Some Results on Fast Correlation Attacks

Fredrik Jönsson

LUND UNIVERSITY

Ph.D. Thesis, 24 May, 2002

Fredrik Jönsson
Department of Information Technology
Lund University
P.O. Box 118
S-221 00 Lund, Sweden
e-mail: fredrik.jonsson@it.lth.se
http://www.it.lth.se/fredrikj

Printed in Sweden
KFS AB, 2002

ISRN: LUTEDX/TEIT-02/1020-SE
ISBN: 91-7167-024-6

Abstract

This thesis presents new results on fast correlation attacks on stream ciphers. In particular, fast correlation attacks on stream ciphers containing linear shift registers with an arbitrary number of taps, are considered.

A general introduction to stream ciphers and correlation attacks is given. The introduction also presents standard properties of linear feedback shift registers and Boolean functions.

Three different algorithms for fast correlation attacks are presented. The first algorithm transforms a part of the code stemming from the LFSR sequence into a convolutional code, and decodes the convolutional code using the Viterbi algorithm. A theoretical analysis for the algorithm is performed, using a random coding bound for convolutional codes. This algorithm is then modified by using Turbo code techniques. The third algorithm is based on a method to recover multivariate linear polynomials.

An overview and a comparison of recently proposed algorithms for fast correlations attacks, is given.

The LILI-128 keystream generator, a recent stream cipher proposal, is attacked by a fast correlation attack.

Several stream ciphers working over an extension field have been proposed in the last few years. Most algorithms for fast correlation attacks are described over the binary alphabet. An algorithm is presented that generalizes previous work to an attack over any field.

This thesis also propose a new algorithm for decoding a general linear code. This decoding problem have several applications in cryptology, such as the McEliece public key cryptosystem, the Stern identification scheme, and also in correlation attacks.

Contents

Preface

This thesis reports on my work as a Ph.D. student at the Department of Information Technology at Lund University. During this time parts of the material have been presented at various conferences and in journals.

Parts of the material have appeared in the following papers:

- T. Johansson and F. Jönsson, "Improved fast correlation attacks on stream ciphers via convolutional codes", *Advances in Cryptology — EUROCRYPT '99*, Prague, Czech Republic, LNCS 1592, Springer-Verlag, pp. 347–362, 1999.

- T. Johansson and F. Jönsson, "Fast correlation attacks based on turbo code techniques", *Advances in Cryptology — CRYPTO '99*, Santa Barbara, USA, LNCS 1666, Springer-Verlag, pp. 181–197, 1999.

- T. Johansson and F. Jönsson, "Fast correlation attacks through reconstruction of linear polynomials", *Advances in Cryptology — CRYPTO 2000*, Santa Barbara, USA, LNCS 1880, Springer-Verlag, pp. 300–315, 2000.

- F. Jönsson and T. Johansson, "A fast correlation attack on LILI-128", *Information Processing Letters*, Elsevier Science, vol. 81, pp. 127–132, 2002.

- T. Johansson and F. Jönsson, "On the complexity of some cryptographic problems based on the general decoding problem ", to appear in *IEEE Transaction on Information Theory*, 2002.

- ▶ T. Johansson and F. Jönsson, "Theoretical analysis of a correlation attack based on convolutional codes", to appear in *IEEE Transaction on Information Theory*, 2002.

Parts of the material have also appeared as abstract in the following conference procedings:

- ▶ T. Johansson and F. Jönsson, "On the complexity of some cryptographic problems based on the general decoding problem", *Proceedings of 1998 IEEE International Symposium on Information Theory*, Cambridge, Mass., USA, p. 442, 1998.
- ▶ F. Jönsson and T. Johansson, "Theoretical analysis of a correlation attack based on convolutional codes", *Proceedings of 2000 IEEE International Symposium on Information Theory*, Sorrento, Italy, p. 212, 2000.
- ▶ F. Jönsson and T. Johansson, "Correlation attacks on stream ciphers over GF(2^n)", *Proceedings of 2001 IEEE International Symposium on Information Theory*, Washington, D.C., USA, p. 140, 2001.

Acknowledgments

I would like to thank some persons without whose support this thesis would not have been possible.

First of all, Thomas Johansson, my supervisor, whose help, encouragement and patience helped me through the difficult moments.

Then, friends and colleagues at the Department of Information Technology for making it such an inspiring environment to work in. Especially, my fellow collegues in the crypto group, Patrik Ekdahl and Enes Pasalic, for discussions about research and life in general, and also for proof-reading of parts of this thesis. Here, I would also like to thank Stefan Höst for all help with LaTeXrelated issues.

Finally, I would like to thank my family, for always supporting me in my work. Especially, my wife Patricia, and our children Emelie and Hampus, for their love and for giving me the best moments in life.

Fredrik Jönsson

This work was supported by the Swedish Foundation for Strategic Research under the Personal Computing and Communication Grant.

1 Introduction

Today information is often transmitted over digital transmission links and networks. Most of these networks consist of fixed media wires such as copper cables and optical fibres. In the last few years, technology has made it possible also to build wireless networks. Several standards for short-range radio data communication have been developed. These standards offer both mobility and high-speed communication.

A radio channel has some characteristics that need to be taken into account when discussing the security of a communication system. The radio channel is more vulnerable to eavesdropping and jamming compared to fixed media channels. In a mobile network, the terminals often have low power requirements.

When information is transmitted over an open channel we must take precautions to protect the data. The message that is transmitted on the channel should only be readable for the intended receivers. A receiver with a received message should also be able to verify that the message was sent by the claimed sender and that it has not been tampered with during the transmission.

The security issues above, amongst others, are studied in the area of *cryptology*. *Cryptology* is the study of mathematical techniques related to aspects of information security. Cryptology involves both the design of cryptosystems, called *cryptography*, and security analysis, called *cryptanalysis*.

The main topic of this thesis is named correlation attacks on stream ciphers. Stream ciphers is a class of cryptosystems for *confidentiality*, that is, to ensure that the information is kept private to the authorized users. Confidentiality is also called *secrecy* or *privacy*. Stream ciphers have many properties that make them suitable in mobile environments.

A correlation attack is a general method for attacking stream ciphers. For many stream ciphers, correlation attacks have given very efficient attacks.

This introductory chapter is organized as follows. Section 1.1 contains a deeper introduction to cryptography. Furthermore, in Section 1.1 a more formal definition of a cryptosystem for confidentiality is given. Some basic definitions of cryptanalysis is given in Section 1.2. Finally, in Section 1.3 an outline of the thesis is given.

1.1 An Introduction to Cryptography

The history of cryptology is long and there are several fascinating stories to tell. Different systems for encrypting messages have been used by governments and the military to prevent national or military secrets to be revealed by enemies. One of the most famous cipher systems is Enigma, used by Germany in the World War II. It has been argued that when Enigma was broken by the allied, the outcome of the war was changed. Two books that describes classical cryptosystems in a non-technical fashion are *The Codebreakers* by Kahn [48] and *The Code Book* by Singh [87].

The open scientific study of cryptology started around World War II. One of the first and most celebrated scientific papers in the area of cryptology was written by Claude E. Shannon [80]. In [80] Shannon used information theory to give a theoretical foundation of cryptology. From this pioneering work, cryptology as a science has grown. Cryptology uses ideas not only from information theory but also from other fields such as, computer science, probability theory, number theory and abstract algebra.

The traditional way of using cryptography is for confidentiality. A model of a cryptosystem used for *privacy* is illustrated in Figure 1.1. The cryptographic model in Figure 1.1 can be described in the following way. Alice wants to send a private message m to Bob. The message is taken from a set of possible messages or (plaintexts), denoted by $\mathcal{M}$. Since the eavesdropper, here called Eve, can listen to the communication, Alice needs to encrypt the message to keep it private. The two users, Alice and Bob, share a secret key K, where K is taken from the set of possible keys, $\mathcal{K}$, called the *keyspace*. For each $K \in \mathcal{K}$, there is an encryption function, denoted by E_K, that maps a message $m \in \mathcal{M}$ into a ciphertext $c \in \mathcal{C}$, where $\mathcal{C}$ is the set of possible ciphertexts, i.e., $c = E_K(m)$. The ciphertext c is then transmitted over the insecure channel. The key K also specifies a decryption function D_K, where the decryption function is chosen such that $m = D_K(E_K(m))$ for all possible keys K and messages m. Bob, who has the same key as Alice, can retrieve the plaintext m after receiving the ciphertext c by applying the decryption function $m = D_K(c)$.

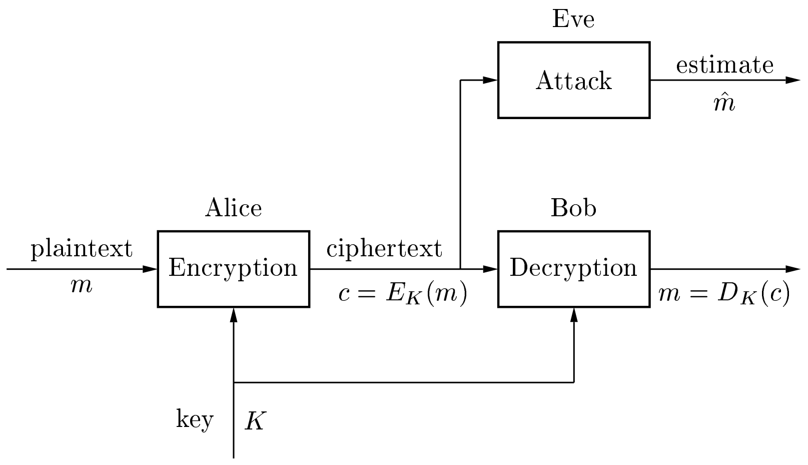

Figure 1.1: Model of a classic cryptosystem

Since Eve does not know the secret key she cannot use the correct decryption function to get the plaintext m. What she can do, however, is to mount an attack and try to estimate the plaintext without knowing the key. It is important to note that we assume that Eve knows all properties of the cryptosystem except the actual key, K, that was used to encrypt the message.

As an example of a classical cryptosystem we describe the Vigenère cipher. The message $\mathbf{m} = m_0, m_1, \ldots$ and the ciphertext $\mathbf{c} = c_0, c_1, \ldots$ are both sequences of letters from the English alphabet, i.e., $m_i, c_i \in \{A, B, \ldots, Z\}$. The key K is a sequence of letters from the English alphabet of length k, $\mathbf{K} = K_1, K_2, \ldots, K_k$. To encrypt a message, the message and the key are transformed to integer sequences by the transformation, $A \leftrightarrow 0, B \leftrightarrow 1, \ldots, Z \leftrightarrow 25$. Denote the integer sequence corresponding to the message and the key by $\mathbf{m}'$ and $\mathbf{K}'$, respectively. The sequences $\mathbf{m}'$ and $\mathbf{K}'$ are then used to calculate a new sequence $\mathbf{c}' = c'_1, c'_2 \ldots$, where

$$c'_i = m'_i + K'_{i \bmod k} \bmod 26, \quad i = 0, 1, 2, \ldots.$$

From the sequence $\mathbf{c}'$ the ciphertext $\mathbf{c}$ is given by the same transformation that was used to transform the message to the integer sequence $\mathbf{m}'$.

Example 1.1: We use a Vigenère cipher to encrypt the message $\mathbf{m} = TOBEORNOTTOBE$. Let the key be $K = HAMLET$. First, the message is transformed into the sequence $\mathbf{m}' = 19, 14, 1, 4, 14, 17, 13, 14, 19, 19, 14, 1, 4$, and the key is transformed into $\mathbf{K}' = 7, 0, 12, 11, 4, 19$. The sequence $\mathbf{c}'$ is then calculated, and we get $\mathbf{c}' = 0, 14, 13, 15, 18, 10, 20, 14, 5, 4, 18, 20, 11$. Finally, we get the ciphertext $\mathbf{c} = AONPSKUOFESUL$. □

In the classical model of cryptography as illustrated in Figure 1.1, Alice and Bob share the same key. This is usually called symmetric-key encryption. In [24] W. Diffie and M. E. Hellman proposed another model for encryption using two separate keys, one for Alice and one for Bob. The elegant property of this model is that the encryption key used by Alice does not have to be kept secret. Since the encryption key can be publicly known, such a system is called a *public-key cryptosystem*. The decryption key, used by Bob, still has to be private and known only to Bob.

The security of public-key cryptosystems relies on the existence of *one-way functions* and *trapdoor one-way functions*.

Definition 1.1: A function f is called a *one-way function* if it is computationally easy to compute $y = f(x)$ for all x but for almost all y it is computationally infeasible to compute the inverse $x = f^{-1}(y)$. □

Formally, one should also provide definitions of the concepts computationally easy and computationally infeasible, mentioned in the definition. However, for our treatment the intuitive meaning is enough. It is not possible to use a one-way function directly for encryption. The receiver Bob has to have an efficient decryption function. Thus we need a trapdoor one-way function.

Definition 1.2: A function f is called a *trapdoor one-way function* if it is a one-way function with the additional property that the inverse is easy to compute if we have knowledge of some extra information, called the trapdoor. □

A *public-key cryptosystem* can be described in the following way. Assume that Bob wants to set up a public-key cryptosystem such that Alice can send a private message to Bob. Bob choose a trapdoor one-way function, denoted by E_e, and transmits it to Alice over the insecure channel. Since Bob know the trapdoor, he can also find the inverse function, denoted by D_d, that is kept private by Bob. The function E_e is then used as the encryption function. As before, let $\mathcal{M}$ be the set of possible plaintexts and $\mathcal{C}$ be the set of possible ciphertexts. Alice can then encrypt a message $m \in \mathcal{M}$ as $c = E_e(m)$. Bob knows the inverse function and can decrypt the ciphertext c as $m = D_d(c)$. Due to the one-way property of E_e, an eavesdropper cannot calculate the plaintext m, even if both E_e and c were observed.

The first public-key cryptosystem was the RSA cryptosystem by Rivest, Shamir, and Adleman [76]. The trapdoor one-way function used in the RSA cryptosystem is based on the difficulty of factoring large integers. Since then, several other public-key cryptosystems have been proposed. However, all public-key cryptosystems used in practice today are much slower than modern symmetric-key cryptosystems, which make them less suitable for encrypting large amounts of plaintext.

The description of cryptology in this section has focused on *privacy*, i.e., methods to make sure that a message can only be read by authorized users. There are also a lot of other subjects in cryptology and a brief description of some other subjects will be made. Assume that we have received a message that is claimed to have been sent from Alice. One question we can ask, is if this message is correct and has not been tampered with by an opponent during the transmission. Cryptographic solutions to make sure that a message has not been altered during transmission is usually called *data integrity*. Another question is how we can be sure that it actually was Alice that transmitted this message. This is called *data origin authentication*. There may also be a need for *entity authentication*, where an entity, or user, can identify itself to other users.

For a more thorough treatment of different subjects in cryptology, different textbooks are available, see for instance [65, 78, 90].

1.2 An Introduction to Cryptanalysis

In Section 1.1 different cryptosystems were discussed. A natural question to ask is how secure those systems are. If a weakness has been found in a cryptosystem, we say that the cryptosystem is broken. It is hard to give a precise definition for when a cryptosystem is broken. The definition of broken will depend on what kind of cryptosystem that has been used, and also on the method used to break it. A definition that often is used for symmetric-key cryptosystems is the the cryptosystem is considered to be broken if we can find the key or the plaintext faster than by an exhaustive key search. However, other definitions can also be considered.

The discussion above leads us into cryptanalysis, the study of the security of cryptographic primitives and functions. When discussing the security of cryptosystems we usually distinguish between *unconditionally* and *conditionally* secure cryptosystems. A unconditionally secure cryptosystem cannot be broken, even with infinite computational resources. The security of a conditionally secure cryptosystem relies on some assumption of the opponent's computing power.

Ideally, we would like to have cryptosystems that are unconditional secure. In [80] Shannon used information theory to analyze unconditional security of cryptosystems. One example of an unconditionally secure cryptosystem is the *one-time pad*.

Definition 1.3: The one-time pad is a cryptosystem where $\mathcal{M} = \mathcal{C} = \mathcal{K} = \mathbb{F}_2^n$. The key $\mathbf{K} = K_1, K_2, \ldots, K_n$ is randomly chosen and may never be reused. A message $\mathbf{m} = m_1, m_2, \ldots, m_n$ is encrypted into the ciphertext $\mathbf{c} = c_1, c_2, \ldots, c_n$, by calculating

$$c_i = m_i + k_i \quad 1 \leqslant i \leqslant n.$$

□

To set up a one-time pad cryptosystem, we need to transmit a key that is at least as long as the plaintext over a secure channel. Then we could instead transmit the plaintext directly over the secure channel. However, such a system might be useful in the case where we have access to a secure channel only in a small period of time. Then we can use this time period to send a key over the channel. This transmitted key can the be used at a later time to communicate over an insecure channel.

From the discussion above we conclude that in most cases an unconditionally secure cryptosystem is impractical. Thus, one must rely on cryptosystems that are conditionally secure. For some cryptosystems one can prove that breaking the cipher is equivalent to solving a computational problem that is supposed to be hard. Examples of such problems are factoring large integers and solving the discrete logarithm problem. This kind of arguments for the security of a cryptosystem is called *provable security*.

For most of the symmetric-key cryptosystems one must rely on a much weaker definition of conditionally secure cryptosystems. The security of a cipher is given as the complexity, or number of operations, of the currently best known attack. For ciphers with such security assumptions there is much research ongoing for improving the best known attacks. A typical example of such research is improvements of fast correlation attacks on stream ciphers.

The different attacks that can be mounted on a cryptosystem can be divided into different classes depending on the power of the attacker. Four of the different classes of attacks are the following.

ciphertext-only This is the worst case for the cryptanalyst. In ciphertext-only attacks the cryptanalyst tries to recover the plaintext or the key, when he observes only the ciphertext.

known-plaintext The aim of a known plaintext attack is to deduce the key, when the cryptanalyst knows some plaintext and the corresponding ciphertext.

chosen-plaintext Like a known plaintext attack, the aim of a chosen plaintext attack is to recover the key. In this case the cryptanalyst can choose any plaintext and gets the corresponding ciphertext.

chosen-ciphertext This case has similarities with chosen plaintext attacks. The difference is that we assume that the cryptanalyst have access to the decryption unit and can decrypt any ciphertext.

The final topic to be discussed in this introductory chapter is the aim of the attack. The most basic aim of an attack is to recover the key from the given ciphertext and eventually some plaintext. This is called a *key recovery attack*. Another common type of attack do not try to recover the key, but instead it directly uses the observed information to recognize the plaintext that was encrypted, or statistical properties of the plaintext.

1.3 Outline of the Thesis

The remaining parts of this thesis are organized as follows. In Chapter 2 an introduction to design and analysis of stream ciphers is given. Linear feedback shift registers, LFSRs, and Boolean functions are two common building blocks of stream ciphers. These two building blocks are defined, and some important properties are given. Chapter 2 also describes some stream cipher proposals, and construction methods.

One of the most efficient types of attacks on stream ciphers are correlation attacks. The principle of correlation attacks is to recover the initial state of one LFSR independently of the other LFSRs in the stream cipher. Fast correlation attacks are improvements of fast correlation attacks, where the initial state is found faster than exhaustive search over the entire state space. Fast correlation attacks are also the main subject of this thesis. Chapter 3 will give an introduction to correlation attacks. A correlation attack is often modelled as a decoding problem. This model is also introduced in Chapter 3.

A fast correlation attack can be modelled as an efficient algorithm for the problem of decoding a received vector to the closest codeword in a general code. In Chapter 4 it is shown how fast correlation attacks can be developed from the problem of decoding a convolutional code. For convolutional codes we can use the Viterbi algorithm to decode efficiently. By using results from convolutional coding it is also possible to derive theoretical results on the performance of the attack.

Turbo codes have proved to give very good performance results in coding theory. Turbo codes are constructed through parallel concatenation of several component codes, where each component code is a convolutional code. Turbo codes are decoded by the BCJR algorithm, which is a decoding algorithm that calculates the a posteriori probabilities of the information symbols. In Chapter 5 we use techniques from turbo codes to build fast correlation attacks.

The main disadvantage with the Viterbi algorithm and the BCJR algorithm is their memory requirements. One wants to find algorithms for correlation attacks that have the same performance as the algorithms based on convolutional codes and turbo codes, but have much smaller memory complexity. In Chapter 6 we present such an algorithm based on the problem of reconstructing a multivariate linear polynomial given noisy observations of points on the polynomial.

Several other proposals for fast correlation attacks have recently been proposed by other authors. In Chapter 7 we shortly review some of them. We also compare the performance of different algorithms for fast correlation attacks.

For some of the proposals there exist theoretical results for the performance. In Chapter 8 we show the importance of such results by an analysis of LILI-128, a recent stream cipher proposal. This chapter also presents some other results that can be interesting for attacks on certain stream ciphers.

To construct an extremely fast stream cipher one can use LFSRs over an extension field. The size of the extension field is chosen to match the word length of computers and processors. However, all of the attacks above are presented over binary field. In Chapter 9 we develop a model for fast correlation attacks over an extension field.

In Chapter 10 we leave the fast correlation attacks, and consider instead what we call the general decoding problem. This problem has several cryptographic applications. One of these applications is in fact correlation attacks. A new algorithm for the general decoding problem is proposed in this chapter.

Finally, in Chapter 11 we give some final remarks and conclusions. The most important contributions of this thesis are summarized.

2 Stream Ciphers

In Section 1.1, cryptosystems for confidentiality were introduced. They were divided into symmetric-key and public-key cryptosystems. Symmetric-key primitives can further be divided into *block ciphers* and *stream ciphers.* In a block cipher, we break up the message to encrypt into blocks of a fixed length. The message is then encrypted one block at a time. There exist several block cipher proposals and standards that are used in practice. For an overview of some of them, see [65, Chapter 7].

Stream ciphers, on the other hand, encrypt each message symbol individually with a time-varying function. Stream ciphers have several properties that make them suitable for use in telecommunication applications. Stream ciphers are in general fast and have low hardware complexity. They also have limited error propagation, and since each symbol is encrypted individually there is no need for a large buffer.

This introductory chapter to stream cipher is organized as follows. In Section 2.1 the basic principles of stream ciphers are given. Several stream ciphers are based on linear feedback shift registers, LFSRs. Some properties of LFSRs that are important for analysis of stream ciphers are given in Section 2.2. Nonlinear Boolean functions can be used to destroy the linearity of the sequences generated by LFSRs. In Section 2.3 an introduction to Boolean functions are given. Some construction methods for LFSR based stream ciphers are given in Section 2.4. Section 2.4 also presents some LFSR based stream cipher that have been proposed. There also exist proposals of stream ciphers that are not based on LFSRs. A discussion of such stream ciphers is given in Section 2.5. In Section 2.6 an overview of different methods for attacking stream ciphers is given.

2.1 Introduction to Stream Ciphers

In this section some basic principles of stream ciphers will be presented. First we start by giving a definition of stream ciphers. Let $\mathcal{M}$ be the set of possible plaintext symbols, $\mathcal{C}$ be the set of possible ciphertext symbols, $\mathcal{Z}$ be the set of possible keystream symbols, and let $\mathcal{K}$ be the set of possible keys. Furthermore, let $\mathbf{m} = m_1, m_2, \ldots$ be a plaintext sequence that we are going to encrypt. The stream cipher contains a keystream generator that produces a pseudorandom sequence, called the keystream, $\mathbf{z} = z_1, z_2, \ldots$, where $z_i \in \mathcal{Z}$ for $i \geqslant 1$. In general, the ith symbol in the keystream, z_i, is a function of the key $K \in \mathcal{K}$ and the previous plaintext symbols $m_1, m_2, \ldots, m_{i-1}$, i.e., $z_i = f_i(K, m_1, m_2, \ldots, m_{i-1})$. The keystream together with an encryption function, $e_{z_i}(m)$, are used to encrypt the message $\mathbf{m}$ symbol by symbol, as

$$c_i = e_{z_i}(m_i), \quad i = 1, 2, \ldots,$$

where c_i, z_i, and m_i are the ith symbols in the ciphertext, keystream, and plaintext sequence, respectively. To decrypt, the receiver also produces the same keystream sequence and decrypt with the keystream and a decryption function. Often, the plaintext, the ciphertext, and the keystream sequence are all sequences of binary digits, that is $\mathcal{M} = \mathcal{C} = \mathcal{Z} = \mathbb{F}_2$.

Depending on the structure of the keystream generator we can divide stream ciphers into two categories, *synchronous* and *self-synchronizing*.

Definition 2.1: In a *synchronous* stream cipher, the keystream is generated independently of the plaintext and the ciphertext. □

A synchronous stream cipher can be represented by a finite state machine, as illustrated in Figure 2.1. Let the state of the keystream generator at time i be denoted by σ_i. The next state σ_{i+1} is then determined by a next state function f, which takes the current state, σ_i, and the key, K, as input, i.e.,

$$\sigma_{i+1} = f(\sigma_i, K).$$

Then, the keystream symbol z_i is produced as a function of the current state and the key, denote this function by g,

$$z_i = g(\sigma_i, K).$$

The key K will also determine the initial state σ_0 of the keystream generator.

Since the keystream from a synchronous keystream generator neither depends on the plaintext, nor on the ciphertext, there is no error propagation. I.e., if a certain symbol in the ciphertext has been corrupted by transmission error, the rest of the ciphertext will not be affected.

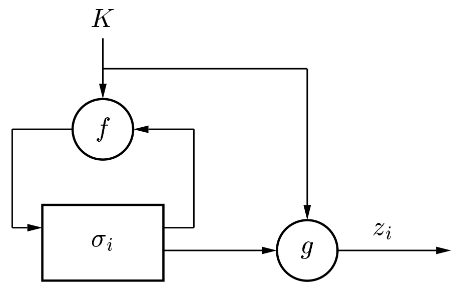

Figure 2.1: The keystream generator of a synchronous stream cipher as a finite state machine.

To be able to decrypt correctly the receiver has to be in perfect synchronization with the sender. If synchronization is lost the decryption will not work correctly and the information is lost. Thus, there is a need for mechanisms for detecting lost synchronization and for re-initialization.

Due to the synchronization property, synchronous stream ciphers are vulnerable to active attacks, where an adversary can insert or delete symbols to the ciphertext sequence. It will also make it possible for an adversary to change some of the ciphertext symbols and still create a valid ciphertext sequence. Thus, we need to use additional techniques to guarantee message authentication.

One of the most common types of synchronous stream cipher is the *binary additive stream cipher*. A binary additive stream cipher is a synchronous stream cipher where the plaintext, ciphertext, and keystream all are binary sequences, and furthermore, the encryption function is XOR, i.e.,

$$c_i = m_i + z_i,$$

see Figure 2.2.

As mentioned previously in this section, another type of stream ciphers is the *self-synchronizing* stream ciphers.

Definition 2.2: A self-synchronizing or asynchronous stream cipher is a stream cipher where the keystream is generated as a function of the key, K, and at most t previous ciphertext symbols. □

As for synchronous stream ciphers we can define a state σ_i also for a self-synchronizing stream cipher. Here the state is taken as the t previous cipher-

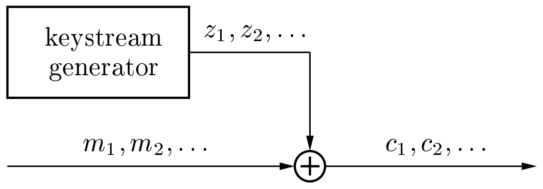

Figure 2.2: Principle of binary additive stream ciphers

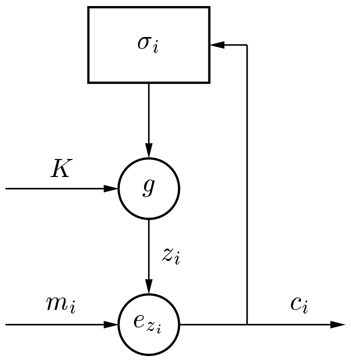

Figure 2.3: Principle of self-synchronizing stream ciphers.

text symbols,

$$\sigma_i = (c_{i-1}, c_{i-2}, \ldots, c_{i-t}).$$

The ith keystream symbol, z_i, is then generated as a function, denoted by g, of the initial state and the key,

$$z_i = g(\sigma_i, K).$$

From the definition of the state we observe that we need to have an initial state defined by a initial value for $i < 0$. This initial value may be public. The principle of self-synchronizing stream ciphers is illustrated in Figure 2.3.

Since the state depends only on the last t ciphertext symbols, the keystream will automatically be re-synchronized after a limited time, if some ciphertext symbols are lost during transmission. If a single error occurs on the channel, the decryption of the next t ciphertext symbols will be affected. Thus, the error propagation is worse for self-synchronizing stream ciphers

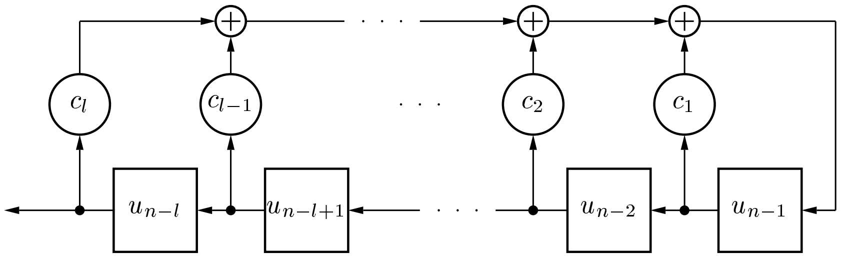

Figure 2.4: General form of a linear feedback shift register.

compared with synchronous stream ciphers. The self-synchronization property will also make it harder to detect insertion or deletion of false ciphertext digits by an active adversary. Thus, there is a need for additional methods to guarantee message authentication.

There exists several general methods for construction of stream ciphers. One common method is to use a block cipher in different feedback modes, see [65, Section 7.2.2]. In fact, this is the most common way of constructing self-synchronizing stream ciphers. For synchronous stream ciphers other constructions are also common. Many synchronous keystream generators that are based on linear feedback shift registers have been proposed. Before we describe some stream cipher proposals, we introduce some concepts from linear feedback shift registers and Boolean functions.

2.2 Linear Feedback Shift Registers

A common component in keystream generators is a *linear feedback shift register*, LFSR. A linear feedback shift register produces a sequence, $\mathbf{u} = u_0, u_1, \ldots$, satisfying the linear recurrence function,

$$u_n = \sum_{j=1}^{l} c_j u_{n-j}, \quad n = l, l+1, \ldots,$$

where l is the length of the LFSR, and $u_i \in \mathbb{F}_q$, $i \geqslant 1$. The general form of a linear feedback shift register is illustrated in Figure 2.4. A LFSR consists of l delay elements, where each delay element, also called *stage*, can store an element, or digit, in $\mathbb{F}_q$. The l stages, $(u_{n-l}, u_{n-l+1}, \ldots, u_{n-1})$, is together called the *state* of the shift register. Each *feedback coefficient* c_j, $j = 1, \ldots, l$ is an element in $\mathbb{F}_q$. Using the feedback coefficients, we define the *feedback polynomial*, or connection polynomial, to be $g(x) = 1 - c_1 x - c_2 x^2 - c_{l-1} x^{l-1} - c_l x^l$.

As an alternative to the feedback polynomial one can use the *characteristic polynomial*, $f(x) = x^l - c_1x^{l-1} - c_2x^{l-1} - \cdots - c_{l-1}x - c_l$. The first L output symbols, $u_0, u_1, \ldots, u_{l-1}$, are initially loaded into the l stages. These symbols loaded into the LFSR, together form the *initial state*.

The output sequences from linear feedback shift registers have many interesting properties. For the properties presented here it is assumed that the feedback polynomial is *non-singular*, i.e., $c_l \neq 0$. Let $\mathbf{u} = u_0, u_1, \ldots$ be a LFSR output sequence. For each $\mathbf{u} = u_0, u_1, \ldots$, there exists a positive integer T, called the *period*, such that $u_i = u_{i+T}$ for all $i \geqslant 0$.

Definition 2.3: The feedback polynomial $g(x)$ is called *irreducible* if it cannot be written as the product of two polynomials with coefficients in $\mathbb{F}_q$ and positive degree. If the root x of an irreducible polynomial $g(x)$ of degree l is a generator of the multiplicative group of all the non-zero elements of $\mathbb{F}_{q^l}$, $g(x)$ is called a *primitive* polynomial. □

For a LFSR with a primitive feedback polynomial we have the following theorem that we state without a proof [35].

Theorem 2.1: Consider a LFSR of length l and feedback polynomial $g(x)$, where $g(x)$ is a primitive polynomial of degree l over $\mathbb{F}_q$. Then each of the $q^l - 1$ non-zero initial states of the LFSR produces a sequence with period $q^l - 1$. □

A LFSR with a primitive feedback polynomial is called a maximum-length LFSR. One can also show that the output sequence from maximum-length LFSRs has many nice statistical properties that are desirable in a keystream generator, provided that the LFSR is not initialized with all zeroes [65].

Another interesting property of linear feedback shift registers is that they can be used to generate any sequence of finite length. Assume that we have a given sequence, denoted by $\mathbf{u}^n$, of length n, $\mathbf{u}^n = u_0, u_1, \ldots, u_{n-1}$. Since we know that there is at least one LFSR that can generate $\mathbf{u}^n$, we would like to find the shortest LFSR that can generate $\mathbf{u}^n$. We do not care what comes after the n symbols. To calculate the shortest LFSR we can use the Berlekamp-Massey algorithm [59]. If the sequence length is n, the complexity of finding the shortest LFSR using the Berlekamp-Massey algorithm is $O(n^2)$. The length of the shortest LFSR that can generate a given sequence, $\mathbf{u}^n$, of length n, is called the *linear complexity* of $\mathbf{u}^n$, denoted by $L(\mathbf{u}^n)$. If the sequence is the all-zero sequence, $\mathbf{u}^n = 0, 0, \ldots$, the linear complexity is defined to be zero.

The main drawback with LFSR sequences is that if we are given a sequence of L consecutive output symbols, then, due to the linearity, we can calculate the output symbol at an arbitrary time instance. Hence, one cannot use a maximum-length LFSR directly as a keystream generator. Instead one needs to combine several LFSRs in different ways, to destroy the linearity and get a sequence with good random properties and a large period.

A standard method to produce binary random-like sequences from LFSR sequences is to combine the output of several binary LFSRs by a nonlinear function f with some desired properties. Here f is a Boolean function in n variables. The purpose is to destroy the linearity of the LFSR sequences and hence provide the resulting sequence with a large linear complexity.

2.3 Boolean Functions

A Boolean function, $f(\mathbf{x})$, takes a binary vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, $x_i \in \mathbb{F}_2$ $1 \leqslant i \leqslant n$, as input and outputs one bit, i.e.,

$$f : \mathbb{F}_2^n \to \mathbb{F}_2 .$$

Boolean functions are used in many different binary keystream generators based on LFSRs. Their purpose in the keystream generators is often to destroy the linearity introduced by the LFSRs. A Boolean function $f(\mathbf{x})$ can uniquely be expressed in algebraic normal form. I.e., there are unique constants $a_0, a_1, \ldots, a_n, a_{12}, \ldots, a_{12\cdots n} \in \mathbb{F}_2$ such that

$$\begin{aligned} f(x_1, x_2 \ldots, x_n) =& a_0 + a_1x_1 + \ldots + a_nx_n \\ & + a_{12}x_1x_2 + a_{13}x_1x_3 + \ldots + a_{12\cdots n}x_1x_2\cdots x_n, \end{aligned}$$

where addition and multiplication are in $\mathbb{F}_2$. If the number of variables in the Boolean function is small, a truth table is often used. A truth table lists the functions output value for all possible inputs. As an example we give the truth table of the Boolean function

$$f(x_1, x_2, x_3) = x_1 + x_1x_2 + x_2x_3$$

in Table 2.1. In the cryptographic applications there are several properties of Boolean functions that are interesting to investigate. For a Boolean function we say that a product of m variables is an m-th order product. The first order products are usually called the linear terms.

Definition 2.4: The algebraic degree, of simply degree, of a Boolean function $f(\mathbf{x})$ is defined to be the number of variables in the highest order product of $f(\mathbf{x})$, when $f(\mathbf{x})$ is written in algebraic normal form. The algebraic degree of $f(\mathbf{x})$ is denoted by $deg(f)$. □

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

Table 2.1: The truth table of the Boolean function $f(x_1, x_2, x_3) = x_1 + x_1x_2 + x_2x_3$.

An n variable Boolean function $f(\mathbf{x})$ is *balanced* if the output column in the truth table contains an equal number of 0's and 1's. Alternatively, $f(\mathbf{x})$ is balanced if $P(f(\mathbf{x}) = 0) = P(f(\mathbf{x}) = 0) = \frac{1}{2}$, when $\mathbf{x}$ is chosen uniformly in $\mathbb{F}_2^n$.

Let $\mathcal{F}_n$ be the set of all Boolean functions in n variables, and let $\mathcal{A}_n$ be the set of all affine functions in n variables. The Hamming distance between two functions $f(\mathbf{x}), g(\mathbf{x}) \in \mathcal{F}_n$ is defined as,

$$d_H(f, g) = |\{\mathbf{x}|f(\mathbf{x}) \neq g(\mathbf{x}), \mathbf{x} \in \mathbb{F}_2^n\}|.$$

Definition 2.5: We define the *nonlinearity* of a Boolean function $f(\mathbf{x})$, denoted by N_f, as the Hamming distance to the nearest affine function, i.e.,

$$N_f = \min_{g \in \mathcal{A}_n} d_H(f, g).$$

□

In most of the cryptographic applications, we would like that the correlation between an individual input variable and the output variable is small.

