New cryptanalysis of LFSR-based stream ciphers and decoders for p-ary QC-MDPC codes

Isaac Andrés Canales Martínez

Thesis for the degree of Philosophiae Doctor (PhD)
University of Bergen, Norway
2022

UNIVERSITY OF BERGEN

New cryptanalysis of LFSR-based stream ciphers and decoders for p-ary QC-MDPC codes

Isaac Andrés Canales Martínez

Thesis for the degree of Philosophiae Doctor (PhD)
at the University of Bergen

Date of defense: 18.03.2022

Year: 2022

Title: New cryptanalysis of LFSR-based stream ciphers and decoders for p-ary QC-MDPC codes

Name: Isaac Andrés Canales Martínez

Print: Skipnes Kommunikasjon / University of Bergen

Acknowledgments

Some of the computations were performed on resources provided by UNINETT Sigma2 - the National Infrastructure for High Performance Computing and Data Storage in Norway.

Abstract

The security of modern cryptography is based on the hardness of solving certain problems. In this context, a problem is considered *hard* if there is no known polynomial time algorithm to solve it. Initially, the security assessment of cryptographic systems only considered adversaries with *classical* computational resources, i.e., digital computers. It is now known that there exist polynomial-time *quantum* algorithms that would render certain cryptosystems insecure if large-scale quantum computers were available. Thus, adversaries with access to such computers should also be considered. In particular, cryptosystems based on the hardness of integer factorisation or the discrete logarithm problem would be broken. For some others such as symmetric-key cryptosystems, the impact seems not to be as serious; it is recommended to at least double the key size of currently used systems to preserve their security level. The potential threat posed by sufficiently powerful quantum computers motivates the continued study and development of *post-quantum cryptography*, that is, cryptographic systems that are secure against adversaries with access to quantum computations.

It is believed that symmetric-key cryptosystems should be secure from quantum attacks. In this manuscript, we study the security of one such family of systems; namely, stream ciphers. They are mainly used in applications where high throughput is required in software or low resource usage is required in hardware. Our focus is on the cryptanalysis of stream ciphers employing linear feedback shift registers (LFSRs). This is modelled as the problem of finding solutions to systems of linear equations with associated probability distributions on the set of right hand sides. To solve this problem, we first present a multivariate version of the correlation attack introduced by Siegenthaler. Building on the ideas of the multivariate attack, we propose a new cryptanalytic method with lower time complexity. Alongside this, we introduce the notion of *relations modulo a matrix* B, which may be seen as a generalisation of parity-checks used in fast correlation attacks. The latter are among the most important class of attacks against LFSR-based stream ciphers. Our new method is successfully applied to hard instances of the filter generator and requires a lower amount of keystream compared to other attacks in the literature. We also perform a theoretical attack against the Grain-v1 cipher and an experimental attack against a toy Grain-like cipher. Compared to the best previous attack, our technique requires less keystream bits but also has a higher time complexity. This is the result of joint work with Semaev.

Public-key cryptosystems based on error-correcting codes are also believed to be secure against quantum attacks. To this end, we develop a new technique in code-based cryptography. Specifically, we propose new decoders for quasi-cyclic moderate density parity-check (QC-MDPC) codes. These codes were proposed by Misoczki et al. for use in the McEliece scheme. The use of QC-MDPC codes avoids attacks applicable when using low-density parity-check (LDPC) codes and also allows for keys with short size. Although we focus on decoding for a particular instance of the p-ary QC-MDPC scheme, our new decoding algorithm is also a general decoding method for p-ary MDPC-like schemes. This algorithm is a bit-flipping decoder, and its performance is improved by varying thresholds for the different iterations. Experimental results demonstrate that our decoders enjoy a very low decoding failure rate for the chosen p-ary QC-MDPC instance. This is the result of joint work with Guo and Johansson.

Contents

List of Figures

List of Tables

List of Algorithms

CHAPTER 1

Introduction

Communication is one of the fundamental processes within human civilization. The discovery of rudimentary forms of it, such as paintings in caves, shows that humanity has tried to share information since ancient times. Communication can occur in different ways, for example, it can be verbal, non-verbal or written. The development of language is perhaps among the most important events in human history. Having a mutually understood set of rules for communication (e.g., symbols or sounds) facilitates the exchange of information between sender and receiver.

Information may have different levels of importance. We might consider some information irrelevant and not care about who has access to it. Some other information could be valuable, we may even invest resources in ensuring that it is securely transmitted and stored. The meaning of security can vary depending on the context. For instance, we may have a situation in which security means that the information is accessible only to the entitled entities. In another situation, security may mean to ensure that the transmitted information is actually being sent by the real sender and not by someone else. Information security can be defined in terms of security services like:

- Confidentiality: the information is protected from unauthorised access or disclosure.
- Authentication: ensure that an entity is indeed who it claims to be.
- Integrity: ensure that the information is not altered or destroyed in an unauthorised manner.
- Non-repudiation: avoid that an entity involved in the communication process denies having participated in it.
- Availability: ensure that the information is accessible and usable by an entitled entity.

This list is by no means exhaustive. Actually, important efforts have been made in defining these and other security services [Tec00; Stu91].

As human society evolves, the means of communication also evolve. Perhaps the most striking change occurred during the 20th century with the development of digital communication. Given the ease of access to devices like digital computers and smartphones, we are able to communicate practically at all times. The current status of digital communication allows us to enjoy services tailored to our personal needs and preferences, and in some cases has drastically reduced, even replaced, human interaction. Historically, it was believed that only governments and big organisations were concerned about the security of their information. Nowadays, however, everyone communicating through a public insecure communication channel (e.g., the internet) can be the target of an attack. Never before has the need to securely exchange and store sensitive information been more required.

Cryptography can be defined as the study of mathematical techniques for securing digital information, systems, and distributed computations against adversarial attacks [KL14]. Some security services can be attained by only employing basic cryptographic primitives or tools, like *cryptosystems*, *signature schemes* and *hash functions*. These tools can also be used in more complex *cryptographic protocols*, which are communication protocols to perform a security-related function. In this manuscript, we will focus on certain cryptosystems and attacks against them.

1.1 Cryptosystems

Cryptosystems are cryptographic tools used mainly to achieve confidentiality. Formally, a cryptosystem can be defined [Buc04] as a tuple $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ such that

1. $\mathcal{P}$ is a set called *plaintext space* and its elements are called *plaintexts*;
2. $\mathcal{C}$ is a set called *ciphertext space* and its elements are called *ciphertexts*;
3. $\mathcal{K}$ is a set called *key space* and its elements are called *keys*;
4. $\mathcal{E} = \{E_k : k \in \mathcal{K}\}$ is a family of functions $E_k : \mathcal{P} \to \mathcal{C}$ and its elements are called *encryption functions*;
5. $\mathcal{D} = \{D_k : k \in \mathcal{K}\}$ is a family of functions $D_k : \mathcal{C} \to \mathcal{P}$ and its elements are called *decryption functions*, and
6. For each $k_e \in \mathcal{K}$, there is $k_d \in \mathcal{K}$ such that $D_{k_d}(E_{k_e}(p)) = p$ for all $p \in \mathcal{P}$.

As it is usual in the literature, our friends Alice and Bob will yet again establish communication through an insecure channel. Alice wants to send a message m, the plaintext, to Bob. She does not want anyone but Bob to get the information contained in the message. To achieve this, Alice *encrypts* the message m using the *encryption key* k_e, i.e., she gets the ciphertext $c = E_{k_e}(m)$. Alice sends c to Bob through the insecure channel. Upon receiving c, Bob *decrypts* c using the *decryption key* k_d to recover the original message as $m = D_{d_k}(c)$. This interaction is depicted in Figure 1.1.

Cryptosystems can be classified as *symmetric*, or *private-key*, cryptosystems and *asymmetric*, or *public-key*, cryptosystems. In the first type, the key used for encryption is the same as that used for decryption, i.e., $k_e = k_d$. When a symmetric cryptosystem

Figure 1.1. Usage of a cryptosystem to communicate through an insecure channel.

is to be used, the communicating parties must share the secret key before they start sending messages. Once the key has been established, it must be kept secret, since anyone possessing it will be able to decrypt the transmitted messages. Asymmetric cryptosystems, on the other hand, make use of different keys for encryption and decryption. Diffie and Hellman introduced the idea of public-key cryptography in their seminal paper [DH76] of 1976. In these cryptosystems, the encryption (or public) key k_e is made public and the decryption (or secret) key k_d must be kept secret. It is required that obtaining k_d from k_e is infeasible. Encryption can be done in principle by everybody (since the encryption key is public) and only the owner of the private key is able to decrypt the messages.

Securely exchanging the secret key is an important problem when using private-key cryptography. However, this problem can be solved easily employing public-key cryptography. The communicating parties can use a public-key cryptosystem within a *key exchange protocol* (Diffie-Hellman protocol [DH76]) to agree on a shared secret, the key for a private-key cryptosystem. Alternatively, the key can be encrypted with a public-key cryptosystem and then transmitted. This process is called *hybrid cryptography*.

In general, public-key cryptosystems are slower in performing the encryption and decryption operations compared to private-key cryptosystems. It is then advisable to use public-key cryptosystems to exchange short messages and private-key cryptosystems to exchange long messages.

1.2 Attacks against cryptosystems and security

In order to define what a successful attack against a cryptosystem is (or other cryptographic primitives and protocols), we need to specify the goal of the attacker. One natural goal is to recover the secret key. This is one of the strongest goals to achieve (perhaps the strongest one), since the attacker would then be able to decrypt all further communication. Another goal might be to fully or partially recover the plaintext corresponding to a given ciphertext. It might also be to distinguish whether a given string of symbols is a ciphertext obtained with a cryptosystem or a randomly generated string.

It is also important to specify the power or capabilities the attacker has when performing the attack. Here, it is common to follow the idea behind one of *Kerckhoffs's Principles* [Ker83]: the security of a cryptosystem must lie in the choice of its keys only, everything else (including the encryption and decryption functions) should be considered public information. There are several *threat* or *attack models* that capture the capabilities of the attacker. Many textbooks and monographs on cryptography provide detailed explanations; see for example [KL14; SP17; Buc04]. Here we mention some of them, in increasing order of power for the attacker, and give a brief explanation:

- *Ciphertext-only attack*. In this model, the attacker has access only to a number of ciphertexts.

- *Known-plaintext attack*. The attacker has access to plaintexts and their corresponding ciphertexts.

- *Chosen-plaintext attack*. As in the previous model, the attacker can obtain plaintext/ciphertext pairs, but the plaintexts are chosen by the attacker. (In a public-key cryptosystem such attacks are always possible since the encryption key is public.)

- *Chosen-ciphertext attack*. The attacker can choose ciphertexts and get the corresponding plaintexts.

In all cases, the same secret key is used for all encryptions and decryptions. In the last three cases, the attacker must obtain information on a plaintext different from the ones already known and the ones corresponding to the ciphertexts used while performing the attack. Also, in a given model, the attacker has the capabilities of the previous weaker models, e.g., a chosen-plaintext attack implies ciphertext-only and known-plaintext attacks.

Once the attacker's goal and capabilities are defined, the security of a cryptosystem can be analysed. If the attacker is able to fulfil its goal given the attack model, we may consider the cryptosystem to be "broken" under this attack. Otherwise, we may consider the cryptosystem to be secure.

The security of a cryptosystem may be supported by rigorous formal proofs which show that the cryptosystem satisfies a given definition under certain clearly specified assumptions. When using this approach to proving security, we say that the cryptosystem is *provably secure*. A proof of security is always relative to the considered definitions and assumptions made. The proof may be irrelevant if the definitions and/or assumptions are incorrect, or if the model does not match the adversary's real capabilities. Another approach to proving security is *computational security*. Here, the idea is to show that, currently, it is computationally infeasible to break the system, i.e., the attacker cannot break the cryptosystem in a reasonable amount of time using a reasonable amount of computational resources. However, the drawback of this approach is that cryptosystems considered computationally secure now might become insecure in the future. Regardless of the approach, the security of modern cryptosystems relies on the assumption that a problem is *hard* to solve. In this context, hard means that there is no known algorithm to solve the problem in question within reasonable (e.g., *polynomial*) time.

The security notions above do not necessarily imply security in the real world. For instance, cryptographic implementations in hardware and software may introduce vulnerabilities that the models cannot capture. Attacks against implementations which take advantage of these vulnerabilities are known as *side channel attacks*. They exploit information that can be gathered from the target device, such as power consumption, timing information, electromagnetic information and even sound. Examples of these are timing attacks, fault attacks, power analysis attacks and cache attacks.

1.3 Some private-key primitives

Private-key cryptography encompasses different primitives like *stream ciphers*, *block ciphers* and *hash functions*, among others. Here, we give a brief presentation of stream and block ciphers. The main topic of this manuscript is on a particular type of stream ciphers. We briefly present block ciphers due to their importance in practical cryptographic applications.

1.3.1 Stream ciphers

A *stream cipher* generates a *pseudorandom* sequence of symbols called *keystream* or *running key*. Informally, a pseudorandom sequence is a sequence that is difficult to distinguish from a true random sequence. The keystream is produced by the cipher's *keystream generator* whose initial *state* is determined by the secret key (and possibly some additional parameters like an *initialisation vector*). The state of the generator is updated constantly in order to produce the keystream symbols. Stream ciphers can be classified according to how the state of the generator is updated. In a *synchronous* stream cipher, the keystream is generated independently of the plaintext and ciphertext. An *asynchronous* or *self-synchronising* stream cipher, on the other hand, employs some symbols from the ciphertext to produce the keystream.

In order to encrypt a message, the keystream is "added" symbol by symbol to the plaintext to produce the ciphertext. Decryption is achieved by generating the same keystream and "subtracting" it from the ciphertext. When the symbols are bits, the addition and subtraction operations correspond to bitwise XOR. Figure 1.2 shows, in a general level, encryption and decryption using a synchronous stream cipher.

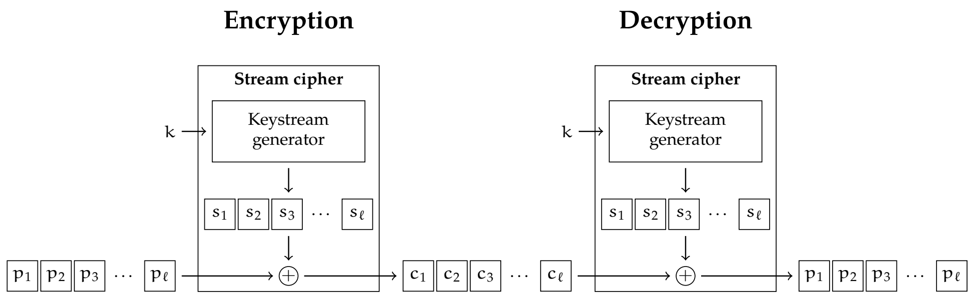


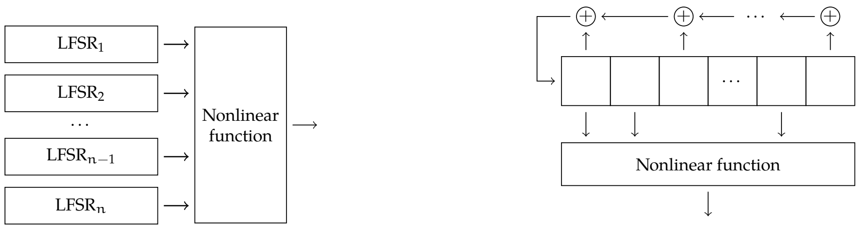
Figure 1.2. Encryption and decryption using a synchronous stream cipher with the secret key k.

Linear feedback shift registers (LFSRs) have been extensively used in the design of stream ciphers. An LFSR can be seen as an array of n cells along with a linear feedback loop involving some of the cells. The content of the array is called the *state* of the LFSR. Without loss of generality, we will assume the feedback loop is connected to the left-most cell. The state is updated at each "clock tick" by shifting the contents of the cells to the right and the new value of the left-most cell is computed by the feedback loop. At each clock tick, the LFSR outputs the value of the right-most cell, thus producing an output sequence. Figure 1.3a depicts a model of an LFSR. These devices are popular due to ease and efficiency of implementation and the good statistical properties of the generated sequence. Due to the linearity of the feedback, however, LFSRs are not used directly to produce the keystream of a stream cipher. Some nonlinearity must be added for this purpose.

(**a**) Linear feedback shift register. (**b**) Nonlinear feedback shift register.

Figure 1.3. Models of feedback shift registers used in the design of stream ciphers.

One way to add nonlinearity is by employing a *nonlinear feedback shift register* (NFSR). NFSRs are similar to LFSRs except that, as the name implies, the feedback is a nonlinear function on the cells of the array. Figure 1.3b depicts a model of an NFSR. Another option to add nonlinearity is to "combine" the output sequences of several LFSRs using a nonlinear function. This construction is known as *combination generator* and the nonlinear function is also called the combining function. An alternative is to use a nonlinear function that takes as input the values of some cells from an LFSR and the output sequence is then given by the output of the function. This construction is known as *filter generator* and the nonlinear function is also called the filtering function. Figures 1.4a and 1.4b show models of a combination and filter generator, respectively.



(**a**) Combination generator. (**b**) Filter generator.

Figure 1.4. Some keystream generator constructions employing LFSRs.

Modern stream ciphers combine LFSRs, NFSRs, the constructions above and other elements in different ways to generate the keystream. Nowadays, the use of stream ciphers is much lower compared to block ciphers. The latter may work as stream ciphers by employing certain *modes of operations*. However, dedicated stream ciphers are still needed when particularly high throughput is required in software or exceptionally low resource usage is required in hardware.

1.3.2 Block ciphers

A *block cipher* performs encryption and decryption in blocks of symbols. Let n be the length of the blocks. The cipher specifies an encryption algorithm that uses the secret key (and some additional parameters like an initialisation vector) to compute the length-n ciphertext from a given length-n plaintext. The cipher also specifies the decryption algorithm to recover the plaintext corresponding to the given ciphertext and secret key (and additional required parameters). The whole plaintext is divided in blocks of length n and encrypted block by block according to a *mode of operation* (see for example [SP17; KL14; NIS] for more details). If the length of the plaintext is not a multiple of n, then it is *padded* (i.e., data is added to the plaintext) so that all blocks have length n. Figure 1.5 shows, in a general level, encryption and decryption using a block cipher.

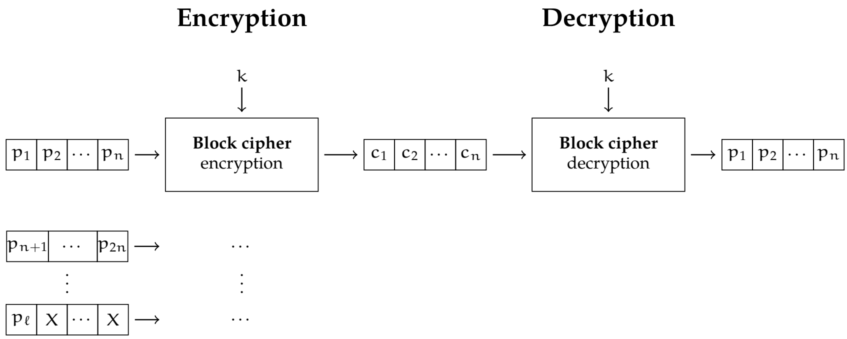


Figure 1.5. Encryption and decryption using a block cipher with the secret key k. The plaintext is divided in blocks of length n; padding is required when the length ℓ of the plaintext is not a multiple of n. The mode of operation will dictate the encryption and decryption process when the amount of data is larger than one block.

It is important for the security of a block cipher that every bit in the input affects many bits in the output, ideally every bit. A technique towards achieving this is the *confusion-diffusion* approach. The idea is to have two simple layers of operations that shuffle and mix the input data. One layer corresponds to the confusion part and the other to the diffusion part of the process. Applying these two layers together corresponds to what is called a *round*. Several rounds are applied iteratively, thus helping ensure that one bit of the input affects many output bits.

A *Substitution-permutation network* (SPN) is a practical construction that follows the confusion-diffusion approach. An SPN makes use of a set of fixed permutations called *S-boxes* and their outputs are mixed according to a given *mixing permutation*. Each round in an SPN takes as input a block of data and its own round key. All round keys are derived from the secret key, also called master key in this context. At a high level, the input block is mixed with the corresponding round key, then the S-boxes and mixing permutation are applied to produce the output block. The output of a round is fed as the input block to the next round along with its corresponding round key. It is customary that the output of the final round undergoes a final mixing step to produce the final output of the SPN. Given the secret key, any SPN is invertible. Figure 1.6 shows the high level structure of the first two rounds of an SPN.

Feistel networks are another approach for designing block ciphers. Compared to an SPN, a Feistel network may use functions that are not invertible. This characteristic allows the cipher to have "less structure" compared to the inherent structure an SPN has due to the usage of invertible components only. S-boxes and mixing permutations may be used as well, however, any type of functions can be employed. A Feistel network is also composed of rounds and makes use of round keys derived from the secret (master) key. The length-n input to the i-th round is divided into two halves, denoted

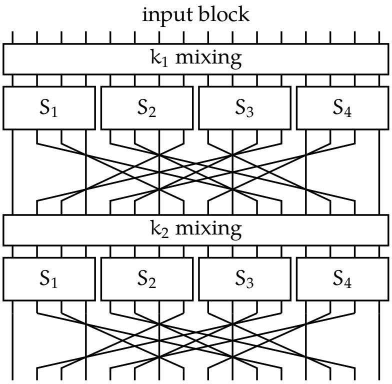


Figure 1.6. Model of a substitution-permutation network.

L_{i-1} and R_{i-1}. The output (L_i, R_i) of that round is given by

$$L_i = R_{i-1} \quad \text{and} \quad R_i = L_{i-1} \oplus f_i(k_i, R_{i-1}),$$

where k_i is the round key for the i-th round. Figure 1.7 depicts the high level overview of the first three rounds of a Feistel network.

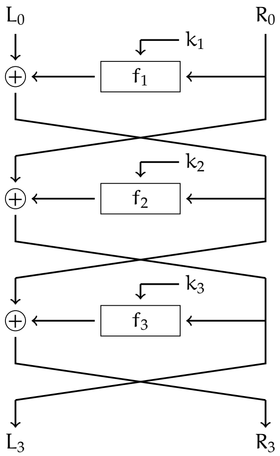


Figure 1.7. Model of a Feistel network.

1.4 Public-key cryptosystems

As mentioned before, public-key cryptosystems make use of different keys, a public encryption key k_e and a secret decryption key k_d. It is required that obtaining k_d from k_e is computationally infeasible. Additionally, the encryption function should be easy to compute, while the decryption function (i.e., its inverse) should be hard to compute for anyone not knowing the corresponding decryption key. A function that is easy to compute but hard to invert is called a *one-way function*. Even though decryption alone is hard, knowing the decryption key makes it possible to recover the plaintext from the ciphertext. The decryption key can then be considered as a *trapdoor*. A function that is one-way but becomes easy to invert with the knowledge of certain information is called a *trapdoor function*. The security of public-key cryptosystems is based on problems believed to be hard and conjectured one-way functions based on those problems.

1.4.1 Integer factorisation and discrete logarithm

The security of many public-key cryptosystems rely on problems from number theory that are believed to be hard. In this context, hard means that there are no known polynomial-time algorithms for solving them. Rivest, Shamir and Adleman created one of the best known public-key cryptosystems, the RSA cryptosystem [RSA78], whose security is based on the hardness of factoring large integers. Another cryptosystem based on integer factorisation is the Rabin cryptosystem [Rab79]. The ElGamal cryptosystem [ElG85a; ElG85b] and many elliptic curve cryptosystems base their security on the difficulty of the discrete logarithm problem.

The best known algorithms for solving the problems above are nonpolynomial in time. However, they are better than brute force and must be considered when assessing the security of cryptosystem relying on those problems. Among the algorithms for integer factorisation, we can mention Pollard's $p-1$ algorithm, Pollard's rho algorithm and the number field sieve. For the discrete logarithm, we have the Pohlig-Hellman algorithm, the baby-step/giant-step algorithm and the index calculus algorithm, among others. For further details on these algorithms, we refer to the existing literature, for example [KL14; SP17; Buc04].

1.4.2 Post-quantum public-key cryptosystems

The algorithms in Section 1.4.1 belong to a class called *classical* algorithms since they are performed by conventional digital (or classical) computers. If large-scale quantum computers are built, they will be able to efficiently solve problems that are hard for classical computers. Particularly, they would solve the factorisation and discrete logarithm problems in polynomial time [Sho97]. Hence, many of the public-key cryptosystems currently in use would be broken. *Post-quantum* or *quantum-resistant* cryptography refers to cryptographic systems that are secure against attacks by both quantum and classical computers. The impact on the security of symmetric-key cryptography will not be as serious. Grover's quantum search algorithm [Gro96] provides a quadratic speed-up compared to search algorithms on classical computers. Doubling the key size would be sufficient to preserve security. Additionally, exponential speed up for search algorithms seems unfeasible, which indicates that symmetric-key cryptography is still serviceable in the post-quantum era [Ben+97].

The search for algorithms believed to be resistant against classical and quantum attacks has been mainly focused on public-key cryptography. We give a brief overview of the main classes of post-quantum cryptographic systems:

- Lattice-based cryptography. Cryptosystems in this class employ objects called *lattices*. These cryptosystems enjoy strong provable security proofs based on worst-case hardness, relatively efficient implementations and simplicity. Some efficient constructions for practical use, however, lack a supporting security proof.

- Code-based cryptography. These are cryptosystems in which the underlying one-way function uses an error-correcting code. While encryption and decryption are efficient, the main disadvantage of most code-based primitives is the very large key sizes.

- Multivariate polynomial cryptography. These schemes are based on the hardness of solving systems of multivariate polynomials over finite fields. Many multivariate cryptosystems have been proposed and several have been broken. Multivariate cryptography has been more successful for signature schemes.

- Isogeny-based cryptography. This class of cryptosystems employ isogenies on supersingular elliptic curves. Even though the discrete logarithm problem can be efficiently solved using a quantum computer, there is no known quantum attack for the isogeny problem on supersingular curves. One of the disadvantages is that there has not been enough analysis to have much confidence in their security.

- Hash-based signatures. Hash-based signatures are digital signatures constructed using hash functions. Their security relies on the collision resistance of the hash function.

The Post-Quantum Cryptography Standardization process, organised by the National Institute of Standards and Technology (NIST), is perhaps one of the most important efforts in post-quantum public-key cryptography. The goal of this process is to evaluate and standardise one or more quantum-resistant public-key cryptographic algorithms. At the time of writing this manuscript, the process is at the final third round, and according to the report from the previous round, NIST expects to select a small number of candidates from round three for standardisation by early 2022. Regarding encryption systems and key exchange, there are 3 lattice-based and 1 code-based proposals. Among the alternate finalists for encryption systems and key exchange, there are 2 lattice-based, 2 code-based and 1 isogeny-based proposals.

Overview

Chapter 2 provides the foundations for the remaining chapters. We give the relevant results and statements without proofs. Results on finite fields, polynomials over finite fields, probability and statistics are presented first. Then, we introduce relevant results on Boolean functions, LFSRs, LFSR sequences and the required background on coding theory.

The main topic of this manuscript is cryptanalysis of LFSRs-based stream ciphers. Particularly, we focus on key recovery attacks against the filter generator in Chapter 3. We first present this device with more detail. There are different classes of key recovery attacks targeting the filter generator. Fast correlation attacks are, perhaps, among the most important class of attacks. We describe the original idea and briefly present some subsequent variations. We also describe some deterministic attacks, which are interesting due to how these cryptanalytic techniques exploit the characteristics of the filter generator. The approach and techniques in algebraic attacks are different from the ones studied in this manuscript, however, we briefly present them due to their general relevance.

In Chapter 4, we present a new key recovery attack against the filter generator. This chapter is based on joint work with Semaev. First, we model the attack as a more general problem: finding the solution of multiple systems of linear equations with associated probability distributions on the set of solutions. A first attempt to solving this problem is the multivariate correlation attack. This can be seen as a generalisation of the original correlation attack by Siegenthaler [Sie85]. The drawback of the multivariate attack is its high time complexity. We then introduce a new method with lower time complexity, the test-and-extend algorithm. This novel algorithm requires (i) the computation of *relations modulo* B, where B is a matrix over a finite field, and (ii) a set of probability distributions induced by these relations. The relations can be seen as a generalisation of parity-checks used in fast correlation attacks. Different techniques for computing the distributions associated to these relations are presented. We apply our new algorithm to some hard instances of the filter generator and conclude the chapter showing a theoretical application against the Grain-v1 cipher [HJM07].

Chapter 5 is on new decoders for quasi-cyclic moderate density parity-check (QC-MDPC) codes over a finite field $\mathbb{F}_p$. This chapter is based on joint work with Guo and Johansson. We first provide the required background on p-ary MDPC schemes. Then, we present a bit-flipping decoding algorithm for a particular instance of these schemes. We improve the decoding failure rate of the algorithm by varying thresholds. We also introduce two techniques to obtain these thresholds. We then show our experimental results of applying the novel decoder to the chosen p-ary QC-MDPC instance.

The contributions of this work are:

- New methods for cryptanalysis of LFSR-based stream ciphers. Namely, the multivariate correlation attack and the test-and-extend algorithm.

- We introduce relations modulo a matrix B and two procedures to obtain them. Also, various techniques to compute the probability distributions induced by these relations are shown.

- New numerical and theoretical results on cryptanalysis of hard instances of the filter generator. Particularly, our new test-and-extend algorithm allows successful recovery of the LFSR's initial state using a low number of keystream bits (see Section 4.7.3) compared to published attacks.

- An improvement in the number of keystream bits required to recover the LFSR's initial state for Grain-v1 with a trade-off on time complexity. This is done with the multivariate correlation attack.

- Computation of linear approximations to Grain-v1 with higher correlation than that reported in [Tod+18].

- To the best of our knowledge, a new highly parallelisable method to compute the FFT of a large input vector.

- A novel decoding algorithm for the p-ary MDPC scheme. The basic idea is to vary the decision thresholds at each iteration.

- Two methods for obtaining the thresholds for the decoder. The first one uses a theoretical analysis analogous to the one done by Gallager [Gal62] for LDPC codes. The second is a heuristic approach and it yielded the best decoding results.

CHAPTER 2

Preliminaries

In this chapter, we present definitions and known results that are fundamental throughout this monograph. The results are presented without proofs. Section 2.1 introduces the relevant algebraic objects and some of their properties. We give the necessary background on probability and statistics in Section 2.2. Section 2.3 contains the definitions and results on Boolean functions. In Section 2.4, linear feedback shift registers and their sequences are discussed. Then, the basics of coding theory are presented in Section 2.5. The background for Chapter 5 on the McEliece cryptosystem and MDPC codes is in Section 5.1. We do not present that content here since it is not required in the other chapters. We refer the reader to the different sources throughout this chapter and Section 5.1 for further details and proofs.

2.1 Algebra

We will assume familiarity with basic algebraic structures and maps between them. Particularly, we assume background on groups, rings and polynomials. However, here we state relevant definitions and results for the remaining sections and chapters. We refer to the existing literature, e.g. [LN96; Lan02], for a thorough treatment of the different topics covered here.

2.1.1 Definitions

Let $(R, +, \cdot)$ be a ring. We will refer to the operations $+$ and $\cdot$ as addition and multiplication, respectively, and we will simply use R to denote $(R, +, \cdot)$. Let $a, b \in R$. The additive inverse of a will be denoted $-a$, $b + (-a)$ will be written $b - a$, and $a \cdot b$ will be written ab.

A ring R is called *commutative* if commutativity holds for multiplication, i.e., $ab = ba$ for all $a, b \in R$. A ring R is called *ring with identity* if it has an identity with respect to multiplication, i.e., there is an element e such that $ae = ea = a$ for all $a \in R$.

Unless otherwise stated, a ring will be a commutative ring with identity. In the rest, R will denote a ring. We use 0 to represent the *zero element* of a ring, i.e., its identity element with respect to addition. The (multiplicative) identity will be denoted by 1. In general, 1 might be equal to 0 in R; in that case, R is the *zero ring* and it contains only the zero element. We will assume that $1 \neq 0$.

Definition 2.1.1. A *subring* of R is a subset S of R that is itself a ring under $+$ and $\cdot$.

Definition 2.1.2. An *ideal* of R is a subset I of R that is a subring of R and for all $a \in I$ and $r \in R$, $ar \in I$.

Definition 2.1.3. The smallest ideal of R containing an element $a \in R$ is the ideal $(a) = \{ar : r \in R\}$. It is the ideal *generated by* a. If an ideal I of R is generated by one element, I is called a *principal* ideal.

Definition 2.1.4. Let I be an ideal of R. The *quotient ring* of R modulo I, denoted by R/I, is the ring with sum and multiplication given by $(a + I) + (b + I) = (a + b) + I$ and $(a + I)(b + I) = (ab) + I$, respectively.