Definition 2.6: An n variable Boolean function is defined to be t-th order *correlation immune*, if for any t-tuple of independent identically distributed binary random variables $X_{i_1}, X_{i_2}, \ldots, X_{i_t}$, we have

$$I(X_{i_1}, X_{i_2}, \ldots, X_{i_t}; Y) = 0, \quad 1 \leqslant i_1 < i_2 < \cdots < i_m \leqslant n,$$

where $Y = f(X_1, X_2, \ldots, X_n)$, and $I(X;Y)$ denotes the mutual information [22]. $\square$

A Boolean function that is both balanced and t-th order correlation immune is called a *t-resilient function.*

The properties above are often investigated through the Walsh transform.

Definition 2.7: For a Boolean function, $f : \mathbb{F}_2^n \to \mathbb{F}$, the *Walsh transform* of $f(\mathbf{x})$ is defined to be the real-valued function $F(\omega)$ over the vector space $\mathbb{F}_2^n$ given by

$$F(\omega) = \sum_{\mathbf{x}} (-1)^{f(\mathbf{x}) \oplus \omega \cdot \mathbf{x}},$$

where the dot product of vectors $\mathbf{x}$ and ω is defined as $x \cdot \omega = x_1\omega_1 + \ldots + x_n\omega_n$. $\square$

The Hamming distance between a Boolean function $f(\mathbf{x})$ and an affine function $g(\mathbf{x}) = \omega \cdot \mathbf{x} + b$, where $b \in \mathbb{F}_2$, can be calculated with the Walsh transform as

$$d_H(f, g) = 2^{n-1} - \frac{(-1)^b F(\omega)}{2}.$$

Thus, the nonlinearity of $f(x)$ can be obtained from the Walsh transform as

$$N_f = 2^{n-1} - \frac{1}{2} \max_{\omega} |F(\omega)|.$$

In [37] the following theorem to express correlation immunity in terms of Walsh transforms was proven.

Theorem 2.2: A Boolean function is t-th order correlation immune if and only if

$$\mathcal{F}(\omega) = 0, \quad \forall \omega \in \mathbb{F}_2^n | 1 \leqslant w_H(\omega) \leqslant t,$$

where $w_H(\omega)$ is the Hamming weight of ω, i.e., the number of nonzero positions in ω. $\square$

A Boolean function $f(\mathbf{x})$ is balanced if and only if $\mathcal{F}(0) = 0$. Hence, we see that the Walsh transform is an important tool when investigating properties of Boolean functions.

2.4 Some Keystream Generators

In this section methods for construction of LFSR-based stream ciphers are presented. The LFSR-based stream ciphers can be divided into different basic design principles. The basic designs can then be combined to get more complex and hopefully more secure stream ciphers. Three basic construction methods are, *Nonlinear combination generators*, *Nonlinear filter generators*, and *Clock-controlled generators*.

In a nonlinear combination generator several linear feedback shift registers are used in parallel. Denote the number of LFSRs by n. The keystream sequence is generated by a nonlinear Boolean function, f, of the output of the n shift registers. The principle of a combination generator is illustrated in Figure 2.5.

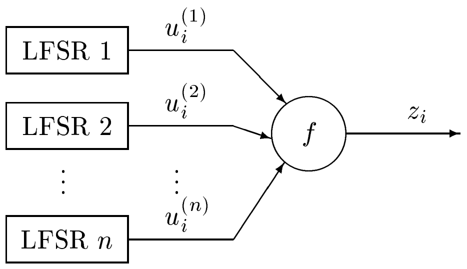

Figure 2.5: Principle of nonlinear combination generators.

To get a secure nonlinear combination generator we need to find a function that is correlation immune and have high nonlinearity. As shown by Siegenthaler [81], there is a tradeoff between these properties. To circumvent this tradeoff, the memoryless function f can be replaced by a finite state machine with memory [63]. A recent stream cipher proposal that is based on this principle is the ciphering function E_0 used in Bluetooth [83].

Instead of using several LFSRs one can use one single LFSR and generate the keystream as a nonlinear function f of the stages of the LFSR. Such a keystream generator is called a nonlinear filter generator. The function f is then called the filtering function. The principle of nonlinear filter generators is illustrated in Figure 2.6. Also for nonlinear filter generators we can replace the memoryless filtering function with a finite state machine. One filter generator with a finite state machine that has been proposed recently is the SNOW cipher [25]. To improve the encryption speed in software, operations in SNOW are defined over $\mathbb{F}_{2^{32}}$.

The third basic method of constructing keystream generators is by clock-controlled generators. In a clock-controlled keystream generator, the output

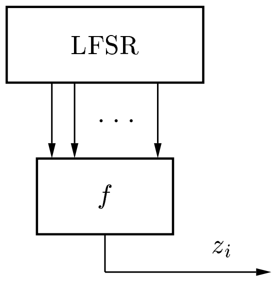

Figure 2.6: Principle of nonlinear filter generators.

of one or several LFSRs controls the clocking of other shift registers. Two examples of clock controlled keystream generators are the shrinking generator [19] and the alternating step generator [36]. The self-shrinking generator [64] is a modification of the shrinking generator with only one LFSR. Another clock controlled generator that is used in practice is the cipher A5 [6,26], used in GSM phones.

As mentioned before, it is also possible to combine the basic construction methods to get more complex generators. A common combination that has been proposed is to use the stages of a clock-controlled shift register as input variables to a nonlinear filtering function. Two proposed families of such generators are the LILI keystream generators [23,86], and the t-class SOBER stream ciphers [77].

2.5 Other Stream Ciphers

Several methods for constructing keystream generators using LFSR as internal components have been given in Section 2.4. In this section we give a brief overview of generators that are not based on linear feedback shift registers. Some keystream generators have been proposed that use the same principles as in Section 2.4 but has replaced the LFSR with another sequence generator. Two common components that has been proposed as an alternative to LFSRs are the *feedback with carry shift register* and the *lagged Fibbonacci generator*.

Feedback with carry shift registers, FCSR, were proposed by Klapper and Goresky [51]. The principle of a FCSR is as follows. In the feedback of the shift registers the content of the tapped stages are added as integers to the content of the memory. The least significant bit of the sum is then fed back to form the new symbol. The remaining higher order bits form the content of the memory. FCSRs can be analyzed using the algebra over the 2-adic numbers, instead of the algebra over finite fields used to analyze LFSRs [52].

The lagged Fibbonacci generator, proposed by Knuth [54, p. 27] is based on addition in $\mathbb{Z}_n$ where n is chosen to match the register size of CPUs. In the feedback the new symbol is calculated as addition modulus n of some of the stages of the generator. The lagged Fibbonaci can be implemented fast in software. However, we can get the same improvement for linear feedback shift register by using a shift register over a finite field $\mathbb{F}_{2^n}$.

There have also been proposed some stream ciphers that are entirely optimized for software implementations. Their structure have very little in common with keystream generators based on LFSRs. Hence, we will not give much attention to these software optimized stream ciphers here. Examples of such stream ciphers are RC4 and SEAL [20]. RC4 is proprietary, but alleged descriptions that are output compatible with certified implementations have been proposed [78].

There also exist stream ciphers that are provably secure conditioned that certain number theoretic problems are hard, e.g. [8,66]. These stream ciphers are much slower than LFSR based stream ciphers and software optimized stream ciphers.

2.6 Cryptanalysis of Stream Ciphers

Most of the analysis that has been performed on synchronous stream ciphers assume a known-plaintext attack. For synchronous stream ciphers a known-plaintext attack is equivalent to an attack on the keystream generator with an observed keystream sequence of a certain length. Assume that the attacker has a message $\mathbf{m} = (m_1, m_2, \ldots, m_N)$, which has been encrypted into the ciphertext $\mathbf{c} = (c_1, c_2, \ldots, c_N)$. From this message-ciphertext pair the keystream can easily be determined. As an example we consider an additive stream cipher where, $c_i = m_i + z_i$. Given $\mathbf{m}$ and $\mathbf{c}$ we can calculate the keystream $\mathbf{z} = (z_1, z_2, \ldots, z_N)$ that was used to encrypt $\mathbf{m}$ as $z_i = m_i + c_i$, for $1 \leqslant i \leqslant N$. Thus, we might assume that the attacker has observed a keystream sequence of length N, $\mathbf{z} = (z_1, z_2, \ldots, z_N)$.

Known-plaintext attacks on synchronous stream ciphers are divided into *key recovery attacks*, where we try to recover the secret key from the observed keystream, and *distinguishing attacks*, where we try to distinguish the observed keystream from a truly random sequence.

To show the importance that a stream cipher withstands a distinguishing attack we use the following example.

Example 2.1: Assume that we have a database where a user can retrieve different files that are stored in the database. Furthermore, assume that the different files have the same size and that they are mutually uncorrelated. For simplicity we can assume that the database contains only two files. Denote these two files by $\mathbf{m}^{(1)}$ and $\mathbf{m}^{(2)}$, respectively. Since it should not be possible for a user to see what files another users downloads the information is encrypted by a stream cipher.

Alice downloads a file and a ciphertext $\mathbf{c}$ is transmitted from the database to Alice. Bob who also can listen to the channel receives the ciphertext. Since Bob does not know Alice's secret key he can not decrypt to find out what file Alice downloaded. However, if a distinguisher exists he can find out what file that was transmitted in the following way.

First, Bob starts by downloading the first file $\mathbf{m}^{(1)}$. Then he creates a vector $\hat{\mathbf{z}} = \mathbf{m}^{(1)} + \mathbf{c}$, using Alice's ciphertext $\mathbf{c}$. If $\mathbf{m}^{(1)}$ was the file that Alice downloaded then $\hat{\mathbf{z}} = \mathbf{m}^{(1)} + \mathbf{c} = \mathbf{z}$, where $\mathbf{z}$ is the keystream that was used to encrypt Alice's message. Otherwise if $\mathbf{m}^{(2)}$ was the downloaded file then $\hat{\mathbf{z}} = \mathbf{m}^{(1)} + \mathbf{m}^{(2)} + \mathbf{z}$, which is a random sequence since $\mathbf{m}^{(1)}$ and $\mathbf{m}^{(2)}$ are uncorrelated. Hence, by applying the distinguisher to the vector $\hat{\mathbf{z}}$ Bob can determine whether $\hat{\mathbf{z}}$ is a random sequence, or a sequence generated from the keystream generator in the stream cipher. □

Distinguishing attacks have been used to attack several recent stream cipher proposals [18, 27, 58]. One can expect that distinguishing attacks will be more important in the future, since several distinguishing attacks have been proposed recently and the theoretical framework has increased.

In a traditionally known-plaintext attack on stream ciphers the attacker tries to recover the secret key given the observed keystream sequence. These attacks are called key-recovery attacks. For this setting of the attack there exists both general methods that are applicable to all synchronous stream ciphers, and also methods that attacks a certain stream cipher proposal.

One of the most important classes of general attacks on stream ciphers is *correlation attacks*. Correlation attacks were first proposed by Siegenthaler [81]. In [62], Meier and Staffelbach presented a modification called *fast correlation attack*. Subsequently, other algorithms for fast correlation attacks have also been proposed. Since fast correlation attacks is the major subject of this thesis, a deeper introduction to fast correlation attacks is given in Chapter 3.

Other classes of key-recovery attacks are the *linear consistency attack* [97] and the *linear syndrome attack* [98] proposed by Zeng, Yang, and Rao. For nonlinear filter generators with certain properties the *inversion attack* [33] by Golić has been shown to be efficient. Some proposals for divide and conquer attacks on stream ciphers are given in [84]. For stream ciphers there also exists general time/memory tradeoff attacks, see [5].

Several attacks on specific stream ciphers have also been proposed, see e.g., [6, 18, 26, 27, 34, 41, 47, 53, 58]

2.7 Summary

In this chapter some basic properties of stream ciphers were presented. We defined synchronous, self-synchronizing, and additive stream ciphers. Two topics that are often used in connection with stream ciphers are LFSRs and Boolean functions. Many important properties of Boolean functions are most easily described by the Walsh transform.

There exist several methods and principles to construct stream ciphers. In this chapter some of these methods were given, along with some stream ciphers that have been proposed. In this chapter we could only give a brief introduction to stream ciphers. To study the design of stream ciphers further see, e.g., [65, Chapter 6].

The chapter ended with a brief introduction to cryptanalysis of stream ciphers. Most attacks on synchronous stream ciphers assume a known plaintext scenario. The attacks were divided into distinguishing attacks and key recovery attacks. The main topic of this thesis is correlation attacks, which is one of the most important classes of key recovery attacks on stream ciphers.

3
Correlation Attacks and Fast Correlation Attacks

In the previous chapter an introduction to design and analysis of stream ciphers was given. In this chapter we will continue with cryptanalysis of stream ciphers and give an introduction to correlation attacks and fast correlation attacks. Correlation attacks is one of the most important general classes of attacks on LFSR based stream ciphers. The original correlation attack was proposed by Siegenthaler in [82]. In his attack nonlinear combination generators were considered. The principle behind correlation attacks is the following. Assume that an attacker can find a correlation between the output of one of the shift registers in the generator and the keystream. The attacker can then mount a divide-and-conquer type of attack and try to find the initial state of this LFSR independently of the other LFSRs.

When applying Siegenthaler's correlation attack, an exhaustive search over all possible initial states of the LFSR is made. In [61, 62] Meier and Staffelbach gave two algorithms for fast correlation attacks, called Algorithm A and Algorithm B, respectively, in which it is not necessary to run through all possible initial states. These algorithms have very good performance when the feedback polynomial has low weight, or if a multiple of the feedback polynomial has low weight and relatively low degree. After this pioneering work, lots of research have been made to improve the performance of fast correlation attacks and also to make fast correlation attacks applicable to feedback polynomials with arbitrary number of taps.

A brief overview of the initial work by Siegenthaler is given in Section 3.1. Correlation attacks can also be modelled as a decoding problem. This model of correlation attacks is introduced in Section 3.2. In Section 3.3 the fast

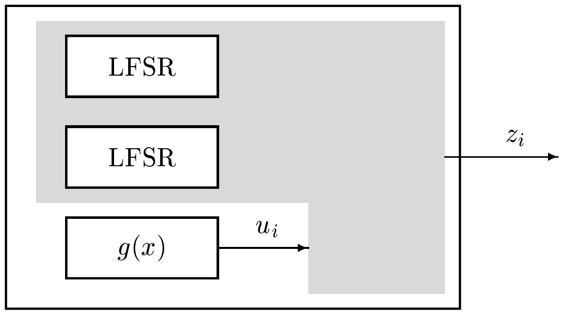

Figure 3.1: A sufficient requirement for a correlation attack, $P(u_i = z_i) \neq 0.5$.

correlation attacks proposed by Meier and Staffelbach are presented. Some of the early improvements of fast correlation attacks are given in Section 3.4.

3.1 Correlation Attacks

The original correlation attack was proposed by Siegenthaler [82]. In [82] the attack was described as a ciphertext only attack. Since a ciphertext only attack requires that there exists redundancy in the plaintext message, one usually applies correlation attacks in known plaintext scenarios, where there are no requirements of redundancy in the message.

Assume that we have observed a keystream sequence, $\mathbf{z}$, of length N, $\mathbf{z} = z_1, z_2, \ldots, z_N$. The keystream sequence is generated from a generator with n different LFSRs. If one can find a correlation between the output of one of the shift registers, called the target LFSR, and the keystream, i.e., $P(u_i = z_i) \neq 0.5$, where u_i is the output of the LFSR and z_i is the known keystream symbol, then one can try to find the initial state in a "divide-and-conquer" type of attack on the target LFSR, see Figure 3.1. Furthermore, we assume that the feedback polynomial $g(x)$ of the target LFSR is known. There is no requirement of any further structure in the generator. The only thing that matters is the fact that we can find a correlation.

In Section 2.4 several standard methodologies for destroying the linear properties of LFSR sequences and produce a keystream sequence with a large linear complexity were given. One of these methodologies is the nonlinear combination generator, illustrated in Figure 3.2.

It is worth noticing that for a nonlinear combination generator there always exists a correlation between the generator output z_i and either one or a set of M LFSR output symbols $\{u_i^{(i_1)}, u_i^{(i_2)}, \ldots, u_i^{(i_M)}\}$. It is well known

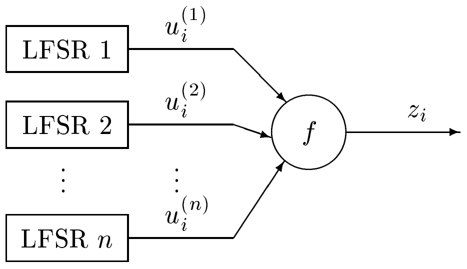

Figure 3.2: Principle of a nonlinear combination generator

that if f is an $(M-1)$-resilient (but not M-resilient) function then there exists a correlation, which can be expressed in the form

$$P(z_i = u_i^{(i_1)} + u_i^{(i_2)} + \cdots + u_i^{(i_M)}) \neq 0.5.$$

It is also known that there is a tradeoff between the resiliency and the nonlinearity of f, and hence M must be rather small [81].

Returning to the previously mentioned correlation attacks, the above overview demonstrates that finding a low complexity algorithm that successfully can use the existing correlation in order to determine a part of the secret key can be a very efficient way of attacking such stream ciphers.

The principle of the original attack proposed by Siegenthaler is the following. Assume that the LFSR we consider has length l and that the correlation between the keystream sequence and the LFSR sequence is $1-p$, where $p < 0.5$. Here we also assume that the feedback polynomials of the LFSRs are known. This is different from the setting in [82] were an additional exhaustive search was made over all primitive polynomials of degree l.

For a binary LFSR of length l there are 2^l possible initial states. For each possible initial state $\mathbf{u}_0 = (u_1, u_2, \ldots, u_l)$ the LFSR output sequence $\mathbf{u} = (u_1, u_2, \ldots, u_N)$ is generated. Let β be defined as $\beta = N - d_H(\mathbf{u}, \mathbf{z})$, where $d_H(\mathbf{u}, \mathbf{z})$ is the Hamming distance between $\mathbf{u}$ and $\mathbf{z}$, i.e., the number of positions in which $\mathbf{u}$ and $\mathbf{z}$ differ.

There are two hypothesis to be considered.

H_1**:** The guessed initial state is correct.

H_0**:** The guessed initial state is wrong.

Under these two hypothesis, β is binomially distributed with mean value

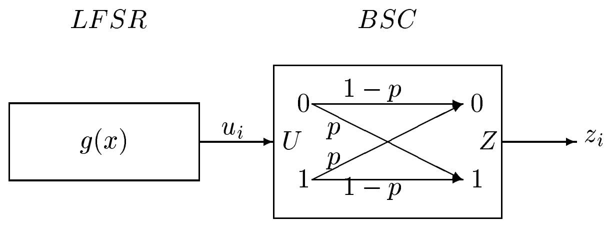

Figure 3.3: Model for a correlation attack

$m_{\beta|H_i}$ and variance $\sigma^2_{\beta|H_i}$, where

$$m_{\beta|H_1} = Np, \quad \sigma^2_{\beta|H_1} = Np(1-p), \tag{3.1}$$

$$m_{\beta|H_0} = \tfrac{N}{2}, \quad \sigma^2_{\beta|H_0} = \frac{N}{4}. \tag{3.2}$$

Hence, we see that if we run through all possible initial states, and if N is large enough, β will with high probability take its largest value when $\mathbf{u}_0$ is the correct initial state.

If a correlation can be found for each of the LFSRs in the generator, the complexity of finding the correct key is reduced from $\prod_{j=1}^{n}(2^{l_j}-1)$ to $\sum_{j=1}^{n}(2^{l_j}-1)$, where l_j is the length of LFSR j and n is the number of shift registers in the generator. The correlation attack by Siegenthaler can also be modelled as a decoding problem, as we will show in the next section.

3.2 Correlation Attacks as a Decoding Problem

Correlation attacks, as illustrated in Figure 3.1, are often viewed as a decoding problem, see for instance [15, 61, 62, 71, 82]. Recall that the LFSR has length l and the set of possible LFSR sequences is denoted by $\mathcal{L}$. Clearly, $|\mathcal{L}| = 2^l$ and for a fixed length N the set of all truncated sequences from $\mathcal{L}$ is also a linear $[N, l]$ block code [57], referred to as $\mathcal{C}$. Thus, the LFSR sequence $\mathbf{u} = (u_1, u_2, \ldots, u_N)$ is regarded as a codeword from $\mathcal{C}$ and the keystream sequence $\mathbf{z} = (z_1, z_2, \ldots, z_N)$ is regarded as the received channel output. From the definition of the correlation between u_i and z_i, we can describe each z_i as the output from the binary symmetric channel, BSC, when u_i was transmitted. The correlation probability $1-p$, defined by $1-p = P(u_i = z_i)$, gives p as the crossover probability (error probability) in the BSC. Without loss of generality we can assume $p < 0.5$. This is all shown in Figure 3.3.

The cryptanalyst's problem can be formulated as follows. Given a length N received word $\mathbf{z} = (z_1, z_2, \ldots z_N)$ as output of the BSC(p), find the length

N codeword from $\mathcal{C}$ that was transmitted.

From simple coding arguments [80], it can be shown that the length N should be at least around $N_0 = l/(1-h(p))$ for unique decoding, where $h(p)$ is the binary entropy function. If the length of the output sequence N is modest but allows unique decoding, say $N = N_0 + D$, where D is a constant, the fastest methods for decoding are probabilistic decoding algorithms. This decoding problem is studied deeper in Chapter 10, where algorithms applicable for this case are given.

3.3 Fast Correlation Attacks

For received sequences of large length *fast correlation attacks* [61, 62] are sometimes applicable. By fast correlation attacks we mean attacks that are significantly faster than exhaustive search over the target LFSR. If exhaustive search has computational complexity $O(2^l)$ then a fast correlation attack should have computational complexity at most $O(2^k)$, where $2^k \ll 2^l$. In [62] Meier and Staffelbach presented two algorithms for fast correlation attacks. Instead of the exhaustive search as originally suggested in [82], the algorithms are based on using certain parity check equations created from the feedback polynomial of the LFSR. The algorithms have two different phases. In the first phase, a set of suitable parity check equations is found. The second phase uses these parity check equations in a fast decoding algorithm to recover the transmitted codeword, and hence, the initial state of the LFSR.

Parity check equations in [62] were created in two separate steps. Let $g(x) = 1 + c_1x + c_2x^2 + \ldots + c_lx^l$ denote the feedback polynomial, and t the number of taps of the LFSR, i.e., the number of nonzero coefficients of $g(x)$ is $t+1$. Symbol number i of the LFSR sequence, u_i, can then be written as $u_i = c_1u_{i-1} + c_2u_{i-2} + \ldots + c_lu_{i-l}$. Since the weight of $g(x)$ is $t+1$, we get in this way $t+1$ different parity check equations for u_i. Secondly, using the fact that $g(x)^j = g(x^j)$ for $j = 2^k$, parity check equations are also generated by repeatedly squaring the polynomial $g(x)$. The obtained parity check equations are then valid in each index position of $\mathbf{u}$. The total number of parity check equations, denoted m, that can be found in this two steps is

$$m \approx (t+1)\log(\frac{N}{2l}),$$

where log uses base 2 [62].

The parity check equations can be written as

$$\begin{aligned} u_i + b_1 &= 0, \\ u_i + b_2 &= 0, \\ &\vdots \\ u_i + b_m &= 0, \end{aligned} \tag{3.3}$$

where each b_j is the sum of t different symbols in the LFSR sequence $\mathbf{u}$. If we substitute the symbols of the LFSR sequence with the observed keystream symbols in Equations (3.3) we get the following set of expressions,

$$\begin{aligned} z_i + y_1 &= L_1, \\ z_i + y_2 &= L_2, \\ &\vdots \\ z_i + y_m &= L_m, \end{aligned} \tag{3.4}$$

where y_j is the sum of t different keystream symbols from the same positions as the symbols of the LFSR sequence used in b_j, and $L_i \in \mathbb{F}_2$, $1 \leqslant i \leqslant m$.

In the second phase, the m parity check equations for each position u_i, $1 \leqslant i \leqslant N$ is used in a decoding algorithm. If $P(z_i = u_i) = (1-p) = 1/2 + \epsilon$ the probability $s = P(y_j = b_j)$ can be calculated [13] as

$$s = P(y_j = b_j) = \frac{1}{2} + 2^{t-1}\epsilon^t. \tag{3.5}$$

Assume that h out of the m equations in (3.4) hold. Then by using Equation (3.5) we can calculate the conditional probability [62],

$$\begin{aligned} p^\star &= P(z_i \neq u_i | h \text{ equations hold}) \\ &= \frac{p(1-s)^h s^{m-h}}{p(1-s)^h s^{m-h} + (1-p)(1-s)^{m-h} s^h}. \end{aligned} \tag{3.6}$$

Two different decoding methods was suggested in [61,62]. The algorithms were called, Algorithm A and Algorithm B, respectively. The algorithm A is a one pass algorithm were $p^\star$ is calculated for each observed symbol. The l positions with highest value of $(1-p^\star)$ are then used to find the correct initial state of the LFSR.

Algorithm B, which has received most attention is an iterative algorithm. In each iteration the conditional probabilities in Equation (3.6) is recalculated until convergence, or the maximum number of iterations is made. For a complete description of the decoding methods, see [62].

The algorithms mentioned above work well when the LFSR contains few taps, but for LFSRs with many taps the algorithms fail. The reason for this failure is that for LFSRs with many taps each parity check equation gives a very small average correction and hence many more equations are needed in order to succeed. Or in other words, the maximum correlation probability p that the algorithms can handle is much lower if the LFSR has many taps (e.g. around $l/2$). Due to the fact that the above attacks require the feedback polynomial $g(x)$ to have a low weight, one usually refrain from using such feedback polynomials in stream cipher design.

3.4 Improved Fast Correlation Attacks

After the introduction of fast correlation attacks by Meier and Staffelbach, several proposals have been made that improves their initial ideas. In this section we will present some of the early improvements that were made. The improvements can be divided into two categories. The methods in the first category are used to find more parity check equations, and also to find low weight parity check equations for feedback polynomials with arbitrary weight. The methods in the second category use more powerful iterative decoding methods to improve the original result by Meier and Staffelbach.

One method for finding parity check equations was suggested by Mihaljević and Golić in [71]. The algorithm proposed in [71] is based on a matrix representation of the LFSR. Assume that we have observed a keystream of length N and that the target LFSR has length l and feedback polynomial $g(x) = 1 + c_1x + \cdots + c_lx^l$. Let $\mathbf{u}_N$ be the state of the LFSR at time N, i.e., $\mathbf{u}_N = (u_{N+1}, u_{N+2}, \ldots, u_{N+l})$. There exists an $l \times l$ matrix A,

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & \ldots & 0 \\ 0 & 0 & 1 & 0 & \ldots & 0 \\ \ldots & \ddots & \ddots & \ldots & \ldots & \ldots \\ 0 & \ldots & \ldots & 0 & 0 & 1 \\ c_l & c_{l-1} & \ldots & c_3 & c_2 & c_1 \end{pmatrix},$$

such that the state at time N can be expressed as $\mathbf{u}_N = A^j\mathbf{u}_{N-j}^T$. Hence, by taking all powers of the form A^j, $j = 1, \ldots, N$ we get a set of N parity check equations all involving u_{N+1}. Keep all orthogonal parity check equations with weight t. Let m_t be the expected number of parity check equations found in the algorithm. From [71] we have the following result,

$$m_t = \frac{N}{l2^l} \cdot \binom{l}{t}.$$

The problem with the method in [71] is that it requires a very large value of N to find sufficient many parity check equations when l is increased. As an example, assume that we would like to find 100 parity check equations of weight 3 for an LFSR with length 64. To achieve this we need $N \approx 2^{61}$ bits. Hence, this method is not practical for large l.

Another method for finding parity check equations with low weight was proposed by Chepyzhov and Smeets in [15]. The proposed method is based on a known problem in coding theory. From Section 3.2 we know that the LFSR output sequences can be regarded as codeword from an $[l, N]$ linear code $\mathcal{C}$. There exists a code $\mathcal{H}$ such that,

$$\mathbf{c} \cdot \mathbf{h}^T = 0, \quad \forall \mathbf{c} \in \mathcal{C}, \mathbf{h} \in \mathcal{H}. \tag{3.7}$$

The code $\mathcal{H}$ is called the dual code to $\mathcal{C}$. From (3.7) we see that finding a parity check equations with low weight is equivalent to finding a codeword in $\mathcal{H}$ with low weight. In [15] it was proposed to use information set decoding to find a codeword in $\mathcal{H}$ of low weight. Also for this method the computational complexity will be to high for longer LFSRs.

In [74] Penzhorn proposed a method of finding low weight parity check equations using division of polynomials. If $g(x) = 1 + c_1x + c_2x^2 + \ldots + c_lx^l$ is the feedback polynomial of an LFSR, i.e., the LFSR output sequence satisfies $u_n = \sum_{j=1}^{l} u_{n-j}c_j$, then by a division algorithm we try to find a polynomial $h(x)$ with low weight such that $h(x) \equiv 0 \bmod g(x)$. Furthermore, Penzhorn showed that finding sufficiently many parity check equations with weight 3 is not feasible, instead one has to search for weight 4 parity check equations. The algorithm in [74] has a computational complexity of order $2^{2l/3}$ for finding weight 4 parity check equations when the available number of positions is of the order $2^{k/3}$.

All of the methods presented above improve the fast correlation attacks by searching for more low weight parity check equations. It has also been proposed the use other iterative decoding methods. These iterative decoding methods all use the low weight parity check equations of the form found by the algorithms above. For an overview of some of these iterative decoding algorithms and their performance, see [17, 28].

3.5 Summary

In this chapter the basic principles of correlations attack were presented. It was shown that correlation attacks can be viewed as the problem of decoding a given vector to a codeword from a linear code.

The original correlation attack by Siegenthaler, and the fast correlation attack by Meier and Staffelbach were briefly presented. The chapter also gave an introduction to some of the early improvements of fast correlation attacks.

The methods proposed to find low weight parity check equations have too high complexity to be practical when the length of the LFSR is increased. Hence, the fast correlation attacks based on low weight parity check decoding methods are only practical for LFSRs with low weight feedback polynomials. To find fast correlation attacks that are practical also for feedback polynomials of any weight we can try other approaches. In the following chapters we will look at such proposals.

4
An Algorithm Based on Convolutional Codes

If the fast correlation attacks described in Chapter 3 should be efficient it is required that the LFSR has a feedback polynomial with low weight. The goal of the algorithm presented in this chapter is to achieve a similar performance and a similar low complexity as the algorithms in Chapter 3, but for feedback polynomials of arbitrary weight.

The principle of the algorithm is to transform a part of the code $\mathcal{C}$ stemming from the LFSR sequence into a convolutional code. The convolutional code can then be decoded efficiently using the Viterbi algorithm.

Most of previous results regarding performance of correlation attacks have been based entirely on simulations. In this chapter, we use random coding bounds for convolutional codes to give a theoretical analysis of the presented algorithm. The results from the theoretical derivation are verified by simulations.

The chapter is organized as follows. In Section 4.1 we describe the new algorithm based on low-rate convolutional codes. This algorithm was originally proposed at Eurocrypt'99 [44]. The theoretical analysis of the attack is given in Section 4.2. Section 4.3 presents some simulation results and verifies the obtained theoretical results. In Section 4.4 we exemplify the theoretical results by calculating the complexity for an attack on a combining generator. Finally, in Section 4.5 we give some conclusions and possible extensions.

4.1 Fast Correlation Attacks Based on Convolutional Codes

The general idea behind the algorithm to be described, is to consider slightly more advanced decoding algorithms *including memory*, but which still have a low decoding complexity. This allows weaker restrictions on the parity check equations that can be used, leading to many more and more powerful equations.

As most other algorithms for correlation attacks it has two phases. In the first phase the algorithm transforms a part of the code $\mathcal{C}$ stemming from the LFSR sequences into a convolutional code. The encoder of this convolutional code is created by finding suitable parity check equations from $\mathcal{C}$. In the decoding step the Viterbi algorithm is used for decoding. It is assumed that the reader is familiar with basic concepts regarding convolutional codes, see [40, 44].

Let B be a fixed memory size and let R denote the rate. In a convolutional encoder with memory B and rate $R = 1/(m+1)$ the information sequence

$$\mathbf{u} = u_0, u_1, \ldots$$

is encoded as the code sequence

$$\mathbf{v} = \mathbf{v}_0, \mathbf{v}_1, \ldots = v_0^{(0)}, v_0^{(1)}, \ldots, v_0^{(m)}, v_1^{(0)}, v_1^{(1)}, \ldots, v_1^{(m)}, \ldots$$

where

$$\mathbf{v}_n = u_n \mathbf{g}_0 + u_{n-1}\mathbf{g}_1 + \ldots + u_{n-B}\mathbf{g}_B. \tag{4.1}$$

Here, each $\mathbf{g}_i$, $0 \leqslant i \leqslant B$, is a vector of length $(m+1)$. The purpose of the first pass of the algorithm is to find suitable parity check equations that will determine the vectors $\mathbf{g}_i, 0 \leqslant i \leqslant B$, defining the convolutional code.

Let us start with the linear code $\mathcal{C}$ stemming from the LFSR sequences. From Section 3.2 we know that there is a corresponding $l \times N$ generator matrix, here referred to as G_{LFSR}, such that $\mathbf{u} = \mathbf{u}_0 G_{LFSR}$, where $\mathbf{u}_0$ denotes the initial state of the LFSR, i.e., $\mathbf{u}_0 = (u_1, u_2, \ldots, u_l)$. The generator matrix is furthermore considered to be written in systematic form, $G_{LFSR} = \begin{pmatrix} I_l & Z \end{pmatrix}$, where I_l is the $l \times l$ identity matrix.

We are now interested in finding parity check equations that involve a current symbol u_n, and an arbitrary linear combination of the B previous symbols $u_{n-1}, \ldots, u_{n-B}$, together with at most t other symbols. Clearly, t should be small and in this description $t = 2$ or $t = 3$ are mainly considered.

To find these desired equations, start by considering the index position $n = B + 1$. Introduce the following notation for the generator matrix,

$$G_{LFSR} = \begin{pmatrix} \mathbf{h}_1 & \mathbf{h}_2 & \cdots & \mathbf{h}_N \end{pmatrix}, \tag{4.2}$$

i.e., $\mathbf{h}_i$ is the ith column of G_{LFSR}. Thus, the ith LFSR output symbol can be calculated as $u_i = \mathbf{u}_0\mathbf{h}_i$.

Parity check equations for u_{B+1} with weight t outside the first $B+1$ positions can then be found by finding linear combinations of t columns such their sum is all zero in the last $l-B-1$ positions and one in position $B+1$. Assume that we have found t columns such that

$$(\mathbf{h}_{i_1} + \mathbf{h}_{i_2} + \ldots + \mathbf{h}_{i_t})^T = (c_B, c_{B-1}, \ldots, c_1, 1, 0, 0, \ldots, 0), \tag{4.3}$$

where each c_j, $1 \leqslant j \leqslant B$ can take any value. By combining Equation (4.2) and Equation (4.3) we get the following parity check equation

$$u_{i_1} + u_{i_2} + \ldots + u_{i_t} = u_{B+1} + \sum_{j=1}^{B} c_j u_{B+1-j}. \tag{4.4}$$

It directly follows from the cyclic structure of the LFSR sequences that Equation (4.4) is valid for any index position n simply by shifting all the symbols in time, resulting in

$$u_n + \sum_{j=1}^{B} c_j u_{n-j} = u_{n-B-1+i_1} + u_{n-B-1+i_2} + \ldots + u_{n-B-1+i_t}. \tag{4.5}$$

Assume that the above procedure gives us a set of m different parity check equations for LFSR output symbol u_n, written as

$$\begin{array}{c} u_n + \sum_{j=1}^{B} c_j^{(1)} u_{n-j} = b_n^{(1)}, \\ u_n + \sum_{j=1}^{B} c_j^{(2)} u_{n-j} = b_n^{(2)}, \\ \vdots \\ u_n + \sum_{j=1}^{B} c_j^{(m)} u_{n-j} = b_n^{(m)}, \end{array} \tag{4.6}$$

where $b_n^{(k)} = \sum_{j=1}^{\leqslant t} u_{n-B-1+i_j^{(k)}}$, $1 \leqslant k \leqslant m$ is the sum of (at most) t positions in $\mathbf{u}$. I.e., $b_n^{(k)}$ is the right-hand side of (4.5) for the kth parity check equation. Identifying the parity check equations from (4.6) with the described form of the convolutional code as in (4.1) gives

$$\begin{pmatrix} \mathbf{g}_0 \\ \mathbf{g}_1 \\ \vdots \\ \mathbf{g}_B \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \ldots & 1 \\ 0 & c_1^{(1)} & c_1^{(2)} & \ldots & c_1^{(m)} \\ 0 & c_2^{(1)} & c_2^{(2)} & \ldots & c_2^{(m)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & c_B^{(1)} & c_B^{(2)} & \ldots & c_B^{(m)} \end{pmatrix}, \tag{4.7}$$

where we have added the systematic bit $v_n^{(0)} = u_n$. Thus, we have obtained a rate $R = \frac{1}{m+1}$ convolutional code with memory B.

For the case $t = 2$ the parity check equations can be found in a very simple way as follows. A parity check equation with $t = 2$ is found if two columns from G_{LFSR} have the same value when restricted to the last $l-B-1$ entries. Hence, we simply put each column of G_{LFSR} into one of 2^{l-B-1} different "buckets", sorted according to the value of the last $l-B-1$ entries. Each pair of columns in each bucket will provide us with one parity check equation, provided u_{B+1} is included. For $t = 3$ we store the columns in the same way as for $t = 2$. To find a parity check equation we run through all pair of columns, add them, and look in the bucket corresponding to the last $l-B-1$ entries of the sum.

For larger values of t we can use an algorithm by Chose, Joux, and Mitton [16] to find parity check equations. The algorithm in [16] is based on a sort and match algorithm by Schroeppel and Shamir [79]. The algorithm in [16] has computational complexity $O(N^{\lceil t/2 \rceil})$ and memory complexity $O(N^{\lfloor t/4 \rfloor})$.