Definition 2.1.5. If there exists a positive integer n such that $nr = 0$ for every $r \in R$, then the least such positive integer n is called the *characteristic* of R. If no such integer n exists, R has characteristic 0.

A *polynomial* over R is an expression of the form

$$a_0 + a_1x^1 + \cdots + a_nx^n,$$

where n is a nonnegative integer, $a_i \in R$ and x is a symbol not belonging to R, called the *indeterminate*. The arithmetic of polynomials over a ring R is analogous to that of (the more familiar) polynomials with real or complex coefficients; see for example [LN96] for precise definitions. The *zero polynomial*, denoted by 0, is the polynomial whose coefficients are all equal to 0. Let $f(x) = \sum_{i=0}^{n} a_ix^i$ be a polynomial over R that is not the zero polynomial. Then, $a_n \neq 0$ is called the *leading coefficient*, a_0 the *constant term* and n the degree of $f(x)$. If $f(x)$ is the zero polynomial, its degree is $-\infty$. Polynomials of degree $\leqslant 0$ are called *constant polynomials*. A *monic polynomial* is a polynomial with leading coefficient equal to 1.

Definition 2.1.6. The set of polynomials over a ring R together with polynomial sum and multiplication form a ring called the *polynomial ring* over R and it is denoted by $R[x]$.

Definition 2.1.7. A *field* F is a commutative ring such that the nonzero elements of F form a group under multiplication. If F contains a finite number of elements, F is a *finite field*.

Definition 2.1.8. Let F be a field. A *subfield* of F is a subset K of F that is itself a field under $+$ and $\cdot$. F is called an *extension* (*field*) of K. If $K \neq F$, K is a *proper subfield* of F.

Definition 2.1.9. A field containing no proper subfields is called a *prime field*.

Definition 2.1.10. The intersection of all subfields of a field F is again a subfield of F. It is called the *prime subfield* of F and it is a prime field.

In the rest, F will denote a field. If F is an extension of K, then F may be viewed as a vector space over K.

Definition 2.1.11. Let F be an extension of K. The dimension of the vector space F over K is called the *degree* of F over K.

Definition 2.1.12. A polynomial $f \in F[x]$ is said to be *irreducible over* F, or *irreducible in* $F[x]$, if f has positive degree and $f = gh$ with $g, h \in F[x]$ implies that either g or h is a constant polynomial.

Definition 2.1.13. An element $a \in F$ is called a *root*, or a *zero*, of the polynomial $f \in F[x]$ if $f(a) = 0$.

Definition 2.1.14. Let K be a subfield of F and $\theta \in F$. If θ satisfies a polynomial equation $a_n\theta^n + \cdots + a_1\theta + a_0 = 0$ with $a_i \in K$ not all being 0, then θ is said to be *algebraic* over K.

Definition 2.1.15. If $\theta \in F$ is algebraic over K, then the unique monic polynomial $g \in K[x]$ generating the ideal $J = \{f \in K[x] \mid f(\theta) = 0\}$ of $K[x]$ is called the *minimal polynomial* of θ over K.

Definition 2.1.16. Let F be an extension of K and $f \in K[x]$ be of positive degree. Then f is said to *split* in F if f can be written as a product of linear factors in $F[x]$, i.e., if there exist elements $\alpha_1, \ldots, \alpha_n \in F$ such that

$$f(x) = a(x - \alpha_1) \cdots (x - \alpha_n),$$

where a is the leading coefficient of f. The field F is a *splitting field* of f over K if f splits in F.

A splitting field F of f over K is the smallest field containing all the roots of f, i.e., no proper subfield of F that is an extension of K contains all the roots of f.

2.1.2 Finite fields and polynomials over finite fields

Theorem 2.1.17 ([LN96, Corollary 1.45])**.** *A finite field has prime characteristic.*

Particularly, if $p \in \mathbb{Z}$ is prime, the finite field with p elements has characteristic p.

Theorem 2.1.18 ([LN96, Theorem 1.78])**.** *Let F be a field of prime characteristic p. The prime subfield of F is isomorphic to the finite field with p elements.*

Theorem 2.1.19 ([LN96, Lemma 2.1, Theorem 2.2, Theorem 2.5])**.**

- *Let F be a finite field containing a subfield K with q elements. Then F has q^m elements, where m is the degree of F over K.*
- *Let F be a finite field. Then F has p^n elements, where the prime p is the characteristic of F and n is the degree of F over its prime subfield.*
- *For every prime p and every positive integer n there exists a finite field with p^n elements.*

Let p be a prime integer. The finite field with p elements will be denoted by $\mathbb{F}_p$. The finite field with $q = p^n$ elements will be denoted by $\mathbb{F}_q$. An extension field of $\mathbb{F}_q$ of degree m will be denoted by $\mathbb{F}_{q^m}$.

Let p be the characteristic of $\mathbb{F}_q$, where $q = p^n$, and let $f \in \mathbb{F}_p[x]$ be irreducible of degree n. The elements of $\mathbb{F}_q$ can be represented as polynomials in $\mathbb{F}_p[x]$ of degree less than n. Then, we may regard $\mathbb{F}_q$ as the ring $\mathbb{F}_p[x]/(f)$.

Theorem 2.1.20 ([LN96, Theorem 2.8]). *For every finite field $\mathbb{F}_q$ the multiplicative group $\mathbb{F}_q^*$ of nonzero elements of $\mathbb{F}_q$ is cyclic.*

Definition 2.1.21. A generator of the cyclic group $\mathbb{F}_q^*$ is called a *primitive element* of $\mathbb{F}_q$.

Definition 2.1.22. Let $f \in \mathbb{F}_q[x]$ be a nonzero polynomial. If $f(0) \neq 0$, the least positive integer e for which $f(x)$ divides $x^e - 1$ is called the *order* of f (sometimes also called the *period* of f or the *exponent* of f). If $f(0) = 0$, then $f(x) = x^h g(x)$, where $h \in \mathbb{N}$ and $g \in \mathbb{F}_q[x]$ with $g(0) \neq 0$ are uniquely determined; the order of f is then defined to be the order of g.

Theorem 2.1.23 ([LN96, Corollary 3.4]). *If $f \in \mathbb{F}_q[x]$ is an irreducible polynomial over $\mathbb{F}_q$ of degree m, then the order of f divides $q^m - 1$.*

Definition 2.1.24. A polynomial $f \in \mathbb{F}_q[x]$ of degree m is called a *primitive* polynomial over $\mathbb{F}_q$ if it is the minimal polynomial over $\mathbb{F}_q$ of a primitive element of $\mathbb{F}_{q^m}$.

A primitive polynomial over $\mathbb{F}_q$ of degree m can be described as a monic polynomial which is irreducible over $\mathbb{F}_q$ and has a root $\alpha \in \mathbb{F}_{q^m}$ that is a primitive element of $\mathbb{F}_{q^m}$.

Theorem 2.1.25 ([LN96, Theorem 3.16]). *A polynomial $f \in \mathbb{F}_q[x]$ of degree m is a primitive polynomial over $\mathbb{F}_q$ if and only if f is monic, $f(0) \neq 0$, and the order of f is equal to $q^m - 1$.*

The remaining definitions and results are applicable to an arbitrary field F. In the context of this manuscript, we are interested in the case that F is a finite field.

Definition 2.1.26. Let n be a positive integer. The splitting field of $x^n - 1$ over a field F is called the *n-th cyclotomic field* over F and denoted by $F^{(n)}$. The roots of $x^n - 1$ in $F^{(n)}$ are called the *n-th roots of unity* over F and the set of all these roots is denoted by $E^{(n)}$.

Theorem 2.1.27 ([LN96, Theorem 2.42]). *Let n be a positive integer and F a field of characteristic p. If p does not divide n, then $E^{(n)}$ is a cyclic group of order n with respect to multiplication in $F^{(n)}$.*

Definition 2.1.28. Let F be a field of characteristic p and n a positive integer not divisible by p. Then a generator of the cyclic group $E^{(n)}$ is called a *primitive n-th root of unity* over F.

Definition 2.1.29. Let F be a field of characteristic p, n a positive integer not divisible by p and ζ a primitive n-th root of unity over F. Then the polynomial

$$C_n(x) = \prod_{\substack{s=1 \\ \gcd(s,n)=1}}^{n} (x - \zeta^s)$$

is called the *n-th cyclotomic polynomial* over F.

The polynomial $C_n(x)$ is independent of the choice of ζ. It has degree $\phi(n)$ and its roots are all the $\phi(n)$ different primitive n-th roots of unity over F, where ϕ is Euler's totient function.

2.2 Probability and Statistics

2.2.1 Basic definitions

Definition 2.2.1. A *sigma algebra* of a set S, denoted by $\mathcal{S}$, is a collection of subsets of S satisfying the following properties:

- $\emptyset \in \mathcal{S}$.
- If $A \in \mathcal{S}$, then $A^C \in \mathcal{S}$, where $A^C = S \setminus A$ is the complement of A.
- If $A_1, A_2, \dots \in \mathcal{S}$, then $\bigcup_{i=1}^{\infty} A_i \in \mathcal{S}$.

Definition 2.2.2. The *sample space* of an experiment is the set of all possible outcomes for that experiment. An *event* is a subset of the sample space.

The terms set and event may be used interchangeably. Let A be an event. If the outcome of an experiment is in the set A, we say that the event occurs.

Definition 2.2.3. Two events A and B are *mutually exclusive* if A and B are disjoint, i.e., $A \cap B = \emptyset$. The events $A_1, A_2, \dots$ are *pairwise mutually exclusive* if they are pairwise disjoint, i.e., $A_i \cap A_j = \emptyset$ for all $i \neq j$.

Definition 2.2.4. Let S be a sample space and $\mathcal{S}$ a sigma algebra of S. A *probability function* is a function P with domain $\mathcal{S}$ satisfying the following properties:

- $P(A) \geqslant 0$ for all $A \in \mathcal{S}$.
- $P(S) = 1$.
- If $A_1, A_2, \dots \in \mathcal{S}$ are pairwise disjoint, then $P(\bigcup_{i=1}^{\infty} A_i) = \sum_{i=1}^{\infty} P(A_i)$.

If S is a finite set, a probability function can equivalently be defined as follows [CB02, Theorem 1.2.6]: Let $S = \{s_1, \dots, s_n\}$ and $\mathcal{S}$ a sigma algebra of S. Let $p_1, \dots, p_n$ be nonnegative numbers that sum to 1. For any $A \in \mathcal{S}$, define

$$P(A) = \sum_{\{i : s_i \in A\}} p_i,$$

where the sum over an empty set is defined to be 0.

Definition 2.2.5. Let A and B be events in S, and $P(B) > 0$. Then, the *conditional probability of* A *given* B, denoted by $P(A|B)$, is

$$P(A|B) = \frac{P(A \cap B)}{P(B)}.$$

Conditional probabilities can be understood as the situation in which the original sample space S has been updated to the sample space B. The probability of any event A is then adjusted with respect to B.

Definition 2.2.6. Two events A and B are *independent* if $P(A \cap B) = P(A)P(B)$. A collection of events $A_1, \dots, A_n$ are *mutually independent* if $P(\bigcap_{j=1}^{k} A_{i_j}) = \prod_{j=1}^{k} P(A_{i_j})$ for any subcollection $A_{i_1}, \dots, A_{i_k}$.

If A and B are independent events, from the definition of conditional probability, we have that $P(A|B) = P(A)$. That is, the occurrence of B does not affect the probability of the event A.

The probability of the events $A_1, \ldots, A_n$ occurring at the same time is $P(A_1 \cap \cdots \cap A_n)$. We will use $P(A_1, \ldots, A_n)$ to denote $P(A \cap \cdots \cap A_n)$.

Definition 2.2.7. A *random variable* is a function defined on a sample space into $\mathbb{R}$.

When a random variable is defined, a new sample space is also defined, namely, the image of the random variable. Let S be a sample space, let P be a probability function of S and let us define a random variable X with image $\mathcal{X}$. The *induced* probability function P_X on (the sample space) $\mathcal{X}$ is defined as follows: For any set $A \subset \mathcal{X}$,

$$P_X(X \in A) = P(\{s \in S : X(s) \in A\}).$$

We will use uppercase letters to denote random variables and lowercase letters to denote the realised values of the variables. We will also simply write $P(\cdot)$ instead of $P_X(\cdot)$.

Definition 2.2.8. A random variable X is *discrete* if its image is countable (i.e., a finite set or a countably infinite set). If the image of X is uncountably infinite, then X is a *continuous* random variable.

Definition 2.2.9. Let X and Y be random variables. If, for every $A \subset \mathbb{R}$, $P(X \in A) = P(Y \in A)$, then X and Y are *identically distributed*.

Remark that if X and Y are identically distributed random variables, they are not necessarily equal, i.e., it does not imply that $X = Y$.

Definition 2.2.10. The *cumulative distribution function* (*cdf*) of a random variable X is defined by

$$F_X(x) = P(X \leqslant x), \quad \text{for all } x.$$

The cdf F_X completely determines the probability distribution of a random variable X. If the random variables X and Y are identically distributed, then $F_X(x) = F_Y(x)$ for every x [CB02, Theorem 1.5.10].

Definition 2.2.11. The *probability mass function* (*pmf*) f_X of a discrete random variable X is

$$f_X(x) = P(X = x), \quad \text{for all } x.$$

The *probability density function* (*pdf*) f_X of a continuous random variable X is the function that satisfies

$$F_X(x) = \int_{-\infty}^{x} f_X(y)\,dy, \quad \text{for all } x.$$

The pmf or pdf contains the same information as the cdf. Hence, either $F_X(x)$ or $f_X(x)$ can be used to describe a probability distribution. If X has a distribution given by $F_X(x)$ (or $f_X(x)$), it is customary to write $X \sim F_X(x)$ (or $X \sim f_X(x)$); similarly, if X and Y have the same distribution, we may write $X \sim Y$.

Definition 2.2.12. Let $f(x)$ be a probability distribution with sample space $\mathcal{X}$. The *support* of $f(x)$ is the set $\{x \in \mathcal{X} : f(x) > 0\}$.

Any function of a random variable is also a random variable. Let X be a random variable with sample space $\mathcal{X}$ and let $Y = g(X)$ with sample space $\mathcal{Y}$. Also, let g^{-1} denote the inverse map of g, defined by $g^{-1}(A) = \{x \in \mathcal{X} : g(x) \in A\}$, where $A \subset \mathcal{Y}$. The probability distribution of Y can be described in terms of that of X: For any set A,

$$P(Y \in A) = P(g(X) \in A) = P(X \in g^{-1}(A)).$$

Definition 2.2.13. The *expected value* or *mean* of a random variable X, with image $\mathcal{X}$, is

$$E[X] = \begin{cases} \sum_{x \in \mathcal{X}} x f_X(x) = \sum_{x \in \mathcal{X}} x P(X = x) & \text{if } X \text{ is discrete,} \\ \int_{-\infty}^{\infty} x f_X(x) dx & \text{if } X \text{ is continuous.} \end{cases}$$

Definition 2.2.14. The *variance* of a random variable X is

$$\mathrm{Var}(X) = E[(X - E[X])^2].$$

The non-negative square root of $\mathrm{Var}(X)$ is the *standard deviation* of X.

In general, the expected value or variance of a random variable may not exist. In the rest, we will not consider that case. The expected value is linear, i.e., for any two random variables X and Y, and a constant a, $E[aX + Y] = aE[X] + E[Y]$. Then, we have that

$$\begin{aligned} \mathrm{Var}(X) &= E[X^2 - 2XE[X] + E[X]^2] \\ &= E[X^2] - 2E[X]E[X] + E[X]^2 \\ &= E[X^2] - E[X]^2; \end{aligned}$$

since $E[X]$ is a constant, $E[E[X]] = E[X]$ and the second equality holds.

Definition 2.2.15. Let X and Y be random variables. The *covariance* of X and Y is

$$\mathrm{Cov}(X, Y) = E[(X - E[X])(Y - E[Y])].$$

Expanding the product in the definition of covariance, we have that

$$\mathrm{Cov}(X, Y) = E[XY] - E[X]E[Y].$$

Notice that $\mathrm{Cov}(X, X) = \mathrm{Var}(X)$. If $\mathrm{Cov}(X, Y) = 0$, X and Y are said to be *uncorrelated*. If X and Y are independent random variables, then $\mathrm{Cov}(X, Y) = 0$ [CB02, Theorem 4.5.5].

Definition 2.2.16. An n-*dimensional random vector*, or *multivariate random variable*, is a function defined on a sample space that takes values in $\mathbb{R}^n$.

The sample space of a random vector $(X_1, \dots, X_n)$ is a subset of $\mathbb{R}^n$.

Definition 2.2.17. A random vector is a *discrete* random vector if its sample space is countable, and it is a *continuous* random vector if its sample space is uncountably infinite.

We will use bold letters to denote the multivariate case. For example, $\mathbf{X}$ will denote the random vector $(X_1, \dots, X_n)$ and $\mathbf{x}$ will denote the realised value $(x_1, \dots, x_n)$.

Definition 2.2.18. Let $\mathbf{X} = (X_1, \dots, X_n)$ be a random vector. The *joint cumulative distribution function* (*joint cdf*) of $\mathbf{X}$ is defined by

$$F_{\mathbf{X}}(\mathbf{x}) = P(X_1 \leqslant x_1, \dots, X_n \leqslant x_n).$$

If $\mathbf{X}$ is a discrete random vector, its *joint probability mass function* (*joint pmf*) is the function

$$f_{\mathbf{X}}(\mathbf{x}) = P(X_1 = x_1, \dots, X_n = x_n).$$

If $\mathbf{X}$ is a continuous random vector, its *joint probability density function* (*joint pdf*) is the function $f_{\mathbf{X}}$ that satisfies

$$F_{\mathbf{X}}(\mathbf{x}) = \int_{-\infty}^{x_1} \cdots \int_{-\infty}^{x_n} f_{\mathbf{X}}(\mathbf{x}) dx_1 \dots dx_n.$$

Let $\mathbf{X} = (X_1, \dots, X_n)$ be a random vector. The probability distribution of each X_i is described by its pmf or pdf $f_{X_i}(x)$. In the context of the joint pmf or joint pdf, the function $f_{X_i}(x)$ is called the *marginal pmf* or *marginal pdf* of X_i. The concept of the marginal distribution can be extended to a subset of variables.

Definition 2.2.19. Let $\mathbf{X}_1, \dots, \mathbf{X}_n$ be random vectors with joint pdf or joint pmf $f_{\mathbf{X}_1,\dots,\mathbf{X}_n}(\mathbf{x}_1, \dots, \mathbf{x}_n)$. Let the marginal pdf or pmf of $\mathbf{X}_i$ be denoted by $f_{\mathbf{X}_i}(\mathbf{x}_i)$. Then, $\mathbf{X}_1, \dots, \mathbf{X}_n$ are *mutually independent random vectors* if, for every $(\mathbf{x}_1, \dots, \mathbf{x}_n)$,

$$f_{\mathbf{X}_1,\dots,\mathbf{X}_n}(\mathbf{x}_1, \dots, \mathbf{x}_n) = \prod_{i=1}^{n} f_{\mathbf{X}_i}(\mathbf{x}_i).$$

If all $\mathbf{X}_i$ have dimension one, then they are *mutually independent random variables*.

Definition 2.2.20. If the random variables $X_1, \dots, X_n$ are mutually independent and the marginal pmf or marginal pdf of each X_i is the same function, then $X_1, \dots, X_n$ are called *independent and identically distributed* random variables. This is commonly abbreviated as i.i.d. random variables.

Definition 2.2.21. The *expected value* of a random vector $\mathbf{X} = (X_1, \dots, X_n)$ is the vector

$$E[\mathbf{X}] = (E[X_1], \dots, E[X_n]).$$

Definition 2.2.22. The *covariance matrix* of a random vector $\mathbf{X} = (X_1, \dots, X_n)$ is the $n \times n$ matrix over $\mathbb{R}$ whose entry in the i-th row and j-th column is $\mathrm{Cov}(X_i, X_j)$, $1 \leqslant i, j \leqslant n$. That is,

$$\mathrm{Cov}(\mathbf{X}) = \begin{pmatrix} \mathrm{Var}(X_1) & \mathrm{Cov}(X_1, X_2) & \cdots & \mathrm{Cov}(X_1, X_n) \\ \mathrm{Cov}(X_2, X_1) & \mathrm{Var}(X_2) & \cdots & \mathrm{Cov}(X_2, X_n) \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{Cov}(X_n, X_1) & \mathrm{Cov}(X_n, X_2) & \cdots & \mathrm{Var}(X_n) \end{pmatrix}.$$

2.2.2 Some probability distributions

Here we present some common probability distributions. These distributions are defined by a function depending on certain parameters. The characteristics of the distributions vary according to the values of the parameters. If $f(\cdot)$ is the function defining a probability distribution (i.e., the pmf/pdf or cdf), it is customary to write $f(\cdot|\theta)$ to emphasize the parameter θ.

Discrete uniform distribution

Let $n_0, n_1 \in \mathbb{Z}$ such that $n_1 \geqslant n_0$, and let $n = n_1 - n_0 + 1$. The *discrete uniform distribution* is the distribution with pmf given by

$$f(x|n_0, n_1) = \frac{1}{n}, \quad x = n_0, \dots, n_1.$$

If a (discrete) random variable X has a discrete uniform distribution, then

$$E[X] = \frac{n_0 + n_1}{2} \quad \text{and} \quad \mathrm{Var}(X) = \frac{n^2 - 1}{12}.$$

Normal distribution

The *normal distribution* with parameters μ and σ^2, denoted by $\mathbf{N}(\mu, \sigma^2)$, is the continuous distribution with pdf

$$f(x|\mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}.$$

The parameters μ and σ^2 are, respectively, the mean and the variance of the distribution; $\sigma \geqslant 0$ is the standard deviation. The *standard normal distribution* is the special case with $\mu = 0$ and $\sigma^2 = 1$. If $X \sim \mathbf{N}(\mu, \sigma^2)$, then

$$E[X] = \mu \quad \text{and} \quad \mathrm{Var}(X) = \sigma^2.$$

We will use $P(\mathbf{N}(\mu, \sigma^2) < x)$ to denote the pdf, i.e.,

$$P(\mathbf{N}(\mu, \sigma^2) < x) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{1}{2}\left(\frac{y-\mu}{\sigma}\right)^2} dy.$$

Let $X \sim \mathbf{N}(\mu, \sigma^2)$. If $Z = (X - \mu)/\sigma$, then $Z \sim \mathbf{N}(0, 1)$, i.e., Z has a standard normal distribution. Conversely, if $Z \sim \mathbf{N}(0, 1)$, then $X = \sigma Z + \mu \sim \mathbf{N}(\mu, \sigma^2)$.

Multivariate normal distribution

The *multivariate normal distribution* is a continuous distribution with parameters $\boldsymbol{\mu} \in \mathbb{R}^n$, the mean, and $Q \in \mathbb{R}^{n \times n}$, the covariance matrix, and is denoted by $\mathbf{N}(\boldsymbol{\mu}, Q)$. It is a generalisation of the univariate normal distribution to higher dimensions. The pdf of a multivariate normal distribution is

$$f(\mathbf{x}|\boldsymbol{\mu}, Q) = \frac{1}{(2\pi)^{n/2}|Q|^{1/2}} e^{-\frac{(\mathbf{x}-\boldsymbol{\mu})^T Q^{-1}(\mathbf{x}-\boldsymbol{\mu})}{2}},$$

where $|Q|$ is the determinant of Q. The covariance matrix is symmetric and positive semi-definite. When $|Q| = 0$, Q^{-1} does not exist and for such *singular* distributions, the probability mass is concentrated on a linear subspace of $\mathbb{R}^n$; the probabilities of singular distributions can still be computed (see [GB09], for example). Let $\mathbf{X} \sim \mathbf{N}(\boldsymbol{\mu}, Q)$, then the mean and variance are given, respectively, by

$$E[\mathbf{X}] = \boldsymbol{\mu} \quad \text{and} \quad \mathrm{Cov}(\mathbf{X}) = Q.$$

2.2.3 Central limit theorem

Theorem 2.2.23 (Central limit theorem [Bil95]). *Let $X_1, \ldots, X_n$ be a sequence of i.i.d. random variables with $E[X_i] = \mu$ and finite $Var(X_i) = \sigma^2$, $i = 1, \ldots, n$. Define $S_n = X_1 + \cdots + X_n$, then*

$$\lim_{n\to\infty} P\left(\frac{S_n - n\mu}{\sigma\sqrt{n}} < x\right) \to P(\mathbf{N}(0,1) < x),$$

i.e., $\frac{S_n - n\mu}{\sigma\sqrt{n}}$ has a limiting standard normal distribution.

In simple words, the central limit theorem (CLT) says that the sum of many i.i.d. random variables will be approximately normally distributed (even if the original variables are not normally distributed). The following result is a variant when the random variables are independent but not identically distributed:

Theorem 2.2.24 (Lyapunov's CLT [Bil95]). *Let $X_1, \ldots, X_n$ be a sequence of independent random variables with $E[X_i] = \mu_i$ and finite $Var(X_i) = \sigma_i^2$, $i = 1, \ldots, n$. Define $S_n = X_1 + \cdots + X_n$ and $s_n^2 = \sum_{i=1}^n \sigma_i^2$. If for some positive δ, Lyapunov's condition*

$$\lim_{n\to\infty} \frac{1}{s_n^{2+\delta}} \sum_{i=1}^{n} E\left[|X_i|^{2+\delta}\right] = 0$$

holds, then

$$\lim_{n\to\infty} P\left(\frac{S_n - \sum_{i=1}^n \mu_i}{s_n} < x\right) \to P(\mathbf{N}(0,1) < x).$$

2.2.4 Random sample and hypothesis testing

Definition 2.2.25. The random variables $X_1, \ldots, X_n$ are called a *random sample of size* n *from the population* $f(x)$ if $X_1, \ldots, X_n$ are i.i.d. random variables with pdf or pmf $f(x)$.

The joint pdf or pmf of a sample $X_1, \ldots, X_n$ is given by

$$f(x_1, \ldots, x_n) = \prod_{i=1}^{n} f(x_i),$$

where all the marginal densities $f(x)$ are the same function since $X_1, \ldots, X_n$ are identically distributed. If the population pdf or pmf can be parametrised by θ, the joint pdf or pmf is

$$f(x_1, \ldots, x_n|\theta) = \prod_{i=1}^{n} f(x_i|\theta),$$

where the same value of θ is used in each term in the product.

Definition 2.2.26. Let $X_1, \ldots, X_n$ be a random sample of size n from a population. Also, let $T(x_1, \ldots, x_n)$ be a real-valued or vector-valued function whose domain includes the sample space of $(X_1, \ldots, X_n)$. Then, the random variable or random vector $Y = T(X_1, \ldots, X_n)$ is called a *statistic*.

A *hypothesis* is a statement about a population. The goal of a hypothesis test is to decide, based on a sample from the population, which of two complementary hypotheses is true. These hypotheses are called the *null hypothesis* and the *alternative hypothesis*. They are denoted by H_0 and H_1, respectively. Let θ be a parameter, then $H_0 : \theta = \theta_0$ and $H_1 : \theta = \theta_1$, where θ_0, θ_1 are some possible values for θ.

Definition 2.2.27. A *hypothesis test* is a rule that specifies:

- The sample values for which the decision is to accept H_0 as true.
- The sample values for which H_0 is rejected and H_1 is accepted as true.

A hypothesis test is typically specified in terms of a *test statistic* $T(X_1, \ldots, X_n)$, which is a function of the sample $X_1, \ldots, X_n$.

Definition 2.2.28. Let $f(\mathbf{x}|\theta)$ denote the joint pdf or pmf of the sample $\mathbf{X} = (X_1, \ldots, X_n)$. Then, given that $\mathbf{X} = \mathbf{x}$ is observed, the function of θ defined by

$$L(\theta|\mathbf{x}) = f(\mathbf{x}|\theta) = \prod_{i=1}^{n} f(x_i|\theta)$$

is called the *likelihood function*.

The likelihood is equal to the probability that an outcome $\mathbf{x}$ is observed when the value of the parameter is θ. It is therefore equal to a probability density over $\mathbf{x}$, not over the parameter θ.

Likelihood-based tests are widely used in hypothesis testing (e.g., likelihood ratio test). In simple terms, we accept H_0 if $k \cdot L(\theta_0|\mathbf{x}) > L(\theta_1|\mathbf{x})$, for some $k \geqslant 0$. According to the Neyman-Pearson lemma [NP33], this is a *most powerful test*; we refer the reader to existing texts, e.g. [CB02], for further details.

2.3 Boolean functions

We will use $\mathbb{F}_2$ to denote the finite field with two elements and $\mathbb{F}_2^n$ to denote the vector space of dimension n over $\mathbb{F}_2$. The symbol $+$ will be used to denote addition in general. The symbol $\oplus$ will be used to specifically denote addition over $\mathbb{F}_2$. If it is clear by the context, $+$ and $\oplus$ might be used interchangeably for addition over $\mathbb{F}_2$.

Definition 2.3.1. A *Boolean function* is a function $f : \mathbb{F}_2^n \to \mathbb{F}_2$. It is customary to say that f is a function in n variables. The set of all n-variable Boolean functions is denoted by $\mathcal{BF}_n$.

Definition 2.3.2. Let f and g be Boolean functions on $\mathbb{F}_2^n$. The *Hamming weight* $w_H(f)$ of f is the size of the set $\{x \in \mathbb{F}_2^n : f(x) \neq 0\}$, the *support* of f. The *Hamming distance* $d_H(f, g)$ between f and g is the size of the set $\{x \in \mathbb{F}_2^n : f(x) \neq g(x)\}$; it is equal to $w_H(f \oplus g)$.

Definition 2.3.3. An *affine function* is a Boolean function with algebraic degree at most 1, i.e.,

$$f(x) = a_n x_n \oplus \cdots \oplus a_1 x_1 \oplus a_0, \quad a_i \in \mathbb{F}_2.$$

A *linear function* is an affine function with $a_0 = 0$.

Definition 2.3.4. Let a Boolean function f be viewed as a function valued in $\mathbb{Z}$. The *Fourier-Hadamard transform* is the linear mapping that maps f to the function $\widehat{f}$ defined on $\mathbb{F}_2^n$ by

$$\widehat{f}(u) = \sum_{x \in \mathbb{F}_2^n} f(x)(-1)^{u \cdot x},$$

where $u \cdot x$ denotes some inner product in $\mathbb{F}_2^n$. The *Fourier-Hadamard spectrum* of f is the string of all values $\widehat{f}(u)$, where $u \in \mathbb{F}_2^n$.

We will use the dot product as the inner product in $\mathbb{F}_2^n$, i.e., $u \cdot v = \bigoplus_{i=1}^n u_i v_i$. The *fast Fourier-Hadamard transform* (FFT) is an efficient algorithm to compute $\widehat{f}$; it is shown in Algorithm 2.1. The FFT takes as input a vector (table) with the values of f for all $x \in \mathbb{F}_2^n$ ordered in lexicographical order with respect to x, i.e., its entries are $(f(0,\dots,0,0), f(0,\dots,0,1), \dots, f(1,\dots,1,0), f(1,\dots,1,1))$. The complexity of the FFT is $O(N \log_2 N)$ arithmetic operations [Car21], where $N = 2^n$.

Algorithm 2.1 Fast Fourier-Hadamard transform

Input: Vector F of values $f(x)$ for all $x \in \mathbb{F}_2^n$ in lexicographical order with respect to x.
Output: Fourier-Hadamard spectrum of f.
Let $F_{i,j}$ denote a vector of 2^i integers, where $0 \leqslant j \leqslant 2^{n-i} - 1$ for all $i = 0, \dots, n$. Also, let $F = (F_0, \dots, F_{2^n - 1})$ and $F_{0,j} = F_j$, $j = 0, \dots, 2^n - 1$.

```
for i = 1,...,n do
    for j = 0,...,2^(n-i) - 1 do
        F_{i,j} = (F_{i-1,2j} + F_{i-1,2j+1}, F_{i-1,2j} - F_{i-1,2j+1})
    end for
end for
return F_{n,0}
```

Definition 2.3.5. The *sign function* of a Boolean function f is

$$f_\chi(x) = (-1)^{f(x)}.$$

Definition 2.3.6. The *Walsh transform* of a Boolean function f, denoted by W_f, is the Fourier-Hadamard transform of its sign function, i.e.,

$$W_f(u) = \sum_{x \in \mathbb{F}_2^n} (-1)^{f(x) \oplus u \cdot x}.$$

The *Walsh spectrum* of f is the string of all values $W_f(u)$, where $u \in \mathbb{F}_2^n$.