For each defined codeword symbol $v_n^{(i)}$ in the convolutional code one has an estimate of that symbol from the transmitted sequence $\mathbf{z}$. We refer to this sequence as the received sequence and denote it by $(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_N)$, where $\mathbf{r}_i = (r_i^{(0)}, r_i^{(1)}, \ldots, r_i^{(m)})$. From the right hand side of (4.6) we have $v_n^{(k)} = b_n^{(k)}$, where $b_n^{(k)} = \sum_{j=1}^{\leqslant t} u_{n-B-1+i_j^{(k)}}$, $1 \leqslant k \leqslant m$. The received sequence is constructed as $r_n^{(k)} = \sum_{j=1}^{\leqslant t} z_{n-B-1+i_j^{(k)}}$, for $1 \leqslant k \leqslant m$ and $r_n^{(0)} = z_n$. From the construction of the received sequence we can calculate the probability that a codesymbol of the convolutional code is correctly received as

$$\begin{aligned} P(v_n^{(k)} = r_n^{(k)}) &= P(b_n^{(k)} = r_n^{(k)}) \\ &= P\Big(\sum_{j=1}^{\leqslant t} u_{n-B-1+i_j^{(k)}} = \sum_{j=1}^{\leqslant t} z_{n-B-1+i_j^{(k)}}\Big). \end{aligned} \tag{4.8}$$

To recover the initial state of the LFSR it is enough to decode l consecutive information bits correctly. Optimal decoding (ML-decoding) of the convolutional code using the Viterbi algorithm can thus be performed. The channel model used in the decoding phase is a binary symmetric channel, BSC, with crossover probability defined by (4.8). For binary symmetric channels ML-decoding is equivalent to minimum distance decoding, and hence in our case, there is no need for floating point arithmetic when implementing the Viterbi algorithm.

The original Viterbi algorithm assumes that the convolutional encoder starts in state $\mathbf{0}$. However, in this application there is neither a predefined starting state, nor a predefined ending state for the trellis corresponding to the convolutional code. For each possible starting state $\sigma_B = (s_1, s_2, \ldots, s_B)$, let $d_H(\sigma_B, \mathbf{z}_B)$, where $\mathbf{z}_B = (z_1, z_2, \ldots, z_B)$ and $d_H(\sigma_B, \mathbf{z}_B)$ denotes the Hamming distance between σ_B and $\mathbf{z}_B$, be the initial metric for that state when we start the Viterbi algorithm at $n = B$. Then one runs the Viterbi algorithm over l information symbols. At depth $B + l$ we search for the ending state σ_{B+l} with minimum metric. The decoder output is then the information sequence corresponding to the surviving path from one of the starting states σ_B to the ending state σ_{B+l} with minimum metric. It is important to note that the decoding algorithm presented here is slightly improved compared to the decoding algorithm originally proposed in [44]. This concludes the general description and we give a detailed summary of the algorithm in Figure 4.1.

In: An observed keystream sequence $\mathbf{z} = (z_1, z_2, \ldots, z_N)$, the systematic $l \times N$ generator matrix in the form

$$G_{LFSR} = \begin{pmatrix} \mathbf{h}_1 & \mathbf{h}_2 & \ldots & \mathbf{h}_N \end{pmatrix},$$

and the algorithm parameters B and t.

1. For $l + B + 1 \leqslant i_1 < i_2 < \ldots < i_t \leqslant N$ find all t-tuples of columns $\mathbf{h}_{i_1}, \mathbf{h}_{i_2}, \ldots, \mathbf{h}_{i_t}$ such that

$$(\mathbf{h}_{i_1} + \mathbf{h}_{i_2} + \ldots + \mathbf{h}_{i_t})^T = (c_B, c_{B-1}, \ldots, c_1, 1, \underbrace{0, 0, \ldots, 0}_{l-B-1}),$$

where each c_j, $1 \leqslant j \leqslant N$, can take any value. Then add

$$(u_{n-B}, \ldots, u_n, 0, \ldots, 0) \cdot (\mathbf{h}_{i_1} + \ldots + \mathbf{h}_{i_t}) + \sum_{j=1}^{\leqslant t} u_{n-B-1-i_j} = 0$$

to the set of parity check equations as in (4.6).

2. From this set, calculate $\mathbf{g}_0, \mathbf{g}_1, \ldots, \mathbf{g}_B$ as in (4.7). Create a received vector $\mathbf{r}$ from $\mathbf{z}$ by $r_n^{(0)} = z_n$ and $r_n^{(k)} = \sum_{j=1}^{\leqslant t} z_{n-B-1+i_j^{(k)}}$, for $1 \leqslant k \leqslant m$, where $i_1^{(k)}, i_2^{(k)}, \ldots, i_t^{(k)}$, $1 \leqslant k \leqslant m$ are the indices determined in step 1.

Decoding part:

3. For each starting state σ_B, let $d_H(\sigma_B, \mathbf{z}_B)$ be the initial metric.

4. Decode the received sequence $\mathbf{r}$ using the Viterbi algorithm from $n = B$ until $n = B + l$. Output the estimated information sequence $(\hat{u}_{B+1}, \hat{u}_{B+2}, \ldots, \hat{u}_{B+l})$. Finally, calculate the corresponding initial state of the LFSR.

Figure 4.1: A description of the algorithm for fast correlation attacks based on convolutional codes.

4.2 Theoretical Analysis

To simplify the theoretical analysis, we consider some small modifications in the algorithm described in the previous section. Firstly, we consider the convolutional code to be time-varying. This can be achieved by applying a random permutation to the columns of G_{LFSR} and searching for new parity check equations for each time index. For a more thorough description we refer to [43]. Secondly, we exclude the systematic symbol $v_n^{(0)} = u_n$, and the rate of our convolutional code is $R = 1/m$.

The principle of the analysis is the following. We start by calculating the average number of parity check equations that we will find by the proposed algorithm. This gives us the rate of the convolutional code. Depending on the value of t we calculate the error probability in an equivalent binary symmetric channel for the convolutional code. Given the number of equations and the error probability of the BSC we can use results from convolutional coding to get a bound on the burst error probability of the convolutional code. This burst error probability bounds the probability that the proposed attack fails. Finally, we fix the rate to be $R \approx R_0$, where R_0 is the computational cutoff rate [40]. This gives us a rate close to capacity and yet a small error probability. Based on these assumptions, the result of the analysis is the required initial correlation for given keystream length N, LFSR length l, and algorithm parameters B and t.

Lemma 4.1: Let m be the number of parity check equations in (4.6) and let $E[m]$ be the expected value of m. Then,

$$E[m] \approx \frac{\binom{N-l-B}{t}}{2^{l-B}} \approx \frac{N^t}{t!} 2^{B-l}.$$

$\square$

Proof: The number of different linear combinations of t columns from the last $N-l-B$ columns of G_{LFSR} is $\binom{N-l-B}{t}$. Since the rows of G_{LFSR} is LFSR sequences of length N, the entries g_{ij} of G_{LFSR} is approximately independent identically distributed random variables with, $P(g_{ij} = 0) = 1/2$. Thus, each of the 2^{l-B-1} different possible values of the last $l-B-1$ positions for each column, $\mathbf{h}_i$, of G_{LFSR} have the same probability.

Hence, we get in average $\binom{N-l-B}{t}/2^{l-B-1}$ linear combinations with all-zero in the last positions. Among these linear combinations one half depends on u_n. Thus,

$$E[m] \approx \frac{1}{2} \cdot \frac{\binom{N-l-B}{t}}{2^{l-B-1}} = \frac{\binom{N-l-B}{t}}{2^{l-B}} \approx \frac{N^t}{t!} 2^{B-l},$$

where the last approximation is valid if t is small, and $N \gg l + B$. $\blacksquare$

In the model of a correlation attack presented in Chapter 2, the correlation between the LFSR sequence $\mathbf{u}$ and the keystream $\mathbf{z}$ is $P(u_i = z_i) = 1-p$. This can be modelled as a binary symmetric channel with error probability $p = \frac{1}{2} - \delta$.

In our case, we consider an "embedded" convolutional code. Then the received symbol $r_n^{(k)}$ corresponding to codeword symbol $v_n^{(k)}$ is given as the sum of t keystream symbols, $r_n^{(k)} = \sum_{j=1}^{t} z_{n-B-1+i_j^{(k)}}$. If the expected number of equations are much less than the observed length, that is $E[m] \ll N$, then the probability that a position z_j appears in more than one equation is small, and thus, different $r_n^{(k)}$'s can be considered to be independent. Hence, the error probability in Equation (4.8) can be modelled as a binary symmetric channel with error probability $p' = \frac{1}{2} - \epsilon$.

Lemma 4.2: Let $p = \frac{1}{2} - \delta = P(z_n \neq u_n)$, and $p' = \frac{1}{2} - \epsilon = P(r_n^{(k)} \neq v_n^{(k)})$. Then

$$\epsilon = 2^{t-1}\delta^t.$$

□

Proof: We prove by induction. Let $t = 1$, clearly we have $\epsilon = \delta$. Assume that the expression is true for $t = s - 1$. Using Bayes rule we get that $p' = P(\sum_{j=1}^{s} z_{n-B-1+i_j^{(k)}} \neq v_n^{(k)})$ can be expressed as

$$p' = P(\sum_{j=1}^{s-1} z_{n-B-1+i_j^{(k)}} = v_n^{(k)} | z_{n-B-1+i_s^{(k)}} \neq v_n^{(k)}) P(z_{n-B-1+i_s^{(k)}} \neq v_n^{(k)})$$
$$+ P(\sum_{j=1}^{s-1} z_{n-B-1+i_j^{(k)}} \neq v_n^{(k)} | z_{n-B-1+i_s^{(k)}} = v_n^{(k)}) P(z_{n-B-1+i_s^{(k)}} = v_n^{(k)}).$$

Since the channel is memoryless we get

$$\begin{aligned} p' &= P(\sum_{j=1}^{s-1} z_{n-B-1+i_j^{(k)}} = v_n^{(k)}) P(z_{n-B-1+i_s^{(k)}} \neq v_n^{(k)}) \\ &+ P(\sum_{j=1}^{s-1} z_{n-B-1+i_j^{(k)}} \neq v_n^{(k)}) P(z_{n-B-1+i_s^{(k)}} = v_n^{(k)}), \end{aligned}$$

and since the lemma holds for $t = s - 1$ by assumption we get

$$\begin{aligned} p' &= (\frac{1}{2} + 2^{s-2}\delta^{s-1})(\frac{1}{2} - \delta) + (\frac{1}{2} - 2^{s-2}\delta^{s-1})(\frac{1}{2} + \delta) \\ &= \frac{1}{2} - 2\delta 2^{s-2}\delta^{s-1} \\ &= \frac{1}{2} - 2^{s-1}\delta^{s}. \end{aligned}$$

■

Let P_B be the burst error probability [40], i.e., the probability that we do not decode the convolutional code correctly. To get a bound on the expected burst error probability, $E[P_B]$, for our convolutional code we use the following result, taken from [40, p. 215]. The average is taken over the ensemble of all random rate $R = 1/m$ time-varying convolutional codes encoded by a polynomial generator matrix of memory B.

Lemma 4.3: The expected burst error probability of a rate $R = 1/m$, time-varying convolutional code encoded by a polynomial, time-varying generator matrix of memory B is upper-bounded by

$$E[P_B] \leqslant c(R)2^{-R_0 Bm}, 0 \leqslant R < R_0,$$

where R_0 is the computational cutoff rate for the BSC, defined as

$$R_0 = 1 - \log(1 + 2\sqrt{p(1-p)}),$$

and where

$$c(R) = \frac{1}{2^{(R_0 - R)m} - 1}.$$

□

Using the bound for the expected burst error probability we can calculate when the proposed attack will succeed. Let the correlation probability be $1/2 + \delta$, i.e.,

$$P(u_i = z_i) = 1/2 + \delta.$$

The values of δ for which the attack is successful is given by the following theorem.

Theorem 4.4: With probability $1 - p_e, p_e < 1$, the proposed attack succeeds if

$$\delta \geqslant \frac{1}{2}\left(\frac{4(1+d)\ln 2 \cdot 2^{l-B}}{\binom{N-l-B}{t}}\right)^{\frac{1}{2t}},$$

where the correlation probability is $P(z_i = u_i) = 1/2 + \delta$,

$$p_e \leqslant l \frac{1}{2^d - 1} 2^{-(1+d)B},$$

and d is any fixed positive real number. □

Proof: We assume that the randomly generated code satisfies the bound in Lemma 4.3, and that $R = R_0$. For a BSC with $p = 1/2 - \epsilon$,

$$R_0 = 1 - \log(1 + 2\sqrt{\frac{1}{4} - \epsilon^2}).$$

Since ϵ is very small, we use Taylors formula to get

$$\sqrt{\frac{1}{4} - \epsilon^2} = \frac{1}{2} - \epsilon^2 - \epsilon^4 + 2\epsilon^6 + o(\epsilon^8).$$

The cutoff rate can then be written as

$$R_0 = 1 - \log(2 - 2\epsilon^2 - 2\epsilon^4 - 4\epsilon^6 + o(\epsilon^8)) = -\log(1 - \epsilon^2 - \epsilon^4 - 2\epsilon^6 + o(\epsilon^8)).$$

Using the standard expression for $\ln(x)$ we get

$$\log(1 - \epsilon^2 - \epsilon^4 - 2\epsilon^6 + o(\epsilon^8)) = \frac{1}{\ln 2}(-\epsilon^2 - \frac{3}{2}\epsilon^4 - \frac{10}{3}\epsilon^6 + o(\epsilon^8)),$$

and

$$R_0 = \frac{1}{\ln 2}(\epsilon^2 + \frac{3}{2}\epsilon^4 + \frac{10}{3}\epsilon^6 + o(\epsilon^8)) = \frac{\epsilon^2}{\ln 2} + o(\epsilon^4),$$

for small ϵ. Using Lemma 4.2 we insert $\epsilon = 2^{t-1}\delta^t$, and R_0 can be expressed as

$$R_0 = \frac{2^{2(t-1)}\delta^{2t}}{\ln 2} + o(\delta^{4t}).$$

From Lemma 4.3 we have an expression for the burst error probability given that $R < R_0$. Ideally we want R to be as close to R_0 as possible. This will, however, make $c(R)$ to increase and we might not get a useful bound on the error probability. Thus, we need to have a code rate slightly less than R_0, say

$$\frac{1}{m} = R = \frac{R_0}{1 + d},$$

where d is a fixed positive real number. By inserting the average number of equations, $E[m]$, from Lemma 4.1 and the expression for R_0 we get

$$\frac{\binom{N-l-B}{t}}{2^{l-B}} \geqslant \frac{(d+1)\ln 2}{2^{2(t-1)}\delta^{2t}},$$

which give us

$$\delta \geqslant \frac{1}{2}\left(\frac{4(d+1)\ln 2 \cdot 2^{l-B}}{\binom{N-l-B}{t}}\right)^{\frac{1}{2t}},$$

The probability that the attack fails is bounded by, $p_e < l \cdot E[P_B]$, where

$$E[P_B] \leqslant c(R)2^{-R_0 Bm} = \frac{1}{2^d-1}2^{-(1+d)B}.$$

Hence, we get

$$p_e \leqslant l\frac{1}{2^d-1}2^{-(1+d)B}.$$

■

By using that $\binom{N-l-B}{t} < N^t/t!$ we can rewrite Theorem 4.4 as the following corollary.

Corollary 4.5: With probability $1-p_e, p_e < 1$, the proposed attack succeeds if

$$N \geqslant \frac{1}{4\delta^2}(4(1+d)\ln 2 \cdot t!)^{\frac{1}{t}} \cdot 2^{\frac{l-B}{t}},$$

where the correlation probability is $P(z_i = u_i) = 1/2 + \delta$,

$$p_e \leqslant l\frac{1}{2^d-1}2^{-(1+d)B},$$

and d is any fixed positive real number. □

4.3 Simulation Results

In order to verify the theoretical results a large number of simulations were made. Most of the simulations use a LFSR with length $l = 40$, and a weight 17 feedback polynomial. The LFSR has the following feedback polynomial,

$$\begin{aligned} g(x) &= 1 + x + x^3 + x^5 + x^9 + x^{11} + x^{12} + x^{17} + x^{19} \\ &\quad + x^{21} + x^{25} + x^{27} + x^{29} + x^{32} + x^{33} + x^{38} + x^{40}. \end{aligned}$$

The algorithm parameter t was chosen to $t = 2$ and $t = 3$. The memory B varied between $B = 7$ and $B = 17$. The simulation results are compared with the theoretical results in Theorem 4.4. In the theoretical results we let the distance parameter d be $d = 0.2$. These results are plotted in Figure 4.2 and Figure 4.3, respectively. In these figures the maximal value of p is plotted when N is varied. The simulation results are based on the suboptimal

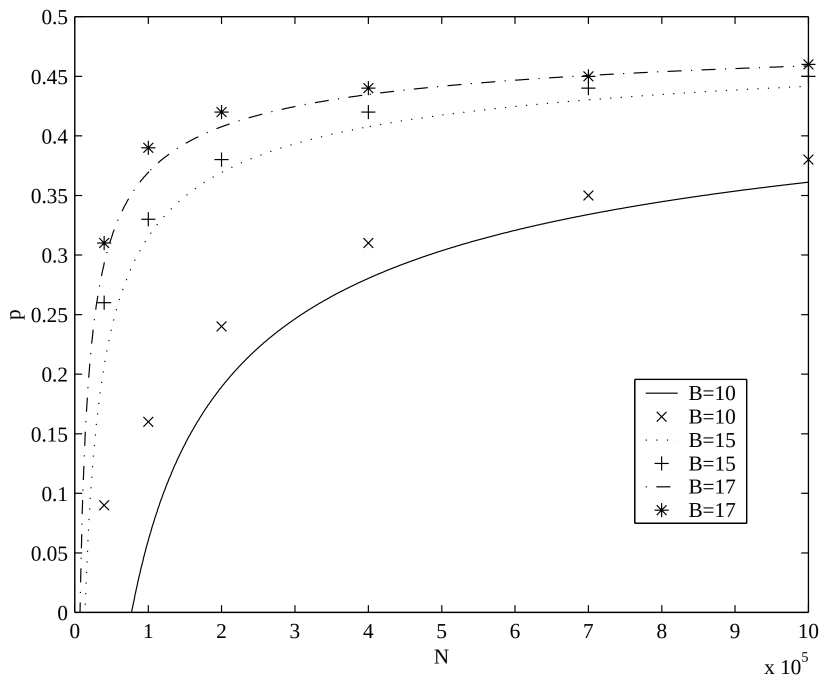

Figure 4.2: Comparison between simulations and theoretical results for LFSR length $l = 40$ and $t = 2$.

decoding method described in Section 4.1. From Figure 4.2 and Figure 4.3 we can see that for large values of the constant B the bound in Theorem 4.4 is rather tight. It is also important to note that one of the conditions for Theorem 4.4 to hold is that $N \gg m$. However, from the simulations we can see that also in the case when $N < m$ the results from Theorem 4.4 is close to the actual performance of the algorithm.

We can see in Figure 4.2 that for small B the simulation shows better performance than one could expect by the results from Theorem 4.4. One explanation for this behavior is that in these cases we get rather few equations and the systematic symbol that we exclude in the theoretical analysis has a bigger impact on the result. We should also note that it is possible to have a rate larger than R_0 and still have a quite low probability of algorithm failure.

From the simulations we can also make a comment regarding the complexity of decoding when the value of the parameter t is varied. Assume that we have $p = 0.39$. With $t = 2$ and $N = 100000$ we need memory $B = 17$. The time for a successful decoding in this case was approximately 90 seconds.

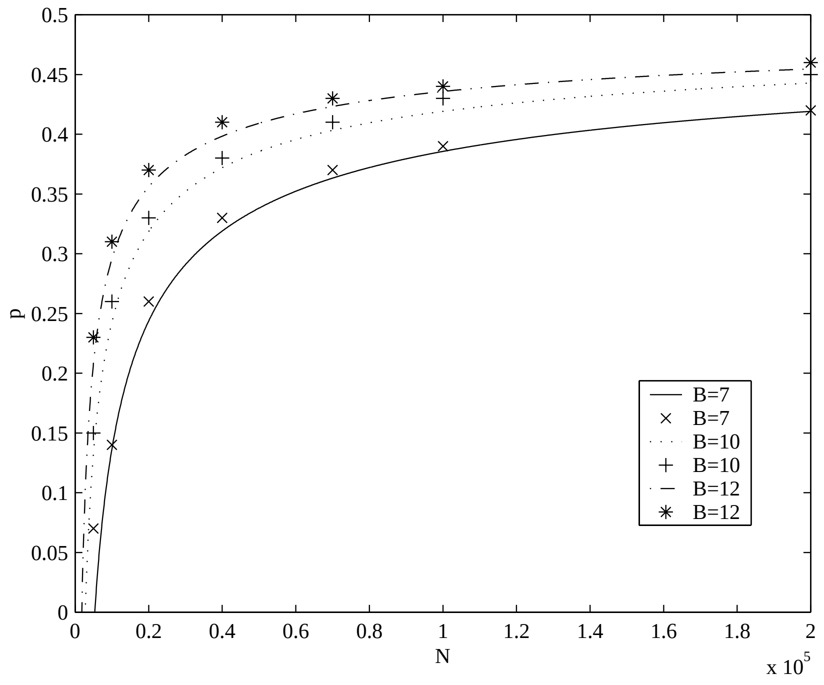

Figure 4.3: Comparison between simulations and theoretical results for LFSR length $l = 40$ and $t = 3$.

With $t = 3$ we only need $B = 7$ to get $p = 0.39$ for $N = 100000$. In this latter case the decoding takes approximately 1.5 seconds. Thus, we can conjecture that it is desirable to increase the value of t. However, due to the complexity in the precomputation we can not use a too large value of t.

Some simulations were also made for $l = 60$. The result is tabulated in Table 4.1. From Table 4.1 we can see that the results in Theorem 4.4 is rather tight also for $l = 60$.

N	t	B	p_{th}	p_{sim}
$100 \cdot 10^6$	2	20	0.42	0.43
$600 \cdot 10^3$	3	18	0.37	0.38

Table 4.1: Comparison between simulations and theoretical results for LFSR length $l = 60$.

4.4 Example

In this section we illustrate the importance of having theoretical results for the performance of correlation attacks by an example, in which we calculate the complexity of attacking a combining generator. The principle of a nonlinear filter generator is to generate the keystream as some nonlinear function of the output from M LFSRs, see Figure 4.4.

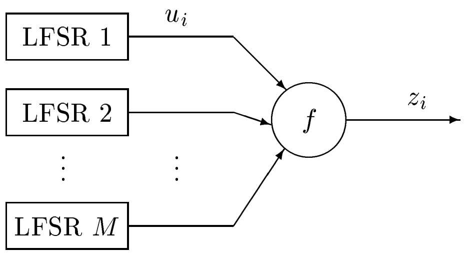

Figure 4.4: Principle of nonlinear combination generators.

Assume that LFSR 1 has length $l = 89$, and that there exists a correlation of size $P(z_i = u_i) = 0.53125$. If we let $B = 10$ and $d = 0.1$, we can calculate the required amount of observed keystream symbols for having a successful attack. From Corollary 4.5 we get $N \approx 2^{35.8}$ and $p_e \leqslant 0.6$, where we used $\delta = 0.03125$.

The decoding complexity of the attack can be calculated as follows. From Lemma 4.2 the average number of parity check equations is $m \approx 2^{25.5}$. The number of states, $|\sigma|$, in the Viterbi decoder is 2^B. If we use ML-decoding the decoding complexity, denoted by C_{dec}, can be calculated as

$$C_{dec} = l \cdot m \cdot 2^B \approx 2^{42}.$$

The precomputation for this case is approximately 2^{71} table lookups.

Note that the complexity is given as the average number of bit operations in the decoding phase and the average number of table lookups in the precomputation. Thus, for the decoding phase, m bit operations are done significantly faster than m clock cycles. On the other hand, in the precomputation phase we search for the existence of a certain vector in a large sorted list of vectors. By using a hash table this can be made almost in constant time. However, it will still require more than one clock cycle per operation.

4.5 Summary

We have presented a new algorithm for fast correlation attacks. The algorithm is based on low-rate convolutional codes. In a theoretical analysis, using random coding bounds for convolutional bounds, we derive its performance. The analysis results in a bound which determines the required number of observed keystream symbols needed in a successful attack. If we compare the theoretical bound with results obtained by simulations we observe that the bound is rather tight. Thus we conclude that the results from the theoretical analysis can be used as a tool for evaluating the security of stream ciphers based on linear feedback shift registers, where the shift register length makes simulations impossible.

5

An Algorithm Using Turbo Code Techniques

In the algorithm presented in the previous chapter, decoding a convolutional codes with large memory was a major issue. Since the starting state of the trellis is unknown, one cannot use low memory decoding algorithms such as sequential decoding. Instead, decoding with the memory consuming Viterbi algorithm must be used. For correlations close to 0.5 the rate of the convolutional code must be extremely low. From Lemma 4.1 we see that for long shift registers, the memory of the convolutional encoder must be large if sufficiently many parity check equations should be found. Unfortunately, due to the memory requirements of the Viterbi algorithm it is not possible to use too large memory. In fact, what limits the attack in Chapter 4 is not the computational complexity but rather the memory complexity. To be able to improve the attack one would like to find a method of decreasing the rate of the convolutional code without increasing the memory.

In coding theory, turbo codes have turned out to be an efficient coding/decoding method, which achieve performance close to the Shannon limit. The basic principle of turbo codes is to use two convolutional codes in parallel with some kind of permutation in between, see [3,4]. Figure 5.1 illustrates the principle of turbo codes. The information symbols are the input to both encoders. Before the sequence of information symbols enters the second encoder, a permutation π is applied. To decode a turbo code an iterative decoding algorithm is used. The decoding algorithm is a suboptimal decoding algorithm based on a posteriori probability decoding of the constituent codes. The memory complexity of the decoding algorithm for turbo codes is the same as the maximum memory complexity for decoding the constituent

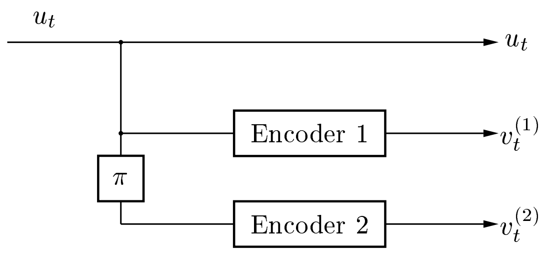

Figure 5.1: Principle of turbo codes.

codes. Hence, we see that it is possible to decrease the rate without increase the memory complexity of the decoding algorithm.

This chapter presents a fast correlation attack, which use techniques from turbo codes to get a code with low rate and reasonable low memory complexity. The code is then decoded by an iterative decoding algorithm, well known from general decoding techniques of turbo codes. In Section 5.1, the iterative decoding method for convolutional codes is presented in a basic algorithm using only one code. In Section 5.2, the idea of several "parallel" codes is introduced and the main algorithm of this chapter is described. Simulation results are presented in Section 5.3. In Section 5.4, a parallel decoding version is proposed, and in Section 5.5 we conclude with some possible extensions.

5.1 A Basic Algorithm Using One Constituent Code

Algorithm B by Meier and Staffelbach [62] calculates an a posteriori probability for each symbol in the received sequence and then iteratively tries to improve these probabilities by recalculating them. The procedure is based on very simple (memoryless) parity checks. The method of Chapter 4 uses instead convolutional codes but uses a simple Viterbi decoding procedure on a small part of the received sequence.

The ideas to be proposed try to combine the best parts of both methods into a single algorithm. In this section a basic construction is presented. This basic algorithm uses one convolutional code (Chapter 4 method) and then applies an APP (a posteriori probability) decoding algorithm in order to provide an a posteriori probability for each symbol in a certain part of the received sequence. Optimal APP decoding (also referred to as MAP decoding) on a convolutional code can be performed by the famous BCJR algorithm [1], or variations of it. The a posteriori probabilities are then fed back as a priori probabilities and in this fashion the procedure is iterated until convergence. This is now described in more detail.

The first step involves computing parity check equations for a convolutional code with fixed memory B. We follow the procedure of Section 4.1 and compute all parity check equations with $t = 2$ involving the particular index position $B + 1$, i.e., find all parity check equations of the form

$$u_{i_1} + u_{i_2} = u_{B+1} + \sum_{j=1}^{B} c_j u_{B+1-j}. \tag{5.1}$$

Parity checks for index position $B + 1 + i$ are then immediately obtained through a cyclic shift of the original parity checks with i steps, as in (4.5). We refer to Section 4.1 for a review of the details. Write the m obtained parity check equations in the form

$$\begin{array}{c} u_n + \sum_{j=1}^{B} c_j^{(1)} u_{n-j} = u_{n-B-1+i_1^{(1)}} + u_{n-B-1+i_2^{(1)}}, \\ u_n + \sum_{j=1}^{B} c_j^{(2)} u_{n-j} = u_{n-B-1+i_1^{(2)}} + u_{n-B-1+i_2^{(2)}}, \\ \vdots \\ u_n + \sum_{j=1}^{B} c_j^{(m)} u_{n-j} = u_{n-B-1+i_1^{(m)}} + u_{n-B-1+i_2^{(m)}}. \end{array} \tag{5.2}$$

The convolutional code is defined by all codeword sequences $\mathbf{v}$,

$$\mathbf{v} = \ldots v_n^{(0)} v_n^{(1)} \ldots v_n^{(m)} v_{n+1}^{(0)} v_{n+1}^{(1)} \ldots v_{n+1}^{(m)} \ldots, \qquad n > B, \tag{5.3}$$

where

$$v_n^{(0)} = u_n, \quad v_n^{(k)} = u_n + \sum_{j=1}^{B} c_j^{(k)} u_{n-j}, \quad 1 \le k \le m.$$

The second step is the APP decoding phase. After observing a keystream sequence $\mathbf{z}$, construct a sequence $\mathbf{r}$ acting as a received sequence for the convolutional code by

$$\mathbf{r} = \ldots r_n^{(0)} r_n^{(1)} \ldots r_n^{(m)} r_{n+1}^{(0)} r_{n+1}^{(1)} \ldots r_{n+1}^{(m)} \ldots, \qquad n > B,$$

where

$$r_n^{(0)} = z_n, \quad r_n^{(k)} = z_{n-B-1+i_1^{(k)}} + z_{n-B-1+i_1^{(k)}}, \quad 1 \le k \le m.$$

Assume that the correlation between the keystream and the considered LFSR output sequence is $1 - p$, i.e., we have $P(z_n = u_n) = P(r_n^{(0)} = v_n^{(0)}) = 1 - p$. From the correlation and the construction of $r_n^{(k)}$ we get

$$P(r_n^{(k)} = v_n^{(k)}) = (1-p)^2 + p^2, \quad 1 \le k \le m.$$

Basic algorithm		
$B = 13$	$B = 14$	$B = 15$
0.20	0.23	0.26

Table 5.1: Maximum p for some different algorithms when $N = 40000$ and $B = 13, 14, 15$.

Then decode the constructed sequence $\mathbf{r}$ stemming from a codeword $\mathbf{v}$ of the convolutional code using an APP decoding algorithm. The original BCJR algorithm requires storage of the whole trellis. However, suboptimal versions of the BCJR algorithm, see [38, 92], remove this problem with a negligible decrease in performance. This procedure provides us with the a posteriori probabilities for the information sequence, i.e.,

$$P(v_{B+1}^{(0)}|\mathbf{r}), P(v_{B+2}^{(0)}|\mathbf{r}), \ldots, P(v_l^{(0)}|\mathbf{r}).$$

Finally, since $v_{B+1}^{(0)} = u_{B+1}, v_{B+2}^{(0)} = u_{B+2}, \ldots$ this information is fed back as new a priori probabilities for $(u_{B+1}, u_{B+2}, \ldots, u_l)$ and the a priori probabilities of the codeword sequence $\mathbf{v}$ of the convolutional code is recalculated. The decoding procedure is performed a second time, and this procedure is iterated $2-5$ times (until convergence). A summary of the algorithm is given in Figure 5.2.

We end by presenting some simulation results for the basic algorithm in Table 5.1. We choose to use the same case as tabulated in [44, 71], which is based on a LFSR with length $l = 40$, and a weight 17 feedback polynomial. The performance of the algorithm is approximately the same as the results for the algorithm using the Viterbi algorithm in Chapter 4. This is what can be expected since we have the same codes as in Chapter 4.

In: The $l \times N$ generator matrix G_{LFSR} for the code generated by a LFSR, the received sequence $\mathbf{z}$, the error probability p, and the number of iterations I.

1. For each position n, $B+1 \leqslant n \leqslant l$, in G_{LFSR}, find the set of parity check equations of the form (5.2) and construct the convolutional code.

   Decoding phase:

2. After receiving $\mathbf{z}$, construct the received sequence $\mathbf{r}$ by

$$r_n^{(0)} = z_n, \quad r_n^{(j)} = z_{n-B-1+i_1^{(j)}} + z_{n-B-1+i_2^{(j)}}, \quad 1 \leqslant j \leqslant m.$$

3. Update

$$P(v_n^{(0)}) = P(u_n), \quad B+1 \leqslant n \leqslant l.$$

   Run the APP decoding algorithm and calculate the a posteriori probabilities

$$P(v_{B+1}^{(0)}|\mathbf{r}), P(v_{B+2}^{(0)}|\mathbf{r}), \ldots, P(v_l^{(0)}|\mathbf{r}).$$

   Since $v_n^{(0)} = u_n$, set

$$P(u_{B+1}) \leftarrow P(v_{B+1}^{(0)}|\mathbf{r}), \ldots, P(u_l) \leftarrow P(v_l^{(0)}|\mathbf{r}).$$

4. If the number of iterations $< I$ go to 3. Otherwise select the most probable value for each of the symbols $u_{B+1}, u_{B+2}, \ldots, u_{B+l}$, calculate the initial state $\mathbf{u}_0$ and check if it is correct.

Figure 5.2: A description of the basic algorithm.

5.2 Description of an Algorithm Based on Turbo Code Techniques

One of the most revolutionary ideas in coding theory the last decade has been the introduction of turbo codes. The original turbo code [3] consists of two convolutional codes, where the information bits are directly fed into one of them and an interleaved version of the same information bits are fed into the other convolutional code. The fundamentally new idea was the proposed decoding scheme, which uses an iterative procedure. Decode the first code using an APP decoding algorithm, which provides a posteriori probabilities for all information symbols. Use these as a priori information when decoding the second code using again APP decoding. The obtained a posteriori probabilities are now used as a priori information when decoding the first code a second time, and the procedure continues in this iterative fashion. For more details on iterative decoding and decoding of turbo codes, see, e.g. [38].

Much the same ideas as described above can be applied to our decoding problem. Instead of using just one fixed convolutional code, as in the basic algorithm described in Section 5.1, we will show how to find and use two or more different convolutional codes. When we have only one convolutional code as in Chapter 4 and in Section 5.1 it was sufficient to decode only over l information symbols. However, in turbo coding it is known that one would like to have a large interleaver size to get better performance [3]. To be able to have a larger interleaver size we increase the number of information symbols to J. In this chapter the value of J is chosen to be $J = 10 \cdot B + l$. The different convolutional codes are obtained by randomly permuting the index positions of the original code in the interval $B + 1 \dots J$.

A problem arises, however, since we need parity check equations for permuted versions of the code $\mathcal{C}$. The shifting technique will no longer provide this for all indices, since after the column permutation the new code has no longer the cyclic properties of $\mathcal{C}$. To overcome this problem we simply search for all valid parity check equations in each index position. It will increase the precomputation time by a factor $J - B$, but for the case $t = 2$ this is not at all a problem. Hence, this procedure will create different parity check equations for different index positions, thus leading to a *timevarying* convolutional code (in opposite to the code in Section 5.1). Also the *number* of parity checks will vary with n.

To find parity check equations for permutations of $\mathcal{C}$ we do as follows. We still assume that $t = 2$. Let π be a permutation that randomly permutes only the first J indices, written as,

$$\pi = \left(\begin{array}{cccc} \pi_1 & \pi_2 & \dots & \pi_N \end{array} \right),$$

where $\pi_i = i$ for $i > J$. As in Section 4.1 we use the following notation for the generator matrix,

$$G_{LFSR} = \begin{pmatrix} \mathbf{h}_1 & \mathbf{h}_2 & \dots & \mathbf{h}_N \end{pmatrix},$$

i.e., $\mathbf{h}_i$ is the ith column of G_{LFSR}. If the matrix G_{LFSR} is permuted according to the permutation π we get the following matrix,

$$\pi(G_{LFSR}) = \begin{pmatrix} \mathbf{h}_{\pi_1} & \mathbf{h}_{\pi_2} & \dots & \mathbf{h}_{\pi_N} \end{pmatrix}. \tag{5.4}$$

Assume that we want to find parity check equation for an index position π_n, where $B+1 \leq n \leq J$. Apply suitable row operation to the matrix in (5.4) to get a matrix, denoted by G'_{LFSR}, of the form,

$$G'_{LFSR} = \begin{pmatrix} \hat{\mathbf{h}}'_{\pi_1} & \hat{\mathbf{h}}'_{\pi_2} & \dots & \hat{\mathbf{h}}'_{\pi_{n-B-1}} & I_{B+1} & \hat{\mathbf{h}}'_{\pi_{n+1}} & \dots & \hat{\mathbf{h}}'_{\pi_N} \\ \check{\mathbf{h}}'_{\pi_1} & \check{\mathbf{h}}'_{\pi_2} & \dots & \check{\mathbf{h}}'_{\pi_{n-B-1}} & 0 & \check{\mathbf{h}}'_{\pi_{n+1}} & \dots & \check{\mathbf{h}}'_{\pi_N} \end{pmatrix}, \tag{5.5}$$

where I_{B+1} is the $B+1 \times B+1$ identity matrix, $\hat{\mathbf{h}}'_{\pi_i}$ and $\check{\mathbf{h}}'_{\pi_i}$ are the first $B+1$ and the last $l-B-1$ positions, respectively, of the ith column of the matrix in (5.4) after the row operations have been performed.