Definition 2.3.7. A Boolean function f in n variables is *balanced* if its outputs are equally distributed over $\{0,1\}$. In other words, f maps 2^{n-1} vectors in $\mathbb{F}_2^n$ to 0 and the other 2^{n-1} vectors to 1.

From the definition of the Walsh transform, f is balanced if and only if $W_f(0) = 0$.

Definition 2.3.8. The *bias* (also *correlation* or *imbalance*) of a Boolean function f is

$$\mathcal{E}(f) = W_f(0) = \sum_{x \in \mathbb{F}_2^n} (-1)^{f(x)}.$$

Definition 2.3.9. The *nonlinearity* of a Boolean function f, denoted by $nl(f)$, is the minimum Hamming distance between f and affine functions.

The nonlinearity can be computed using the Walsh transform [Car21]:

$$nl(f) = 2^{n-1} - \frac{1}{2} \max_{u \in \mathbb{F}_2^n} |W_f(u)|.$$

Hence, a function has high nonlinearity if and only if all values of its Walsh spectrum have low magnitudes. The value $\max_{u \in \mathbb{F}_2^n} |W_f(u)|$ is called the *linearity* of f.

Definition 2.3.10. Let f be a Boolean function and $a \in \mathbb{F}_2^n$. The *derivative of* f *in direction* a is the function $D_a f(x) = f(x) \oplus f(a + x)$.

Definition 2.3.11. The *autocorrelation function* of a Boolean function f is the function

$$\Delta_f(a) = W_{D_a f}(0) = \sum_{x \in \mathbb{F}_2^n} (-1)^{f(x) \oplus f(a+x)},$$

where $a \in \mathbb{F}_2^n$.

2.4 Linear feedback shift registers and sequences

Let F be a finite field. A *feedback shift register* is a state machine which produces a sequence of elements of F. The device consists of n *cells* or *stages* and receives a clock input to update the content of the cells and produce an output. The *state* of the register is the value of the cells viewed as a length-n vector; $(s_t, s_{t+1}, \dots, s_{t+n-1})$ is the state at time t. The cells are initially loaded with n elements of F; they define the *initial state*. At every clock cycle, the content of the first cell is the output. To update the state, the content of the i-th cell is transferred into the $(i-1)$-th cell, $i = 2, \dots, n$, while the content of the n-th cell is computed as a function of the current state by the *feedback function* f. Figure 2.1 depicts a feedback shift register.

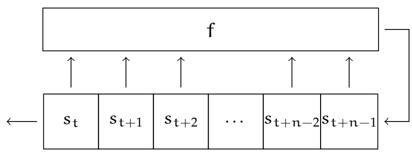


Figure 2.1. Feedback shift register.

Definition 2.4.1. A feedback shift register is *linear* if its feedback function is linear, i.e., it can be expressed as

$$f(x_1, \dots, x_n) = c_1 x_1 + \dots + c_n x_n,$$

where $c_1, \dots, c_n \in F$.

Unless otherwise stated, we will assume that the feedback shift registers are binary, i.e., the output sequence consists of elements of $\mathbb{F}_2$. The rest of this section is focused on linear feedback shift registers; they will be referred to as LFSRs. The feedback function is then a linear Boolean function in n variables.

Definition 2.4.2. A *linear recurrence* is an equation of the type

$$s_{t+n} = c_1 s_{t+n-1} + c_2 s_{t+n-2} + \cdots + c_n s_t. \tag{2.1}$$

Any sequence satisfying (2.1) is called a *linear recurring sequence*.

Theorem 2.4.3 ([Gol17, Theorem 2.2])**.** *Let the sequence $s_1, s_2, \dots, s_N$ denote the succession of values for a given cell of an LFSR. Then, s_N satisfies a linear recurrence where the coefficients c_i are elements of $\mathbb{F}_2$ and do not depend on N.*

Given the operation of a shift register, the sequence of values of the first cell and all the other cells are the same, except for a shift or delay. This shift is of one position with the second cell, two positions with the third cell and so on. Thus, all cells of an LFSR satisfy the same linear recurrence. This means that the output sequence and the whole state satisfy the recurrence. The output sequence generated by an LFSR will be denoted as $\{s_t\}_{t\geqslant 1}$. The coefficients c_i of the linear recurrence satisfied by $\{s_t\}_{t\geqslant 1}$ are called the *feedback coefficients* of the LFSR that generated it.

Definition 2.4.4. The *feedback polynomial* or *connection polynomial* of a sequence $\{s_t\}_{t\geqslant 1}$ and of the LFSR which produced it is the degree-n polynomial

$$f(x) = 1 - \sum_{i=1}^{n} c_i x^i,$$

where c_i are the coefficients of the linear recurrence satisfied by $\{s_t\}_{t\geqslant 1}$. The *characteristic polynomial* of the sequence $\{s_t\}_{t\geqslant 1}$ and of the LFSR which produced it is the reciprocal of the feedback polynomial, i.e.,

$$f^*(x) = x^n f(1/x) = x^n - \sum_{i=1}^{n} c_i x^{n-i}.$$

Definition 2.4.5. A length-n LFSR is *non-singular* if the degree of its feedback polynomial is equal to n (i.e., if the feedback coefficient c_n is not zero).

Definition 2.4.6. Let $s_1, s_2, \dots$ be a sequence of elements of a nonempty set S. If there exists integers $p > 0$ and $n_0 \geqslant 0$ such that $s_{n+p} = s_n$ for all $n > n_0$, then the sequence is called *ultimately periodic* and p is called the *period* of the sequence. The smallest number among all possible periods of an ultimately periodic sequence is called the *least period* of the sequence.

Definition 2.4.7. An ultimately periodic sequence $s_1, s_2, \dots$ with least period p is called *periodic* if $s_{n+p} = s_n$ holds for all $n \geqslant 1$.

Theorem 2.4.8 ([Gol17, Theorem 2.1])**.** *The sequence $\{s_t\}_{t\geqslant 1}$ generated by a non-singular length-n LFSR is periodic with period $p \leqslant 2^n - 1$.*

Theorem 2.4.9 ([Gol17, Theorem 2.3])**.** *The period of $\{s_t\}_{t\geqslant 1}$ is the smallest positive integer p for which its feedback polynomial divides $x^p - 1$.*

Definition 2.4.10. A sequence generated by a length-n LFSR has *maximum length* if its period is $p = 2^n - 1$.

Theorem 2.4.11 ([Gol17, Theorem 2.4]). *If the sequence generated by an LFSR has maximum length, its feedback polynomial is irreducible.*

We are interested in sequences with maximum length. In order to obtain such sequences, irreducibility of the feedback polynomial is a necessary condition, but not sufficient. When the feedback polynomial $f(x)$ of $\{s_t\}_{t\geqslant 1}$ is irreducible, then the period of $\{s_t\}_{t\geqslant 1}$ is equal to the exponent of $f(x)$ [Gol17].

Theorem 2.4.12 ([Gol17, Theorem 3.1]). *If a sequence has an irreducible feedback polynomial of degree n, the period of the sequence is a factor of $2^n - 1$.*

Theorem 2.4.13 ([Gol17, Theorem 3.2]). *Every factor a of $2^n - 1$ which is not a factor of any number $2^s - 1$ with $s < n$ occurs as the exponent of irreducible polynomials of degree n. Precisely, there are $\phi(a)/n$ irreducible polynomials of degree n with exponent a, where ϕ is Euler's totient function.*

When $2^n - 1$ is prime (a *Mersenne prime*), every irreducible polynomial of degree n corresponds to a sequence of maximum length (by theorem 2.4.12). When $2^n - 1$ is not prime, maximum-length sequences are generated by irreducible polynomials of degree n with "maximum exponents" (exponents equal to $2^n - 1$), i.e., primitive polynomials of degree n. The factors of $C_p(x)$, the p-th cyclotomic polynomial, are the irreducible polynomials of order p. Particularly, the factors of $C_{2^n-1}(x)$ have degree n and there are $\phi(2^n - 1)/n$ of them [Gol17]. The feedback polynomial being primitive is the necessary and sufficient condition for $\{s_t\}_{t\geqslant 1}$ to have maximum length.

Definition 2.4.14. A *pseudo-noise sequence*, or *PN sequence*, is a maximum-length linear recurring sequence over $\mathbb{F}_2$. I.e., $s_1, s_2, \dots$, with $s_i \in \mathbb{F}_2$, is a PN sequence if and only if it is a sequence which satisfies a linear recurrence

$$s_t = \sum_{i=1}^{n} c_i s_{t-i},$$

where $c_i \in \mathbb{F}_2$, and has period $p = 2^n - 1$.

Any sequence generated by an LFSR with feedback polynomial f is also generated by any LFSR whose feedback polynomial is a multiple of f [Can11]. Also, there is a sequence generated by an LFSR with feedback polynomial f which can be generated by a shorter LFSR if and only if f is not irreducible over $\mathbb{F}_2$ [Can11].

Definition 2.4.15. Let $\{s_t\}_{t\geqslant 1}$ be any linear recurring sequence. The characteristic polynomial of the shortest LFSR which generates $\{s_t\}_{t\geqslant 1}$ is called the *minimal polynomial* of the sequence.

Definition 2.4.16. The degree of the minimal polynomial of a linear recurring sequence is the *linear complexity* of the sequence. It is equal to the length of the shortest LFSR which generates the sequence.

The operation of a length-n LFSR can be seen as a linear operator on its states. Let $(s_t, s_{t+1}, \dots, s_{t+n-1})^\mathsf{T}$ be the state of the LFSR at time $t \geqslant 1$. The $n \times n$ matrix

$$M = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ c_n & c_{n-1} & c_{n-2} & \cdots & c_1 \end{pmatrix}$$

"implements" the operation of the LFSR:

$$M \cdot (s_t, s_{t+1}, \dots, s_{t+n-1})^T = \left(s_{t+1}, s_{t+2}, \dots, s_{t+n-1}, \sum_{i=1}^{n} c_i s_{t+n-i} \right)$$
$$= (s_{t+1}, s_{t+2}, \dots, s_{t+n-1}, s_{t+n})$$

is the state of the LFSR at time $t+1$. Therefore, any state at time $t \geqslant 1$ can be expressed in terms of the initial state as

$$S_t^T = M^{t-1} \cdot S_1^T \tag{2.2}$$

and each symbol s_t, $t \geqslant 1$, can be written as a linear combination

$$s_t = h_{t,1}s_1 + h_{t,2}s_2 + \cdots + h_{t,n}s_n, \tag{2.3}$$

where $h_{t,j} \in \mathbb{F}_2$ are given by the first row of M^{t-1}.

2.5 Error-correcting codes

It is possible for digital information to be corrupted by noise when transmitted over a communication channel (or kept in a storage medium). Error-correcting codes may be used to overcome this. The main idea is to *encode* the information sequence, or message, $\mathbf{u} = (u_0 \dots u_{k-1}) \in \mathbb{F}_2^k$ to obtain a *codeword* $\mathbf{v} = (v_0 \dots v_{n-1}) \in \mathbb{F}_2^n$, where $k < n$. The codeword is then sent through the noisy communication channel. At the other end of the channel, the received sequence $\mathbf{r} = (r_0 \dots r_{n-1}) \in \mathbb{F}_2^n$, is *decoded* to obtain an *estimate* $\widehat{\mathbf{u}}$ of the original message. This process is depicted in Figure 2.2. The codeword contains the information sequence and some redundancy in order to detect and correct errors. If $\widehat{\mathbf{u}} \neq \mathbf{u}$, then a *decoding error* occurs.

Figure 2.2. Usage of error-correcting codes to transmit information.

Alternatively, the output of decoding may be an estimate $\widehat{\mathbf{v}}$, and decoding error means that $\widehat{\mathbf{v}} \neq \mathbf{v}$. Define the *error* vector $\mathbf{e} = (e_0 \dots e_{n-1}) \in \mathbb{F}_2^n$ as $\mathbf{e} = \mathbf{r} - \mathbf{v}$. Then, the decoder may equivalently output an estimate $\widehat{\mathbf{e}}$ and $\widehat{\mathbf{e}} \neq \mathbf{e}$ means a decoding error.

There are different noisy communication channel models. Here, we will only consider the *binary symmetric channel* (BSC). It is a binary-input and binary-output channel model, where the probability of transmission error is given by the *crossover probability* ϵ. The BSC is a *memoryless* channel because an output symbol depends only on its corresponding transmitted symbol. Figure 2.3 depicts a BSC.

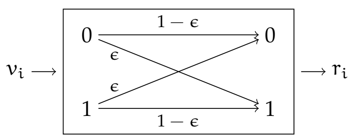


Figure 2.3. Binary symmetric channel with crossover probability ϵ.

In the rest of this section, we will continue using boldface letters to denote vectors.

2.5.1 Linear codes

Here we briefly introduce *linear codes*. In the discussion above, we assumed information to be binary. Linear codes will be presented in a more general setting where the information symbols are elements of a finite field $\mathbb{F}_p$. We refer to [MS81] for a thorough presentation of these codes.

Definition 2.5.1. Let p be a prime integer. An $[n, k]$*-linear code* $\mathcal{C}$ of *length* n and *dimension* k is a k-dimensional subspace of $\mathbb{F}_p^n$. The *rate* of $\mathcal{C}$ is k/n.

Let $\mathcal{C}$ be an $[n, k]$-linear code. A vector $\mathbf{v} \in \mathcal{C}$ is called a *codeword*. The *size* of $\mathcal{C}$ is the number of codewords, i.e., $|\mathcal{C}| = p^k$. It is customary to call $\mathcal{C}$ a *binary* code when $p = 2$, and p-*ary* otherwise. In the rest, we will assume $\mathcal{C}$ to be an $[n, k]$-linear code.

Definition 2.5.2. A *generator matrix* of $\mathcal{C}$, denoted by G, is a $k \times n$ matrix whose rows are the vectors of a basis of $\mathcal{C}$. This matrix defines the code as

$$\mathcal{C} = \{\mathbf{u}G \mid \mathbf{u} \in \mathbb{F}_p^k\}.$$

A generator matrix G is called *systematic* if the information symbols $u_0 \dots, u_{k-1}$ appear unchanged at the beginning of the codeword, i.e., G has the form

$$G = (I_k \,|\, M),$$

where I_k is the $k \times k$ identity matrix.

Definition 2.5.3. A *parity-check matrix* of $\mathcal{C}$, denoted by H, is an $r \times n$ matrix, $r = n - k$, that defines $\mathcal{C}$ as

$$\mathcal{C} = \{\mathbf{v} \in \mathbb{F}_p^n \mid H\mathbf{v}^\mathsf{T} = 0\},$$

i.e., $\mathcal{C}$ is the kernel of H.

A generator matrix G and a parity-check matrix H of $\mathcal{C}$ are related by

$$GH^\mathsf{T} = 0 \quad \text{or} \quad HG^\mathsf{T} = 0.$$

Let $r = n - k$. If $\mathcal{C}$ has a systematic matrix $G = (I_k \,|\, M)$, that code has a parity-check matrix $H = (-M^\mathsf{T} \,|\, I_r)$, where I_r is the $r \times r$ identity matrix. Given a parity-check matrix H, a generator matrix can be obtained as follows: take H into the form $(M \,|\, I_r)$ using row operations, then $G = (I_k \,|\, -M^\mathsf{T})$. The *syndrome* of $\mathbf{v} \in \mathbb{F}_p^n$, relative to H, is defined as $H\mathbf{v}^\mathsf{T}$. Notice that the syndrome of $\mathbf{v}$ is zero if and only if $\mathbf{v} \in \mathcal{C}$.

Definition 2.5.4. The *dual code* of $\mathcal{C}$, denoted by $\mathcal{C}^\perp$, is the linear code spanned by the rows of any parity-check matrix of $\mathcal{C}$.

Definition 2.5.5. A linear code is *cyclic* if every cyclic shift of a codeword is also a codeword. A linear code is *quasi-cyclic* (QC) if there exists an integer n_0 such that a cyclic shift by n_0 positions of a codeword is also a codeword.

Let $\mathcal{C}$ be quasi-cyclic code with $n = n_0\ell$, for some integer ℓ. Then, the generator and parity-check matrices of $\mathcal{C}$ can be constructed by circulant blocks of size $\ell \times \ell$. Only one vector completely determines a given block (e.g., the first row or first column). Hence, only one vector per block is needed to completely determine the generator and parity-check matrices.

In the remaining of this section, we return to binary codes, i.e., the symbols are elements of $\mathbb{F}_2$.

2.5.2 Low-density parity-check codes

A *low-density parity-check* (LDPC) code is a linear code defined by a sparse parity-check matrix with constant low row weight. Here we focus on the notions of LDPC codes relevant to this manuscript. We refer to the original paper [Gal62] or [JZ15] for further details.

The parity-check matrix of a *regular* $[n, j, w]$-LDPC code is such that each column contains a small fixed number j of 1's and each row contains a small fixed number w of 1's. For example, the following matrix is a parity-check matrix for a regular $[12, 2, 3]$-LDPC code:

$$H = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \end{pmatrix}.$$

Irregular LDPC codes admit matrices where the number of 1's in the rows/columns varies; here, we assume LDPC codes to be regular.

Given an $r \times n$ parity-check matrix H, a generator matrix G of size $k \times n$ can be obtained as for linear codes above. If $\mathbf{u} = (u_0, \ldots, u_{k-1})$ is the information sequence, the corresponding codeword is

$$\mathbf{v} = \mathbf{u}G.$$

Decoding - Gallager's bit-flipping and iterative algorithms

Gallager introduced two decoding schemes for LDPC codes in [Gal62]. Let the codeword $\mathbf{v} = (v_0 \ldots v_{n-1})$ be sent through a BSC and $\mathbf{r} = (r_0 \ldots r_{n-1})$ be the received sequence. The parity-checks for a symbol r_i are given by the rows of H whose i-th entry is 1.

The *bit-flipping* algorithm is shown in Algorithm 2.2. As the name implies, this algorithm flips the values of the symbols r_i according to the number of parity-checks that are satisfied. At each iteration, the parity checks are computed with the decided value $\widehat{\mathbf{v}}$ and the process stops when all parity-checks are satisfied or after a number of iterations.

Algorithm 2.2 Gallager's bit-flipping algorithm

1: Set $\widehat{\mathbf{v}} = \mathbf{r}$.
2: **repeat**
3: For each $\widehat{v}_i$, $i = 0, \ldots, n-1$, count the number of unsatisfied parity-checks.
4: Flip the value of the bits $\widehat{v}_i$ with largest number of unsatisfied parity-checks.
5: **until** all parity-checks are satisfied or the maximum number of iterations is reached.
6: Return $\widehat{\mathbf{v}}$ if all parity-checks are satisfied, $\perp$ otherwise.

Gallager's iterative algorithm is a decoding technique in the so-called *belief propaga-*

tion algorithms. It considers the log likelihood ratios

$$\log \frac{P(v_t = 0|\mathbf{r}, S)}{P(v_t = 1|\mathbf{r}, S)},$$

where S is the event that the transmitted symbols satisfy the j parity-check constraints on v_t, $0 \leqslant t < n$. We do not give the details of the algorithm and refer to the original paper [Gal62] or [JZ15] for details. Algorithm 3.6 in Section 3.2.2 is based closely on Gallager's iterative algorithm.

2.5.3 Convolutional codes

Here we briefly present convolutional codes; we focus only on codes whose characteristics are relevant for this manuscript. We refer to [JZ15] for an in-depth treatment of these codes.

Convolutional codes can be thought of as linear codes where the infinite information sequence

$$\mathbf{u} = \mathbf{u}_0\mathbf{u}_1 \cdots = u_0^{(1)}u_0^{(2)} \ldots u_0^{(k)}u_1^{(1)}u_1^{(2)} \ldots u_1^{(k)} \ldots$$

is encoded by a binary *convolutional encoder* of rate $R = k/n$, $k \leqslant n$, into the infinite code sequence

$$\mathbf{v} = \mathbf{v}_0\mathbf{v}_1 \cdots = v_0^{(1)}v_0^{(2)} \ldots v_0^{(n)}v_1^{(1)}v_1^{(2)} \ldots v_1^{(n)} \ldots.$$

Here we will assume that $k = 1$, then $\mathbf{u}_i = u_i \in \mathbb{F}_2$.

A convolutional encoder consists of a shift register of *memory* (or length) m and generates n output sequences, denoted by $v_0^{(i)}v_1^{(i)} \ldots$, $i = 1, \ldots, n$. The initial state of the shift register is the zero state; the feedback symbol at each clock cycle is computed from the current state and the current information symbol according to the linear feedback. The output sequences are generated linearly from the state of the shift register and feedback symbol. The code sequence is obtained by interleaving the n output sequences. Figure 2.4 depicts an example model of a convolutional encoder. We will focus on convolutional encoders without feedback.

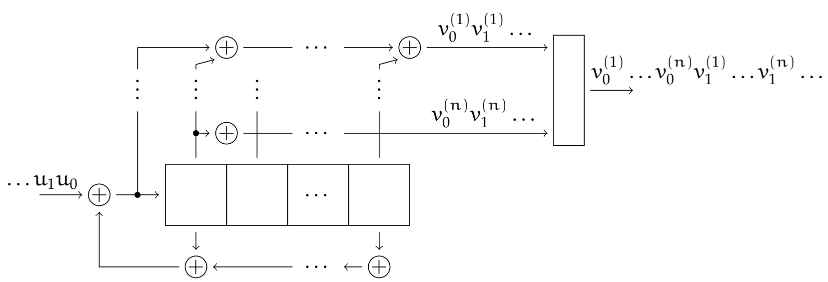


Figure 2.4. Example model of a convolutional encoder.

When the encoder has no feedback, we may write

$$\mathbf{v}_t = (v_t^{(1)} \ldots v_t^{(n)}) = f(u_t, u_{t-1}, \ldots, u_{t-m}),$$

where $f : \mathbb{F}_2^{m+1} \to \mathbb{F}_2^n$ is required to be linear. Due to the linearity of f, we can express $\mathbf{v}_t$ as

$$\mathbf{v}_t = u_t G_0 + u_{t-1} G_1 + \cdots + u_{t-m} G_m,$$

where each G_i, $0 \leqslant i \leqslant m$, is a binary matrix of size $1 \times n$. Then, the code sequence can be written as

$$\mathbf{v}_0\mathbf{v}_1 \cdots = (u_0u_1 \ldots)G,$$

where

$$G = \begin{pmatrix} G_0 & G_1 & \cdots & G_m & & 0 \\ & G_0 & G_1 & \cdots & G_m & \\ 0 & & \ddots & \ddots & \ddots & \ddots \end{pmatrix}.$$

The matrix G is the generator matrix and G_i, $0 \leqslant i \leqslant m$, are called the *generator submatrices*.

Let the *encoder state* at time t, denoted by σ_t, be the content of the shift register:

$$\sigma_t = u_{t-1}u_{t-2} \ldots u_{t-m}.$$

An encoder has, therefore, at most 2^m different states at time t. We now create a graph called *trellis*: consider all possible states σ_t, $t = 0, 1, \ldots$, as vertices in a graph, and add an edge between σ_t and σ_{t+1} if and only if there is an information symbol u_t that at time t updates the state σ_t to σ_{t+1}; also, label each edge from σ_t to σ_{t+1} with $\mathbf{v}_t$. For example, the encoder of rate $R = 1/2$ in Figure 2.5 has the corresponding trellis in Figure 2.6. A convolutional code may also be called a *trellis code* since the set of codeword sequences corresponds with the set of paths in the trellis.

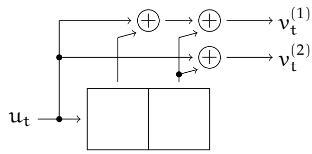


Figure 2.5. Example of a convolutional encoder of rate $R = 1/2$.

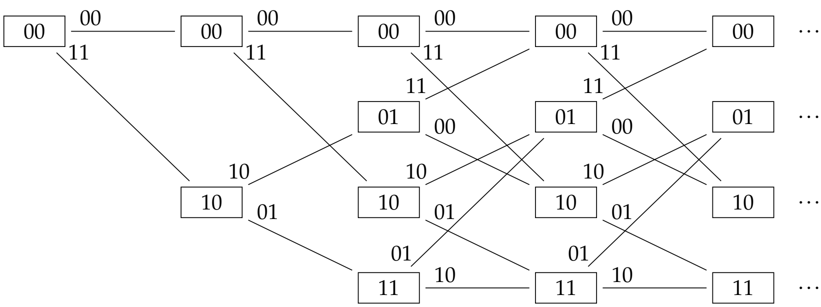


Figure 2.6. Example of a trellis code of rate $R = 1/2$.

Decoding - Viterbi algorithm

Suppose that a codeword $\mathbf{v}$ is sent through a BSC and the sequence $\mathbf{r}$ is received. The Viterbi algorithm [Vit67] may be used for efficient maximum likelihood decoding. The algorithm traverses the trellis and decides a value $\widehat{\mathbf{v}}$ for the codeword $\mathbf{v}$ that maximises the likelihood $P(\mathbf{r}|\mathbf{v})$. The *Viterbi metric* is given by

$$\mu(\mathbf{r}, \mathbf{v}) = \sum_t \mu_b(\mathbf{r}_t, \mathbf{v}_t) = \sum_t \sum_i \mu_s(r_t^{(i)}, v_t^{(i)}),$$

where μ_s, the *Viterbi symbol metric*, is a well-chosen function that counts the number of matched symbols in the current branch of the trellis (see for example [JZ15]). The quantity $\mu_b(\mathbf{r}_t, \mathbf{v}_t)$ is called the *Viterbi branch metric*. Algorithm 2.3 shows the Viterbi algorithm. Let $\mathbf{r}$ be a sequence of length N generated by a convolutional encoder of rate k/n, the time complexity of the Viterbi algorithm on $\mathbf{r}$ is $O(nN2^k)$.

Algorithm 2.3 Viterbi algorithm

1: Assign the Viterbi metric 0 to the initial node. Set $t = 0$.
2: **for** each node at level $t + 1$ **do**
3: For each predecessor at level t, find the sum of that predecessor's Viterbi metric and the branch metric of the connecting branch.
4: Get the maximum of the sums above and assign it to the current node. Label the node with the shortest path to it.
5: **end for**
6: If the end of the trellis is reached, set $\widehat{\mathbf{v}}$ to the value of a path with largest Viterbi metric and return $\widehat{\mathbf{v}}$. Otherwise, increment t and go to step 2.

2.5.4 Turbo codes

Here we briefly present *turbo codes* [BGT93] which informally may be defined as a "concatenation" of convolutional codes. The important feature of turbo codes is their iterative decoding techniques: multiple decoders operate on the received sequence to give *soft decisions*, i.e., estimates based on probabilities. We present turbo codes with characteristics relevant to this manuscript; we refer to [BGT93; JZ15] for a thorough treatment of these codes.

A turbo code encoder of rate $R = 1/(n+1)$ consists of n parallel identical convolutional encoders, referred to as *constituent encoders*, and n *permutors*. The encoder takes as input the length N information sequence $\mathbf{u} = u_0 \dots u_{N-1}$. Each of the permutors takes the information sequence and generates a permuted version $\mathbf{u}^{(i)} = \pi_i(\mathbf{u})$, where π_i is a permutation of N symbols. Each sequence $\mathbf{u}^{(i)}$ is then fed to the i-th convolutional encoder which produces the *parity* sequence $\mathbf{v}^{(i)} = v_0^{(i)} \dots v_{N-1}^{(i)}$. Let $\mathbf{v}^{(0)} = \mathbf{u}$. The code sequence is obtained as

$$\mathbf{v} = \mathbf{v}_0 \mathbf{v}_1 \dots \mathbf{v}_{N-1},$$

where $\mathbf{v}_t = v_t^{(0)} v_t^{(1)} \dots v_t^{(n)}$, $t = 0, \dots, N-1$. Figure 2.7 depicts a model of a turbo code encoder.

Decoding - BCJR algorithm

Suppose that a codeword $\mathbf{v}$ is sent through a BSC and the sequence $\mathbf{r} = \mathbf{r}_0 \mathbf{r}_1 \dots \mathbf{r}_{N-1}$ is received, where $\mathbf{r}_t = r_t^{(0)} r_t^{(1)} \dots r_t^{(n)}$, $t = 0, \dots, N-1$. Decoding for turbo codes

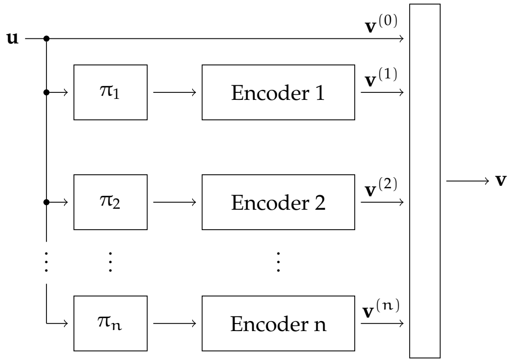


Figure 2.7. Model a turbo code encoder.

is done by iteratively executing n *a posteriori probability* (APP) decoders, one for each of the constituent codes. Each *constituent decoder* uses *a priori probabilities* $P(u_t = 0)$, $t = 0, \dots, N-1$, to produce the so-called *a posteriori probability* for all symbols u_t. At each iteration, a decoder computes the a posteriori probabilities and these are used as a priori probabilities for the next decoder. The probabilities computed at the last iteration are an approximation of $P(u_t = 0|\mathbf{r})$, $t = 0, \dots, N-1$; these probabilities are used to decide a value $\widehat{\mathbf{v}}$. Figure 2.8 shows a turbo decoder with two constituent codes.

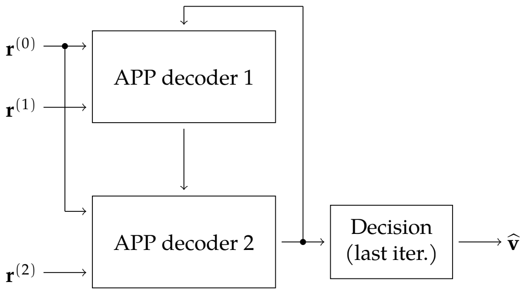


Figure 2.8. Example of a decoder for a turbo code with two constituent codes.

The BCJR algorithm [Bah+74] is one of the most popular APP decoding algorithms for convolutional codes. This algorithm is commonly chosen as the constituent decoder for turbo decoders. Given the input sequences $\mathbf{r}^{(0)}$ and $\mathbf{r}^{(i)}$, the BCJR algorithm computes the probabilities

$$P(u_t = 0|\mathbf{r}^{(0)}, \mathbf{r}^{(i)}), \quad t = 0, \dots, N-1.$$

We refer to the original source [Bah+74] or [JZ15] for the details of the algorithm.

CHAPTER 3

Cryptanalysis of the filter generator

The goal of a key recovery attack against a stream cipher is to get the secret key used to generate the given keystream. The secret key is used to initialise various components of the cipher. On devices employing linear feedback shift registers (LFSRs), the key is used to derive their initial states. A filter generator is an example of such devices.

In this chapter we will focus on key recovery attacks against the filter generator, i.e., to recover the initial state of the LFSR given the keystream. We present different known techniques to perform these type of attacks. In the context of this manuscript, fast correlation attacks and deterministic attacks are the most relevant. Algebraic attacks are not directly related with our work, however, we briefly present them due to their general relevance.

3.1 The device

Let $\mathbb{F}_q$ be a finite field. A filter generator is a keystream generator consisting of an LFSR over $\mathbb{F}_q$ of length n and a function $f : \mathbb{F}_q^\ell \to \mathbb{F}_q$, called the filtering function. The LFSR's feedback taps are defined by its degree-n primitive feedback polynomial

$$g(x) = c_0 - c_1 x - c_2 x^2 - \cdots - c_n x^n,$$

where $c_i \in \mathbb{F}_q$ and $c_0 = 1$. In general, $g(x)$ may not be primitive, however, we will assume it is so that the LFSR sequence has maximum period (see Section 2.4[1]). The state of the LFSR at time i is $S_i = (s_i, s_{i+1}, \ldots, s_{i+n-1})$. The inputs to f are the values in the LFSR cells with indices $k_1, \ldots, k_\ell$, where $0 \leqslant k_1 < \cdots < k_\ell \leqslant n-1$. Alternatively, the inputs may be specified by k_1 and the spacings $\gamma_1, \ldots, \gamma_{\ell-1}$, where $\gamma_i = k_{i+1} - k_i$.

[1]Section 2.4 focuses on binary LFSRs, i.e., over $\mathbb{F}_2$. However, many results there can be generalised to LFSRs over arbitrary finite fields; see for example [LN96].

The keystream symbol z_i at time i is computed as

$$z_i = f(s_{i+k_1}, \ldots, s_{i+k_\ell}).$$

We will focus on binary filter generators. Then, $g(x) \in \mathbb{F}_2[x]$, f is a Boolean function in ℓ variables and $s_i, z_i \in \mathbb{F}_2$ may be referred to as bits. Figure 3.1 shows a model of a filter generator.