Then put each column $\mathbf{h}_{\pi_i}$, $J < i \leqslant N$ together with its index position into one of 2^{l-B-1} different "buckets", sorted according to the column value. Each pair of columns in each bucket will provide us with one valid parity check equation (of the form (5.1)) for index position π_n, provided u_{π_n} is included. Finally, since the number of parity checks will vary with n, we introduce $m(n)$ as the number of found parity checks for index position π_n. The parity check equations for index position π_n is written as

$$\begin{aligned} u_{\pi_n} + \sum_{j=1}^{B} c_j^{(1)} u_{\pi_{n-i}} &= u_{i_1}^{(1)} + u_{i_2}^{(1)}, \\ u_{\pi_n} + \sum_{j=1}^{B} c_j^{(2)} u_{\pi_{n-i}} &= u_{i_1}^{(2)} + u_{i_2}^{(2)}, \\ &\vdots \\ u_{\pi_n} + \sum_{j=1}^{B} c_j^{(m(n))} u_{\pi_{n-i}} &= u_{i_1}^{(m(n))} + u_{i_2}^{(m(n))}. \end{aligned} \tag{5.6}$$

For each n, the constants defining the parity check equations,

$$\begin{pmatrix} \mathbf{g}_0(n) \\ \mathbf{g}_1(n) \\ \vdots \\ \mathbf{g}_B(n) \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 0 & c_1^{(1)} & c_1^{(2)} & \dots & c_1^{(m(n))} \\ 0 & c_2^{(1)} & c_2^{(2)} & \dots & c_2^{(m(n))} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & c_B^{(1)} & c_B^{(2)} & \dots & c_B^{(m(n))} \end{pmatrix}, \tag{5.7}$$

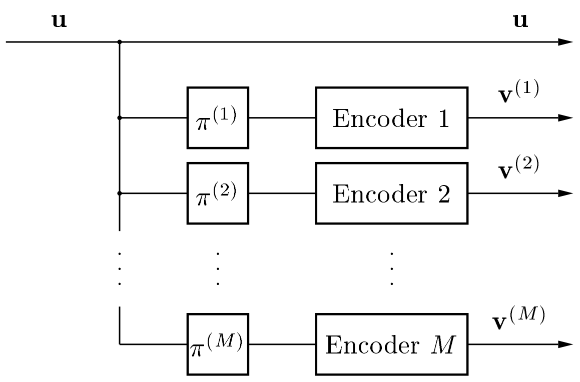

Figure 5.3: Model of the encoder structure used in the algorithm.

must be stored in order to build the trellis in the decoding phase.

To get M constituent encoders the steps above are repeated M times with different permutations. The encoder structure with M constituent parallel encoders is illustrated in Figure 5.3. The information sequence $\mathbf{u} = u_1, u_2, \ldots, u_J$ is the first J symbols in the LFSR output sequence, and $\mathbf{v}^{(k)}$ is the code sequence from the kth encoder. The code sequence from the kth code can also be written in the form of (5.3) as

$$\mathbf{v}^{(k)} = v_{B+1}^{(1,k)}, v_{B+1}^{(2,k)}, \ldots, v_{B+1}^{(m_{B+1}^{(k)},k)}, \ldots, v_{B+2}^{(1,k)}, v_{B+2}^{(2,k)}, \ldots, v_{B+2}^{(m_{B+2}^{(k)},k)}, \ldots,$$

where $v_n^{(j,k)}$ is the jth code symbol from code k at time n.

For each defined codeword symbol, $v_n^{(j,k)}$, we can construct an estimate from the observed keystream sequence $\mathbf{z}$. From (5.6) for the kth code we get, $v_n^{(j,k)} = u_{i_1}^{(j,k)} + u_{i_2}^{(j,k)}$. An estimate for $v_n^{(j,k)}$ is then given as $r_n^{(j,k)} = z_{i_1}^{(j,k)} + z_{i_2}^{(j,k)}$. Hence, we can construct sequences, referred to as received sequences, $\mathbf{r}^{(1)}, \mathbf{r}^{(2)}, \ldots, \mathbf{r}^{(M)}$, where

$$\mathbf{r}^{(k)} = r_{B+1}^{(1,k)}, r_{B+1}^{(2,k)}, \ldots, r_{B+1}^{(m_{B+1}^{(k)},k)}, \ldots, r_{B+2}^{(1,k)}, r_{B+2}^{(2,k)}, \ldots, r_{B+2}^{(m_{B+2}^{(k)},k)}, \ldots,$$

$1 \leqslant k \leqslant M$. For the information sequence $u_1, u_2, \ldots, u_J$ we get an estimate as $\mathbf{r}^{(0)} = z_1, z_2, \ldots, z_J$. The correlation between the LFSR output sequence and the keystream gives the probability that an estimate is correct as, $P(z_j = u_j) = 1 - p$ and $P(r_n^{(j,k)} = v_n^{(j,k)}) = 1 - 2p + p^2$ when $t = 2$.

In the decoding phase the M constituent codes are decoded one by one in a serial and iterative algorithm. The decoding algorithm is the same as the

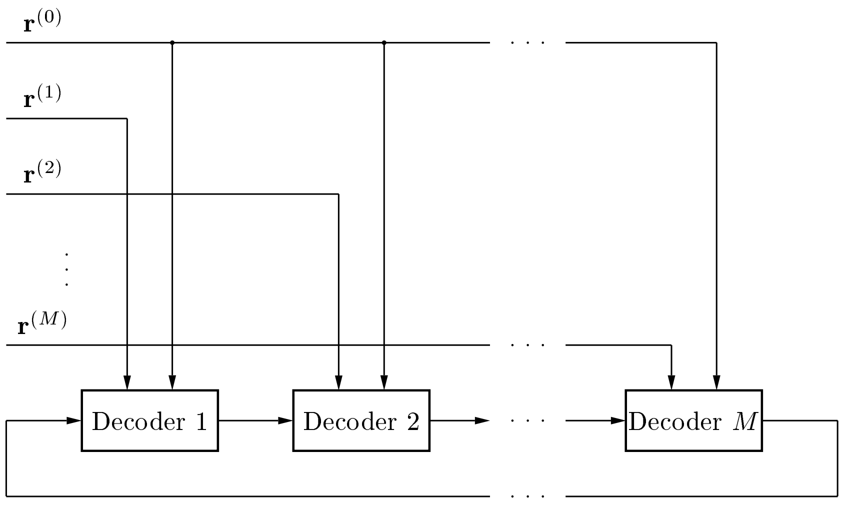

Figure 5.4: Model of the information flow in a turbo decoding algorithm.

decoding algorithm used for turbo codes [3]. In this chapter we follow the description of decoding of turbo codes in [40, Chapter 7]. Start by decoding the first code by an APP-decoding algorithm, such as the BCJR algorithm [1]. After the first decoder in the first iteration, we get the APP-probabilities,

$$P(u_n = 0|\mathbf{r}^{(0)}\mathbf{r}^{(1)}) \quad 1 \leqslant n \leqslant J, \tag{5.8}$$

for the information sequence. Using Bayes rule, (5.8) can be written as

$$P(u_n = 0|\mathbf{r}^{(0)}\mathbf{r}^{(1)}) = \frac{P(u_n = 0)P(\mathbf{r}^{(0)}\mathbf{r}^{(1)}|u_n = 0)}{P(\mathbf{r}^{(0)}\mathbf{r}^{(1)})}. \tag{5.9}$$

Define $\mathbf{r}^{(0)}_{\setminus n}$ to be the sequence $\mathbf{r}^{(0)}_{\setminus n} = (z_1, z_2, \ldots, z_{n-1}, z_{n+1}, z_{n+12}, \ldots, z_J)$, i.e, the symbol z_n is excluded from $\mathbf{r}^{(0)}$. Furthermore, since

$$P(\mathbf{r}^{(0)}\mathbf{r}^{(1)}|u_n = 0) = P(\mathbf{r}^{(0)}_{\setminus n}\mathbf{r}^{(1)}|u_n = 0)P(z_n|u_n = 0),$$

(5.9) can be written as,

$$P(u_n = 0|\mathbf{r}^{(0)}\mathbf{r}^{(1)}) = \frac{P(u_n = 0)P(z_n|u_n = 0)P(\mathbf{r}^{(0)}_{\setminus n}\mathbf{r}^{(1)}|u_n = 0)}{P(\mathbf{r}^{(0)}\mathbf{r}^{(1)})}. \tag{5.10}$$

For $u_n = 1$ we can derive the following APP probabilities,

$$P(u_n = 1|\mathbf{r}^{(0)}\mathbf{r}^{(1)}) = \frac{P(u_n = 1)P(z_n|u_n = 1)P(\mathbf{r}^{(0)}_{\setminus n}\mathbf{r}^{(1)}|u_n = 1)}{P(\mathbf{r}^{(0)}\mathbf{r}^{(1)})}. \qquad (5.11)$$

Combining (5.10) and (5.11) we get the following likelihood ratio,

$$\frac{P(u_n = 0|\mathbf{r}^{(0)}\mathbf{r}^{(1)})}{P(u_n = 1|\mathbf{r}^{(0)}\mathbf{r}^{(1)})} = \frac{P(u_n = 0)P(z_n|u_n = 0)P(\mathbf{r}^{(0)}_{\setminus n}\mathbf{r}^{(1)}|u_n = 0)}{P(u_n = 1)P(z_n|u_n = 1)P(\mathbf{r}^{(0)}_{\setminus n}\mathbf{r}^{(1)}|u_n = 1)}. \qquad (5.12)$$

Since the sequence $\mathbf{u}$ is generated by an LFSR we might assume that $P(u_n = 0) = P(u_n = 1) = 1/2$. We define the *intrinsic* information for the nth symbol, denoted by Λ_n^{int}, to be the ratio,

$$\Lambda_n^{\text{int}} = \frac{P(z_n|u_n = 0)}{P(z_n|u_n = 1)}.$$

The intrinsic information does not depend on the code, and hence, it will not vary through the iterations. Furthermore, define the *extrinsic* information for the nth symbol from the first decoder after the first iteration, denoted by $\Lambda_n^{\text{ext}(1)}(1)$, to be,

$$\Lambda_n^{\text{ext}(1)}(1) = \frac{P(\mathbf{r}^{(0)}_{\setminus n}\mathbf{r}^{(1)}|u_n = 0)}{P(\mathbf{r}^{(0)}_{\setminus n}\mathbf{r}^{(1)}|u_n = 1)}.$$

Hence, the likelihood ratio of the APP probabilities in (5.12), denoted by $\Lambda_n^{(1)}(1)$, can be expressed as

$$\Lambda_n^{(1)}(1) = \Lambda_n^{\text{int}}\Lambda_n^{\text{ext}(1)}(1). \qquad (5.13)$$

In the second decoder in the first iteration, the extrinsic information from the first decoder, $\Lambda_n^{\text{ext}(1)}(1)$, is used as a priori information when the ratio of the a posteriori probabilities $\Lambda_n^{(2)}(1)$ after the second decoder in the first iteration. Following (5.13) $\Lambda_n^{(2)}(1)$ can be written as

$$\Lambda_n^{(2)}(1) = \Lambda_n^{\text{int}}\Lambda_n^{\text{ext}(1)}(1)\Lambda_n^{\text{ext}(2)}(1). \qquad (5.14)$$

This is then continued for all M decoders. Hence, the ratio of the a posteriori probabilities $\Lambda_n^{(M)}(1)$ after the last decoder in the first iteration can be written as

$$\Lambda_n^{(M)}(1) = \Lambda_n^{\text{int}}\Lambda_n^{\text{ext}(1)}(1)\Lambda_n^{\text{ext}(2)}(1)\ldots\Lambda_n^{\text{ext}(M)}(1). \qquad (5.15)$$

After all M constituent codes have been decoded in the first iteration, the a posteriori probabilities are fed back to the first decoder. In (5.15) we see that the extrinsic information from all the decoders are present. To avoid too much dependency the extrinsic information from the kth decoder in the previous iteration is excluded in the next iteration. Hence, the ratio of the a priori probabilities into the kth decoder in the ith iteration is given as

$$\begin{aligned}\Lambda_n^{(k)}(i) =&\Lambda_n^{\mathrm{ext}(1)}(i)\Lambda_n^{\mathrm{ext}(2)}(i)\ldots\Lambda_n^{\mathrm{ext}(k-1)}(i)\\ &\Lambda_n^{\mathrm{ext}(k+1)}(i-1)\Lambda_n^{\mathrm{ext}(k+2)}(i-1)\ldots\Lambda_n^{\mathrm{ext}(M)}(i-1).\end{aligned} \tag{5.16}$$

The decoding procedure above is iterated I rounds. After decoding the Mth code in the Ith iteration, the resulting ratio of the a posteriori probabilities are given by

$$\Lambda_n^{(M)}(I) = \Lambda_n^{\mathrm{int}}\Lambda_n^{\mathrm{ext}(1)}(I)\Lambda_n^{\mathrm{ext}(2)}(I)\ldots\Lambda_n^{\mathrm{ext}(M)}(I). \tag{5.17}$$

If $\Lambda_n^{(M)}(I) > 1$ then z_n is decoded 0, otherwise z_n is decoded to 1. The typical number of iterations that has to be made for the decoding to converge is approximately 15–20. A comprehensive description of the procedure for M constituent codes/decoders is given in Figure 5.5.

In: The $l \times N$ generator matrix G_{LFSR} for the code generated by the LFSR, the received sequence $\mathbf{z}$, the error probability p, the number of iterations I, and the number of constituent codes M.

1. Let $\pi_1, \pi_2 \ldots, \pi_M$ be M random permutations permuting indices $B+1, \ldots, J$ and leaving the other indices fixed. Let $G_1 = \pi_1(G_{LFSR}), G_2 = \pi_2(G_{LFSR}), \ldots, G_M = \pi_M(G_{LFSR})$ be generator matrices for M different codes. For each G_i, $1 \leqslant i \leqslant M$, find all parity checks of the form (5.6).

   Decoding phase:

2. After receiving $\mathbf{z}$, construct the received sequences $\mathbf{r}^{(0)}, \mathbf{r}^{(1)}, \mathbf{r}^{(2)}, \ldots, \mathbf{r}^{(M)}$ by

$$\mathbf{r}^{(0)} = z_1, z_2, \ldots, z_J,$$

$$\mathbf{r}^{(k)} = r_{B+1}^{(1,k)}, \ldots, r_{B+1}^{(m_{B+1}^{(k)},k)}, \ldots, r_{B+2}^{(1,k)}, \ldots, r_{B+2}^{(m_{B+2}^{(k)},k)}, \ldots,$$

   where,

$$r_n^{(j,k)} = z_{i_1}^{(j,k)} + z_{i_2}^{(j,k)}, \quad 1 \leqslant j \leqslant m_k(n), 1 \leqslant k \leqslant M, n > B.$$

3. Initiate Λ_n^{int}. If $z_n = 0$ then $\Lambda_n^{\text{int}} = \frac{1-p}{p}$, otherwise $\Lambda_n^{\text{int}} = \frac{p}{1-p}$. Let $\Lambda_n^{\text{ext}(k)}(0) = 1$, $1 \leqslant k \leqslant M$, and let $i = 1$.

4. Decode each constituent code with an APP decoding algorithm. The ratio of the a priori probabilities to the kth decoder is

$$\begin{aligned}\Lambda_n^{(k)}(i) =& \Lambda_n^{\text{ext}(1)}(i)\Lambda_n^{\text{ext}(2)}(i) \ldots \Lambda_n^{\text{ext}(k-1)}(i) \\ & \Lambda_n^{\text{ext}(k+1)}(i-1)\Lambda_n^{\text{ext}(k+2)}(i-1) \ldots \Lambda_n^{\text{ext}(M)}(i-1).\end{aligned}$$

   Calculate the extrinsic information $\Lambda_n^{\text{ext}(k)}(i)$. Let $i \leftarrow i+1$.

5. If $i < I$ go to 4., otherwise calculate the resulting ratio of the a posteriori probabilities

$$\Lambda_n^{(M)}(I) = \Lambda_n^{\text{int}}\Lambda_n^{\text{ext}(1)}(I)\Lambda_n^{\text{ext}(2)}(I) \ldots \Lambda_n^{\text{ext}(M)}(I),$$

   for each of the symbols $u_{5B+1}, u_{5B+2}, \ldots, u_{5B+l}$, calculate the initial state $\mathbf{u}_0$, and check it for correctness.

Figure 5.5: A description of the fast correlation attack using turbo decoding algorithm.

B	Chapter 4	$M = 1$	$M = 2$	$M = 4$	$M = 8$	$M = 16$
12	0.12	0.18	0.21	0.22	0.23	0.25
13	0.19	0.20	0.22	0.24	0.25	0.26
14	0.22	0.23	0.24	0.26	0.27	0.28
15	0.26	0.26	0.27	0.29	0.30	0.30

Table 5.2: Maximum p for turbo algorithm with $B = 12, \ldots, 15$ and varying M when $N = 40000$.

B	Chapter 4	$M = 1$	$M = 2$	$M = 4$	$M = 8$	$M = 16$
10	0.31	0.31	0.33	0.34	0.35	0.36
11	0.34	0.34	0.36	0.37	0.38	0.38
12	0.36	0.37	0.38	0.38	0.39	0.39
13	0.37	0.39	0.40	0.40	0.41	0.41

Table 5.3: Maximum p for turbo algorithm with $B = 10, \ldots, 13$ and varying M when $N = 400000$.

5.3 Simulation Results

In this section we present some simulation results for algorithm based on turbo codes. The parameter values are $J = 10B + l$ and $I = 15$. In Table 5.2 we show the maximum error probability for a received sequence of length $N = 40000$ when the memory B is varying in the range $10 - 13$ and the number of constituent codes is $1, 2, 4, 8$ and 16. Table 5.3 then shows the same for length $N = 400000$. In the tables we also give the results obtained by the algorithm proposed in Chapter 4.

We can see the performance improvement with growing M for fixed B. A few comments regarding computational complexity and memory requirements are in place.

If one uses the suboptimal APP decoding algorithm in [92] the memory requirements will be roughly the same as in Viterbi decoding. The computational complexity for the algorithm in [92] is roughly a factor 3 higher compared to the Viterbi algorithm, since it runs through the trellis three times. There are also slightly different operations performed in the algorithms. The computational complexity is then further increased a factor M when the turbo algorithm with M constituent codes are considered. Finally, we iterate at least twice. To conclude, for fixed parameters B and N, the turbo algorithm have roughly the same memory requirements, but an increase of computational complexity of at least a factor $6M$.

It is important to note that in many cases, the possible performance is not limited by the computational complexity, but rather, limited by the required memory. For example, if $N = 40000$, the maximal memory size that our current implementation could handle for the basic algorithm on a regular PC (for the example given in Table 5.1) was $B = 17$, but this required only roughly half an hour CPU time. Hence, in this case we do not consider the penalty of increased computational complexity to be severe.

5.4 Variant Based on Parallel Decoding

As can be seen from the description of the turbo algorithm, it is not directly parallelizable (the APP decoding can be partly parallelized). Since it is very reasonable to assume that the opponent can compute in parallel, we shortly describe a modified turbo algorithm. Assume that we have access to M different processors. We can then construct M different constituent convolutional codes exactly as described in Section 5.1 using some suitable memory B. After having received the keystream $\mathbf{z}$, the received sequences $\mathbf{r}^{(0)}, \mathbf{r}^{(1)}, \ldots, \mathbf{r}^{(M)}$ are constructed. Next, processor number i works as the APP decoder for code number i. Since the decoders are working in parallel, the extrinsic information can not be used in other decoders in the same iteration. Hence, we must modify the expressions for calculating the ratio of the a priori probabilities. Hence, the ratio of the a priori probabilities entering the kth decoder in the ith iteration is given as

$$\begin{aligned}\Lambda_n^{(k)}(i) =& \Lambda_n^{\mathrm{ext}(1)}(i-1)\Lambda_n^{\mathrm{ext}(2)}(i-1)\ldots\Lambda_n^{\mathrm{ext}(k-1)}(i-1)\\ & \Lambda_n^{\mathrm{ext}(k+1)}(i-1)\Lambda_n^{\mathrm{ext}(k+2)}(i-1)\ldots\Lambda_n^{\mathrm{ext}(M)}(i-1).\end{aligned} \tag{5.18}$$

Each decoder then outputs the extrinsic information. As before, the ratio of the a posteriori probabilities after decoding all the M codes in the Ith iteration, is given as

$$\Lambda_n^{(M)}(I) = \Lambda_n^{\mathrm{int}}\Lambda_n^{\mathrm{ext}(1)}(I)\Lambda_n^{\mathrm{ext}(2)}(I)\ldots\Lambda_n^{\mathrm{ext}(M)}(I).$$

The structure is depicted in Figure 5.6. Some simulation result for $N = 40000$ are given in Table 5.4. As seen from Table 5.4, the performance of the parallel decoding algorithm is almost the same as for the ordinary turbo decoding algorithm. However, more iterations have to be made to get convergence. The explanation of this behavior is that the extrinsic information in the parallel decoding algorithm is not used until the next iteration. On the other hand, for the serial decoding algorithm the extrinsic information is used already in the decoding of the next code in the same iteration.

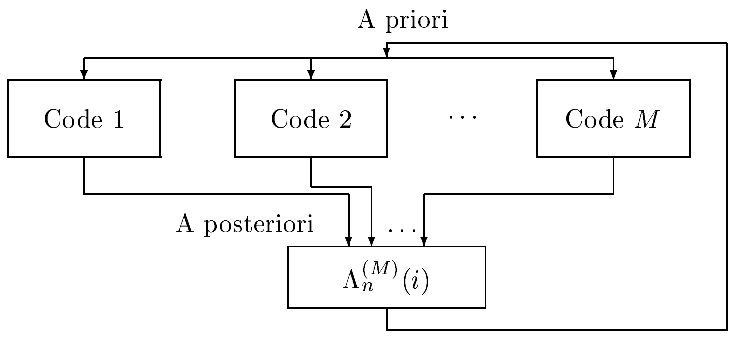

Figure 5.6: Structure of a parallel turbo decoding algorithm.

	$M = 1$	$M = 2$	$M = 4$	$M = 8$	$M = 16$
Turbo	0.20	0.22	0.24	0.25	0.26
Parallel turbo	0.20	0.22	0.23	0.23	0.24

Table 5.4: Maximum p for the parallel turbo algorithm when $N = 40000$ and $B = 13$.

5.5 Summary

In this chapter we have shown how iterative decoding techniques based on the ideas from the construction and decoding of turbo codes can be used as a basis for correlation attacks on stream ciphers. The performance has been demonstrated through simulations.

Although we can reduce the memory complexity of the decoder by using several parallel convolutional codes, we still have problem with the memory requirements. When the length of the LFSR is long we need a fairly large value of B to find any parity check equations in the precomputation phase.

The proposed iterative decoding techniques have opened for other possibilities that can be considered in the future. An alternative to the decoding structure in this work, as given in Figure 5.1, could be the following. Consider all index positions that are used to build parity check equations for the convolutional code. Now consider these positions as information symbols for another convolutional code, and find parity checks for this code. Decoding this code will provide APP probabilities for its information symbols and hence more reliable parity checks for the first code. This idea is easily generalized to more complex decoding structures.

6 An Algorithm Based on Reconstruction of Linear Polynomials

In Chapter 4, a fast correlation attack based on convolutional decoding was presented. The main drawback with this algorithm is that a large memory is needed in the decoding process. To circumvent the problem with the memory consumption, a method based on several parallel convolutional codes, as presented in Chapter 5, can be used. Even if the memory requirements were slightly reduced, a fast correlation attacks based on convolutional codes still require to much memory for being practical for long shift registers. Chebyzhov, Johansson, and Smeets proposed a new method for fast correlation attacks in [13]. The method they proposed has approximately the same performance as the algorithm based on convolutional codes in Chapter 4, but it has a very low memory consumption in the decoding phase.

In this section we model the fast correlation attack as the problem of learning a binary linear multivariate polynomial. Algorithms for polynomial reconstruction with queries can be modified through some general techniques used in fast correlation attacks. The result is a new and efficient way of performing fast correlation attacks, with memory complexity the same as the method proposed in [13].

This chapter is organized as follows. In Section 6.1 we give the preliminaries on the standard model that is used for cryptanalysis and reformulate this into a polynomial reconstruction problem. In Section 6.2 we review an algorithm by Goldreich, Rubinfeld and Sudan [32] that solves the polynomial reconstruction problem with queries in polynomial time. In Section 6.3 we derive a new algorithm for fast correlation attacks, inspired by the previous section. The algorithm was originally proposed in [45]. In Section 6.4 we present simulation results and a comparison with other algorithms, and in Section 6.5 a sketch of a theoretical analysis of the algorithm is presented. In Section 6.6 we present a sequential version of the new algorithm, i.e, this algorithm builds a tree of possible candidates and searches through it.

6.1 Preliminaries and Model

Most authors [15, 43, 44, 61, 62, 71, 82] use the approach of viewing fast correlation attacks as a decoding problem over the binary symmetric channel. However, in this section we show that fast correlation attacks can equivalently be viewed as the problem of learning a linear multivariate polynomial.

Recall from Section 3.2 that the target LFSR have length l and feedback polynomial $g(x)$. The number of possible LFSR sequences is 2^l and the known keystream sequence $\mathbf{z} = (z_1, z_2, \ldots, z_N)$ is of length N.

The assumed correlation between u_i and z_i is described by the correlation probability $1/2 + \epsilon$, defined by $1/2 + \epsilon = P(u_i = z_i)$, where $0 < \epsilon < 1/2$. The problem of cryptanalysis is the following. Given the received word $(z_1, z_2, \ldots, z_N)$ as the output of the stream cipher, find the initial state (or at least a part of it) of the target LFSR.

Let the unknown initial state of the target LFSR be denoted by

$$\mathbf{u}_0 = (u_1, u_2, \ldots, u_l). \tag{6.1}$$

From Section 3.2 we know that there exists an $l \times N$ matrix, G_{LFSR}, of the form

$$G_{LFSR} = \begin{pmatrix} \mathbf{h}_0 & \mathbf{h}_1 & \ldots & \mathbf{h}_N \end{pmatrix}, \tag{6.2}$$

such that $u_i = \mathbf{u}_0 \mathbf{h}_i$. Hence, we can express each u_i as some known linear combination of the initial state $\mathbf{u}_0$, i.e.,

$$u_i = h_{i1} u_1 + h_{i2} u_2 + \cdots + h_{il} u_l, \tag{6.3}$$

where h_{ij} is the element in row j of column i in G_{LFSR}.

Define the *initial state polynomial*, denoted $U_0(\mathbf{x})$, to be

$$U_0(\mathbf{x}) = U_0(x_1, x_2, \ldots, x_l) = u_1 x_1 + u_2 x_2 + \cdots + u_l x_l. \tag{6.4}$$

We introduce the variables $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$, where $\mathbf{x}_i = \mathbf{h}_i^T$, for $1 \leqslant i \leqslant N$, i.e.,$\mathbf{x}_i = (h_{i1}, h_{i2}, \ldots, h_{il})$. With this notation, we can express each u_i as being the initial state polynomial evaluated in some known point $\mathbf{x}_i$,

$$u_i = U_0(\mathbf{x}_i), \quad i \geqslant 1. \tag{6.5}$$

The correlation between u_i and z_i can be described by introducing a noise vector

$$\mathbf{e} = (e_1, e_2, \ldots, e_N), \tag{6.6}$$

where $e_i \in \mathbb{F}_2$ are independent random variables for $1 \leq i \leq N$ and $P(e_i = 0) = 1/2 + \epsilon$. Then we model the correlation by writing $\mathbf{z} = \mathbf{u} + \mathbf{e}$, giving

$$\mathbf{z} = (U_0(\mathbf{x}_1) + e_1, U_0(\mathbf{x}_2) + e_2, \ldots, U_0(\mathbf{x}_N) + e_N), \tag{6.7}$$

where $\mathbf{x}_i$ are known l-tuples for all $1 \leqslant i \leqslant N$. In conclusion, we have reformulated our problem into the following.

The output vector z consists of a number of noisy observations of an unknown polynomial $U(\mathbf{x})$ evaluated in different known points $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$. The task of the attacker is to determine the unknown polynomial $U(\mathbf{x})$.

6.2 Learning Polynomials with Queries

In computational learning theory (see e.g., [32] and its references), one might want to consider the following general *reconstruction problem:*

Given: An oracle (black box) for an arbitrary unknown function $f : F^l \to F$, a class of functions $\mathcal{F}$ and a parameter δ.

Problem: Provide a list of all functions $g \in \mathcal{F}$ that agree with f on at least a δ fraction of the inputs.

The general reconstruction problem can be interpreted in several ways. We consider only the paradigm of learning with persistent noise. Here we assume that the output of the oracle is derived by evaluating some specific function in $\mathcal{F}$ and then adding noise to the result. A lot of work on different settings for this problem can be found.

We will now pay special attention to the work of Goldreich, Rubinfeld and Sudan in [32]. They consider a case of the reconstruction problem when the hypothesis class $\mathcal{F}$ is the set of linear polynomials in l variables (actually, any polynomial degree d was considered in [32], but we are only interested in the linear case). In the binary case ($F = \mathbb{F}_2$), they demonstrate an algorithm that given $\epsilon > 0$ and provided oracle access to an arbitrary function $f : \mathbb{F}_2^l \to \mathbb{F}$, runs in time poly$(l/\epsilon)$ and outputs a list of all linear functions in l variables that agree with f on at least $\delta = 1/2 + \epsilon$ of the output.

Let us immediately describe the procedure. First, the problem description can be as follows. On a selected input $\mathbf{x}$, the oracle evaluates an unknown linear function $p(\mathbf{x})$, adds a noise value e, and outputs the result $p(\mathbf{x}) + e$. On the next oracle access, the function is evaluated in a new point and a new noise value is added.

The algorithm for solving the above problem given in [32] is a generalization of an algorithm given in [31] (in the binary case that we consider they coincide). Consider all polynomials of the form

$$p(\mathbf{x}) = \sum_{i=1}^{l} c_i x_i.$$

The algorithm uses the concept of *b-prefixes*, called i-prefixes in [32], which is defined to be all polynomials that can be expressed in the form

$$p(x_1, x_2, \ldots, x_b, 0, 0, \ldots, 0).$$

This means that an b-prefix is a polynomial in l variables in which only the first b variables appear.

The algorithm proceeds in l rounds, so that in the bth round we have a list of candidates for the b-prefixes of $p(\mathbf{x})$. The list of b-prefixes is generated by extending the list of $(b-1)$-prefixes from the previous round in all possible ways, i.e., by adding or not adding the x_b variable to each of the members of the $(b-1)$-prefixes. Hence the list is doubled in cardinality. After the extension, a screening process takes place. The screening process guarantees that the b-prefix of the correct solution passes with high probability and that not too many other prefixes pass.

The screening process is done by testing each candidate prefix, denoted $(c_1, c_2, \ldots, c_b)$, as follows. Pick $n = \mathrm{poly}(l/\epsilon)$ sequences uniformly from $\mathbb{F}_2^{l-b}$. For each such sequence, denoted $(s_{b+1}, \ldots, s_l)$, and for every $\xi \in \mathbb{F}_2$, estimate the quantity

$$P(\xi) = P\left(f(\mathbf{r}, \mathbf{s}) = \sum_{j=1}^{b} c_j r_j + \xi \right),$$

when $\mathbf{r} = (r_1, r_2, \ldots, r_b)$ is chosen randomly from $\mathbb{F}_2^i$. Here $(\mathbf{r}, \mathbf{s})$ denotes the vector $(r_1, \ldots, r_b, s_{b+1}, \ldots, s_l)$. All these probabilities can be approximated simultaneously by using a sample of $\mathrm{poly}(l/\epsilon)$ sequences $(r_1, \ldots, r_i)$. A candidate is considered to pass the test if for at least one sequence $(s_{i+1}, \ldots, s_l)$ there exists ξ such that the estimate $P(\xi)$ is greater than $1/2 + \epsilon/3$. It is shown in [32] that the correct candidate passes the test with overwhelming probability, and that not too many other candidates do. For more details on this algorithm, we refer to [32].

6.3 Fast Correlation Attacks Based on Algorithms for Learning Polynomials

We observe the similarities between our correlation attack problem described as a polynomial reconstruction problem as in Section 6.1, and the problem of learning polynomials with queries as described in the previous section.

Since queries are essential in the polynomial time algorithm of the previous section, it can not be applied directly to the correlation attack problem. *In the query case, sample points given to the oracle can be chosen, whereas for correlation attacks the sample points are simply randomly selected.* The latter problem is actually a well-known problem also in learning theory, called "learning parity with noise", and it is commonly believed to be hard, see [7,49]. Nevertheless, we are interested in finding as efficient correlation attacks as possible, and we will now derive an algorithm that is inspired by the results presented in the previous section.

Let us first briefly review our problem formulation. The recovery of the initial state of the target LFSR is viewed as the problem of recovering an unknown binary linear polynomial $U_0(\mathbf{x})$ in l variables. To our disposal, we have a number N of noisy observations of this polynomial (the output sequence), denoted

$$\mathbf{z} = (z_1, z_2, \ldots, z_N).$$

The noise is such that

$$P(z_i = U_0(\mathbf{x}_i)) = 1/2 + \epsilon, \quad 1 \leqslant i \leqslant N,$$

where $\mathbf{x}_i$ are known random l-tuples for all $1 \leqslant i \leqslant N$.

Our problem in applying the algorithm described in Section 6.2 is the fact that we are not able to select the points $\mathbf{x}_i$ ourselves. Instead, we are restricted to use the values given by the N columns of G_{LFSR}. This can to some extent be compensated for by the following observation [44].

Assume that we have noisy observations z_{i_1} and z_{i_2} of the polynomial $U(\mathbf{x})$ in two points $\mathbf{x}_{i_1}$ and $\mathbf{x}_{i_2}$, respectively, i.e., $P(z_{i_1} = U_0(\mathbf{x}_{i_1})) = 1/2 + \epsilon$ and $P(z_{i_2} = U_0(\mathbf{x}_{i_2})) = 1/2 + \epsilon$. Since $U_0(\mathbf{x})$ is a linear polynomial, we can get an observation in the point $\mathbf{x}_{i_1} + \mathbf{x}_{i_2}$ as

$$\begin{aligned} z_{i_1} + z_{i_2} &= U_0(\mathbf{x}_{i_1}) + e_{i_1} + U_0(\mathbf{x}_{i_2}) + e_{i_2} \\ &= U_0(\mathbf{x}_{i_1} + \mathbf{x}_{i_2}) + e_{i_1} + e_{i_2}. \end{aligned} \tag{6.8}$$

The probability that $z_i + z_j = U_0(\mathbf{x}_i + \mathbf{x}_j)$ is given as

$$\begin{aligned} P(z_{i_1} + z_{i_2} = U_0(\mathbf{x}_{i_1} + \mathbf{x}_{i_2})) &= P(z_{i_1} + z_{i_2} = U_0(\mathbf{x}_{i_1}) + U_0(\mathbf{x}_{i_2})) \\ &= P(z_{i_1} = U_0(\mathbf{x}_{i_1}))P(z_{i_2} = U_0(\mathbf{x}_{i_2})) \\ &\quad + P(z_{i_1} \neq U_0(\mathbf{x}_{i_1}))P(z_{i_2} \neq U_0(\mathbf{x}_{i_2})) \\ &= (1/2 + \epsilon)^2 + (1/2 - \epsilon)^2 \\ &= 1/2 + 2\epsilon^2. \end{aligned} \tag{6.9}$$

Next, observe that we do not have to restrict ourselves to addition of just two sample points, but can consider any sum of t points. Hence, any $\sum_{j=1}^{t} z_{i_j}$,

$i_1, \ldots i_t \in \{1, 2, \ldots, N\}$, will be a noisy observation of $U(\sum_{j=1}^{t} \mathbf{x}_{i_j})$ with noise level

$$P(\sum_{j=1}^{t} z_{c_j} = U_0(\sum_{j=1}^{t} \mathbf{x}_{i_j})) = 1/2 + 2^{t-1}\epsilon^t. \tag{6.10}$$

For convenience, we introduce the notation $\hat{\mathbf{x}} = \sum_{j=1}^{t} \mathbf{x}_{i_j}$, $\hat{z} = \sum_{j=1}^{t} z_{i_j}$, and $\hat{e} = \sum_{j=1}^{t} e_{i_j}$ and write

$$U_0(\hat{\mathbf{x}}) = \hat{z} + \hat{e},$$

where $\hat{e}$ is a binary random variable with $P(e = 0) = 1/2 + 2^{t-1}\epsilon^t$, from (6.10).

If we want to use the algorithm in Section 6.2 we must feed the oracle with $\hat{\mathbf{x}}$ points of a special form. An idea in the algorithm to be described is to construct such points by adding suitable vectors $\mathbf{x}_{i_j}$ in such a way that their sum is of the required form. Clearly, the noise level increases with the number of vectors in the sum, so we are interested in having as few vectors as possible summing to the desired form. On the other hand, allowing only very few vectors in the sum will give us only very few $\hat{\mathbf{x}}$ vectors of the desired form. Hence, there is a tradeoff for the value of the constant t. We return to this issue in the theoretical analysis.

Also, we introduce a slightly modified version of the algorithm from Section 6.2. The new version includes a squared distance used in the test in the screening procedure. We first consider a version in which the idea of b-prefixes is removed. In Section 6.6 we elaborate on the idea of b-prefixes. A description of the basic algorithm is given in Figure 6.1.