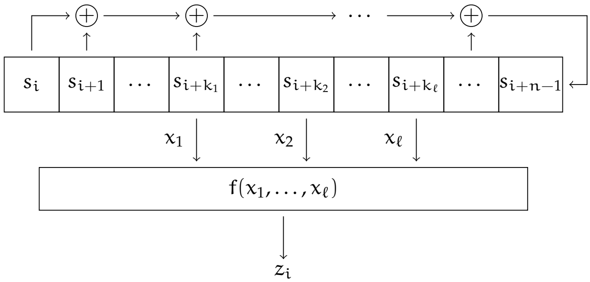


Figure 3.1. Model of a filter generator.

A key recovery attack against the filter generator consists in recovering the LFSR's initial state used to produce a given keystream. The feedback polynomial $g(x)$, the filtering function f and the indices $k_1, \ldots, k_\ell$ are considered to be known information. The available length-N keystream sequence will be denoted by $\{z_i\}_{1 \leqslant i \leqslant N}$. The output sequence of the LFSR will be denoted by $\{s_i\}_{1 \leqslant i \leqslant N}$.

3.2 Fast correlation attacks

Fast correlation attacks are a class of key recovery attacks against stream ciphers. They are applicable when the cipher's keystream generator employs LFSRs. The general idea is to exploit the correlation between the keystream and the sequence produced by one of the LFSRs. An important fact about nonlinear Boolean functions is that linear correlations always exist [Sie84]. For a filter generator, we assume the correlation between the LFSR sequence and the keystream is given by the correlation probability

$$p = \Pr(z_i = s_i) = 1/2 + \epsilon, \quad \epsilon \neq 0.$$

The quality of the correlation can be measured by $|\epsilon|$. If it is close to $1/2$, the correlation is good and the cipher is weak against fast correlation attacks. However, when $|\epsilon|$ is close to zero, the correlation is low and fast correlation attacks may be inefficient. More generally, the correlation may be given by $p = \Pr(z_i = s_{i+j_1} + \cdots + s_{i+j_w}) = 1/2 + \epsilon$.

Fast correlation attacks can be seen as a decoding problem. Any length-N sequence $s_1, \ldots, s_N$ produced by the LFSR is a codeword of a linear code $\mathcal{C}$ of length N and dimension n defined by $g(x)$. Recall that each s_i can be written in terms of the initial state $S_1 = (s_1, \ldots, s_n)$ as $s_i = \sum_{j=1}^{n} h_{i,j} s_j$ (see (2.3)). The $n \times N$ generator matrix of $\mathcal{C}$ is

$$G = \begin{pmatrix} h_{1,1} & h_{2,1} & \ldots & h_{N,1} \\ h_{1,2} & h_{2,2} & \ldots & h_{N,2} \\ \vdots & \vdots & \ddots & \vdots \\ h_{1,n} & h_{2,n} & \ldots & h_{N,n} \end{pmatrix} \tag{3.1}$$

and $(s_1, \ldots, s_N) = S_1 G$. The keystream $z_1, \ldots, z_N$ is the result of transmitting $s_1, \ldots, s_N$ over a binary symmetric channel (BSC) with crossover probability $1 - p$ (see Figure 2.3), and we have that

$$(z_1, \ldots, z_N) = (s_1, \ldots, s_N) + (e_1, \ldots, e_N),$$

where $e_i \in \mathbb{F}_2$ and $\Pr(e_i = 1) = 1 - p$. The attack recovers the initial state of the LFSR (i.e., the information symbols) by decoding the keystream relative to the code $\mathcal{C}$.

A *parity-check equation*, or simply a parity-check, is a polynomial

$$h(x) = 1 + x^{j_1} + \cdots + x^{j_{w-1}} \equiv 0 \bmod g(x).$$

The number of nonzero terms in a parity-check is its *weight*. If the weight is small, the parity-check is said to have *low-density* or *low-weight*. The LFSR sequence bits satisfy the linear relation

$$s_i + s_{i-j_1} + \cdots + s_{i-j_{w-1}} = 0.$$

Since the parity-checks hold for the LFSR sequence, evaluating these equations with the keystream bits leak information that can be used to recover the LFSR's initial state. Fast correlation attacks employ low-density parity-checks. The feedback polynomial and the amount of available keystream bits affect the number of low-density parity-checks that can be obtained.

Generally, fast correlation attacks consist of (i) obtaining low-density parity-checks and (ii) recovering the initial state/decoding using these equations. Obtaining the parity-checks may be considered the pre-computation phase of the attack, while recovering the initial state/decoding may be considered the main phase.

Some characteristics of the filter generator may render fast correlation attacks less efficient. The feedback polynomial should be primitive and with high degree. This is to achieve maximum period and high linear complexity of the LFSR sequence. Also, the weight of $g(x)$ should not be low. This will make it harder to find many useful parity-checks. The filtering function f should be balanced to guarantee good statistical properties on the keystream. Also, f should have high nonlinearity in order to reduce the correlation probability.

We briefly present the original fast correlation attack by Meier and Staffelbach [MS89] in Section 3.2.1. The initial algorithms in [MS89] have led to attacks employing different tools and approaches. Section 3.2.2 briefly describes some of the various contributions in fast correlation attacks. An overview of the many developments can be found in [Ågr+12] and [Mei11] as well. More recently, Todo et al. [Tod+18] presented an attack based on a "commutative" property of parity-checks. We briefly describe this attack in Section 3.2.3. Section 3.2.4 contains a summary of the time complexity and some reported results of the attacks described here.

3.2.1 The original idea

The original fast correlation attack was presented by Meier and Staffelbach in [MS89] as an improvement of the correlation attack by Siegenthaler [Sie85]. The latter is presented as a key recovery attack against the combination generator (see Figure 1.4a). It exploits the correlation between the output of the combining function f (i.e., the keystream) and one of the inputs x_i of f (the output sequence of the i-th LFSR). Let the combination generator consist of m LFSRs of length n_i, $i = 1, \ldots, m$. The worst case for a brute force attack has complexity $O(\prod_{i=1}^{m}(2^{n_i}-1))$ (since the all zero initial state is ignored). Siegenthaler's attack performs exhaustive search over all initial states for a target LFSR. The complexity is then reduced to $O(N\sum_{i=1}^{m}2^{n_i})$ [Sie85]. We refer to the original paper for the details and analysis. Fast correlation attacks attempt to recover the LFSR's initial state without trying all possible values.

Recall that the output sequence of the LFSR satisfies a relation

$$s_i = c_1 s_{i-1} + c_2 s_{i-2} + \cdots + c_n s_{i-n}, \tag{3.2}$$

where the coefficients c_j are determined by the feedback polynomial $g(x)$. Let w be the number of nonzero c_j, $1 \leqslant j \leqslant n$. Then, (3.2) can be rewritten as the following equation with $w+1$ terms:

$$\sum_{\{j:0\leqslant j\leqslant n, c_j\neq 0\}} s_{i-j} = 0. \tag{3.3}$$

Equation (3.3) is a parity-check of weight $w+1$. These parity-checks are valid for shifted versions of the LFSR sequence and can be written as

$$\sum_{\{j:0\leqslant j\leqslant n, c_j\neq 0\}} s_{t-j} = 0$$

for $t = k+1, \ldots, N$, where $k = \max(\{j : 1 \leqslant j \leqslant n, c_j \neq 0\})$. Therefore, each s_i appears in approximately $w+1$ parity-check equations. Every multiple of $g(x)$ yields a valid parity-check. Particularly, for exponents 2^k, we have that $g(x)^{2^k} = g(x^{2^k})$. Thus, we can get more parity-checks of weight $w+1$ if N is large enough. From the analysis in [MS89], the average number m of parity-checks that can be found for each s_i is

$$m \approx \log_2\left(\frac{N}{2n}\right)(w+1).$$

The m parity-checks for each s_i can be written as

$$\begin{aligned} s_i + b_i^{(1)} &= 0,\\ s_i + b_i^{(2)} &= 0,\\ &\vdots\\ s_i + b_i^{(m)} &= 0, \end{aligned}$$

where each $b_i^{(j)} = \sum_{k=1}^{w} s_{i_k}$ for some w different terms of the LFSR sequence. By using the terms of the keystream instead of those of the LFSR sequence in the equations above, we get

$$L_i^{(j)} = z_i + y_i^{(j)}, \quad j = 1, \ldots, m,$$

where $y_i^{(j)} = \sum_{k=1}^{w} z_{i_k}$ for the corresponding w different terms of the keystream. Notice that $L_i^{(j)}$ is not necessarily equal to 0.

Recall that the keystream is correlated to the LFSR sequence with probability

$$p = \Pr(z_i = s_i) \neq 1/2.$$

Let $s = \Pr(b_i^{(j)} = y_i^{(j)})$. This probability is a function of p and w, and $s = s(p, w)$ can be computed by the following recursion:

$$\begin{aligned} s(p, w) &= ps(p, w-1) + (1-p)(1-s(p, w-1)), \\ s(p, 1) &= p. \end{aligned} \tag{3.4}$$

For each z_i, some number h of the equations $L_i^{(j)}$ hold (are equal to 0) and $m-h$ do not hold (are equal to 1). This allows us to compute the *a posteriori* probability of $z_i = s_i$:

$$\begin{aligned} p^* &= \Pr\left(z_i = s_i \mid h \text{ equations } L_i^{(j)} \text{ hold}\right) \\ &= \frac{ps^h(1-s)^{m-h}}{ps^h(1-s)^{m-h} + (1-p)(1-s)^h s^{m-h}}. \end{aligned} \tag{3.5}$$

Algorithm 3.1 One-pass algorithm by Meier and Staffelbach

Input: The keystream $\{z_i\}_{1 \leqslant i \leqslant N}$ and the parity-check equations.

1: Compute p^* for each z_i.
2: Choose the n bits z_i having the highest values p^* and set this to be the reference guess I_0, i.e., $s_i = z_i$.
3: Try modifications of I_0 with Hamming distance $0, 1, 2, \ldots$ to find the initial state. For each modification of I_0, generate the keystream and check it against $\{z_i\}_{1 \leqslant i \leqslant N}$.

Algorithm 3.1 shows the one-pass algorithm (algorithm A) from [MS89]. I_0 is the reference guess obtained by selecting the n bits z_i with highest p^*. Assuming that exactly r bits in I_0 are incorrect, the maximum number of trials in step 3 is $A(n, r) = \sum_{i=0}^{r} \binom{n}{i} \leqslant 2^{H(\theta)n}$ where $H(\cdot)$ is the binary entropy function and $\theta = r/n$. The complexity of the algorithm is estimated to be $O(2^{cn})$ for $0 \leqslant c \leqslant 1$ [MS89], and under favourable conditions $c \ll 1$. Meier and Staffelbach show that the value of c decreases with large N and p far from $1/2$. This algorithm is suitable when the weight of $g(x)$ is small (< 10) and p is close to 0.75.

A second method proposed by Meier and Staffelbach makes several passes over the sequence $\{z_i\}_{1 \leqslant i \leqslant N}$. This approach updates the probabilities p^* and "flips" some of the keystream bits to recover the LFSR sequence and, therefore, the initial state.

Assume that the bits z_i have different probabilities p_i. Then, a generalisation of the equations to compute s and p^* is needed. For equations (3.4), this generalisation is

$$\begin{aligned} s(p_1, \ldots, p_w, w) &= p_w s(p_1, \ldots, p_{w-1}, w-1) + (1-p_w)(1 - s(p_1, \ldots, p_{w-1}, w-1)), \\ s(p_1, 1) &= p_1. \end{aligned}$$

For z_i, let $s^{(j)} = s(p_{i_1}, \ldots, p_{i_w}, w)$, where i_k are the indices of the w terms in $y_i^{(j)}$ for equation $L_i^{(j)}$. Also, let J be the set of indices j of all the equations $L_i^{(j)}$ and let H be the set of indices j of the equations $L_i^{(j)}$ that hold. Then, the generalisation of equation (3.5) is

$$\begin{aligned} p^* &= \Pr\left(z_i = s_i \mid h \text{ equations } L_i^{(j)} \text{ hold}\right) \\ &= \frac{p_i \prod_{j \in H} s^{(j)} \prod_{j \in J \setminus H} (1 - s^{(j)})}{p_i \prod_{j \in H} s^{(j)} \prod_{j \in J \setminus H} (1 - s^{(j)}) + (1-p_i) \prod_{j \in H} (1 - s^{(j)}) \prod_{j \in J \setminus H} s^{(j)}}. \end{aligned}$$

Algorithm 3.2 shows the iterative algorithm (algorithm B) from [MS89]. It uses two thresholds: p_{thr} and N_{thr}. We present the equations to compute the two thresholds; we refer to the original paper for the details on how to obtain these equations. Let

$$I(p, m, h) = \left(\sum_{i=0}^{h} \binom{m}{i} (1-p)(1-s)^i s^{m-i} \right) - \left(\sum_{i=0}^{h} \binom{m}{i} p s^i (1-s)^{m-i} \right)$$

and let h_{max} be the value of h that maximises $I(p, m, h)$ for the given p and m. Then

$$p_{thr} = \frac{1}{2} \left(p^*(p, m, h_{max}) + p^*(p, m, h_{max} + 1) \right)$$

and

$$N_{thr} = \left(\sum_{i=0}^{h} \binom{m}{i} \left(p s^i (1-s)^{m-i} + (1-p)(1-s)^i s^{m-i} \right) \right) N.$$

p_{thr} is used to determine the number of keystream bits that should be flipped. When this number is larger or equal to N_{thr}, the corresponding bits z_i are flipped, otherwise, the computation of p^* is iterated. The complexity of the algorithm is estimated to grow linearly with the length of the LFSR, i.e., is of order $O(n)$ [MS89]. This algorithm is suitable when the weight of $g(x)$ is small (< 10, preferably 2 or 4) even when p is very close to 0.5.

Algorithm 3.2 Iterative algorithm by Meier and Staffelbach

Input: The keystream $\{z_i\}_{1 \leqslant i \leqslant N}$ and the parity-check equations.
Let $\alpha (\approx 5)$ be the number of iterations.

Compute the thresholds p_{thr} and N_{thr}.
for $r = 1, 2, \ldots$ **do**
  for $i = 1, \ldots, \alpha$ **do**
    For each z_i, compute p^* and assign $p_i = p^*$.
    Compute $N_w = |\{i \mid p_i < p_{thr}\}|$.
    if $N_w \geqslant N_{thr}$ **then**
      break
    end if
  end for
  Complement all bits z_i with $p_i < p_{thr}$ and reset all probabilities p_i to p.
  if If all bits z_i satisfy the parity-checks **then**
    break
  end if
end for
Terminate with $s_i = z_i$, $i = 1, \ldots, N$.

The efficiency of both algorithms depend on the weight of the parity-checks, the correlation probability p and N. Given a fixed value of p, the algorithms present better performance when low-density parity-checks are employed. We refer to the original paper [MS89] for a more detailed description and analysis of the algorithms.

3.2.2 Some techniques for fast correlation attacks

We now present some of the various contributions in fast correlation attacks. Some of them introduce methods to obtain parity-checks from general feedback polynomials.

The attack by Meier and Staffelbach requires $g(x)$ to have low weight. Recall that $g(x)$ has degree n, the state of the LFSR at time i is $S_i = (s_i, s_{i+1}, \dots, s_{i+n-1})$, the LFSR output sequence is $s_1, \dots, s_N$ and the keystream sequence is $z_1, \dots, z_N$. Also, a parity-check is a low-weight multiple $h(x)$ of $g(x)$ such that $h(0) = 1$. Given the keystream, the maximum admissible degree of the parity-checks is $N - 1$. Finally, recall that the correlation probability between the keystream and the LFSR sequence is $p = \Pr(z_i = s_i) = 1/2 + \epsilon$, $\epsilon \neq 0$.

Finding parity-checks - A method by Golić

In [MS89], Meier and Staffelbach briefly discuss a method to find parity-checks from arbitrary polynomials. In [Gol96a], however, Golić points out an erroneous assumption in that method and introduces a new one which finds all parity-checks with weight at most $2k+1$. Golić's method is presented in Algorithm 3.3. A b match means two different residues such that

$$(x^{i_1} + \cdots + x^{i_k}) + (x^{i'_1} + \cdots + x^{i'_k}) \equiv b \pmod{g(x)}.$$

For $b = 0$, the parity-checks may be divided by a suitable power of x to satisfy $h(0) = 1$.

Algorithm 3.3 Finding parity-checks - Golić

Input: $g(x)$, maximum degree m and a positive integer k.
Output: All polynomial multiples of $g(x)$ of degree at most m with weight at most $2k + 1$.