Let us give an intuitive explanation of the algorithm. We first note that the algorithm recovers the first B bits of the initial state, namely $(u_1, \ldots, u_B)$. The remaining part of the initial state can be recovered in a similar way. However, recover the remaining bits is a simpler problem, and we can ignore the complexity of recover them.

Now consider the case of one hypothesized value of $(u_1, \ldots, u_B)$. We want to check whether this value, denoted $(\hat{u}_1, \ldots, \hat{u}_B)$, is correct or not. This is done by first selecting a certain $(l-B)$-tuple $\mathbf{s}^{(k)} = (s_1^{(k)}, s_2^{(k)}, \ldots, s_{l-B}^{(k)})$, and then by finding all linear combinations of t vectors in $\{\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_N\}$,

$$\hat{\mathbf{x}}^{(k)} = \sum_{j=1}^{t} \mathbf{x}_{i_j}, \tag{6.11}$$

having the special form

$$\hat{\mathbf{x}}^{(k)} = (\hat{x}_1, \ldots, \hat{x}_B, s_1^{(k)}, \ldots, s_{l-B}^{(k)}), \tag{6.12}$$

for arbitrary values of $\hat{x}_1, \ldots, \hat{x}_B$ (not all zero). The complexity of this precomputation step depends on t, and by using some simple birthday-paradox arguments, one can show that the computation can be done in $O(N^{\lceil t/2 \rceil})$ using $O(N^{\lfloor t/2 \rfloor})$ storage. When $t > 3$, the memory complexity can be improved to $O(N^{\lfloor t/2 \rfloor})$, by using the improvement proposed in [18].

The main observation is that the relation between $U(\hat{\mathbf{x}}^{(k)})$ and $\hat{z}^{(k)}$ can, from our previous arguments, be written in the form

$$U(\hat{\mathbf{x}}^{(k)}) = \hat{z}^{(k)} + \hat{e}, \tag{6.13}$$

where $\hat{e}$ represents the noise having a noise level of $P(\hat{e} = 0) = 1/2 + 2^{t-1}\epsilon^t$. Now (6.13) is equivalently expressed as

$$\sum_{j=1}^{B} u_j \hat{x}_j + \sum_{j=B+1}^{l} u_j s_j^{(k)} = \hat{z}^{(i)} + \hat{e}, \tag{6.14}$$

and this can be rewritten as

$$\sum_{j=1}^{B} (u_j + \hat{u}_j)\hat{x}_j + \sum_{j=B+1}^{l} u_j s_j^{(k)} + \hat{e} = \sum_{j=1}^{B} \hat{u}_j \hat{x}_j + \hat{z}^{(k)}. \tag{6.15}$$

Recall that $W = \sum_{j=B+1}^{l} u_j s_j^{(k)}$ in (6.15) is a fixed binary random variable for all linear combinations of the special form that we required, i.e., for a fixed $\mathbf{s}(k)$ we will have either $W = 0$ for all our $\hat{\mathbf{x}}^{(k)}$s, or $W = 1$.

Consider a correct hypothesized value, i.e., $\hat{u}_j = u_j$ for $j = 1, \ldots, B$, and assume that we have $S^{(k)}$ equations. Let α be the number of times we have

$$\sum_{j=1}^{B} \hat{u}_j \hat{x}_j = \hat{z}^{(k)}.$$

Then α simply counts the number of times the right hand side in (6.15) is zero. Since $\sum_{j=1}^{B} (u_j + \hat{u}_j)\hat{x}_j = 0$, the probability for the left hand side to be zero is then $P(W + \hat{e} = 0)$. This probability is either $1/2 - 2^{t-1}\epsilon^t$ or $1/2 + 2^{t-1}\epsilon^t$ for all equations, depending on whether $W = 0$ or $W = 1$. Thus α has a binomial distribution $Bin(S^{(k)}, p)$, with p being one of the two probabilities above.

However, if the hypothesized valued was wrong, then

$$P(\sum_{j=1}^{B} (u_j + \hat{u}_j)\hat{x}_j = 0) = 1/2,$$

and hence, it will result in α being binomial distributed, $Bin(S^{(k)}, p)$, with $p = 1/2$.

In order to separate the two hypothesis we calculate the number of times we have $\sum_{j=1}^{B} \hat{u}_j \hat{x}_j = \hat{z}^{(k)}$, denote this by α. Then a squared distance $((S^{(k)} - 2 \cdot \alpha)^2)$ is used. In Section 6.5 we show that under certain conditions, the squared distance is close to optimal. If we have enough points, i.e., we can create enough different $\hat{\mathbf{x}}$ as linear combinations of at most t $\mathbf{x}_i$'s, we will be able to separate the two hypotheses. However, the number of linear combinations for a particular $\mathbf{s}^{(k)}$ value is limited. Hence, we also run through a lot of different $\mathbf{s}^{(k)}$ values. Each such value of $\mathbf{s}^{(k)}$ gives a squared distance, and we sum them all up to become our overall distance, denoted by β.

In: $\mathbf{z} = (z_1, \ldots, z_N)$, $[\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N]$, and constants t, B and n.

1. (Precomputation) Select n different $(l - B)$-tuples $\mathbf{s}^{(1)}, \mathbf{s}^{(2)}, \ldots, \mathbf{s}^{(n)}$. For each $\mathbf{s}^{(k)}$, find all linear combinations of the form $\hat{\mathbf{x}}^{(k)} = \sum_{j=1}^{t} \mathbf{x}_{i_j}$ which are of the special form
$$\hat{\mathbf{x}}^{(k)} = (\hat{x}_1, \ldots, \hat{x}_B, \mathbf{s}_1^{(k)}, \ldots, \mathbf{s}_{l-B}^{(k)}),$$
for arbitrary values of $\hat{x}_1, \ldots, \hat{x}_B$. Store all $\hat{\mathbf{x}}^{(k)}$ together with all $\hat{z}^{(k)} = \sum_{j=1}^{t} z_{i_j}$. Let the set of all such pairs for a given value of $\mathbf{s}^{(k)}$ have cardinality $S^{(k)}$.
2. Run through all 2^B values of the constants $(\hat{u}_1, \ldots, \hat{u}_B)$ as follows. Set $\beta \leftarrow 0$.
3. For each $\mathbf{s}^{(k)}$, run through all $S^{(k)}$ stored pairs $\{(\hat{\mathbf{x}}^{(k)}, \hat{z}^{(k)})\}$, calculate the number of times we have
$$\sum_{j=1}^{B} \hat{u}_j \hat{x}_j = \hat{z}^{(k)},$$
and denote this by α. Update
$$\beta \leftarrow \beta + (S^{(k)} - 2 \cdot \alpha)^2.$$
4. If β is the highest received value so far, store $(\hat{u}_1, \ldots, \hat{u}_B)$. Set $\beta \leftarrow 0$.

Out: Output $(\hat{u}_1, \ldots, \hat{u}_B)$ having the highest value of β.

Figure 6.1: A description of the basic algorithm.

6.4 Simulation Results

In this section we present some simulation results for the basic algorithm described in Section 6.3. Simulations are presented for $t = 2$ and $t = 3$. In general, increasing t will increase the performance at the cost of an increased precomputation time and increased memory requirement in precomputation.

The first simulations are for the same feedback polynomial and the same length of the observed keystream as in [11,16,70] and the previous chapters, i.e., a feedback polynomial with $l = 40$. Table 6.1 shows the maximum error probability $p = 1/2 - \epsilon$ for the basic algorithm when the received sequence is of length $N = 400000$. The parameter B is varying in the range $13 - 16$ and n is in the set $n \in \{1, 2, 4, 8, \ldots, 512\}$. As a particular example, when

$N = 400000$

n	$B = 13$	$B = 14$	$B = 15$	$B = 16$
1	0.30	0.32	0.34	0.36
2	0.32	0.34	0.36	0.38
4	0.34	0.36	0.38	0.40
8	0.36	0.38	0.40	0.41
16	0.38	0.39	0.41	0.42
32	0.39	0.40	0.42	0.43
64	0.40	0.41	0.42	0.44
128	0.41	0.42	0.43	0.44
256	0.42	0.43	0.43	0.45
512	0.42	0.44	0.44	0.45

Table 6.1: Maximum $p = 1/2 - \epsilon$ for the basic algorithm with $t = 2$, $B = 13, \ldots, 16$, varying n, and $N = 400000$.

$B = 16, n = 256$ we reach $p = 0.45$ having 400000 known keystream symbols. The running time is less than 3 minutes, and the precomputation time is negligible.

It is important to observe that for a fixed running time, the performance increases with increasing n (up to a certain point). The table entries $\{B = 16, n = 1\}$, $\{B = 15, n = 4\}$, $\{B = 14, n = 16\}$, $\{B = 13, n = 64\}$ all have roughly the same computational complexity, but an increasing performance with n can be observed.

More interesting is perhaps to show simulation results for longer LFSRs, as was done in [13]. We present results for the basic algorithm when $l = 60$ using a feedback polynomial of weight 13. In Table 6.2 we show the required computational complexity and the achieved correlation probability for different algorithm parameters. The implementations were written in

$l = 60$, $t = 2$

N	k	n	time	p
$40 \cdot 10^6$	23	1	96 sec	0.35
$40 \cdot 10^6$	22	2	48 sec	0.36
$40 \cdot 10^6$	21	4	25 sec	0.36
$40 \cdot 10^6$	25	1	26 min	0.40
$40 \cdot 10^6$	24	2	13 min	0.40
$40 \cdot 10^6$	23	4	6.5 min	0.40
$40 \cdot 10^6$	22	8	3.3 min	0.41
$40 \cdot 10^6$	25	4	106 min	0.43

$l = 60$, $t = 3$

N	k	n	time	p
$1.5 \cdot 10^5$	24	1	4.5 min	0.3
$1.5 \cdot 10^5$	23	2	2.3 min	0.3
$1.5 \cdot 10^5$	22	4	69 sec	0.3
$1.5 \cdot 10^5$	25	1	18 min	0.32
$1.5 \cdot 10^5$	24	2	9.2 min	0.32
$1.5 \cdot 10^5$	23	4	4.6 min	0.32

Table 6.2: Performance of the basic algorithm with $l = 60$ when $t = 2$ and $t = 3$, respectively.

C and the running times were measured on a Sun Ultra-80 running under Solaris.

We can compare with other suggested methods. Actually, in the special case of $n = 1$, our proposed algorithm will coincide with the method in [13]. This enables us to see the improvement in Table 6.2, by observing the decrease of decoding time when n increases (for a fixed p).

An important advantage for the proposed method is the storage complexity. The attacks based on convolutional and turbo codes in Chapter 4 and Chapter 5, respectively, uses a trellis with 2^B states. Hence, the size of B is limited to $20-30$ in practise, due to the fact that it must be kept in memory during decoding. On the other hand, the memory requirements for the algorithms presented in this paper remain constant when B increases. Also, the proposed algorithm can be trivially parallelized, and hence the only limiting factor is the total computational complexity.

6.5 Theoretical Analysis

In this section we sketch some results for a theoretical analysis of the proposed algorithms. First, we give an expression for the expected number of linear combinations of the form (6.11), i.e., the expected value of the parameter $S^{(k)}$ in the algorithm. Let $E[S^{(k)}]$ be the expected number of linear combinations of the form (6.11) that can be created from t out of N random vectors $\mathbf{x}_1, \ldots, \mathbf{x}_N$. From Lemma 4.1 we get that $E[S^{(k)}]$ is given as

$$E[S^{(k)}] = \frac{\binom{N}{t}}{2^{l-B}}. \tag{6.16}$$

Next, we show that using β as defined in Section 6.3 gives close to optimal performance. We start by considering the case when we have one fixed value of $\mathbf{s}^{(k)}$. Then we generalize to the case with several different $\mathbf{s}^{(\mathbf{k})}$ vectors.

Assume that for the given $\mathbf{s}^{(k)}$ we have created $S^{(k)}$ noisy observations of the polynomial $U(\mathbf{x})$. The expected value of $S^{(k)}$ is then given by (6.16). Assume further that we are considering a particular candidate $(\hat{u}_1, \ldots, \hat{u}_B)$, and that we have found α observations such that $\sum_{j=1}^{B} \hat{u}_j \hat{x}_j = \hat{z}^{(k)}$. Consider two hypothesis H_0, and H_1. Let H_1 be the hypothesis that the candidate is correct, and H_0 that the candidate is wrong.

Introduce the random variable $W = \sum_{j=B+1}^{l} u_j s_j^{(k)}$. Define p_0 as $p_0 = P(e = 0) = 1/2 + 2^{t-1}\epsilon^t$. Furthermore, $P(W = 0) = 1/2$. We showed in Section 6.3 the following distribution for α:

$$\begin{aligned} &\alpha | H_0 \in Bin(S, 1/2), \\ &\alpha | H_1, W = 0 \in Bin(S, p_0), \\ &\alpha | H_1, W = 1 \in Bin(S, (1 - p_0)). \end{aligned} \tag{6.17}$$

The estimate of $(u_1, \ldots, u_B)$ is taken as the $(\hat{u}_1, \ldots, \hat{u}_B)$ for which $P(H_1|\alpha)$ is maximal. However, it is not possible to calculate $P(H_1|\alpha)$ directly. Instead, we can equivalently choose the estimate as the $(\hat{u}_1, \ldots, \hat{u}_B)$ for which the likelihood ratio

$$\Lambda = \frac{P(H_1|\alpha)}{1 - P(H_1|\alpha)} = \frac{P(H_1|\alpha)}{P(H_0|\alpha)} = \frac{P(\alpha|H_1)P(H_1)}{P(\alpha|H_0)P(H_0)},$$

is maximal. Using (6.17) we get

$$\Lambda = \frac{\frac{1}{2}(\binom{S^{(k)}}{\alpha} p_0^{\alpha}(1 - p_0)^{S^{(k)}-\alpha} + \binom{S^{(k)}}{\alpha} p_0^{S^{(k)}-\alpha}(1 - p_0)^{\alpha})}{\binom{S^{(k)}}{\alpha}(\frac{1}{2})^{S^{(k)}}} \cdot \frac{P(H_1)}{P(H_0)}.$$

The expression above can also be written as [9]

$$\Lambda = (4p_0(1 - p_0))^{S^{(k)}/2} \cosh(\ln(\frac{p_0}{1 - p_0})(\frac{S^{(k)}}{2} - \alpha)) \cdot \frac{P(H_1)}{P(H_0)}.$$

Since $P(H_0)$, $P(H_1)$, and p_0 do not depend on α, it follows that maximizing Λ is equivalent to taking the candidate for which

$$\cosh(\ln(\frac{p_0}{1-p_0})(\frac{S^{(k)}}{2}-\alpha))$$

is maximum.

In our case it is more convenient to use the loglikelihood ratio $\lambda = \ln(\Lambda)$, and then, it follows that maximizing λ is equivalent to taking the candidate for which

$$\ln\cosh(\ln(\frac{p_0}{1-p_0})(\frac{S^{(k)}}{2}-\alpha)) \tag{6.18}$$

is maximum. If we would like to approximate (6.18) we can use the following

$$\ln\cosh x \approx \begin{cases} |x| - \ln 2 & |x| > 1, \\ \frac{x^2}{2} & |x| < 1. \end{cases}$$

Hence, if

$$\ln(\frac{p_0}{1-p_0})(\frac{S^{(k)}}{2}-\alpha)$$

is small, the chosen distance is close to being statistically optimal. In an implementation of the algorithm it might be possible to use the value of (6.18) directly. One of the main reason to use the quadratic distance, is that the theoretical analysis will be simplified.

This derivation holds when we have one value of $\mathbf{s}$. Now we assume that we have n different $\mathbf{s}^{(k)}$ values $(\mathbf{s}^{(1)}, \mathbf{s}^{(2)}, \ldots, \mathbf{s}^{(n)})$ and corresponding $\alpha = (\alpha^{(1)}, \alpha^{(2)}, \ldots, \alpha^{(n)})$. Furthermore, assume that the we have the same value of p_0 for all n different $\mathbf{s}^{(k)}$s and that $E[S^{(1)}] = E[S^{(2)}] = \cdots = E[S^{(n)}] = S$. By observing that

$$P(\alpha|H_0) = P(\alpha^{(1)}|H_0)P(\alpha^{(n)}|H_0)\cdots P(\alpha^{(2)}|H_0)$$

we see that the distance β, where

$$\beta = \sum_{k=1}^{n}(2\alpha - S^{(k)})^2,$$

is close to being statistically optimal.

Finally in this section, we give a sketch of an analysis of the performance of the algorithm when we use the quadratic distance measure. Let $\gamma = 2\alpha - S^{(k)}$. From (6.17) we get

$$\begin{aligned} E[\gamma|H_0] = 0, &\quad \mathrm{Var}(\gamma|H_0) = S^{(k)}, \\ E[\gamma|H_1, W=0] = (2p_0 - 1)S^{(k)}, &\quad \mathrm{Var}(\gamma|H_1, W=0) = 4S^{(k)}p_0(1-p_0), \\ E[\gamma|H_1, W=1] = (1 - 2p_0)S^{(k)}, &\quad \mathrm{Var}(\gamma|H_1, W=1) = 4S^{(k)}p_0(1-p_0). \end{aligned} \tag{6.19}$$

If we approximate the binomial distribution for α with a normal distribution, we can also approximate the distribution for γ with a normal distribution. Since β is given as $\beta = \sum_{k=1}^{n}(\gamma^{(k)})^2$, we get that β is approximately chi-square distributed [75]. Under the hypothesis H_0, the probability distribution for β can be approximated with a central chi-square distribution. Similarly, under the hypothesis H_1, the probability distribution for β can be approximated with a noncentral chi-square distribution.

6.6 A Sequential Algorithm

In this section we want to elaborate around the idea of using b-prefixes from Section 6.2 and modify the proposed algorithm into a sequential algorithm.

Instead of simply selecting the candidate $(\hat{u}_1, \ldots, \hat{u}_B)$ having the highest value of β, we would now want to have a set of surviving candidates. These are then extended by incrementing, in our case, B by one. This extension doubles the number of candidates, since each surviving candidate can be extended in the $(B+1)$th position by either 0 or 1. But before the next extension, we run a screening procedure that removes a substantial part of the candidates.

This is a straightforward usage of the idea of b-prefixes. From our perspective, it does introduce some small practical problems. The major problem is that we now must store a large set of possible candidates. The performance of our algorithms is highly connected with the computational complexity. If a large memory must be used in our algorithm, some degradation in complexity is likely to appear in practice. This is the reason for presenting a slightly different approach. Essentially we use, instead of an l round algorithm, a tree structure for all candidates that are still "alive". The advantage is that, essentially, the memory requirements are removed. Figure 6.3 shows how a version of such an algorithm may look like.

Note that the set Ω is only introduced to simplify the presentation. We do not need to store it. It is a lexicographically ordered set, and when we put new values in Ω, we actually do not need to store anything.

Whether a candidate will survive the test at level b or not is determined by a threshold value ($threshold(b)$). Increasing the threshold value will throw away more wrong candidates, but will also increase the probability of throwing away the correct candidate.

Comparing with the algorithm of the previous section, this algorithm will have a better performance (fewer tests on average) if we implement it in an efficient way. One important observation in this direction is the fact that all $\hat{\mathbf{x}}^{(k)}$ vectors for a certain k will appear again as valid $\hat{\mathbf{x}}^{(k)}$ vectors for higher b (assuming that we use the same $\mathbf{s}^{(k)}$ vectors). This means that we should not recalculate α on $\hat{\mathbf{x}}^{(k)}$ vectors that have already been used, but rather, we

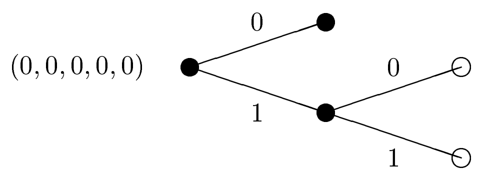

Figure 6.2: The tree in Example 6.1.

store the value of α for all $\mathbf{s}^{(k)}$ values and incorporate this in the calculation for higher b values.

Example 6.1: Assume that the sequential algorithm is applied with $B = 5$. We examine first the value $(u_1, \ldots, u_5) = (0,0,0,0,0)$. Assume that the received β is higher than $threshold(5)$. We then extend this vector with the two possible values for u_6, giving $(0,0,0,0,0,0)$ and $(0,0,0,0,0,1)$. We continue to examine the first of these candidates. Assume that $\beta < threshold(6)$. We continue with the second of these candidates. Assume that in this case $\beta > threshold(6)$. We extend this vector and get $(0,0,0,0,0,1,0)$ and $(0,0,0,0,0,1,1)$ as two new vectors. We continue in this fashion. The tree structure of this procedure is presented in Figure 6.2. □

In: $\mathbf{z} = (z_1, \ldots, z_N)$, $[\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N]$, and constants t, B, n and $threshold(b)$. Let Ω be a list of all B-tuples in lexicographical order.

1. (Precomputation) For each value of b, $B \leqslant b \leqslant l$, set up a screening procedure as given in 2.

2. (Precomputation) Select n different $(l-b)$-tuples $\mathbf{s}^{(1)}, \ldots, \mathbf{s}^{(n)}$. For each $\mathbf{s}^{(k)}$, find all linear combinations of the form $\hat{\mathbf{x}}^{(k)} = \sum_{j=1}^{t} \mathbf{x}_{i_j}$ which are of the special form
$$\hat{\mathbf{x}}^{(k)} = (\hat{x}_1, \ldots, \hat{x}_b, \mathbf{s}^{(k)}),$$
for arbitrary values of $\hat{x}_1, \ldots, \hat{x}_b$. Store $(\hat{\mathbf{x}}^{(k)}, \hat{z}^{(k)} = \sum_{j=1}^{t} z_{i_j})$. Assume that $S^{(k)}$ such pairs have been stored.

3. Take the first value in Ω, denoted $(\hat{u}_1, \ldots, \hat{u}_b)$.

4. For each $\mathbf{s}^{(k)}$, run through all $S^{(k)}$ stored pairs for $\mathbf{s}_i$, calculate the number of times we have
$$\sum_{j=1}^{b} \hat{u}_j \hat{x}_j = \hat{z}^{(k)},$$
and denote this by α. Update
$$\beta \leftarrow \beta + (S^{(k)} - 2 \cdot \alpha)^2.$$

5. If $\beta > threshold(b)$, put both $(\hat{u}_1, \ldots, \hat{u}_b, 0)$ and $(\hat{u}_1, \ldots, \hat{u}_b, 1)$ in Ω. Set $\alpha \leftarrow 0$. If $|\Omega| > 1$ go to 3.

Out: Output all values in Ω that has reached length l.

Figure 6.3: A description of the sequential algorithm.

6.7 Summary

In this chapter we have shown how learning theory can be used as a basis for correlation attacks on stream ciphers. Techniques for reconstructing polynomials have been modified and combined with some general techniques from correlation attacks. The performance has been demonstrated through a sketch of a theoretical analysis as well as through simulations. The simulations show a very good performance.

The problem that arises in a standard correlation attack is equivalent to the problem of learning parity with noise, a well known problem in computational learning theory, commonly believed to be a hard problem. This might indicate that it is hard to find further significant improvements on the problem. One interesting idea would be to examine whether recent results on polynomial reconstruction as a decoding tool for certain error correcting codes [91] can be used. Some results in this direction can be found in [39].

7 A Comparison with Other Proposed Algorithms

After the algorithm based on convolutional codes (Chapter 4) was presented at Eurocrypt'99, a lot of research on fast correlation attacks have taken place. Many other algorithms for fast correlation attacks have been proposed in the last few years. In this chapter we will present four of them and compare their performance with the other algorithms presented in this thesis.

The algorithms for fast correlation attacks to be presented in this chapter all use the standard model as described in Chapter 3. This model is briefly discussed in Section 7.1. In Section 7.2 we will present an algorithm by Canteaut and Trabbia. The algorithm was originally proposed at Eurocrypt 2000 [11]. At FSE 2000 [13] Chebyzhov, Johansson, and Smeets proposed a simple and efficient algorithm for fast correlation attacks. This algorithm is presented in Section 7.3. Mihaljević, Fossorier, and Imai have developed a number of algorithms for fast correlation attacks in the last few years. In Section 7.4 we present their most recent proposal, originally proposed at FSE 2001 [70]. The final algorithm to be presented in this chapter was proposed by Chose, Joux, and Mittel at Eurocrypt 2002 [16]. This algorithm is a modification of the algorithm by Mihaljević, Fossorier, and Imai [70]. A description of this algorithm is given in Section 7.5. In Section 7.6 the performance of different algorithms for fast correlation attacks is compared and discussed.

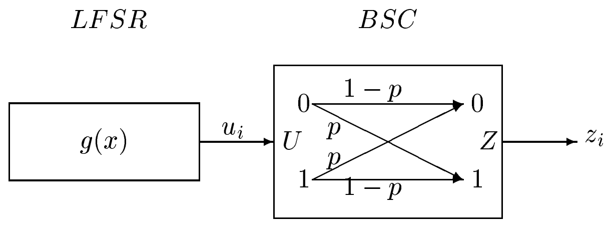

Figure 7.1: Model of a correlation attack

7.1 Model for the Correlation Attacks

The algorithms presented in this chapter all use the standard model for fast correlation attacks. Consider a keystream generator that contains one or several binary LFSRs. Assume that there is a correlation between the keystream and the output of one of the LFSRs, called the target LFSR, in the keystream generator. Recall from Chapter 3 the following notation. Let $u_1, u_2, \ldots$ be the output of the target LFSR, and let $z_1, z_2, \ldots$ be the generated keystream. Furthermore, assume that the correlation is of the form

$$P(z_i = u_i) = 1 - p = \frac{1}{2} - \epsilon, \quad \forall i.$$

Let the length of the target LFSR be l, and let the feedback polynomial be $g(x)$. For the attacker the keystream generator can be modelled as in Figure 7.1.

In the attack, assume that the opponent has observed a keystream sequence of length N, denoted by $\mathbf{z} = z_1, z_2, \ldots, z_N$. The task of the attacker is then to recover the initial state of the target LFSR, given the observed sequence.

7.2 The Algorithm by Canteaut and Trabbia

At Eurocrypt 2000 Canteaut and Trabbia presented an algorithm for fast correlation attacks [11] based on low density parity check codes. The algorithm finds parity check equations of weight 4 and 5, and decodes with a decoding algorithm for low density parity check codes. In the precomputation the following steps are performed. First, calculate all residues $q_i(x) = x^i \bmod g(x)$, for $1 \leqslant i < N$, and store in a table T. Then, compute all possible polynomials of the form $1 + q_{i_1}(x) + \cdots + q_{i_{t-1}}(x)$, for $1 \leqslant i_1 < \cdots < i_{t-1} < N$. If we can find a j such that $q_j(x) = 1 + q_{i_1}(x) + \cdots + q_{i_{t-1}}(x)$ then the polynomial

$$p(x) = 1 + x^{i_1} + x^{i_{t-1}} + x^j$$

is a multiple of $g(x)$ with weight $t+1$, and we add $p(x)$ to the set of parity check equations. From [11] we get that the number of parity check equations of weight $t+1$, denoted $m(t+1)$, found by the algorithm above is approximately

$$m(t+1) \approx \frac{N^t}{t!2^L}. \tag{7.1}$$

After observing a keystream of length N, an estimate of the LFSR initial state is computed by Gallager's iterative decoding algorithm for low density parity check codes [29].

There exist several decoding algorithms that can be used when a set of low weight parity check equations has been found. Comparison of the performance of different decoding algorithms is given by Fossorier, Mihaljević, and Imai in [28], where also some theoretical results are presented. A theoretical discussion of the performance of iterative decoding algorithms suitable for fast correlation attacks is also given in [34, 72].

7.3 The algorithm by Chebyzhov, Johansson, and Smeets

At Fast Software Encryption 2000 Chebyzhov, Johansson, and Smeets proposed a simple algorithm for fast correlation attacks [13]. In Section 3.2 it was shown that correlation attacks can be modelled as decoding of a random (N, l) linear code. However, decoding of random linear codes is considered to be a hard problem. The algorithm proposed in [13] finds an $[m, B]$ linear code associated with the target LFSR, where $1 < B < l$. By ML-decoding of this new code the first B symbols of the initial state of the target LFSR are recovered. When the first B symbols have been recovered it is an easy task to recover the remaining bits.

In the precomputation phase of the algorithm, a method for finding all parity check equations of the form

$$\sum_{j=1}^{B} a_j u_j = u_{i_1} + u_{i_2} + \cdots + u_{i_t}, \tag{7.2}$$

is applied. The method suggested in [13] was based on the methods for finding a low rate convolutional code in Chapter 4. However, one can also use the improved method for finding parity checks proposed in [16]. Denote the total number of parity check equations of the form (7.2), found in the precomputation, by m. Hence, we have the following set of parity check equations

$$\begin{aligned}\textstyle\sum_{j=1}^{B} a_j^{(1)} u_j &= u_{i_1^{(1)}} + u_{i_2^{(1)}} + \cdots + u_{i_t^{(1)}},\\ \textstyle\sum_{j=1}^{B} a_j^{(2)} u_j &= u_{i_1^{(2)}} + u_{i_2^{(2)}} + \cdots + u_{i_t^{(2)}},\\ &\cdots\\ \textstyle\sum_{j=1}^{B} a_j^{(m)} u_j &= u_{i_1^{(m)}} + u_{i_2^{(m)}} + \cdots + u_{i_t^{(m)}}.\end{aligned} \tag{7.3}$$

From the set of parity check equations above we can define a matrix G_2 as

$$G_2 = \begin{pmatrix} a_1^{(1)} & a_1^{(2)} & \dots & a_1^{(m)} \\ a_2^{(1)} & a_2^{(2)} & \dots & a_2^{(m)} \\ \dots & \dots & \dots & \dots \\ a_B^{(1)} & a_B^{(2)} & \dots & a_B^{(m)} \end{pmatrix}.$$

The matrix G_2 can then be considered to be a generator matrix of an $[m, B]$ linear code $\mathcal{C}_2$, and the first B symbols of the initial state $(u_1, u_2 \dots, u_B)$ are the information symbols. Hence, the codewords of $\mathcal{C}_2$ can be written as $(U_1, U_2, \dots, U_m) = (u_1, u_2 \dots, u_B) \cdot G_2$, i.e., $U_j = u_1 a_1^{(j)} + u_2 a_2^{(j)} + \cdots + u_B a_B^{(j)}$. From the right hand side of (7.3) we can calculate a received word $(Z_1, Z_2, \dots, Z_m)$, where $Z_j = z_{i_1^{(j)}} + z_{i_2^{(j)}} + \cdots + z_{i_t^{(j)}}, \quad j = 1, \dots, m$.

In the decoding phase a simple ML-decoding algorithm is used. Run through all 2^B possible values of $(u_1, u_2, \dots, u_B)$. For each $(u_1, u_2, \dots, u_B)$, calculate $(u_1, u_2, \dots, u_B) \cdot G_2$, and calculate the Hamming distance to the received word $(Z_1, Z_2, \dots, Z_m)$. Output the information symbols for the codeword that is closest to the received vector.

One of the interesting properties of the algorithm above is that the performance can be determined theoretically. By using random coding arguments the following theorem was proved in [13].

Theorem 7.1: With given B, l, ϵ, t, the required length N of the observed sequence $z_1, z_2, \dots, z_N$ for the algorithm to succeed is

$$N \approx 1/4 \cdot (2Bt! \ln 2)^{1/t} \cdot \epsilon^{-2} \cdot 2^{\frac{l-B}{t}},$$

assuming $N \gg m$.

7.4 The algorithm by Mihaljević, Fossorier, and Imai

Recently, several algorithms for fast correlation attacks have been proposed by Mihaljević, Fossorier, and Imai [28, 67–70]. In this section we will present their most recent proposal [70]. The algorithm combines exhaustive search over the first B bits with a list decoding algorithm to recover the initial state of the target LFSR.

In the algorithm an algorithm parameter D is introduced, where $D > l$. In the precomputation phase of the algorithm a set of parity check equations for each $k = l+1, l+2, \ldots, D$ is found. Let Ω_k be the set of parity check equations for the kth symbol. The set Ω_k contains all parity check equations of the form

$$\sum_{j=1}^{B} a_j u_j + u_k + u_{i_1} + u_{i_2} + \cdots + u_{i_t} = 0. \tag{7.4}$$

In [70] only the case $t = 2$ was considered.

In the decoding phase of the algorithm, the following steps are performed. Run through all 2^B possible values of the first B symbols of the initial state. For each possible $(u_1, u_2, \ldots, u_B)$ and for each $k = l+1, l+2, \ldots, D$ evaluate the parity check equations, i.e., calculate the number of times that

$$\sum_{j=1}^{B} a_j u_j + z_k + z_{i_1} + z_{i_2} + \cdots + z_{i_t} = 0,$$

for each parity check equation in Ω_k. For each value of $(u_1, u_2, \ldots, u_B)$ two candidate codewords are created. The first candidate is given by the $l - B$ positions with the largest number of satisfied parity check equations. The second candidate is calculated from the $l - B$ positions with the largest number of unsatisfied parity check equations. In this second candidate the bit values are inverted. The candidates are then used in a final correlation check to find the correct initial state.

Some theoretical discussions regarding the performance and complexity were also made in [70]. From these theoretical results, the complexity of a fast correlation attack on the stream cipher LILI-128 was calculated.

7.5 The Algorithm by Chose, Joux, and Mittel

At Eurocrypt 2002, Chose, Joux, and Mittel proposed an improvement of the results in [70]. The algorithm they propose is a small modification of the algorithm by Mihaljević, Fossorier, and Imai. The big improvements reported in [16] do not come from the method used for fast correlation attacks, but rather from some new methods for efficient implementations. These new methods can be applied also to several other proposals for fast correlation attacks.

In the precomputation phase of the attack, Chose, Joux, and Mittel propose a new method for finding parity check equations. The method proposed in [16] is based on a match-and-sort algorithm [79]. Assume that we want to find parity check equations of the form

$$A(\mathbf{u}_0) = u_{i_1} + u_{i_2} + \ldots + u_{i_t} + \sum_{j=1}^{B} c_j u_j, \tag{7.5}$$

where $\mathbf{u}_0$ is the initial state of the target LFSR, $A(\mathbf{u}_0) = \sum_{j=1}^{l} a_j u_j$, and the a_js are some fixed constants. The match-and-sort algorithm proposed in [16] finds parity check equations of the form (7.5) with computational complexity $O(N^{\lceil t/2 \rceil} \log N)$ and with memory complexity $O^{\lfloor (t+1)/4 \rfloor}$.

The new algorithm for finding parity check equations was applied to the problem of finding parity check equations of the form (7.4) for $k = l + 1, \ldots, D$. The decoding step of the algorithm in [16] is a modification of the decoding in [70]. Instead of taking two candidates based on the $l - B$ positions with the largest and smallest number of satisfied parity check equations, the proposed algorithm creates candidates from the $l - B + \delta$ positions having the largest difference between the number of satisfied and unsatisfied parity check equations. The parameter δ is taken as a small integer.

When evaluating the parity check equations for different values of the B bits, it was shown in [16] that it is possible to use the Walsh transform, see Section 2.3. When the improvement is applicable, the computational complexity is reduced from $O(D2^B|\Omega|)$ to $O(D2^B \log|\Omega|)$, where $|\Omega|$ is the average number of parity check equations for each $k = l + 1, \ldots, D$.

7.6 Comparisons Between Different Algorithms

In this section the performance of the algorithms in Chapter 4 and Chapter 6 will be compared with the performance of the algorithms for fast correlation attacks described in this chapter. The comparison is mostly done by simulation results, but also some theoretical discussions are made.

In the simulations, a LFSR with length $l = 40$ is used. The same LFSR is used as the target LFSR in many other simulations of fast correlation attacks, e.g., [11,16,43–45,70]. The programs were all written in the C programming language.

To make the comparisons as fair as possible, we compare the performance of different algorithms for the same value of the parameter t. The parameter t determines the complexity of the precomputation, and hence, when comparing algorithms using the same value of t, the complexity of the precomputation is approximately the same. It is also important to note that for some of the algorithms, e.g., the algorithm based on convolutional codes and the algorithm in [13], the precomputation is done only for one position, while for other algorithms the precomputation must be repeated to get all parity check equations. As an example we can compare the complexity of the precomputation of the algorithm based on recovering of linear polynomials (Chapter 6) for different number of $\mathbf{s}$ vectors. Assume that the computational complexity of finding all parity check equations for some algorithm parameters with only one $\mathbf{s}$ vector is C_{pre}. Now, assume that we have the same algorithm parameters, except that we increase the number of $\mathbf{s}$ vectors to n. The computational complexity of the precomputation is then $n \cdot C_{\text{pre}}$.

We start by comparing the results of the iterative algorithm for fast correlation attacks described in Section 7.2 by Canteaut and Trabbia [11] with the results of the algorithm based on convolutional codes in Chapter 4. The parameter t was set to $t = 3$ and the length of the observed sequence varied between 40000 and 400000. From [11] we get that the computational complexity of the algorithm based on iterative decoding is $C_{CT} \approx 5t\frac{N^t}{t!}2^{-l}$. In Table 7.1 we give the maximum p for which the algorithm converges to the correct initial state with a probability close to one, and we also give the computational complexity of the attack. The results in Table 7.1 were compared

N	p	C_{CT}
40000	0.28	$2^{22.9}$
100000	0.36	$2^{27.8}$
400000	0.44	$2^{35.7}$

Table 7.1: Maximum $p = 1/2 - \epsilon$ for the algorithm by Canteaut and Trabbia, $t = 3$, $l = 40$, varying N.

with the results obtained by the attack in Chapter 4. By Theorem 4.4 the value of the parameter B was calculated such that the algorithm succeeds for the same value of p as in Table 7.1. The theoretical values were then verified by simulations. The results from the simulations are given in Table 7.2. The complexity of the algorithm is given as $C_{cc} \approx l\frac{N^t}{t!}2^{2B-l}$. If we compare the

N	B	p	C_{CT}
40000	6	0.28	$2^{20.6}$
100000	6	0.36	$2^{24.6}$
400000	7	0.44	$2^{32.6}$

Table 7.2: Maximum $p = 1/2 - \epsilon$ for the algorithm based on convolutional codes, $t = 3$, $l = 40$, varying N.

results in Table 7.1 with the results in Table 7.2 we see that the algorithm based on convolutional codes has lower complexity than the algorithm based on iterative decoding. As an example consider $N = 100000$ and $p = 0.36$.

From Table 7.1 we get a computational complexity of $C_{CT} \approx 2^{27.8}$, and from Table 7.2 we get a computational complexity of $C_{CC} \approx 2^{24.6}$. Hence, we see that in this particular case the algorithm based on convolutional codes has approximately a factor of 8 lower computational complexity. This was also verified in the simulations where the algorithm using iterative decoding took 30 seconds compared with the algorithm based on convolutional codes that took 0.2 seconds. It is important to note that the two algorithms uses different operations and the implementation differs a lot. This makes it hard to compare simulation times directly. However, the results clearly indicates that the algorithm based on convolutional codes is more efficient when $t = 3$.

A main advantage of the algorithm based on convolutional codes compared with the algorithm based on iterative decoding is not the improved complexity, but rather that by increasing the parameter B we can succeed in cases when the algorithm in [11] will fail. As an example consider $N = 40000$ and $t = 3$. From Table 7.1 we get that the maximum value of the bit error probability for a successful decoding is $p = 0.28$. If we let $B = 11$ in the attack based on convolutional codes in Chapter 4 we can have a bit error probability of $p = 0.40$ and still have a successful decoding.

Many of the recently proposed algorithms have a parameter B that can be varied and sets the computational complexity and the performance of the algorithms. Through simulations we compare the results from Chapter 4 with the results from the algorithm by Chebyzhov, Johansson, and Smeets [13], and the algorithm by Mihaljević, Fossorier, and Imai [70]. The algorithm proposed by Chose, Joux, and Mittel [16] is not considered since the improvement reported in [16] comes from an algorithmic improvement that can be applied also to the other algorithms for fast correlation attacks. The simulations were made for $N = 400000$, $t = 2$, and $N = 40000$, $t = 3$, with varying value of B.

In Table 7.3 the results from the algorithm based on convolutional codes is tabulated. As before the complexity of the algorithm is calculated as $C_{cc} \approx l\frac{N^t}{t!}2^{2B-l}$. In Table 7.4 the results of the algorithm by Chebyzhov, Johansson,

$N = 400000, t = 2$			$N = 40000, t = 3$		
B	p	C_{cc}	B	p	C_{cc}
10	0.30	$2^{21.5}$	10	0.38	$2^{28.6}$
12	0.35	$2^{25.5}$	12	0.41	$2^{32.6}$
14	0.40	$2^{29.5}$	14	0.42	$2^{36.6}$

Table 7.3: Maximum $p = 1/2 - \epsilon$ for the algorithm based on convolutional codes.

and Smeets are tabulated. For this algorithm the computational complexity can be calculated as $C_{CJS} \approx \frac{N^t}{t!} 2^{2B-l}$. The results of the algorithm by

$N = 400000, t = 2$			$N = 40000, t = 3$		
B	p	C_{CJS}	B	p	C_{CJS}
10	0.16	$2^{16.2}$	10	0.32	$2^{23.3}$
12	0.25	$2^{20.2}$	12	0.35	$2^{27.3}$
13	0.30	$2^{22.2}$	13	0.37	$2^{29.3}$
14	0.32	$2^{24.2}$	14	0.38	$2^{31.3}$
15	0.34	$2^{26.2}$	15	0.39	$2^{33.3}$
17	0.38	$2^{30.2}$	17	0.41	$2^{37.3}$

Table 7.4: Maximum $p = 1/2 - \epsilon$ for the algorithm by Chebyzhev, Johansson, and Smeets.

Mihaljević, Fossorier, and Imai [70] is tabulated in Table 7.5. From [70] we get that the computational complexity of the algorithm can be calculated as $C_{MFI} \approx (D-B)\frac{N^t}{t!} 2^{2B-l}$, if we exclude the final correlation check suggested in [70]. When the value of p is close to 0.5, the complexity of the final correlation check is negligible compared with the the cost of the decoding algorithm.

$N = 400000, t = 2, D = 168$			$N = 40000, t = 3, D = 104$		
B	p	C_{MFI}	B	p	C_{MFI}
10	0.35	$2^{23.2}$	10	0.41	$2^{29.3}$
12	0.40	$2^{27.2}$	12	0.43	$2^{33.3}$
14	0.44	$2^{31.2}$	14	0.44	$2^{37.3}$

Table 7.5: Maximum $p = 1/2 - \epsilon$ for the algorithm by Mihaljević, Fossorier, and Imai.

From the tables we see that for a fixed value of B, the algorithm by Mihaljević, Fossorier, and Imai, can accept a larger value of p and still succeed. However, for a fixed value of B the algorithms also have different complexity. Hence, it is more fair to compare the algorithms for a certain value of p and see which of the algorithms that has the lowest computational complexity. As an example, we pick $N = 40000$, $t = 3$, and $p = 0.41$. From Table 7.5 we get a computational complexity of $2^{29.3}$ for the algorithm by Mihaljević, Fossorier, and Imai. If we compare this results with the results in Table 7.3 and Table 7.4 we see that the algorithm in [70] has the best performance.

The results of the algorithms above should also be compared with the results of the algorithm based on reconstruction of linear polynomials in

Chapter 6. For the case $N = 400000$, $t = 2$, and $p = 0.40$ simulations were made with the algorithm in Chapter 6. The result is given in Table 7.6. The complexity of the algorithm is calculated as $C_{lin} \approx n\frac{N^t}{t!}2^{2B-l}$. Hence,

B	n	C_{lin}
12	512	$2^{29.2}$
11	1024	$2^{28.2}$
10	2048	$2^{27.2}$

Table 7.6: Complexity for the algorithm based on recovering of linear polynomials, for $N = 400000$, $t = 2$, and $p = 0.40$.

we see that for $B = 10$ and $n = 1024$ the algorithm based on recovering of linear polynomials has roughly the same performance as the algorithm by Mihaljević, Fossorier, and Imai.

We can also note that the improvement suggested by Chose, Joux, and Mittel in [16] is also applicable to the algorithm in Chapter 6. Hence, we conclude that the performance of algorithm based on recovering of linear polynomials are among the fastest algorithms for fast correlation attacks.

7.7 Summary

In this chapter, some recent algorithms for fast correlation attacks have been shortly presented, and their performance has been compared. The comparison was made from simulation results on a LFSR with length 40. From the simulations we conclude that the algorithm based on recovery of linear polynomials in Chapter 6 together with the algorithm in [70] is the fastest known algorithm, if the improvements of [16] are incorporated, when applicable. This is the same conclusion as given in [70].

We have not considered the memory complexity of the algorithms. However, all the algorithms, except the algorithms based on convolutional codes and Turbo codes, have a memory complexity that is proportional to the number of parity check equations that are used in the decoding algorithm.

8 Correlation Attack on LILI-128

In this thesis several algorithms for fast correlation attacks have been presented. The algorithms were given in a general model were we have a correlation of the form $P(z_i = u_i) = 1 - p$. In this chapter we will show how fast correlation attacks can be applied to nonlinear filter generators. When attacking nonlinear filter generators, the nonlinearity of the Boolean function will determine the correlation. The correlation attack on nonlinear filters will be exemplified by an attack on the LILI-128 keystream generator.

The LILI-128 keystream generator is a simple and fast generator that uses two binary LFSRs and two functions to generate a binary pseudorandom keystream sequence. LILI-128 is a specific cipher taken from the LILI family of keystream generators, first introduced in [85]. LILI-128 was submitted as a synchronous stream cipher candidate to the NESSIE project [23]. The NESSIE project is a project within the Information Societies Technology (IST) Programme of the European Commission, running years 2000-2002. The main objective of the project is to put forward a portfolio of strong cryptographic primitives that has been obtained after an open call and been evaluated using a transparent and open process.

In the submission [23], the authors conjecture that the complexity of any attack on LILI-128 is at least 2^{112} operations. We quote from [23], “this is a conservative estimate, and the true level of security may be much higher”.

The remaining part of this chapter is organized as follows. We start by giving a description of LILI-128 in Section 8.1. Correlation attacks on nonlinear filter generators are described in Section 8.2. In Section 8.3, we demonstrate that it is possible to mount a fast correlation attack on LILI-128

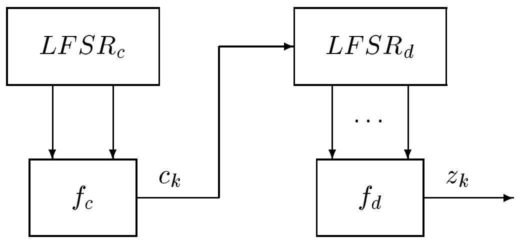

Figure 8.1: The LILI-128 keystream generator.

with complexity around 2^{71} operations. This assumes a received sequence of length around 128MByte and a precomputation phase with complexity 2^{79} table lookups.

8.1 Description of LILI-128

In this section we give a brief description of the LILI-128 keystream generator. For a full description, see [23]. The LILI-128 keystream generator is a clock-controlled nonlinear filter generator. The generator consists of two linear feedback shift registers, denoted by $LFSR_c$ and $LFSR_d$, respectively. The total length of the LFSRs is 128, and at initialization, the 128 bit key provides the initial states of the LFSRs. The generator can be divided into two subsystems. The first system produces an integer sequence that is used to control the clocking of the second subsystem, which in turn produces the keystream. The structure of LILI-128 is illustrated in Figure 8.1.

$LFSR_c$ is a LFSR of length 39 with the feedback polynomial

$$g_c(x) = x^{39} + x^{35} + x^{33} + x^{31} + x^{17} + x^{15} + x^{14} + x^2 + 1.$$

Since $g_c(x)$ is primitive, the sequence produced by $LFSR_c$ is a maximum-length sequence. The contents of stages 12 and 20 of $LFSR_c$ are inputs to the function f_c. The function f_c takes two bits as input and produces an integer c_k, such that $c_k \in \{1, 2, 3, 4\}$. The value of c_k is calculated as

$$c_k = f_c(y_1, y_2) = 2y_1 + y_2 + 1, k \geq 1.$$

The integer sequence $\mathbf{c} = c_1, c_2, \dots$ controls the clocking of $LFSR_d$, in the sense that $LFSR_d$ is clocked c_k times before producing the output value z_k. Thus, from the definition of $\mathbf{c}$, $LFSR_d$ is clocked at least once and at most four times between the output of consecutive keystream bits.

The length of $LFSR_d$ is 89 and the feedback polynomial of $LFSR_d$ is the primitive polynomial

$$g_d(x) = x^{89} + x^{83} + x^{80} + x^{55} + x^{53} + x^{42} + x^{39} + x + 1.$$

The contents of 10 different stages of $LFSR_d$ are inputs to a Boolean function f_d, $f_d : \mathbb{F}_2^{10} \to \mathbb{F}_2$. An exact description of f_d in table form is given in [23]. The output of f_d is the keystream sequence $\mathbf{z} = z_1, z_2, \ldots$.

Finally, the stream cipher description is completed by adding the generator output $\mathbf{z} = z_1, z_2, \ldots$ bitwise to the plaintext, obtaining the ciphertext.

8.2 Correlation Attacks on Nonlinear Filter Generators

Consider a nonlinear filter generator that consists of a LFSR with length l, and a nonlinear filter function $f : \mathbb{F}_2^n \to \mathbb{F}$, see Figure 8.2.

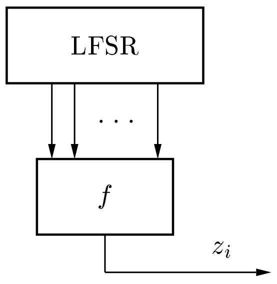

Figure 8.2: Principle of nonlinear filter generators.

When applying a fast correlation attack to a nonlinear filter generator we start by determine the nonlinearity of the Boolean function f. Recall from Section 2.3, that for a Boolean function, $f : \mathbb{F}_2^n \to \mathbb{F}$, the *Walsh transform* of $f(x)$ is defined to be the real-valued function $F(\omega)$ over the vector space $\mathbb{F}_2^n$ given by

$$F(\omega) = \sum_x (-1)^{f(x) \oplus \omega \cdot x},$$

where a dot product of vectors x and ω is defined as $x \cdot \omega = x_1\omega_1 + \ldots + x_n\omega_n$. Furthermore, the *nonlinearity* of a Boolean function $f(x)$, denoted by N_f, is the Hamming distance to the nearest affine function, i.e.,

$$N_f = \min_{g \in \mathcal{A}_n} d_H(f, g),$$

where f and g are the truth tables of $f(x)$ and $g(x)$, $\mathcal{A}_n$ is the set of affine functions on n variables, and $d_H(f,g)$ is the Hamming distance between the two vectors f and g, i.e., the number of positions where f and g differ. The nonlinearity of $f(x)$ can be obtained from the Walsh transform as

$$N_f = 2^{n-1} - \frac{1}{2}\max_{\omega} |F(\omega)|.$$

Let f_l be a linear function such that $d_H(f, f_l) = N_f$. Thus, if we use this function to approximate f we get

$$P(f(x) = f_l(x)) = 1 - \frac{N_f}{2^l}. \tag{8.1}$$

We call the probability in (8.1) the identified correlation.

In general, there exists several affine functions g such that $d_H(f,g) = N_f$. Denote the number of such affine equations by w_f. Hence, we have w_f different affine functions $f_{l_1}, f_{l_2}, \ldots, f_{l_{w_f}}$ such that

$$P(f(x) = f_{l_i}(x)) = 1 - \frac{N_f}{2^l}, \quad 1 \leqslant i \leqslant w_f. \tag{8.2}$$

When we replace f with f_l, we can write the output from f_l as a linear combination of the initial state $\mathbf{u}_0$. Thus, we can find an $l \times N$ matrix G, such that the output sequence $\mathbf{v}$ from f_l can be written as $\mathbf{v} = \mathbf{u}_0 G$. Furthermore, by using all w_f affine functions $\{f_{l_1}, f_{l_2}, \ldots, f_{l_{w_f}}\}$ with the same correlation, we can get w_f different $l \times N$ matrices, denoted $G_1, G_2, \ldots, G_{w_f}$. By concatenation of these matrices we get the following $l \times w_f N$ matrix,

$$G' = \begin{pmatrix} G_1 & G_2 & \cdots & G_{w_f} \end{pmatrix}.$$

The matrix G' can be viewed as a generator matrix of another linear code $\mathcal{C}'$. When we have the matrix G' we can apply one of the algorithms for fast correlation attacks and attack the generator. It is important to note that for the code $\mathcal{C}'$, the cyclic properties used when calculating parity check equations for the algorithm based on convolutional codes in Chapter 4 does not hold.

8.3 Correlation Attack on LILI-128

Now we are ready to proceed with an attack on LILI-128. Assume that we have observed a keystream sequence of length N, $\mathbf{z} = (z_1, z_2, \ldots, z_N)$. Let the output sequence from f_d be denoted by $\mathbf{d} = (d_1, d_2, \ldots, d_M)$, where

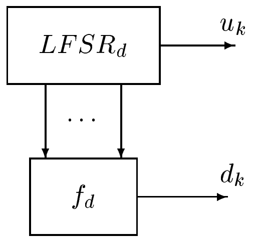

Figure 8.3: The data generation subsystem.

$M > N$, when $LFSR_d$ is *regularly* clocked. This is illustrated in Figure 8.3. Furthermore, define an integer sequence $\mathbf{s} = (s_1, s_2, \ldots, s_N)$, $s_k \in \mathbb{Z}$, where

$$s_k = \sum_{i=1}^{k} c_i, \quad k = 1, \ldots, N,$$

and $c_i \in \{1, 2, 3, 4\}$. Using the sequences $\mathbf{d}$ and $\mathbf{s}$ we can write the observed sequence, $\mathbf{z}$, as

$$z_k = d_{s_k}, \quad k = 1, \ldots, N.$$

The principle of the attack is the following. Guess an initial state of $LFSR_c$, and calculate the decimating sequence $\mathbf{c}$ and the corresponding sequence $\mathbf{s}$, where $s_k = \sum_{i=1}^{k} c_i$, for $k = 1, \ldots, N$. If the guessed initial state of $LFSR_c$ is correct, the sequence $\mathbf{s}$ together with $\mathbf{z}$ will give us the N symbols $d_{s_1}, d_{s_2}, \ldots, d_{s_N}$, since then $z_1 = d_{s_1}$, $z_2 = d_{s_2}$, etc.

Next, we use these N obtained symbols of $\mathbf{d}$ to find the initial state of $LFSR_d$. Denote the sequence produced by $LFSR_d$ (under regular clocking) by $\mathbf{u} = (u_1, u_2, \ldots, u_N)$. From [23], the ith output symbol from f_d, d_i, is given as

$$d_i = f_d(u_{i-89}, u_{i-88}, u_{i-86}, u_{i-79}, u_{i-77}, u_{i-69}, u_{i-59}, u_{i-45}, u_{i-24}, u_{i-9}).$$

The next step is to apply a correlation attack and recover the initial state of $LFSR_d$, by using the method in Section 8.2.

The function f_d used in LILI-128 has nonlinearity $N_{f_d} = 480$ [23]. This implies that we can find a linear function $f_l(x_1, x_2, \ldots, x_{10}) = a_1x_1 + a_2x_2 + \ldots + a_{10}x_{10}$ such that $d_H(f_d, f_l) = 480$. Thus, if we use this function to approximate f_d, we get

$$P(f_d(x) = f_l(x)) = \frac{1024 - 480}{1024} = 0.53125. \tag{8.3}$$

We call the probability in (8.3) the identified correlation. To summarize we have the following,

$$P(d_i = a_1u_{i-89} + a_2u_{i-88} + a_3u_{i-86} + a_4u_{i-79} + a_5u_{i-77} + a_6u_{i-69} + a_7u_{i-59} + a_8u_{i-45} + a_9u_{i-24} + a_{10}u_{i-9}) = 0.53125,$$

where the coefficients a_i, $1 \leqslant i \leqslant 10$, can be determined from the exact description of f_d in [23].

The complete Walsh spectrum of the function f_d is the following. There are 720 different $\omega \in \mathbb{F}_2^{10}$ with $F(\omega) = 0$, 64 values with $F(\omega) = \pm 32$, and 240 values with $F(\omega) = \pm 64$. This means that there exist 240 different affine functions $f_{l_1}, f_{l_2}, \ldots, f_{l_{240}}$ such that

$$P(f_d(x) = f_{l_i}(x)) = 0.53125, \quad 1 \leqslant i \leqslant 240.$$

By using all 240 affine functions $\{f_{l_1}, f_{l_2}, \ldots, f_{l_{240}}\}$ having the same correlation, we can get 240 different $89 \times N$ matrices, denoted $G_1, G_2, \ldots, G_{240}$. By concatenation of these matrices we get the following $89 \times 240N$ matrix,

$$G' = \begin{pmatrix} G_1 & G_2 & \cdots & G_{240} \end{pmatrix}.$$

Let $\mathbf{g}_i$ be the ith column of G', i.e.,

$$G' = \begin{pmatrix} \mathbf{g}_1 & \mathbf{g}_2 & \cdots & \mathbf{g}_{240N} \end{pmatrix}.$$

We now follow the algorithm in Section 6.3 with $n = 1$, (which is equivalent with the algorithm proposed in [13]). Let the algorithm parameter t be $t = 3$, and let $k < 89$ be some fixed value. In the precomputation phase of the attack we find all triples of columns, $\mathbf{g}_{i_1}, \mathbf{g}_{i_2}, \mathbf{g}_{i_3}$, such that

$$(\mathbf{g}_{i_1} + \mathbf{g}_{i_2} + \mathbf{g}_{i_3})^T = (\underbrace{*, *, \ldots, *}_{k}, \underbrace{0, 0, \ldots, 0}_{89-k}), \tag{8.4}$$

where $*$ means an arbitrary value (not all zero).

To find all such triples we proceed as follows. Put each column of G' in a list, sorted according to the value of the last $89 - k$ entries. For each pair of columns $\mathbf{g}_{i_1}, \mathbf{g}_{i_2}$, calculate the value of $\mathbf{g}_{i_1} + \mathbf{g}_{i_2}$ in the last $89 - k$ positions. Check if there exists a column $\mathbf{g}_{i_3}$ such that (8.4) is satisfied. This is easily done since the columns are sorted. Thus, the running time of the precomputation step is roughly N^2 table lookups. Let the number of triples satisfying (8.4) be m. Denote the indices of all such triples as $\{i_1(j), i_2(j), i_3(j)\}$, $1 \leqslant j \leqslant m$.

If i_1, i_2, and i_3 satisfy (8.4), the sum $u_{i_1} + u_{i_2} + u_{i_3}$ is a linear combination of only the first k symbols of the initial state of $LFSR_d$. Thus,

$$(u_{i_1^{(1)}} + u_{i_2^{(1)}} + u_{i_3^{(1)}}, u_{i_1^{(2)}} + u_{i_2^{(2)}} + u_{i_3^{(2)}}, \ldots, u_{i_1^{(m)}} + u_{i_2^{(m)}} + u_{i_3^{(m)}})$$

forms an $[m,k]$-code, denoted by $\mathcal{C}_3$.

Using $\mathbf{z}$ we can calculate

$$Z_j = z_{i_1^{(j)}} + z_{i_2^{(j)}} + z_{i_3^{(j)}}, \quad 1 \leqslant j \leqslant m.$$

The vector $(Z_1, Z_2, \ldots, Z_m)$ acts as a received word for $\mathcal{C}_3$. Due to the correlation between z_i and u_i, we can write $z_i = u_i + e_i$, where e_i is a binary random variable with $P(e_i = 1) = p = P(z_i \neq u_i) = 1 - 0.53125$. It is convenient to write the distribution of e_i as $P(e_i = 1) = 1/2 - \epsilon$. Thus, for the attack on LILI-128 in this paper we get $\epsilon = 1/32 = 0.03125$. Define the error vector for $\mathcal{C}_3$ to be $(E_1, E_2, \ldots, E_m)$, where

$$E_j = e_{i_1^{(j)}} + e_{i_2^{(j)}} + e_{i_3^{(j)}}, \quad 1 \leqslant j \leqslant m.$$

Since the e_i's are independent random variables, it is clear that E_j, for $j = 1, 2, \ldots, m$, are also independent binary random variables with error probability

$$p_3 = P(e_{i_1^{(j)}} + e_{i_2^{(j)}} + e_{i_3^{(j)}} = 1) = 3p(1-p) + p^3 = 1/2 - 4\epsilon^3 \approx 0.49988,$$

when $\epsilon = 0.03125$.

Finally, we decode the code $\mathcal{C}_3$ using Maximum Likelihood decoding (ML-decoding). The algorithm returns a codeword from $\mathcal{C}_3$ that is closest to $(Z_1, Z_2, \ldots, Z_m)$. From this codeword, we get k bits of the initial state of $LFSR_d$. The remaining bits of the initial state are determined in a similar way. It is important to note that obtaining the remaining bits has negligible complexity compared with determining the first k bits, see [13].

The method above has been described for a regularly clocked $LFSR_d$. Considering the case of LILI-128, when $LFSR_d$ is clocked using the sequence $\mathbf{c}$, we need to slightly modify the attack. Since $P(c_i = j) = 0.25$ for $j = 1, 2, 3, 4$, $LFSR_d$ is clocked 2.5 times on average, between successive outputs. Thus, if we have observed N keystream symbols, then $LFSR_d$ has been clocked M times, where $M \approx 2.5N$. The sequence $\mathbf{c}$ determines which N symbols from $d_1, d_2, \ldots, d_M$ that is used to produce the keystream $\mathbf{z}$.

The precomputation in the algorithm takes roughly N^2 table lookups. However, in the precomputation phase of the correlation attack on LILI-128 we need to consider all $240M$ possible symbols when we construct the code $\mathcal{C}_3$. Thus, the precomputation uses approximately $(240M)^2$ table lookups. Each possible initial state of $LFSR_c$ selects N positions, and from the set of equations of the form (8.4) we keep only those that are valid, given the guessed initial state of $LFSR_c$.

8.4 Numerical Result

From the analysis in [13] we get the following result. Let m be the average number of parity check equations of $\mathcal{C}_3$ needed for a successful decoding. The value of m can be calculated as

$$m \approx k \cdot \frac{2\ln 2}{(2\epsilon)^{2t}}, \tag{8.5}$$

see [13]. In the attack presented we have $\epsilon = 0.03125$ and $t = 3$. The matrix G' has the size $89 \times 240M$, where $M = 2.5N$. From the output of the regularly clocked $LFSR_d$ we can expect that there are approximately

$$\frac{(2.5 \cdot 240N)^t}{t!} \cdot 2^{-89+k}$$

linear 3-tuples satisfying (8.4). Each of these parity check equations is valid with probability $(\frac{1}{2.5})^t$ for a given decimated sequence. Thus, for each possible decimation sequence we need on average

$$m = \frac{(240N)^t}{t!} \cdot 2^{-89+k}$$

parity checks. Finally, if we let $t = 3$ the algorithm succeeds, see [13], if

$$N \approx \frac{1}{960} \cdot (k \cdot 12 \cdot \ln 2)^{1/3} \cdot \epsilon^{-2} \cdot 2^{\frac{89-k}{3}}. \tag{8.6}$$

The decoding complexity of the attack can be calculated as follows. We run through 2^k codewords with length m. Thus, the decoding complexity is of the order

$$2^k \cdot k \cdot \frac{2\ln 2}{(2\epsilon)^{2t}}.$$

For LILI-128 with $l = 89$, $\epsilon = 0.03125$, and $t = 3$ we obtain the results tabulated in Table 8.1 for finding the initial state of $LFSR_d$.

As an example from Table 8.1, we choose $k = 5$ to get a complexity of approximately 2^{32}. Since this decoding phase has to be repeated for each possible initial state of $LFSR_c$, we get that the average complexity is $2^{32} \cdot 2^{39} = 2^{71}$. This requires an observed keystream sequence of approximately 1 Gbit, or equivalent 128MB. The precomputation for this case is approximately 2^{79} table lookups.

This result can be compared with the claim in [23], where the authors conjecture that the complexity of an attack is at least 2^{112} operations.

Note that the complexity is given as the average number of bit operations in the decoding phase and the average number of table lookups in the precomputation. Thus, for the decoding phase, m bit operations is done significantly

k	N	C_{dec}
1	$2^{30.5}$	$2^{25.5}$
3	$2^{30.3}$	$2^{29.1}$
5	$2^{29.9}$	$2^{31.8}$
7	$2^{29.4}$	$2^{34.3}$
10	$2^{28.6}$	$2^{37.8}$
15	$2^{27.1}$	$2^{43.4}$
20	$2^{25.6}$	$2^{48.8}$
25	$2^{24.0}$	$2^{54.1}$

Table 8.1: The required length and the decoding complexity to recover the initial state of $LFSR_d$ given the initial state of $LFSR_c$.

faster than m clock cycles. On the other hand, in the precomputation phase we search for the existence of a certain vector in a large sorted list of vectors. By using a hash table technique this can be done in almost constant time. The list of vectors will be large, and thus it must be stored on hard disk instead of the primary memory. However, it is important to point out that it is possible to organize the precomputation such that the table lookups take place in primary memory only. Hence, the precomputation time will take $c \cdot 2^{79}$ clock cycles, where c is a small constant.

8.5 Summary

In this chapter we have presented a method for correlation attacks on nonlinear filter generators. The correlation in the generator was determined from the nonlinearity of the filter function.

This method was used to attack LILI-128, one of the NESSIE stream cipher proposals. The attack takes around 2^{71} operations assuming a received sequence of length around 128MByte using a precomputation phase with complexity 2^{79} table lookups.

9 Correlation Attacks over Extension Fields

Binary LFSR based stream ciphers have often been used in cases where a stream cipher that can be efficiently implemented in hardware is needed. However, their software implementation is rather slow. To get a stream cipher that is fast also for software implementations, we can construct a stream cipher with LFSRs over an extension field, $\mathbb{F}_{2^n}$, where n is chosen to match the word size of the intended platform, typically values are $n = 8, 16,$ or 32. A stream cipher construction over $\mathbb{F}_{2^n}$ can also be used in high speed hardware implementations, since the LFSR outputs n bits in each clock cycle.

The algorithms for fast correlation attacks presented in this thesis were defined over the binary alphabet. When attacking a stream cipher containing a LFSR over $\mathbb{F}_{2^n}$, one can also consider a correlation attack over $\mathbb{F}_{2^n}$. In this chapter we describe such an algorithm for correlation attacks over $\mathbb{F}_q$, and give a theoretical analysis of the performance of the algorithm.

The chapter is organized as follows. In Section 9.1 we show how a correlation attack over $\mathbb{F}_q$ can be modelled as a decoding problem. Furthermore, some known results for decoding random codes are given. In Section 9.2 we generalize the algorithm in [13] to stream ciphers over $\mathbb{F}_{2^n}$. A theoretical analysis of the proposed algorithm is given in Section 9.3.

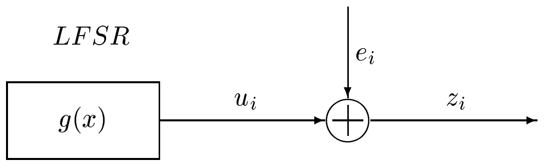

Figure 9.1: Model of the correlation attack.

9.1 Model of the Attack

In Section 3.2 it was shown that correlation attacks over $\mathbb{F}_{\nvDash}$ can be modelled as a decoding problem. In this section we generalize the results from Section 3.2 to correlation attacks over $\mathbb{F}_{||}$. Let $\mathbf{u} = u_1, u_2, \ldots, u_N$, where now $u_i \in \mathbb{F}_q, i = 1, 2, \ldots, N$, be an output sequence from an LFSR in the keystream generator, and let $\mathbf{z} = z_1, z_2, \ldots, z_N$, where $z_i \in \mathbb{F}_q, i = 1, 2, \ldots, N$, be an observed output sequence from a keystream generator in a synchronous stream cipher.

Due to the correlation between u_i and z_i, we can describe each z_i as the output of a discrete memoryless channel, DMC, when u_i was transmitted. Assume that we have a correlation of the form,

$$z_i = u_i + e_i, \tag{9.1}$$

where e_i is non-uniformly distributed. This is all shown in Figure 9.1. For a DMC with the properties above the capacity, denote C, can be calculated as

$$C = \log q + \sum_e P(e_i = e) \cdot \log(P(e_i = e)),$$

where the capacity is reached for $P(u_i = u) = 1/q, \forall u \in \mathbb{F}_q$. It can be shown that we can construct an additive channel as above from any DMC.

Assume that the feedback polynomial, $g(x)$, is known and that the degree of $g(x)$ is l. When consider LFSRs over $\mathbb{F}_q$ we have q^l possible output sequences. The LFSR output sequence, $\mathbf{u}$, can be seen as a codeword from a linear $[N, l]$ q-ary code [57], refereed to as $\mathcal{C}$. The cryptanalyst's problem can be formulated as follows. Given a length N received word $(z_1, z_2, \ldots z_N)$ as output of the DMC, find the length N codeword from $\mathcal{C}$ that was transmitted.

When decoding random linear codes, a central method is Maximum Likelihood decoding, ML-decoding, i.e.,

$$\hat{\mathbf{u}} = \arg\max_{\mathbf{u} \in \mathcal{C}} P(\mathbf{z}|\mathbf{u}).$$

Since the correlation attack succeeds if we can decode correctly, it is interesting to find the probability of error using ML-decoding. Let P_e be the error probability of ML-decoding. Using random coding argument the following theorem was proved by Gallager, Berlekamp, and Shannon.

Theorem 9.1 [[30]]: Let C denote the capacity of the channel and let the code rate $R = l \log q/N$ satisfy $R < C$. Then

$$E[P_e] \leqslant 2^{-N\tau(R)},$$

where $E[P_e]$ is the mathematical expectation of the error probability of ML-decoding in the ensemble of random linear $[l, N]$-codes, $\tau(R)$ is the random coding exponent, and $\tau(R) > 0$ for all $R < C$. □

Thus, we assume that the correlation attack succeeds with a non-negligible probability if $R < C$, where the code rate $R = l \cdot \log q/N$.

9.2 Algorithm Description

In previous section it was shown that a correlation attack over $\mathbb{F}_q$ can be seen as a decoding problem. However, the complexity of decoding the code $\mathcal{C}$ is too complex to get a practical attack for shift registers of high degree. Thus, the goal is to remodel the correlation attack to find an easier decoding problem. The algorithm proposed in this chapter is a modification of the algorithm in [13]. The difference is that we here consider q-ary codes instead of binary linear codes. The principle of the algorithm is as follows. First, find a new code, $\mathcal{C}_2$, which is easier to decode than $\mathcal{C}$. This step is independent of the keystream and, hence, can be made in advance. In the next step, we decode using a received word obtained from the observed keystream sequence.

To get the new code, $\mathcal{C}_2$, we use the following observation. Let G_{LFSR} be a generator matrix for $\mathcal{C}$. The LFSR output sequence can be calculated as

$$\mathbf{u} = (u_1, u_2, \ldots, u_l)G_{LFSR},$$

where $(u_1, u_2, \ldots, u_l)$ is the initial state of the shift register. Write the matrix G_{LFSR} as

$$G_{LFSR} = (\mathbf{g}_1, \mathbf{g}_2, \ldots, \mathbf{g}_N),$$

i.e., $\mathbf{g}_i$ is the i-th column of G_{LFSR}.

Assume that we can find a pair of columns of G_{LFSR}, $\mathbf{g}_{i_1}$, $\mathbf{g}_{i_2}$, such that

$$a_{i_1}\mathbf{g}_{i_1} + a_{i_2}\mathbf{g}_{i_2} = c_1u_1 + c_2u_2 + \ldots + c_ku_k, \qquad (9.2)$$

where $k < l$ and $a_{i_1}, a_{i_2}, c_1, c_2, \ldots, c_k \in \mathbb{F}_q$. Equation (9.2) defines a parity check equation involving the first k positions of the initial state and two more positions of the LFSR-output sequence. Search for all pairs with this property, and denote the number of such equations by m. Denote the indices of the n-th pair of columns with the properties in Equation (9.2) by $i_1^{(n)}, i_2^{(n)}$. Let $U_n = a_{i_1^{(n)}} u_{i_1^{(n)}} + a_{i_2^{(n)}} u_{i_2^{(n)}}$, $1 \leqslant n \leqslant m$, then the sequence (U_1, U_2, U_m) is a codeword from an $[m, k]$ q-ary linear code, denoted by $\mathcal{C}_2$. The received sequence $(Z_1, Z_2, \ldots, Z_m)$ can be obtained from the keystream as $Z_1 = a_{i_1^{(1)}} z_{i_1^{(1)}} + a_{i_2^{(1)}} z_{i_2^{(1)}}, Z_2 = a_{i_1^{(2)}} z_{i_1^{(2)}} + a_{i_2^{(2)}} z_{i_2^{(2)}}, \ldots, Z_m = a_{i_1^{(m)}} z_{i_1^{(m)}} + a_{i_2^{(m)}} z_{i_2^{(m)}}$. Since, $z_{i_1} = u_{i_1} + e_{i_1}$, and $z_{i_2} = u_{i_2} + e_{i_2}$we get

$$\begin{aligned}
Z_n &= a_{i_1^{(n)}} z_{i_1^{(n)}} + a_{i_2^{(n)}} z_{i_2^{(n)}} \\
&= a_{i_1^{(n)}} (u_{i_1^{(n)}} + e_{i_1^{(n)}}) + a_{i_2^{(n)}} (u_{i_2^{(n)}} + e_{i_2^{(n)}}) \\
&= a_{i_1^{(n)}} u_{i_1^{(n)}} + a_{i_2^{(n)}} u_{i_2^{(n)}} + a_{i_1^{(n)}} e_{i_1^{(n)}} + a_{i_2^{(n)}} e_{i_2^{(n)}} \\
&= U_n + a_{i_1^{(n)}} e_{i_1^{(n)}} + a_{i_2^{(n)}} e_{i_2^{(n)}}.
\end{aligned}$$

Let $E_n = a_{i_1^{(n)}} e_{i_1^{(n)}} + a_{i_2^{(n)}} e_{i_2^{(n)}}$. From our model of the DMC we can calculate the distribution of E_n as,

$$P(E_n = e_n) = \sum_{x \in \mathbb{F}_q} P(e_{i_1^{(n)}} = x) \cdot P(e_{i_2^{(n)}} = a_{i_2^{(n)}}^{-1} (e_n - a_{i_1^{(n)}} x)), \forall e_n \in GF(q).$$

As for the other algorithms presented in this thesis, we do not need to restrict the algorithm to use only pair of columns, any t-tuple can be used. We describe the algorithm for a general $t \leqslant 2$ in Figure 9.2.

In: A length l LFSR over $\mathbb{F}_q$ with feedback polynomial $g(x)$, observed keystream $\mathbf{z} = z_1, z_2, \ldots, z_N$, and parameters k and t.

Precomputation:

1. Find all combinations of the form,

$$\sum_{j=1}^{t} a_{i_j} u_{i_j} = c_1 \cdot u_1 + c_2 \cdot u_2 + \ldots + c_k \cdot u_k.$$

Let the number of such sets be m_t. Store the indices $i_1, i_2, \ldots, i_t$ together with $c_1, c_2, \ldots, c_k$, and $a_{i_1}, a_{i_2}, \ldots, a_{i_t}$.

Decoding phase:

2. Compute the received sequence,

$$(Z_1 = \sum_{j=1}^{t} z_{i_j^{(1)}}, Z_2 = \sum_{j=1}^{t} z_{i_j^{(2)}}, \ldots, Z_{m_t} = \sum_{j=1}^{t} z_{i_j^{(m_t)}})$$

3. Decode using exhaustive search through all q^k codewords.

Out: The first k symbols of the initial state.

Figure 9.2: A description of the proposed algorithm for correlation attacks over extension fields.

9.3 Theoretical Analysis

In the theoretical analysis we consider a channel with the following properties. Assume that the received symbol can be written as $z_i = u_i + e_i$, where the distributions of e_i is

$$\begin{aligned} P(e_i = 0) &= 1/q + \delta, \\ P(e_i = e) &= 1/q - \delta/(q-1), \quad \forall e \in \mathbb{F}_q, e \neq 0, \end{aligned}$$

where $q = 2^n$. Let $E_i = \sum_{j=1}^{t} a_{i_j} e_{i_j}$, the distribution of E_i is then given as,

$$\begin{aligned} P(E_i = 0) &= 1/q + (q/(q-1))^{t-1}\delta^t, \\ P(E_i = e) &= 1/q - (q/(q-1))^{t-1}\delta^t/(q-1), \quad \forall e \in \mathbb{F}_q, e \neq 0. \end{aligned}$$

Thus, we get a new channel of the form

$$\begin{aligned} P(E_i = 0) &= 1/q + \epsilon, \\ P(E_i = e) &= 1/q - \epsilon/(q-1), \quad \forall e \in \mathbb{F}_q, e \neq 0, \end{aligned}$$

where $\epsilon = (q/(q-1))^{t-1}\delta^t$.

We can now extend the line of reasoning in [13] from the binary case to the q-ary case. As the procedure is very similat to [13], we omit some details. For a channel of the form above we can approximate the channel capacity, C, as

$$C \approx \frac{q^2\epsilon^2}{2\ln 2(q-1)}.$$

Denote by $\hat{m}_t$ the expected number of equations found by the precomputation step. The value of $\hat{m}_t$ can be calculated as

$$\hat{m}_t = \frac{(q^{t-1}-1)}{q^{l-k}} \cdot \binom{N}{t}.$$

For N large we can approximate $\hat{m}_t$ as

$$\hat{m}_t \approx \frac{N^t q^{t-1-(l-k)}}{t!}.$$

Since the rate of the code is $R = k \log q/m_t$, and we need $R < C$ we get the following theorem.

Theorem 9.2: With given k, l, δ, t, the required length N of the observed sequence for the algorithm to succeed is

$$N \approx \frac{q-1}{q^2} \cdot (2kt! \ln 2 \log q)^{1/t} q^{\frac{l-k}{t}} \delta^{-2},$$

assuming $N \gg m_t$. □

For $q = 2$ this expression is the same as the expression presented in [13] for the binary case.

Example 9.1: Consider a stream cipher over $\mathbb{F}_{2^2}$ where a correlation between the keystream and the output of a LFSR with degree $l = 30$ has been identified. Assume that the correlation can be written as $z_i = u_i + e_i$ with the following probability distribution of e_i. $P(e_i = 0) = 0.4$, $P(e_i = e) = 0.2$, $e \neq 0$. Let the algorithm parameters be $k = 15$ and $t = 2$. From Theorem 9.2 we get a required length of $N \approx 2.5 \cdot 10^6$.

The code $\mathcal{C}$ can also be viewed as a $[2N, 2l]$ binary code. Each element $x \in \mathbb{F}_{2^2}$ can be written as $x = x^{(1)}\alpha + x^{(0)}$, where $x^{(1)}, x^{(0)} \in \mathbb{F}_2$ and α is a primitive element in $\mathbb{F}_{2^2}$. Thus, l symbols from $\mathbb{F}_{2^2}$ are equivalent with $2l$ binary symbols. Assuming the same setting as above, we can transform the correlation attack over $\mathbb{F}_{2^2}$ into a correlation attack over $\mathbb{F}_2$ with binary shift register length 60. The correlation in the binary case will be $p = 0.6$, i.e., $\delta = 0.1$. To reach the same complexity we choose $k = 30$ in the binary case. With this parameter value we get from Theorem 9.2 a required length of $N \approx 7.5 \cdot 10^6$ binary digits. Thus, we need approximately $3.7 \cdot 10^6$ observed keystream symbols over $\mathbb{F}_{2^2}$ using a binary attack, compared with $2.5 \cdot 10^6$ for an attack over $\mathbb{F}_{2^2}$. $\square$

9.4 Summary

In this chapter we have shortly described a correlation attack over $\mathbb{F}_q$. The attack is a generalization of the attack in [13] to $\mathbb{F}_q$. By using random coding bounds we gave a bound on the performance of the algorithm.

By an example it was shown that in some cases an attack over $\mathbb{F}_{2^n}$ is more effective than an attack over $\mathbb{F}_2$. It is also possible to construct examples, where the opposite is true.

10
The General Decoding Problem

In this chapter we leave fast correlation attacks and study what we call the general decoding problem, with several applications in cryptology. The general decoding problem is the problem of decoding a received word to the closest codeword in an arbitrary code. It was shown by Berlekamp, McEliece, and van Tilborg in [2] that the (decisional) general decoding problem is NP-complete. Another related problem in coding theory is the problem of finding the minimum distance of a general code. Vardy showed in [95] that this problem is NP-complete. Assuming $P \neq NP$, there are no algorithms that solve the two problems in polynomial time.

The difficulty of the general decoding problem and the problem of finding the minimum distance has been explicitly used in many cryptosystems, such as Stern identification scheme, McEliece public-key cryptosystem, etc. Implicitly, the general decoding problem is important also in other cryptographic systems, e.g., in stream ciphers, where a fast general decoding algorithm would imply a fast correlation attack in any stream cipher where a correlation can be identified.

In Section 10.1 we give some notation from coding theory and formulates the general decoding problem. In Section 10.2 a new probabilistic algorithm is proposed. This proposed algorithm uses a time consuming Gaussian elimination. In Section 10.3 we propose an improved variant of this algorithm that circumvents this problem. The two algorithms were originally proposed in [42, 46]. A theoretical study of the complexity of these two algorithms is made in Section 10.4. In Section 10.5 we give some simulation results for the algorithms and compare with the theoretical results. Some cryptographic applications of the algorithm are presented in Section 10.6. Finally, we give some conclusions in Section 10.7.

10.1 Notation and Problem Formulation

We start by giving some preliminaries from coding theory, and by defining the general decoding problem. Let $\mathbf{x}$ and $\mathbf{y}$ be two binary vectors of length n, $\mathbf{x}, \mathbf{y} \in \mathbb{F}_2^n$. *The Hamming weight* of $\mathbf{x}$, $w_H(\mathbf{x})$, is the number of nonzero positions of $\mathbf{x}$. *The Hamming distance* between $\mathbf{x}$ and $\mathbf{y}$, $d_H(\mathbf{x}, \mathbf{y})$, is the number of coordinates in which $\mathbf{x}$ and $\mathbf{y}$ differ, i.e.,

$$d_H(\mathbf{x}, \mathbf{y}) = |\{i | 1 \leqslant i \leqslant n, x_i \neq y_i\}|.$$

For binary vectors, $\mathbf{x}$ and $\mathbf{y}$, the Hamming weight and the Hamming distance is related as

$$d_H(\mathbf{x}, \mathbf{y}) = w_H(\mathbf{x} + \mathbf{y}). \tag{10.1}$$

Let $\mathcal{C}$ be an arbitrary $[n, k]$ binary linear code. *The minimum distance*, $d_{\min}$, of the code $\mathcal{C}$ is defined as

$$d_{\min} = \min_{\mathbf{c}, \mathbf{c}' \in \mathcal{C}, \mathbf{c} \neq \mathbf{c}'} d_H(\mathbf{c}, \mathbf{c}').$$

Using Equation (10.1) the minimum distance of a linear code can also be determined as

$$d_{\min} = \min_{\mathbf{c} \in \mathcal{C}, \mathbf{c} \neq 0} w_H(\mathbf{c}).$$

If the minimum distance of a code is not known, we can sometimes estimate it with the result of the Gilbert-Varshamov bound. The bound states that there exists at least one $[n, k]$ binary linear code with minimum distance d_{GV}, where

$$d_{GV} = \max\{d | \sum_{i=0}^{d-1} \binom{n}{i} \leqslant 2^{n-k}\}.$$

We define two types of decoding algorithms for linear codes [93].

Definition 10.1: Let $\mathcal{C}$ be a general linear code, and let $\mathbf{r} \in \mathbb{F}_2^n$. A general decoding algorithm decodes the vector $\mathbf{r}$ into a codeword $\hat{\mathbf{c}} \in \mathcal{C}$, such that the Hamming distance, $d_H(\hat{\mathbf{c}}, \mathbf{r})$, between $\hat{\mathbf{c}}$ and $\mathbf{r}$ is minimal,

$$\hat{\mathbf{c}} = \arg \min_{\mathbf{c} \in \mathcal{C}} d_H(\mathbf{c}, \mathbf{r}).$$

□

A general decoding algorithm is also called a *complete decoding* algorithm.

Definition 10.2: Let $\mathbf{r} \in \mathbb{F}_2^n$ and let t be an integer. A t-bounded distance decoding algorithm decodes (if possible) the vector $\mathbf{r}$ into a codeword $\hat{\mathbf{c}}$, for which

$$d_H(\hat{\mathbf{c}}, \mathbf{r}) \leqslant t.$$

□

To the defined decoding algorithms several decoding problems considered to be hard can be defined, see [2, 95]. It is important to note that the underlying problem for both the McEliece public-key cryptosystem and the Stern identification scheme is *not* proven to be NP-complete.

In general, the output of a t-bounded distance decoding algorithm does not equal the output of a general decoding algorithm. Furthermore, the t-bounded distance decoding does not solve a problem proven to be NP-complete.

Let $\mathcal{E}_t$ be the set of all binary vectors of length n, such that $w_H(\mathbf{e}) \leqslant t, \forall \mathbf{e} \in \mathcal{E}_t$, i.e.,

$$\mathcal{E}_t = \{\mathbf{e} \in \mathbb{F}_2^n | w_H(\mathbf{e}) \leqslant t\}.$$

Assume that the received vector $\mathbf{r} = (r_1, r_2, \ldots, r_n)$ is created as

$$\mathbf{r} = \mathbf{v} + \mathbf{e},$$

where $\mathbf{v} \in \mathcal{C}$ and $\mathbf{e} \in \mathcal{E}_t$. Furthermore, let $\mathbf{c}$ be the estimated codeword from t-bounded distance decoding. From well-known result in coding theory [57] we know that if

$$t \leqslant \left\lfloor \frac{d_{\min} - 1}{2} \right\rfloor,$$

where $d_{\min}$ is the minimum distance of $\mathcal{C}$, then the result from the t-bounded distance decoder equals the result of complete decoding. Furthermore, this result is unique and $\mathbf{c} = \mathbf{v}$. For this setting of the general decoding problem several algorithms have been proposed [10, 55, 56, 60, 88]. These algorithms give an upper bound on the complexity of different cryptographic problems related to the general decoding problem. As an example, consider the McEliece cryptosystem. With the algorithm presented in [10] the average complexity of a successful attack on the McEliece cryptosystem with parameter $n = 1024, k = 524, t = 50$ is $2^{64.2}$.(in binary operations)

The model above assumes that the decoder takes one received word and produces one corresponding estimate. However, for some applications it is sufficient that we can correctly decode one received word from a set of many received words. Using this fact we consider the following setting. Assume that the decoder has a set of m received vectors, and successfully decoding just *one* vector in this set (into its closest codeword) is sufficient.

The problem can then be formulated as follows.

Problem formulation: Assume a set of m received words from a binary linear $[n,k]$ code $\mathcal{C}$ with minimum distance $d_{\min}$. Each received word has at most t errors, where $t \leqslant \lfloor \frac{d_{\min}-1}{2} \rfloor$. Find one codeword $\mathbf{c} \in \mathcal{C}$ such that it has distance at most t to one of the m received words.

This setting of the general decoding problem has also been studied in [12,94].

10.2 A Basic Algorithm

When considering the general decoding problem with one received word, it is possible to use an algorithm for finding low weight codewords, as a decoding algorithm. The algorithm for finding low weight codewords is applied in the following way. Let G be a generator matrix for the code $\mathcal{C}$ and let $\mathbf{r}$ be the received word. Create a new matrix G' by extending G with $\mathbf{r}$ as a new row,

$$G' = \begin{pmatrix} G \\ \mathbf{r} \end{pmatrix}.$$

The matrix G' can then be regarded as a generator matrix of a code $\mathcal{C}'$.

Suppose that we can find a codeword $\mathbf{v} \in \mathcal{C}'$ such that $w_H(\mathbf{v}) \leqslant t$. Since $\mathcal{C}'$ and $\mathcal{C}$ are linear codes, we can obtain $\mathbf{c}$ by adding $\mathbf{v}$ to $\mathbf{r}$, $\mathbf{c} = \mathbf{v} + \mathbf{r}$. Furthermore, $d_H(\mathbf{c},\mathbf{r}) = w_H(\mathbf{v})$. Thus, decoding an $[n,k]$ linear code is transformed to the problem of finding a minimum weight codeword in an $[n,k+1]$ linear code. This method is simple and works fine when it is used to decode the vectors one by one. However, in the case when we have a set of m received words, and look for an estimate for one of the received words, this approach is not possible. We are restricted to using decoding algorithms.

The most effective t-bounded distance decoding algorithms are generalizations of *information set decoding* [93]. Let $N = \{1,2,\dots,n\}$ be the set of all coordinates, and let $I = \{i_1, i_2, \dots, i_k\}$ be a subset with k elements. For the subset I, let $G = (V,W)_I$ denote the decomposition of the generator matrix G onto I, i.e., $V = (G_i)_{i\in I}$ and $W = (G_j)_{j \in N\setminus I}$, where G_i is the ith column of G. This notation follows previous work in this area [10]. Assume some ordering of I, $(i_1, i_2, \dots, i_k)$. V is then the matrix $\begin{pmatrix} G_{i_1} & G_{i_2} & \dots & G_{i_k} \end{pmatrix}$, where G_j is the jth column of G.

Definition 10.3: Let $\mathcal{C}$ be a linear code and let G be a generator matrix for $\mathcal{C}$. An information set I, is a k-element subset of N such that $V = (G_i)_{i\in I}$ has full rank. The complementary set, $J = N\setminus I$ is called a redundancy set. □

The principle of information set decoding is the following. Randomly select an information set and let $\mathbf{r}_{|I}$ be the restriction of $\mathbf{r}$ to I, i.e.,

$$\mathbf{r}_{|I} = (r_{i_1}, r_{i_2}, \ldots, r_{i_k}).$$

Since I is an information set there is exactly one codeword $\mathbf{c} \in \mathcal{C}$ such that $c_{|I} = r_{|I}$. The codeword $\mathbf{c}$ is then a candidate to a codeword close to $\mathbf{r}$. If $d_H(\mathbf{c}, \mathbf{r}) \leqslant t$ we output $\mathbf{c}$. On the other hand, if $d_H(\mathbf{c}, \mathbf{r}) > t$ we choose a new information set and repeat this procedure until $\mathbf{r}$ is successfully decoded.

The algorithm that we propose is a modification of information set decoding, where we allow $\mathbf{r}_{|I}$ to have at most p errors, and the parameter p is an integer, usually $p = 1, 2$, or 3. Furthermore, the algorithm uses an index set L, where $L \subset J$. The process of choosing the candidate codewords is somewhat similar to the Stern algorithm [88].

In: Generator matrix G, matrix with received words R, parameters t, p and l.

Out: Codeword $\mathbf{c}$ such that $d_H(\mathbf{c}, \mathbf{r}_i) \leqslant t$, where $\mathbf{r}_i$ is a word in R.

1. Randomly pick an information set I and form a systematic generator matrix
$$G' = (I_k, Z)_I.$$
2. Decompose the matrix R onto I and add suitable rows from G' to get a new matrix with received words in the form
$$R' = (0_k, T)_I.$$
3. Randomly select an l-element subset L of $J = N \setminus I$.
4. For each row $\mathbf{r}'$ in R', calculate $\mathbf{r}'_{|L}$ and store in a table sorted according to the value of $\mathbf{r}'_{|L}$.
5. Let $\mathcal{K}$ be the set of all linear combinations of at most p rows of G'. For each $\mathbf{c}' \in \mathcal{K}$ calculate $\mathbf{c}'_{|L}$ and look in the table for a received word such that $\mathbf{r}'_{|L} = \mathbf{c}'_{|L}$. If such a codeword exists and if $d_H(\mathbf{r}', \mathbf{c}') \leqslant t$ we have a successful decoding. If no pair $(\mathbf{r}', \mathbf{c}')$ with $d_H(\mathbf{r}', \mathbf{c}') \leqslant t$ is found, go to 1.

Figure 10.1: A description of the basic algorithm

Assume that we have m received words and that each received word is composed of a codeword from $\mathcal{C}$ corrupted by an error vector from $\mathcal{E}_t$. Furthermore, assume that the noise is independent between different codewords and that each error vector has at most t errors. Write the m received words as a matrix,

$$R = \begin{pmatrix} \mathbf{r}_1 \\ \mathbf{r}_2 \\ \vdots \\ \mathbf{r}_m \end{pmatrix},$$

where $\mathbf{r}_1, \ldots, \mathbf{r}_m$ are the different received words.

Randomly pick an information set I, and let J be the redundancy set, $J = N \setminus I$. Let $G = (V_G, W_G)_I$ and $R = (V_R, W_R)_I$ denote the decomposition of the matrices G and R onto I.

Since V_G has full rank, we can use Gaussian elimination to form a matrix

$$G' = (I_k, Z)_I,$$

where I_k is the $k \times k$ identity matrix. The matrix G' is a systematic generator matrix for a code $\mathcal{C}'$ equivalent to $\mathcal{C}$. Add suitable rows from G' to R such that the first k positions of $R = (V_R, W_R)_I$ will be all zero. Denote this new matrix by R',

$$R' = (0_k, T)_I,$$

where 0_k is the $k \times k$ zero matrix. When adding codewords to R', the number of errors in each received word is not changed. Hence, if we manage to decode and get the error vector for one received word in R', then the error pattern for the corresponding received word in R can be obtained.

Define a candidate set $\mathcal{K}$ as the set of all codewords in $\mathcal{C}'$ which have weight at most p in the first k positions, i.e.,

$$\mathcal{K} = \{\mathbf{c}' \in \mathcal{C}' | w_H(\mathbf{c}'_{|I}) \leqslant p\}.$$

The codewords in $\mathcal{K}$ correspond to the linear combinations of at most p rows from G'. Now, consider a row in R', denoted by $\mathbf{r}'$. To find a candidate $\mathbf{c}'$ from $\mathcal{K}$, we use an index set $L = \{j_1, j_2, \ldots, j_l\}$ with l coordinates from J. A candidate codeword $\mathbf{c}' \in \mathcal{K}$ is then given as a codeword from $\mathcal{K}$ such that $\mathbf{r}'_{|L} = \mathbf{c}'_{|L}$. To find a codeword of this form we use the following method. For each row in R', calculate $\mathbf{r}'_{|L}$ and store in a table sorted according to the value of $\mathbf{r}'_{|L}$. The candidate set $\mathcal{K}$ is created by taking all linear combinations of at most p rows of G'. For each candidate codeword $\mathbf{c}'$, calculate $\mathbf{c}'_{|L}$ and look in the table for a received word such that $\mathbf{r}'_{|L} = \mathbf{c}'_{|L}$. If such a codeword exists and if $d_H(\mathbf{r}', \mathbf{c}') \leqslant t$ we have a successful decoding. The algorithm is given in Figure 10.1.

10.3 An Improved Algorithm

The proposed algorithm in the previous section makes one Gaussian elimination on an $k \times n$ matrix in each iteration. Since this Gaussian elimination and the following update of the matrix R use many operations, we also propose another approach. To remove this expensive step we proceed as done in the modification of the Stern algorithm in [10].

Instead of randomly choosing an entirely new information set in each iteration, we create a new information set by modifying only one element of the previous one. To update the information set we use the following observation from [10].

Let I be an information set and let $G = (I_k, Z)_I$ be a systematic generator matrix. Choose $\lambda \in I$, $\mu \in J$, and let $I' = (I \setminus \lambda) \cup \{\mu\}$. Then I' is an information set if and only if $z_{\lambda,\mu} = 1$, where $Z = (z_{i,j})_{i \in I, j \in J}$.

Thus, we create a new matrix $G' = (I_k, Z')_{I'}$ by randomly choosing $\lambda \in I$ and $\mu \in J$ such that $z_{\lambda,\mu} = 1$. Then the λth and the μth columns of $G = (I_k, Z)_I$ and $R = (0_k, T)_I$ are exchanged. Furthermore, suitable row operations are performed to get matrices of the form $G' = (I_k, Z')_{I'}$ and $R' = (0_k, T')_{I'}$.

After this modification for creating the new information set, the algorithm proceeds as previously, see Figure 10.2.

When m is small, the algorithm presented by Canteaut and Chabaud in [10] will perform better than the algorithm in Figure 10.2. However, the algorithm in Figure 10.2 can be slightly modified to make it efficient also for small values of m. We introduce two new parameters k' and p', where $k' \leqslant k$ and $0 \leqslant p - p' \leqslant k - k'$. Step 4 and 5 of the algorithm in Figure 10.2 are then modified as follows.

4'. Split the rows of Z' into two parts, Z'_1 and Z'_2, where Z'_1 is the first k' rows of Z'. For each linear combination of p' rows of Z'_1, calculate $\mathbf{c}'_{1|L}$ and store in a table sorted according to the value of $\mathbf{c}'_{1|L}$.

5'. For each row $\mathbf{r}'$ of R' and for each linear combination $\mathbf{c}'_2$ of $p - p'$ rows of Z'_2, calculate $\mathbf{r}'_{|L} + \mathbf{c}'_{2|L}$ and look in the table for a codeword $\mathbf{c}'_1$ such that $\mathbf{c}'_{2|L} = \mathbf{r}'_{|L} + \mathbf{c}'_{2|L}$. If such a 3-tuple exists and if $w_H(\mathbf{c}'_1 + \mathbf{c}'_2 + \mathbf{r}') \leqslant t$ we have a successful decoding. If no 3-tuple $(\mathbf{c}'_1, \mathbf{c}'_2, \mathbf{r}')$ with $w_H(\mathbf{c}'_1 + \mathbf{c}'_2 + \mathbf{r}') \leqslant t$ is found, go to 1'.

For $p' = p/2$ and $k' = k/2$ the modified algorithm exactly corresponds to the algorithm presented in [10], and for $p' = p$ and $k' = k$ it corresponds to the algorithm in Figure 10.2. For large values of m, the optimal values of p' and k' are $p' = p$ and $k' = k$.

```
In:  Generator matrix $G = (I_k, Z)_I$, matrix with received words
     $R = (0_k, T)_I$, parameters $t$, $p$ and $l$.
Out: Codeword $\mathbf{c}$ such that $d_H(\mathbf{c}, \mathbf{r}_i) \leqslant t$, where $\mathbf{r}_i$ is a received
     word in $R$.
1'.  Choose $\lambda \in I$ and $\mu \in J$ such that $I' = (I \setminus \lambda) \cup \{\mu\}$ is an
     information set. Exchange the $\lambda$th and $\mu$th columns of $G$.
     Form a systematic generator matrix
         $G' = (I_k, Z')_{I'}$.
2'.  Exchange the $\lambda$th and $\mu$th columns of $R$ and form a matrix
     of the form
         $R' = (0_k, T')_{I'}$.
3.   Randomly select an $l$-element subset $L$ of $J = N \setminus I$.
4.   For each row $\mathbf{r}'$ in $R'$, calculate $\mathbf{r}'_{|L}$ and store in a table
     sorted according to the value of $\mathbf{r}'_{|L}$.
5.   Let $\mathcal{K}$ be the set of all linear combinations of at most $p$ rows
     of $G'$. For each $\mathbf{c}' \in \mathcal{K}$ calculate $\mathbf{c}'_{|L}$ and look in the table
     for a received word such that $\mathbf{r}'_{|L} = \mathbf{c}'_{|L}$. If such a codeword
     exists and if $d_H(\mathbf{r}', \mathbf{c}') \leqslant t$ we have a successful decoding. If
     no pair $(\mathbf{r}', \mathbf{c}')$ with $d_H(\mathbf{r}', \mathbf{c}') \leqslant t$ is found, go to 1'.
```

Figure 10.2: A description of the modified algorithm

10.4 Analysis of the Algorithms

In this section we present the work factor for the two algorithms proposed in Figure 10.1 and Section 10.2. The work factor is given as the average number of elementary bit operations for a successful decoding. This measure of complexity has appeared in most previous work in this topic, see e.g., [10, 12, 55]. To derive the work factor we start by deriving an expression for the average number of iterations. Then we analyze the number of bit operations in each iteration. The analysis is essential for finding the values of l and p for which the running time of the algorithms is minimal. In the derivation we assume that the weight of the error vector is exactly t.

Before we derive the complexity of the two algorithms we make a note regarding the implementation of the two algorithms. In the description of the algorithms we used a systematic generator matrix $G' = (I_k, Z)_I$ and a matrix R' with the received words of the form $R' = (0_k, T)_I$. When implementing the algorithms it is not necessary to store the first k columns of the two matrices. This will save memory and it will also decrease the computational complexity of the two algorithms. Hence, we assume in this section that the algorithms are implemented such that only the parts Z and T of the matrices G' and R', respectively, are stored in the memory.

The basic algorithm

First start with the algorithm in Section 10.2. Assume that $m = 1$. Then the probability of a correct decoding is given by the following lemma.

Lemma 10.1: Assume that only one word is received. The probability that the basic algorithm decodes this received word correctly in one iteration, denoted P_1, is

$$P_1 = \sum_{i=0}^{p} \frac{\binom{k}{i}\binom{n-k-l}{t-i}}{\binom{n}{t}}.$$

□

Proof: Let $\mathbf{r}$ be a received word, $\mathbf{r} = \mathbf{c} + \mathbf{e}$, where $\mathbf{c} \in \mathcal{C}$ and $w_H(\mathbf{e}) = t$. Assume that $w_H(\mathbf{e}_{|I}) = i$. Then for the correct codeword, $\mathbf{c}$, $d_H(\mathbf{c}_{|I}, \mathbf{r}_{|I}) = i$. If $i > p$ the algorithm can not correct this error pattern, but if $i \leqslant p$ the received word will be correctly decoded if $w_H(\mathbf{e}_{|L}) = 0$.

Hence, the probability of successful decoding in one iteration is the probability that there are p or less errors in the first k positions and no errors in the next l positions.

There are $\binom{k}{i}$ different possibilities for i errors in k positions, and $\binom{n-k-l}{t-i}$ possibilities that the remaining $t-i$ errors are in the last $n-k-l$ positions. Hence, there are

$$\binom{k}{i}\binom{n-k-l}{t-i}$$

different error patterns that can be decoded successfully in one iteration. The total number of error patterns with t errors is $\binom{n}{t}$, and the probability that a correct decoding is obtained in one iteration is

$$P_1 = \sum_{i=0}^{p} \frac{\binom{k}{i}\binom{n-k-l}{t-i}}{\binom{n}{t}}.$$

■

If we instead have m received words in the set R, we define the algorithm to be successful if at least one word is correctly decoded. Let N_{alg} be the average number of iterations for a correct decoding given m received words. The value of N_{alg} is given by the following theorem.

Theorem 10.2: The average number of iterations for the basic algorithm is

$$N_{alg} = \frac{1}{\sum_{i=1}^{m} \binom{m}{i} (-1)^{(i+1)} P_1^i}, \tag{10.2}$$

where P_1 is given by Lemma10.1, and m is the number of received words. □

Proof: Define P_m as the probability that the algorithm is successful, and P_F the probability of a failure. Clearly, $P_m = 1 - P_F$. Since the error patterns in the different received words are independent we get that $P_F = (1 - P_1)^m$. Hence,

$$P_m = 1 - (1 - P_1)^m = \sum_{i=1}^{m} \binom{m}{i} (-1)^{(i+1)} P_1^i,$$

and $N_{alg} = 1/P_m$. ■

We are interested in the complexity given as the average number of bit operations for a successful decoding. To be able to calculate this complexity we need to know the complexity of each iteration. Let C_{alg} be the expected number of bit operations of one iteration.

Theorem 10.3: The computational complexity of one iteration is

$$C_{alg} = nk^2/2 + mnk/2 + \sum_{i=1}^{p} \binom{k}{i} il + ml + \frac{\sum_{i=0}^{p} \binom{k}{i} mn(i+1)}{2^l}. \tag{10.3}$$

□

Proof: In each iteration the algorithm performs the following operations. A Gaussian elimination which takes $nk^2/2$ operations. Then we add codewords to make R all-zero in the first positions. This takes $mkn/2$ operations. To check the weight in the l positions of G and R, it takes $\sum_{i=1}^{p} \binom{k}{i} il$ and ml operations respectively. The last step consists of checking the weight for the entire codeword if the weight in the l positions is the same. On average we have $\sum_{i=0}^{p} \binom{k}{i} m/2^l$ pairs of codewords and received words for which the weight is the same. For each of these pairs the calculation of the distance takes $n(i+1)$ operations. ■

Let W_{alg} denote the average total complexity, or work factor, for a successful decoding. The value of W_{alg} is calculated as

$$W_{alg} = N_{alg} \cdot C_{alg},$$

where N_{alg} and C_{alg} are given by (10.2) and (10.3) respectively.

The modified algorithm

For the improved algorithm presented in Section 10.3, the information sets in different iterations are dependent, and thus, so are the number of errors in them. However, the algorithm can be modelled by a discrete time random process. In fact, by using the same methods as in [10] we will show that the algorithm can be described as a Markov chain. The following analysis will be similar to the analysis in [10].

Assume that we have one received word $\mathbf{r} = \mathbf{c} + \mathbf{e}$. Furthermore, let I_i be the information set and L_i the index set in the ith iteration. The ith iteration can then be represented by a random variable X_i which corresponds to the number of errors of $\mathbf{r}$ in I_i, i.e., $w_H(\mathbf{e}_{|I_i})$. This random variable takes its values in the set $\{0, 1, \ldots, t\}$. When $X_i \leqslant p$ we also have the possibility that the algorithm succeeds with the decoding. Hence, we introduce a success state and denote this state by S. The state space of the stochastic process $\{X_i\}_{i\in\mathbb{N}}$ is then

$$\mathcal{E} = \{S\} \cup \{0, 1, \ldots, t\}.$$

A Markov chain is a discrete stochastic process in which the distribution at any time in the future depends only on the current state of the process, and not on how that state was reached. For a Markov chain, the following condition is satisfied,

$$\begin{aligned} P(X_i = u_i | X_{i-1} = u_{i-1}, X_{i-2} = u_{i-2}, \ldots, X_0 = u_0) &= \\ P(X_i = u_i | X_{i-1} = u_{i-1}). \end{aligned}$$

A *homogeneous Markov chain* is a Markov chain where the conditional probabilities

$$P(X_i = v | X_{i-1} = u), \quad u, v \in \mathcal{E},$$

do not depend on i. Denote the transition probability $P(X_i = v | X_{i-1} = u)$ by $p_{u,v}$, and the *transition matrix* by P, where $P = (p_{u,v})_{u,v\in\mathcal{E}}$.

Since the new information set I_i only depends on the previous information set I_{i-1}, and the index set L is randomly chosen, the stochastic process $\{X_i\}_{i\in\mathbb{N}}$, defined above, is a homogeneous Markov chain. The Markov chain is completely determined by the transition matrix P and the initial probability vector $\pi_0 = (P(X_0 = u))_{u\in\mathcal{E}}$.

To express the different transition probabilities we define $\beta(u)$ to be

$$\beta(u) = P(X_i = S | w_H(\mathbf{e}_{|I_i}) = u),$$

where $w_H(\mathbf{e}_{|I_i})$ is the number of errors in the information set I_i. The value of $\beta(u)$ is given by the probability that we choose all l positions of L without errors, i.e.,

$$\beta(u) = \begin{cases} \frac{\binom{n-k-t+u}{l}}{\binom{n-k}{l}}, & u \leqslant p, \\ 0, & u > p. \end{cases} \tag{10.4}$$

The transition probabilities can then be calculated as

$$\begin{aligned}
p_{u,u} &= (1-\beta(u))(\frac{k-u}{k}\cdot\frac{n-k-t+u}{n-k}+\frac{u}{k}\cdot\frac{t-u}{n-k}), \quad u \geqslant 0,\\
p_{u,u-1} &= (1-\beta(u-1))(\frac{u}{k}\cdot\frac{n-k-t+u}{n-k}), \quad u \geqslant 1,\\
p_{u,u+1} &= (1-\beta(u+1))(\frac{k-u}{k}\cdot\frac{t-u}{n-k}), \quad u \geqslant 0,\\
p_{u,S} &= (\beta(u)(\frac{k-u}{k}\cdot\frac{n-k-t+u}{n-k}+\frac{u}{k}\cdot\frac{t-u}{n-k})\\
&\quad +\beta(u-1)(\frac{u}{k}\cdot\frac{n-k-t+u}{n-k})\\
&\quad +\beta(u+1)(\frac{k-u}{k}\cdot\frac{t-u}{n-k}), \quad u \geqslant 0,\\
p_{S,S} &= 1,\\
p_{u,v} &= 0, \quad \text{otherwise.}
\end{aligned}$$

The initial state probabilities are given by the vector $\pi_0(u)$ and can be calculated to the following expressions

$$\pi_0(u) = \begin{cases} \dfrac{(1-\beta(u))\binom{t}{u}\binom{n-t}{k-u}}{\binom{n}{k}}, & u \neq S; \\ \displaystyle\sum_{u=0}^{p} \frac{\beta(u)\binom{t}{u}\binom{n-t}{k-u}}{\binom{n}{k}}, & u = S. \end{cases} \tag{10.5}$$

A maximal state subset which cannot be left once it is entered is called a persistent space [50]. Since the persistent space of this Markov chain consists of only the state S, and all other states are non-persistent, the Markov chain associated with the algorithm is an absorbing Markov chain. Since the probability that an absorbing Markov chain is in an absorbing state after n steps tends to 1 as n tends to infinity, we deduce that the proposed algorithm converges.

Let Q be the matrix containing the transition probabilities between the nonpersistent states. The matrix P can then be written as

$$P = \begin{pmatrix} Q & P^* \\ 0 & 1 \end{pmatrix},$$

where P^* is a column vector containing the transition probabilities from a nonpersistent state to S. Now we use some basic results from Markov chain theory. Define the fundamental matrix, first introduced in [50], as

$$R = \sum_{m=0}^{\infty} Q^m = (I_d - Q)^{-1}, \tag{10.6}$$

where I_d is the $d \times d$ identity matrix. For a Markov chain with an absorbing state, fundamental matrix R and initial state probabilities $p_0(u)$, the expected number of iterations until the Markov chain reaches the persistent state, denoted by M, is given as [50]

$$M = \sum_u p_0(u) \sum_v R_{u,v}. \tag{10.7}$$

Let N_{mod} be the average number of iterations for a successful decoding using the modified algorithm. Using (10.7), N_{mod} is given as

$$N_{mod} = \sum_{u=0}^{t} \pi_0(u) \sum_{v=0}^{t} R_{u,v}, \tag{10.8}$$

where $R_{u,v}$ is the element in row u and column v of R calculated by (10.6) and $\pi_0(u)$ is the initial state probabilities calculated by (10.5).

To calculate the total work factor we also need the number of operations in each iteration. The same methods that were used for the original algorithm can be used and the result is given by the following theorem.

Theorem 10.4: The complexity in one iteration is

$$C_{mod} = nk/2 + mn/2 + \sum_{i=1}^{p} \binom{k}{i} il + ml + \frac{\sum_{i=0}^{p} \binom{k}{i} mn(i+1)}{2^l}. \tag{10.9}$$

□

Proof: If we compare the modified algorithm with the original algorithm we see that instead of a Gaussian elimination we only swap one position in the information set. Hence, in the first step we have $k/2$ rows of G and $m/2$ rows of R that has 1 in the column that we swap. The update of G and R then takes in average $nk/2 + mn/2$ operations. After step $1'$ the two algorithms use the same operations and the corresponding complexity can be taken from Theorem 10.3. ■

To get the average number of iterations of the algorithm when we have a set of m received words, we need the probability that the algorithm is successful after N iterations given one codeword. Denote this probability by p_N. From Markov chain theory we get that this probability can be written as

$$p_N = \sum_u \pi_0(u)(L^{-1}\Lambda^N L)_{u,S},$$

where Λ is a diagonal matrix and L is a matrix such that $P = L^{-1}\Lambda L$. Since the error patterns in the different received words are independent, we get that the probability that we succeed after N iterations, given m received words, denoted $p_{N,m}$, is

$$p_{N,m} = 1 - (1 - p_N)^m.$$

This results in

$$p_{N,m} = \sum_{i=1}^{m} \binom{m}{i} (-1)^{(i+1)} \left(\sum_u \pi_0(u)(L^{-1}\Lambda^N L)_{u,S} \right)^i.$$

However, since this expression is rather complex, one can use the following observation. If the average number of iterations given one received word is large, then we can approximate the average number of iterations given m received words as $\frac{N_{mod}}{m}$, where N_{mod} is given by (10.8). Hence, we get that the total work factor for the modified algorithm with a set of m received words is given as

$$W_{mod} \approx \frac{N_{mod} \cdot C_{mod}}{m},$$

where C_{mod} is given by (10.9) and N_{mod} is given by (10.8).

10.5 Simulation and Theoretical Results

In order to check the behavior of the two proposed algorithms we have made simulations for the problem of decoding general linear codes. All simulations were made on a PC with a Pentium II processor running Linux as operating system. The simulation programs were written in C.

First we used a [256,128] linear code. The number of errors to correct was set to $t = 14$, and the number of received words m varied between 1 and 1024.

m	p	l	N_{alg}	$\log_2(W_{alg})$	N_{sim}	$\log_2(T)$
1	2	4	280	29.4	281	10.6
2	2	5	155	28.6	154	9.8
4	2	6	86	27.8	86	9.0
8	2	7	48	27.0	49	8.2
16	2	8	27	26.2	28	7.5
32	2	9	15	25.5	16	6.8
64	2	9	7.9	24.9	7.9	6.1
128	2	10	4.6	24.5	4.7	5.8
256	2	11	2.8	24.3	2.8	5.7
512	2	11	1.7	24.3	1.7	6.3
1024	2	13	1.4	24.7	1.3	6.8

Table 10.1: Simulation results for a [256,128] code using the basic algorithm

The average number of iterations in the simulations were compared with the theoretical results given by the previous section. In Table 10.1 the simulated average number of iterations N_{sim} and time T in milliseconds for a correct decoding is given for the basic algorithm. Furthermore, Table 10.1 shows the average number of iterations N_{alg} and the work factor W_{alg} given by the theoretical analysis in the previous section. The algorithm parameters p and l are chosen such that the work factor is minimal given the code parameters and the number of received words. Note that the values for the time and work factor in Table 10.1 are given as logarithms using base 2. The corresponding results for the modified algorithm are presented in Table 10.2.

From Table 10.1 and Table 10.2 we can see that the time for a successful decoding decreases with a factor of approximately 2^7 when the number of received words increases from 1 to 1024. The fastest decoding algorithm for $m = 1$ presented in [10] has a work factor of $2^{26.5}$. Hence, we see that for $m = 1$ the modified algorithm in Section 10.3 has a work factor of a factor of 4 higher, but when m increases the work factor can be up to a factor of 50 lower.

We observe that when the number of received vectors increases, the optimal value of p decreases. It also seems that the value of l increases for a given p when m increases.

When comparing the running times from an implementation with the theoretical results it is important to keep in mind that the theoretical analysis is based on bit operations, while the computer implementations often use 32-bit word representation. Hence, some of the operations can be made on

m	p	l	N_{mod}	$\log_2(W_{mod})$	N_{sim}	$\log_2(T)$
1	1	7	20118	28.4	20105	10.9
2	1	8	10255	27.5	10190	10.1
4	1	8	5127	26.5	5095	9.1
8	1	9	2619	25.6	2612	8.3
16	1	10	1341	24.7	1333	7.6
32	1	11	688	23.9	668	7.0
64	1	12	354	23.2	344	6.5
128	1	12	177	22.7	167	5.8
256	1	12	89	22.3	79	5.4
512	2	16	8.5	21.7	3.5	3.3
1024	2	17	4.6	21.1	1.8	3.7

Table 10.2: Simulation results for a [256,128] code using the modified algorithm

32 bit simultaneously and the simulation times may be different from the theoretical values.

Since the improved algorithm in Section 10.3 has much lower complexity than the basic algorithm in Section 10.2, the rest of the presented results is given for the modified algorithm.

To check the behavior when the length of the codewords increases we have calculated the work factor for different rate 1/2 codes, i.e., $k = n/2$. Furthermore, we use two different cases depending on the number of received words. The first case uses one received word, $m = 1$, and the second case uses $m = 2^{20}$. The length of the codewords varies between 256 and 1024 and Figure 10.3 shows the work factor for the different codes using the modified algorithm. The parameter t was determined by the Gilbert-Varshamov bound.

From Figure 10.3 we can see that the gain in performance that can be obtained when increasing the number of received words increases when the length of the codewords increases. As a particular example consider the following. When $n = 256$ the gain is 2^7 and for $n = 1024$ the gain has increased to 2^{15}. For large n the number of received words must be large to be able to have maximal gain, but most of the gain is obtained already for small values of m.

Another property to study is the algorithmic behavior for different rates, i.e., let k vary for the same value of n. In Figure 10.4 we give the theoretical performance for different codes with $n = 512$, k varies from 64 to 448, and t was determined by the Gilbert-Varshamov bound. The figure shows the

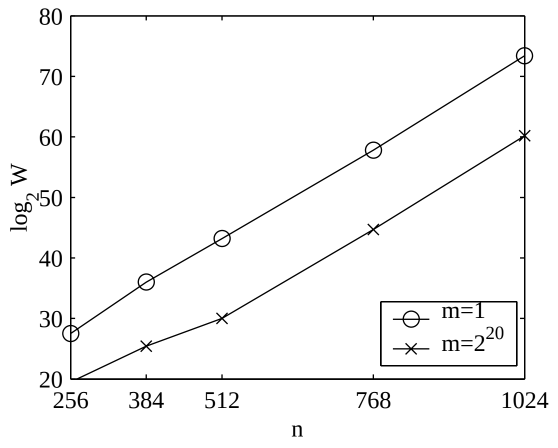

Figure 10.3: The theoretical work factor for different rate 1/2 codes.

results both for $m = 1$ and $m = 2^{20}$.

Decoding one word from a set of m received words is easier than decoding one received word. Nevertheless, for some cryptographic applications it is easy to get a list of m received words and decoding only one of them is equivalent to breaking the system.

To see how much we gain by increasing the value of m, we compare the complexity of the proposed algorithm with the results obtained by Stern's algorithm [88] and the algorithm proposed by Canteaut and Chabaud in [10], which currently are the most efficient algorithms for $m = 1$. We pick the following code parameters, $n = 1024$, $k = 525$, $t = 50$, and $n = 512$, $k = 512$, $t = 56$. This size of the parameters corresponds to the code parameters for the McEliece cryptosystem and the Stern identification scheme, respectively. In Table 10.3 we give the complexity for the modified algorithm for different values of m.

As an example from Table 10.3 consider the McEliece cryptosystem. From [10] we get the work factor $2^{64.2}$ and using the algorithm in Section 10.3 we get the work factor $2^{68.1}$ for $m = 1$ and $2^{49.0}$ for $m = 2^{40}$. Hence, we see that when m is small the algorithm in [10] is better but when $m \geqslant 32$ the algorithm in Section 10.3 gives better performance. For this particular code the value of m that gives best performance was $m = 2^{40}$. However, already for $m = n$ the proposed algorithm has a factor of 2^5 lower complexity than the algorithm in [10].

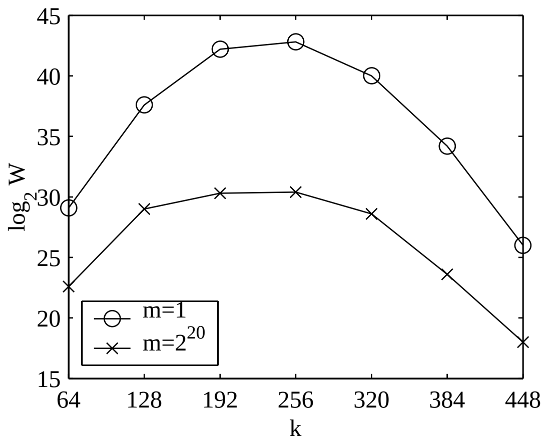

Figure 10.4: The theoretical work factor for different rates when $n = 512$.

	Cryptosystem	
Decoding Algorithm	McEliece $n = 1024$, $k = 524$, $t = 50$	Stern $n = 512$, $k = 256$, $t = 56$
[88]	$2^{69.9}$	$2^{73.5}$
[10]	$2^{64.2}$	$2^{69.9}$
$m = 1$	$2^{68.3}$	$2^{72.6}$
$m = 32$	$2^{64.0}$	$2^{70.0}$
$m = 256$	$2^{61.4}$	$2^{67.5}$
$m = 512$	$2^{60.6}$	$2^{66.9}$
$m = 1024$	$2^{59.7}$	$2^{66.4}$
$m = 2^{15}$	$2^{56.9}$	$2^{65.0}$
$m = 2^{30}$	$2^{50.7}$	$2^{60.8}$
$m = 2^{40}$	$2^{49.4}$	$2^{59.2}$

Table 10.3: Comparison of the complexity of different algorithms for attacks on the McElice cryptosystem and the Stern identification scheme.

10.6 Some Cryptographic Applications

In this section we shortly describe some cryptographic applications for which the proposed decoding algorithms are applicable.

The Stern identification scheme

At Crypto'93 Stern proposed an identification scheme based on syndrome decoding, [89]. All users share a parity check matrix H for a binary $[n, k]$ linear code, denoted by $\mathcal{C}$. Each user has secretly chosen a vector, $\mathbf{s}$, of Hamming weight p, where the parameter p is an integer slightly below the expected minimum distance of the code. The public key, or identification, is computed as $\mathbf{i} = \mathbf{s}H^T$. By an interactive zero-knowledge protocol, any user can identify himself to another by proving he knows $\mathbf{s}$ without revealing it. If an adversary can find a weight p vector with syndrome $\mathbf{i}$, he can identify himself as an authorized user and the system is broken. The original parameters proposed in [89] are $n = 512$, $k = 256$, and $p = 56$.

To apply the proposed algorithm for attacking the Stern identification scheme with m users, one proceeds as follows. Write the parity check matrix as $H = (P^T, I_{n-k})$. A generator matrix G for $\mathcal{C}$ is then given as $G = (I_k, P)$. For a given syndrome $\mathbf{i}$ we can form the word $\mathbf{r} = (\mathbf{0}, \mathbf{i})$. Clearly, $\mathbf{r}H^T = \mathbf{i}$. Assume we have a codeword $\mathbf{c} \in \mathcal{C}$ such that $d_H(\mathbf{c}, \mathbf{r}) = t$. Then let

$$\hat{\mathbf{s}} = \mathbf{c} + \mathbf{r} = \mathbf{c} + (\mathbf{0}, \mathbf{i}).$$

The syndrome of $\hat{\mathbf{s}}$ is then given as

$$\hat{\mathbf{s}}H^T = (\mathbf{c} + (\mathbf{0}, \mathbf{i}))H^T = \mathbf{c}H^T + (\mathbf{0}, \mathbf{i})H^T = \mathbf{0} + \mathbf{i} = \mathbf{i}.$$

Thus we have found a weight t vector, namely $\hat{\mathbf{s}} = (\mathbf{c} + (\mathbf{0}, \mathbf{i}))$, such that its syndrome is $\mathbf{i}$. Thus, decoding the received word $(\mathbf{0}, \mathbf{i})$ can give us the desired vector. Now, for each user, we form the corresponding vector $\mathbf{r}_j = (\mathbf{0}, \mathbf{i}_j)$. The matrix R is then created by using vector $\mathbf{r}_j$ as the jth row of R. Finally, apply the proposed algorithm.

From Table 10.3 we find that in a system with $m = 1024$ users, the average complexity for finding one vector with desired properties is $2^{66.4}$. This can be compared with $2^{69.9}$ which is obtained by the algorithm in [10].

A dual version of the Stern identification scheme that uses a generator matrix of the code was proposed by Véron [96]. In order to reduce the number of transmitted bits, he proposed the parameters $n = 512$, $k = 120$, and $p = 114$. Also here we can apply the proposed algorithm.

The McEliece public-key cryptosystem

See [60] for a complete description of the McEliece cryptosystem. McEliece cryptosystem uses an $[n, k]$ linear code $\mathcal{C}$ from a family of codes for which an efficient decoding algorithm exists. In the original proposal Goppa codes were suggested. Furthermore, assume that the code can correct t errors. The private key consists of an $[n, k]$ linear binary code $\mathcal{C}$, a random $k \times k$ binary invertible matrix S, and a random $n \times n$ permutation matrix P.

The public key is the $k \times n$ matrix $G' = SGP$, where G is a generator matrix of the secret code $\mathcal{C}$. Thus, the matrix G' is a generator matrix for a linear code $\mathcal{C}'$ equivalent to $\mathcal{C}$.

The encryption of a k bit message, $\mathbf{m}$, is given as

$$\mathbf{r} = \mathbf{m}G' + \mathbf{e},$$

where $\mathbf{e}$ is a random error pattern with $w_H(\mathbf{e}) = t$. The n bit vector $\mathbf{r}$ is then transmitted to the receiver.

To decrypt, the receiver starts by creating the vector

$$\mathbf{y} = \mathbf{r}P^{-1} = \mathbf{m}SG + \mathbf{e}P^{-1}.$$

Using the fast decoding algorithm for $\mathcal{C}$ the vector $\mathbf{m}S$ is obtained. Finally, the message is given by $\mathbf{m} = (\mathbf{m}S)S^{-1}$.

The ciphertexts in the McEliece cryptosystem correspond to words of the public code $\mathcal{C}'$ with t errors. Thus, if we can find a decoding method for $\mathcal{C}'$, the system is broken. To apply the proposed algorithm we assume that m ciphertexts are given as m vectors and that decrypting any of them is sufficient for success in the attack. By putting the vectors in a matrix R, the proposed algorithms can be applied.

For the original parameters $n = 1024$, $k = 524$, and $t = 50$ the average complexity for breaking the system given m ciphertexts is given in Table 10.3.

Niederreiter proposed a dual version of this system, see [73]. The plaintext in the Niederreiter cryptosystem is an n bit vector with weight t, and the associated ciphertext is the syndrome. Also here we can apply the proposed algorithms.

A McEliece-based digital signature scheme

In [21] Courtois, Finiasz, and Sendrier proposed a method to achieve a digital signature scheme based on the McEliece public key cryptosystem. In the paper three different versions of the signature scheme are proposed. The versions differ in the length of the signature, but the underlying decoding problem is the same. As in the McEliece public key cryptosystem, the private key is a binary linear $[n, k]$-linear code, denoted by $\mathcal{C}$, for which a fast decoding algorithm exists. The public key is a parity check matrix, H, of a code equivalent to $\mathcal{C}$.

To sign a message $\mathbf{m}$ the sender first applies a secure public hash function and hashes the message to a binary vector of length $n-k$, $\mathbf{y} = h(\mathbf{m})$. The signer uses the fast decoding algorithm of $\mathcal{C}$, known only by the signer, to find an n bit vector $\mathbf{x}$ of weight t such that the syndrome of $\mathbf{x}$ is $\mathbf{y}$. (The signature is obtained by applying a public compression function to $\mathbf{x}$.) When a receiver verifies a given signature he proceeds as follows. From the signature $\mathbf{x}$ the syndrome $\mathbf{y}$ is calculated, $\mathbf{y} = H\mathbf{x}^T$. This syndrome is then compared to the hash value $h(\mathbf{m})$, obtained by hashing the message $\mathbf{m}$.

To successfully attack the system and forge a signature on a random message $\mathbf{m}$, the attacker must find a vector $\mathbf{x}$ such that the syndrome is $h(\mathbf{m})$. Thus, the security of the system is equivalent to syndrome decoding.

In the proposal [21] the following parameters of $\mathcal{C}$ were suggested, $n = 65536$, $k = 65392$, and $t = 11$. For the proposed parameters the complexity of an attack was determined to be 2^{85}, using the decoding algorithm in [10].

We demonstrate how the results of this paper can be used to reduce the complexity of an attack. We assume that the attacker has m randomly generated messages, $\mathbf{m}_1, \mathbf{m}_2, \ldots, \mathbf{m}_m$. The attacker is successful if he can forge a signature on one of the messages. Using the public hash function h, m different syndromes can be created, $\mathbf{y}_i = h(\mathbf{m}_i)$, $i = 1 \ldots m$. The decoding algorithm in Section IV is applied and we search for a binary vector $\mathbf{x}$ of weight t, such that $\mathbf{y}_i = H\mathbf{x}_i^T$ for some i. The complexity of the attack can be estimated by the results in Section V. For example, choosing $m = 2^{44}$, an attack would have complexity $W = 2^{76.5}$ bit operations. In conclusion, by using the algorithm in Section IV the complexity of an attack can be slightly reduced.

Correlation attacks on stream ciphers

In Section 3.2 it was shown that if a correlation between the keystream and one of the LFSR's in the stream cipher is identified, the cipher can be broken by solving a decoding problem.

Let the length of the LFSR be l and let the set of possible LFSR sequences be denoted by $\mathcal{L}$. Clearly, $|\mathcal{L}| = 2^l$ and for a fixed length n the set of all truncated sequences from $\mathcal{L}$ is also a linear $[n, l]$ block code [57], referred to as $\mathcal{C}$. Thus, the LFSR sequence $\mathbf{u} = u_1, u_2, \ldots, u_n$ is regarded as a codeword from $\mathcal{C}$ and the keystream sequence $\mathbf{z} = z_1, z_2, \ldots, z_n$ is regarded as the received channel output. From the definition of the correlation between u_i and z_i, we can describe each z_i as the output from the binary symmetric channel, BSC, when u_i was transmitted. The correlation probability $1-p$, defined by $1 - p = P(z_i = u_i)$, gives p as the crossover probability (error probability) in the BSC. W.l.o.g we can assume $p < 0.5$.

The length n should be at least around $n_0 = l/(1 - h(p))$ for unique decoding, where $h(p)$ is the binary entropy function. If $n \gg n_0$ fast correlation attacks are applicable, however, when n is close to n_0 fast correlation attacks fail since they require long sequences of received keystream symbols. When n is close to n_0 we can apply the algorithm in Figure 10.2 to recover the initial state of the LFSR. To apply information set decoding in an attack on stream ciphers is also considered in [14].

From the cyclic structure of $\mathcal{C}$, all vectors of the form $(u_i, \ldots, u_{n+i})$, $i > 1$, are codewords in $\mathcal{C}$. Hence, we can from the output sequence $z_1, z_2, \ldots$ form many received words, namely $(z_1, \ldots, z_n)$, $(z_2, \ldots, z_{n+1})$, and so on. Even more words can be created by observing that $x \to x^2$ maps into the same code, i.e., $(u_0, u_2, u_4, u_6, \ldots, u_{2n})$ is also a codeword in $\mathcal{C}$. Finally, we add suitable codewords to take each received vector in R to the desired form and proceed as before.

10.7 Summary

A probabilistic decoding algorithm that uses a set of received words in order to improve performance has been suggested. It has been shown that it can be applied to most of the cryptographic applications that rely on the general decoding problem. A significant reduction of the computational complexity compared with Stern's original algorithm and other algorithms has been demonstrated, both theoretically and through simulations for codes of length around 1000.

The results improve the attacks of different cryptosystems based on the general decoding problem. The improvements of the attacks restrict the parameters of cryptosystems where the algorithms are applicable. For the Stern identification scheme with 2^{15} users, the computational complexity of 2^{65} corresponds to roughly 2^{18} hours of work on a PC using our simulation program as an estimate. Since the proposed decoding algorithms are parallelizable, a joint software project with 10000 users would crack the scheme in roughly a day. For the McEliece public key cryptosystem with 2^{15} messages, only a few minutes would be required for the same setting. This should render the proposed parameters to be completely insecure. For the Stern identification scheme the results indicate that length 1024 would be suitable for a secure system. Similarly, for the McEliece public key cryptosystem, we would suggest length 2048 for a secure system.

11 Concluding Remarks

In this thesis, fast correlation attacks on stream ciphers have been studied. After an introduction of correlation attacks and fast correlation attacks, three algorithms for fast correlation attacks were proposed. The proposed algorithms considers stream ciphers with linear feedback shift registers with feedback polynomials having an arbitrary number of taps. The presented results, indicate that it is possible to attack LFSRs with length approximately 100 in practise. From the theoretical results, we can also calculate the complexity of an attack on longer LFSRs.

Recently, many other algorithms for fast correlation attacks have been proposed. The performance of four of them are compared with the performance of the algorithms proposed in this thesis. From the simulations we concluded that the algorithm based on recovering a linear polynomial, is one of the algorithms having best performance.

In future work it would be of interest to develop the algorithms for fast correlation attacks further. In particular, the implementation issues of the algorithms can be considered.

Although the precomputation only has to be performed once, it would be of interest to reduce the complexity in the precomputation for practical implementations. As an example, consider the attack on LILI-128 presented in this thesis. The precomputation takes approximately a factor of 250 more operations compared with the fast correlation attack.

In this thesis, the fast correlation attacks have mainly been presented in a general model. It would be interesting to apply the algorithms to some proposed stream ciphers.

Bibliography

[1] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv. Optimal decoding of linear codes for minimizing symbol error rate. *IEEE Trans. on Inform. Theory*, IT-20:284–287, March 1974.

[2] E. R. Berlekamp, R. J. McEliece, and H.C.A. van Tilborg. On the inherent intractability of certain coding problems. *IEEE Trans. on Inform. Theory*, IT-24:384–386, July 1978.

[3] C. Berrou and A. Glavieux. Near optimum error-correcting coding and decoding: Turbo codes. *IEEE Trans. on Communications*, COM-44:1261–1271, Oct. 1996.

[4] C. Berrou, G. Glavieux, and P. Thitimashima. Near Shannon limit coding and decoding: Turbo-codes. In *Proceedings of ICC'93*, pages 1064–1070, Geneva, Switzerland, May 1993.

[5] A. Biryukov and A. Shamir. Cryptanalytic time/memory/data tradeoffs for stream ciphers. In *Advances in Cryptology—ASIACRYPT 2000*, volume LNCS 1976, pages 1–13. Springer-Verlag, 2000.

[6] A. Biryukov, A. Shamir, and D. Wagner. Real time cryptanalysis of A5/1 on a PC. In *Fast Software Encryption 2000*, volume LNCS 1978, pages 1–18. Springer-Verlag, 2000.

[7] A. Blum, M. Furst, M. Kearns, and R. Lipton. Cryptographic primitives based on hard learning problems. In *Advances in Cryptology—CRYPTO'93*, volume LNCS 547, pages 278–291. Springer-Verlag, 1993.

[8] L. Blum, M. Blum, and M. Shub. A simple unpredictable pseudorandom number genrator. *SIAM Journal on Computing*, vol. 15:364–383, 1986.

[9] L. Brynielsson. Statistische Überlegungen zu einer methode von F. Jönsson. personal communication.

[10] A. Canteaut and F. Chabaud. A new algorithm for finding minimum-weight words in a linear code: Application to McEliece cryptosystem and to narrow-sense BCH codes of length 511. In *IEEE Trans. on Inform. Theory*, volume IT-44, pages 367–378, 1998.

[11] A. Canteaut and M. Trabbia. Improved fast correlation attacks using parity-check equations of weight 4 and 5. In *Advances in Cryptology—EUROCRYPT 2000*, volume LNCS 1807, pages 573–588. Springer-Verlag, 2000.

[12] F. Chabaud. On the security of some cryptosysytems based on error-correcting codes. In *Advances in Cryptology—EUROCRYPT'94*, volume LNCS 950, pages 176–185. Springer-Verlag, 1994.

[13] V. Chepyzhov, T. Johansson, and B. Smeets. A simple algorithm for fast correlation attacks on stream ciphers. In *Fast Software Encryption 2000*, volume LNCS 1978, pages 181–195. Springer-Verlag, 2000.

[14] V. Chepyzhov and B. Smeets. Performance of information set decoding in a correlation attack on certain stream ciphers. unpublished report.

[15] V. Chepyzhov and B. Smeets. On a fast correlation attack on certain stream ciphers. In *Advances in Cryptology—EUROCRYPT'91*, volume LNCS 547, pages 176–185. Springer-Verlag, 1991.

[16] P. Chose, A. Joux, and M. Mitton. Fast correlation attacks: An algorithmic point of view. In *Advances in Cryptology—EUROCRYPT 2002*. To be published in LNCS, 2002.

[17] A. Clark, J. Golic, and E. Dawson. A comparison of fast correlation attacks. In *Fast Software Encryption'96*, volume LNCS 1039, pages 145–158. Springer-Verlag, 1996.

[18] D. Coppersmith, S. Halevi, and C. Jutla. Cryptanalysis of stream ciphers with linear masking. Cryptology ePrint Archive, Report 2002/020, 2002. `http://eprint.iacr.org/`.

[19] D. Coppersmith, H. Krawczyk, and Y Mansour. The shrinking generator. In *Advances in Cryptology—CRYPTO'93*, volume LNCS 547, pages 22–39. Springer-Verlag, 1993.

[20] D. Coppersmith and P. Rogaway. Software-efficient pseudorandom function and the use thereof for encryption. U.S. Patent $\sharp 5,454,039$, 1995.

[21] N. Courtois, M. Finiasz, and N. Sendrier. How to achieve a McEliece-based digital signature scheme. In *Advances in Cryptology—ASIACRYPT 2001*, volume LNCS 2248, pages 157–174. Springer-Verlag, 2001.

[22] T. M. Cover and J. A. Thomas. *Elements of Information Theory.* John Wiley and Sons, New York, 1991.

[23] E. Dawson, A. Clark, Golić J., W. Millan, L. Penna, and L. Simpson. The LILI-128 keystream generator. `http://www.cryptonessie.org/`, 2000. Proceedings of first NESSIE Workshop.

[24] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Trans. on Inform. Theory*, IT-22:644–654, 1976.

[25] P. Ekdahl and T. Johansson. SNOW—a new stream cipher. In *Proceedings of First Open NESSIE Workshop,KU-Leuven*, 2000.

[26] P. Ekdahl and T. Johansson. Another attack on A5/1. In *Proceedings of 2001 IEEE Intern. Symposium on Inform. Theory*, page 160, 2001.

[27] P. Ekdahl and T. Johansson. Distinguishing attacks on SOBER-t16 and t32. In *Fast Software Encryption 2002*, To be published in LNCS.

[28] M.P.C. Fossorier, M.J. Mihaljević, and H. Imai. Critical noise for convergence of iterative probabilistic decoding with belief propagation in cryptographic applications. In *Applied Algebra, Algebraic Algorithms and Error Correcting Codes—AAECC 13*, volume LNCS 1719, pages 282–293. Springer-Verlag, 1999.

[29] R. G. Gallager. *Low-Density Parity-Check Codes.* MIT Press, Cambridge, Mass., 1963.

[30] R. G. Gallager. *Information Theory and Reliable Communication.* John Wiley and Sons, New York, 1968.

[31] O. Goldreich and L. A. Levin. A hard-core predicate for all one-way functions. In *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*, pages 25–32, 1989.

[32] O. Goldreich, R. Rubinfeld, and M. Sudan. Learning polynomials with queries: The highly noisy case. In *36th Annual Symposium on Foundation of Computer Science*, pages 294–303, 1995.

[33] J. Dj. Golić. On the security of nonlinear filter generators. In *Fast Software Encryption'96*, volume LNCS 1039, pages 173–188. Springer-Verlag, 1996.

[34] J.Dj. Golić, M. Salmasizadeh, and E. Dawson. Fast correlation attacks on the summation generator. *Journal of Cryptology*, Vol. 13:245–262, 2000.

[35] S.W. Golomb. *Shift Register Sequences.* Holden-Day, San Fransisco, 1997.

[36] C.G. Günther. Alternating step generators controlled by de Bruijn sequences. In *Advances in Cryptology—EUROCRYPT'87*, volume LNCS 304, pages 91–103. Springer-Verlag, 1988.

[37] X. Guo-Zhen and J.L. Massey. A spectral characterization of correlation-immune combining functions. *IEEE Trans. on Inform. Theory*, IT-34:569–571, 1988.

[38] J. Hagenauer, E. Offer, and L. Papke. Iterative decoding of binary block and convolutional codes. *IEEE Trans. on Inform. Theory*, IT-42:429–445, March 1996.

[39] T. Jakobsen. *Higher-Order Cryptanalysis of Block Ciphers.* Ph. D. thesis, Technical University of Denmark, Denmark, 1999.

[40] R. Johannesson and K. Sh. Zigangirov. *Fundamentals of Convolutional Coding.* IEEE Press, Piscataway, N.J., 1999.

[41] T. Johansson. Reduced complexity correlation attacks on two clock-controlled generators. In *Advances in Cryptology—ASIACRYPT'98*, volume LNCS 1514, pages 342–356. Springer-Verlag, 1998.

[42] T. Johansson and F. Jönsson. On the complexity of some cryptographic problems based on the general decoding problem. In *Proceedings of 1998 IEEE Intern. Symposium on Inform. Theory*, page 442, 1998.

[43] T. Johansson and F. Jönsson. Fast correlation attacks based on turbo code techniques. In *Advances in Cryptology—CRYPTO'99*, pages 181–197. Springer-Verlag, 1999.

[44] T. Johansson and F. Jönsson. Improved fast correlation attacks on stream ciphers via convolutional codes. In *Advances in Cryptology—EUROCRYPT'99*, pages 347–362. Springer-Verlag, 1999.

[45] T. Johansson and F. Jönsson. Fast correlation attacks through reconstruction of linear polynomials. In *Advances in Cryptology—CRYPTO 2000*, pages 300–315. Springer-Verlag, 2000.

[46] T. Johansson and F. Jönsson. On the complexity of some cryptographic problems based on the general decoding problem. *IEEE Trans. on Inform. Theory*, to appear.

[47] F. Jönsson and T. Johansson. A fast correlation attack on LILI-128. *Information Processing Letters*, vol. 81:127–132, 2002.

[48] D. Kahn. *The Codebreakers: The story of Secret Writing.* Macmillan Publishing, New York, 1967.

[49] M. Kearns. Efficient noise-tolerant learning from statistical queries. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*, pages 392–401, 1993.

[50] J. Kemeny and J. Snell. *Finite Markov Chains.* Springer-Verlag, 1960.

[51] A. Klapper and M. Goresky. 2-adic shift registers. In *Fast Software Encryption, Cambridge Security Workshop*, volume LNCS 809, pages 174–178. Springer-Verlag, 1994.

[52] A. Klapper and M. Goresky. Feedback shift registers, combiners with memory, and 2-adic span. *Journal of Cryptology*, Vol. 10:111–147, 1997.

[53] L.R. Knudsen, W. Meier, B. Preneel, V. Rijmen, and S. Verdoolaege. Analysis methods for (alleged) RC4. In *Advances in Cryptology—ASIACRYPT'98*, volume LNCS 1514, pages 327–341. Springer-Verlag, 1998.

[54] D.E. Knuth. *The Art of Computer Programming — Seminumerical Algorithms.* Addison-Wesley, Massachusets, 2nd edition, 1981.

[55] P. J. Lee and E. F. Brickell. An observation on the security of McEliece's public-key cryptosystem. In *Advances in Cryptology—EUROCRYPT'88*, volume LNCS 330, pages 275–280. Springer-Verlag, 1988.

[56] J. S. Leon. A probabilistic algorithm for computing minimum weights of large error-correcting codes. *IEEE Trans. on Inform. Theory*, IT-34:1354–1359, September 1988.

[57] F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error-Correcting Codes.* North-Holland, Amsterdam, 1977.

[58] I. Mantin and A. Shamir. Practical attack on broadcast RC4. In *Fast Software Encryption 2001*, To be published in LNCS.

[59] J. L. Massey. Shift-register synthesis and BCH decoding. *IEEE Trans. on Inform. Theory*, IT-15:122–127, January 1969.

[60] R. J. McEliece. A public-key cryptosystem based on algebraic coding theory. *DSN Progress Report 42–44*, pages 114–116, 1978.

[61] W. Meier and O. Staffelbach. Fast correlation attacks on stream ciphers. In *Advances in Cryptology—EUROCRYPT'88*, volume LNCS 330, pages 301–314. Springer-Verlag, 1988.

[62] W. Meier and O. Staffelbach. Fast correlation attacks on certain stream ciphers. *Journal of Cryptology*, Vol. 1:159–176, 1989.

[63] W. Meier and O. Staffelbach. Correlation properties of combiners with memory in stream ciphers. *Journal of Cryptology*, Vol. 5:67–86, 1992.

[64] W. Meier and O. Staffelbach. The self-shrinking generator. In *Advances in Cryptology—EUROCRYPT'94*, volume LNCS 950, pages 205–214. Springer-Verlag, 1994.

[65] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography.* CRC Press, Boca Raton, 1997.

[66] S. Micali and C.P. Schnorr. Efficient, perfect polynomial random number generators. *Journal of Cryptology*, Vol. 3:157–172, 1991.

[67] M. Mihaljevic, M. Fossorier, and H. Imai. A low-complexity and high-performance algorithm for the fast correlation attack. In *Fast Software Encryption 2000*, volume LNCS 1978, pages 196–212. Springer-Verlag, 2000.

[68] M. Mihaljevic, M. Fossorier, and H. Imai. An algorithm for cryptanalysis of certain keystream generators suitable for high-speed software and hardware implementations. *IEICE Trans. Fundamentals*, vol. E84-A(no.1):311–318, January 2001.

[69] M. Mihaljevic, M. Fossorier, and H. Imai. On decoding techniques for cryptanalysis of certain encryption algorithms. *IEICE Trans. Fundamentals*, vol. E84-A(no.4):919–930, April 2001.

[70] M. Mihaljevic, M. Fossorier, and H. Imai. Fast correlation attack algorithm with list decoding and an application. In *Fast Software Encryption 2001*, To be published in LNCS.

[71] M. Mihaljevic and J. Golić. A fast iterative algorithm for a shift register initial state reconstruction given the noisy output sequence. In *Advances in Cryptology—AUSCRYPT'90*, volume LNCS 453, pages 165–175. Springer-Verlag, 1990.

[72] M. Mihaljevic and J. Dj. Golic. A method for convergence analysis of iterative probabilistic decoding. *IEEE Trans. on Inform. Theory*, vol. 46:2206–2211, 2000.

[73] H. Niederreiter. Knapsack-type cryptosystems and algebraic coding theory. *Probl. Controll and Information Theory*, vol. 15:159–166, 1986.

[74] W. Penzhorn. Correlation attacks on stream ciphers: Computing low weight parity checks based on error correcting codes. In *Fast Software Encryption'96*, volume LNCS 1039, pages 159–172. Springer-Verlag, 1996.

[75] J.G. Proakis. *Digital Communications*. McGraw-Hill, 3rd edition, 1995.

[76] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public key cryptosystems. *Communications of the ACM*, vol. 21:120–126, 1978.

[77] G. Rose and P. Hawkes. The t-class of SOBER stream ciphers. `http://www.home.aone.net.au/qualcomm`.

[78] B. Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. John Wiley and Sons, New York, 2nd edition, 1996.

[79] R. Schroeppel and A. Shamir. A $T = O(2^{n/2})$, $S = O(2^{n/4})$ algorithm for certain NP-complete problems. *SIAM Journal of Computing*, vol. 10:456–464, 1981.

[80] C. E. Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, Vol. 27:656–715, 1949.

[81] T. Siegenthaler. Correlation-immunity of nonlinear combining functions for cryptographic applications. *IEEE Trans. on Inform. Theory*, IT-30:776–780, 1984.

[82] T. Siegenthaler. Decrypting a class of stream ciphers using ciphertext only. *IEEE Trans. on Computers*, C-34:81–85, 1985.

[83] BLUETOOTH SIG. Bluetooth specification. `www.bluetooth.com`.

[84] L. SIMPSON. *Divide and Conquer Attacks on Shift Register Based Stream Ciphers.* Ph.D. Thesis, Queensland University of Technology, Australia, 1999.

[85] L. SIMPSON, E. DAWSON, J. GOLIC, AND M. SALMASIZADEH. Fast correlation attacks on the multiplexer generator. In *Proceedings of 1998 IEEE Intern. Symposium on Inform. Theory*, page 270, 1998.

[86] L. SIMPSON, E. DAWSON, GOLIĆ J., AND W. MILLAN. LILI keystream generator. In *Proceedings of the Seventh AnnualWorkshop on Selected Areas in Cryptology—SAC'2000*, volume LNCS 2012, pages 248–261. Springer-Verlag, 2000.

[87] S. SINGH. *The Code Book.* Fourth Estate, London, 1999.

[88] J. STERN. A method for finding codewords of small weight. In *Coding Theory and Applications*, volume LNCS 388, pages 106–113. Springer-Verlag, 1989.

[89] J. STERN. A new identification scheme based on syndrome decoding. In *Advances in Cryptology—CRYPTO'93*, volume LNCS 388, pages 13–21. Springer-Verlag, 1994.

[90] D. R. STINSON. *Cryptography: Theory and Practise.* CRC Press, Boca Raton, 1995.

[91] M. SUDAN. Decoding of Reed Solomon codes beyond the error-correction bound. *Journal of Complexity*, vol. 13(1):180–193, March 1997.

[92] A. TROFIMOV AND K. SH. ZIGANGIROV. A posteriori probability decoding of convolutional codes. *Problems of Information Transmission*, vol. 4:351–358, 1999.

[93] H. C. A. VAN TILBORG. Coding theory at work in cryptology and vice versa. In *Handbook of Coding Theory*, volume vol. II, pages 1195–1227. North-Holland, 1998.

[94] J. VAN TILBURG. *Security-Analysis of a Class of Cryptosystems Based on Linear Error-Correcting Codes.* Ph.D. Thesis, Technical University Eindhoven, The Netherlands, 1994.

[95] A. VARDY. The intractability of computing the minimum distance of a code. *IEEE Trans. on Inform. Theory*, IT-43:1757–1766, 1997.

[96] P. Véron. *Opérateur trace, Schémas D'identification et Codes de Goppa.* Ph.D. Thesis, University de Toulon et du Var, France, 1995.

[97] K. Zeng, C. H. Yang, and T. R. N. Rao. On the linear consistency test (LCT) in cryptanalysis. In *Advances in Cryptology—CRYPTO'89*, volume LNCS 435, pages 164–174. Springer-Verlag, 1990.

[98] K. Zeng, C. H. Yang, and T. R. N. Rao. An improved linear syndrome algorithm in cryptanalysis with applications. In *Advances in Cryptology—CRYPTO'90*, volume LNCS 537, pages 34–47. Springer-Verlag, 1991.