1: Compute and store all residues $x^i \bmod g(x)$, $i = 1, \dots, m$.
2: Compute and store the residues $x^{i_1} + \cdots + x^{i_k} \bmod g(x)$ for all $\binom{m}{k}$ combinations $1 \leqslant i_1 < \cdots < i_k \leqslant m$.
3: Sort the residues above and find the 0 and 1 matches.

Let $m = N$ and $S = \binom{N}{k}$. Golić's algorithm has space complexity $O(S)$ and time complexity $O(S \log S)$.

Attack based on convolutional codes

Johansson and Jönsson [JJ99b] model the attack as a decoding problem. Recall that, in this setting, $s_1, \dots, s_N$ is a codeword and $z_1, \dots, z_N$ is the received sequence at the other end of a BSC with crossover probability $1 - p$. Let G_{LFSR} be the $n \times N$ generator matrix (3.1) of the linear code obtained from the LFSR. Notice that the initial state of the LFSR appear as the first n symbols of the codeword. Hence, G_{LFSR} is in systematic form and can be written

$$G_{\mathrm{LFSR}} = (I_n Z), \tag{3.6}$$

where I_n is the $n \times n$ identity matrix.

Let w be a small positive integer and $B > 0$ be the *memory* parameter. Johansson and Jönsson find parity-checks that involve a symbol s_i, $i \geqslant B+1$, a linear combination of the B previous symbols $s_{i-1}, \dots, s_{i-B}$ and at most w other symbols. These parity-checks are written

$$s_i + \sum_{j=1}^{B} c_i s_{i-j} + \sum_{j=1}^{\leqslant w} s_{i_j} = 0. \tag{3.7}$$

To exemplify a way to get parity-checks, rewrite the generator matrix as

$$G_{\mathrm{LFSR}} = \begin{pmatrix} I_{B+1} & Z_{B+1} \\ 0_{n-B-1} & Z_{n-B-1} \end{pmatrix}.$$

Parity-checks of weight w outside of the first $B+1$ positions, can be obtained by finding linear combinations of up to w columns of the sub-matrix Z_{n-B-1} that yield the zero vector. Particularly, when $w = 2$, sorting the columns of G_{LFSR} by the last $n - B - 1$ entries and finding collisions will yield parity-checks.

Assume m parity-checks have been found for s_{B+1}. Due to the structure of the LFSR, these parity-checks are valid for any other index i by shifting the symbols involved in the parity-checks. Hence, the parity-checks are written

$$s_i + \sum_{j=1}^{B} c_{j,1} s_{i-j} + b_1 = 0,$$
$$\vdots$$
$$s_i + \sum_{j=1}^{B} c_{j,m} s_{i-j} + b_m = 0,$$

where $b_k = \sum_{j=1}^{\leqslant w} s_{i_j}$ is the sum of at most w different terms.

Johansson and Jönsson create a convolutional encoder of rate $R = 1/(m+1)$ and memory B whose generator matrix is

$$G = \begin{pmatrix} G_0 & G_1 & \cdots & G_B & & 0 \\ & G_0 & G_1 & \cdots & G_B & \\ 0 & & \ddots & \ddots & \ddots & \ddots \end{pmatrix},$$

where $G_0 = \begin{pmatrix} 1 & 1 & \cdots & 1 \end{pmatrix}$ and $G_j = \begin{pmatrix} 0 & c_{j,1} & c_{j,2} & \cdots & c_{j,m} \end{pmatrix}$ for $j = 1, \ldots, B$. Then, the codeword sequence is

$$v = \cdots v_i v_{i+1} \cdots = \cdots v_i^{(0)} \cdots v_i^{(m)} v_{i+1}^{(0)} \cdots v_{i+1}^{(m)} \cdots,$$

where v_i has the form $v_i^{(0)} v_i^{(1)} \ldots v_i^{(m)} = s_i G_0 + s_{i-1} G_1 + \cdots + s_{i-B} G_B$. The received sequence can be constructed from $z_1, \ldots, z_N$ as

$$r = \cdots r_i r_{i+1} \cdots = \cdots r_i^{(0)} \cdots r_i^{(m)} r_{i+1}^{(0)} \cdots r_{i+1}^{(m)} \cdots,$$

where $r_i^{(0)} = z_i$ and $r_i^{(j)} = \sum_{k=1}^{\leqslant w} z_{i_k}$ for $j = 1, \ldots, m$.

To recover the initial state, it suffices to correctly decode n consecutive information symbols. The Viterbi algorithm [Vit67] is used for decoding. Johansson and Jönsson execute the algorithm over $J > n$ information symbols. Algorithm 3.4 summarises the technique in [JJ99b] using $w = 2$ and $J = n + 10B$. The time complexity of the attack is $O\left(Jm2^B\right)$. In the original paper, the performance of the algorithm is analysed with some numerical results. If B is small, the capability of the method decreases. For a given B, the efficiency of the attack depends on p and N.

Algorithm 3.4 Viterbi decoding by Johansson and Jönsson

Input: Set of parity-checks, keystream sequence.
Output: Initial state of the LFSR.

1: Create the received sequence r.
2: Let $\Pr(v_i^{(0)} = r_i^{(0)}) = 1 - p$ and $\Pr(v_i^{(j)} = r_i^{(j)}) = (1-p)^2 + p^2$, for $i = B+1$ to $n + 10B$.
3: For each possible state (initial value) s, let $\log \Pr(s = (z_1, z_2, \ldots, z_B))$ be the initial metric. Use the Viterbi algorithm to decode r from $i = B$ to J. Compute the initial state of the LFSR from the decoded information sequence $(\hat{s}_{5B+1}, \ldots, \hat{s}_{5B+n})$.

Attack based on turbo codes

Johansson and Jönsson [JJ99a] extend their own ideas from the attack above by using turbo codes. Recall that turbo codes "concatenate" convolutional codes. For decoding, the main idea is to use an *a posteriori* probability (APP) decoder (i.e., an algorithm that outputs *a posteriori* probabilities for all information symbols) for each constituent code. The output of the first APP decoder is used as *a priori* probabilities for the second APP decoder. These are then taken as *a priori* for the next decoder and so on. This process continues until convergence or until reaching a maximum number of iterations.

Johansson and Jönsson use $M \geqslant 2$ different convolutional codes in their attack. The first one is obtained as in the attack above using convolutional codes. The subsequent codes are obtained by permuting index positions of the first one in the interval $B + 1 \ldots J$, for some value J; Johansson and Jönsson use $J = n + 10B$, where B is the memory of the convolutional encoders. For the first code, the authors use parity-checks (3.7). For the other codes, new parity-checks must be re-computed. This increases the time complexity during pre-computation, however, it is not a big problem when the weight is $w = 2$. The BCJR algorithm [Bah+74] is used as the APP decoder. The time complexity of the attack is $O\left(6MJm2^B\right)$, where m is the number of parity-check equations. We refer the reader to [JJ99a] for in-depth details on the re-computation of parity-checks and decoding for this attack.

Attack based on reducing the dimension of the underlying code

In the formulation as a decoding problem, fast correlation attacks associate a binary linear $[N, n]$-code $\mathcal{C}$ to the LFSR. Chepyzhov et al. [CJS01] employ instead a new binary linear $[N_2, k]$-code, where $k < n$ and N_2 is defined below. The k information symbols of this code coincide with the first k symbols of the initial state of the LFSR. Hence, by decoding this new code we can recover k symbols of the LFSR's initial state. The remaining symbols can be recovered by repeating the process for the next k bits.

Let each s_i be written in terms of the initial state (see (2.3)):

$$s_i = h_{i,1}s_1 + h_{i,2}s_2 + \cdots + h_{i,n}s_n, \quad i = 1, \ldots, N. \tag{3.8}$$

We search in (3.8) for all N_2 distinct pairs of equations such that

$$h_{i,k+1} = h_{j,k+1},\ h_{i,k+2} = h_{j,k+2},\ \ldots,\ h_{i,n} = h_{j,n},\ 1 \leqslant i \neq j \leqslant N.$$

For all these pairs, the sum $s_i + s_j$ can be written as

$$s_i + s_j = (h_{i,1} + h_{j,1})s_1 + (h_{i,2} + h_{j,2})s_2 + \cdots + (h_{i,k} + h_{j,k})s_k.$$

Notice that this sum is a linear combination of the first k symbols $s_1, \dots, s_k$ only and does not depend on $s_{k+1}, \dots, s_n$. Let $\{i_1, j_1\}, \{i_2, j_2\}, \dots, \{i_{N_2}, j_{N_2}\}$ be the indices of all the pairs. Then, we get a new $[N_2, k]$-code $\mathcal{C}_2$ whose information symbols are $s_i, \dots, s_k$. The corresponding codewords are computed as

$$(s_{i_1} + s_{j_1}, s_{i_2} + s_{j_2}, \dots, s_{i_{N_2}} + s_{j_{N_2}}),$$

the received word is

$$(z_{i_1} + z_{j_1}, z_{i_2} + z_{j_2}, \dots, z_{i_{N_2}} + z_{j_{N_2}})$$

and the crossover probability of the corresponding BSC is

$$p_2 = 2p(1-p) = 1/2 - 2\epsilon^2.$$

To recover $s_1, \dots, s_k$, we need to decode $\mathcal{C}_2$ with the crossover probability p_2. Decoding is done through exhaustive search of all 2^k codewords of $\mathcal{C}_2$ and the one closest to the received sequence is the decoded sequence. The authors also present a generalisation where, instead of finding pairs, they search for sums of w equations that depend on the target k information symbols only. Then, the crossover probability is given by $p_w = 1/2 - 2^{w-1}\epsilon^w$. Experimental results show that using $w = 3, 4$ improves the attack. The time complexity of this method is $O\left(2^k k \frac{2}{(2\epsilon)^{2w}}\right)$.

Attack based on low-density parity-check codes

Canteaut and Trabbia [CT00] find parity-checks with weight w as shown in Algorithm 3.5. The space complexity is $O(N)$ and the time complexity is $O\left(\binom{N-1}{w-2}\right)$.

Algorithm 3.5 Finding parity-checks - Canteaut and Trabbia

Input: $g(x)$, weight w.
Output: All polynomial multiples of $g(x)$ with weight w and degree $< N$.

1: Compute all residues $q_i(x) = x^i \bmod g(x)$, $i = 1, \dots, N-1$. Store them in a table defined by
$$T[a] = \{i \mid q_i(x) = a\}$$
for some a.
2: **for** each set of $w-2$ elements of $\{1, \dots, N-1\}$ **do**
3: Compute $A = 1 + q_{i_1}(x) + \cdots + q_{i_{w-2}}(x)$.
4: For $j \in T[A]$, $1 + x^{i_1} + \cdots + x^{i_{w-2}} + x^j$ is a multiple of $g(x)$ with weight w.
5: **end for**

Using the parity-checks, the initial state is recovered by a decoding technique based on Gallager's iterative algorithm [Gal62]. This method employs the probability $\Pr(s_i = 1 \mid \{z_j\}_{1 \leqslant j \leqslant N}, \mathcal{S})$, where $\mathcal{S}$ is the event that all parity-checks involving s_i are satisfied. Algorithm 3.6 shows the decoding procedure in [CT00]. Canteaut and Trabbia apply their method using parity-checks of weight 4 and 5. The time complexity of the attack is $O(5(w-1)mN)$, where m is the number of parity-checks used. Given a weight w, the required keystream length N for successful recovery of the initial state is a function of w and p.

Algorithm 3.6 Decoding by Canteaut and Trabbia

Input: Set of parity-checks, keystream sequence.
Output: Initial state of the LFSR.

1: For $i = 0$ to $N-1$, initialise $L[i] = \log \frac{p}{1-p}$.
2: **repeat**
3: **for** $i = 0$ **to** $N-1$ **do**
4: $L'[i] = (-1)^{z_i} L[i]$.
5: For every parity-check $s_i + \sum_{j \in J} s_j$ involving s_i,

$$L'[i] = L'[i] + \left(\prod_{j \in J} (-1)^{z_j} \right) \min_{j \in J} L[j].$$

6: $z_i = \begin{cases} 0 & \text{if } L'[i] > 0 \\ 1 & \text{otherwise} \end{cases}$, and $L[i] = |L'[i]|$.
7: **end for**
8: **until** convergence

Attack based on reconstruction of linear polynomials

Johansson and Jönsson [JJ00] model the problem of recovering the LFSR's initial state as a polynomial reconstruction problem. Let each s_i be written in terms of the initial state $(s_1, \ldots, s_n)$ (see (2.3)):

$$s_i = h_{i,1}s_1 + h_{i,2}s_2 + \cdots + h_{i,n}s_n, \quad i = 1, \ldots, N.$$

Define the *initial state polynomial* $U(x)$ as

$$U(x) = U(x_1, x_2, \ldots, x_n) = s_1x_1 + s_2x_2 + \cdots + s_nx_n.$$

Then, each s_i can be expressed as the evaluation of $U(x)$ as

$$s_i = U(h_{i,1}, h_{i,2}, \ldots, h_{i,n}).$$

Let $(e_1, \ldots, e_n)$ be a noise vector, where $e_i \in \mathbb{F}_2$ are independent random variables with $\Pr(e_i = 0) = 1/2 + \epsilon$. Johansson and Jönsson model the correlation between the LFSR and keystream sequences as

$$(z_1, \ldots, z_n) = (s_1, \ldots, s_n) + (e_1, \ldots, e_n),$$

getting

$$(z_1, z_2, \ldots, z_N) = (U(X_1) + e_1, U(X_2) + e_2, \ldots, U(X_N) + e_N),$$

where X_i are known n-tuples for all $1 \leqslant i \leqslant N$. The attack is then reformulated to determining the unknown polynomial $U(x)$ given the noisy observations $(z_1, \ldots, z_N)$ of $U(x)$ evaluated at the different points $X_1, \ldots, X_N$.

To solve the polynomial reconstruction problem, Johansson and Jönsson base their method on the work of Goldreich et al. [GRS95]. However, a direct application of the latter is not possible due to some restrictions in the inputs. To overcome this, Johansson and Jönsson employ sums of the noisy observations. The noise is such that

$$\Pr(z_i = U(X_i)) = 1/2 + \epsilon, \quad 1 \leqslant i \leqslant N,$$

where X_i are known random vectors of length n for all $1 \leqslant i \leqslant N$. Since $U(x)$ is linear, we have that

$$\begin{aligned}\Pr(z_i + z_j = U(X_i + X_j)) &= \Pr(z_i + z_j = U(X_i) + U(X_j)) \\ &= 1/2 + 2\epsilon^2,\end{aligned}$$

and in general

$$\Pr\left(\sum_{j=1}^{w} z_{i_j} = U\left(\sum_{j=1}^{w} X_{i_j}\right)\right) = 1/2 + 2^{w-1}\epsilon^w.$$

Let $\hat{z} = \sum_{j=1}^{w} z_{i_j}$ and $\hat{X} = \sum_{j=1}^{w} X_{i_j}$. Any $\hat{z}$ is a noisy observation of $U(\hat{X})$ and we can write

$$U(\hat{X}) = \hat{z} + e,$$

where $e \in \mathbb{F}_2$ is a random variable with $\Pr(e = 0) = 1/2 + 2^{w-1}\epsilon^w$. Algorithm 3.7 shows the technique in [JJ00].

Algorithm 3.7 Polynomial reconstruction by Johansson and Jönsson

Input: Keystream $z_1, \dots, z_N$, $(X_1, \dots, X_N)$ and constants w, k and m.
Output: Target k values of the initial state of the LFSR.

Pre-computation

1: Select m different vectors $V_1, \dots, V_m$ of length $(n - k)$.
2: **for** each V_i **do**
3: Find all linear combinations $\hat{X}(i) = \sum_{j=1}^{w} X_{i_j}$ such that $\hat{X}(i) = (\hat{x}_1, \dots, \hat{x}_k, V_i)$, for arbitrary values $\hat{x}_1, \dots, \hat{x}_k$, and store $(\hat{X}(i), \hat{z}(i))$, where $\hat{z}(i) = \sum_{j=1}^{w} z_{i_j}$. Let m_i be the number of such pairs.
4: **end for**

Computation

5: **for all** 2^k possible values $(\hat{s}_1, \dots, \hat{s}_k)$ **do**
6: For each V_i, iterate over all m_i stored pairs $(\hat{X}(i), \hat{z}(i))$ to compute the number `num` of times that

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

Update `dist` $=$ `dist` $+ (m_i - 2 \cdot$ `num`$)^2$.
7: If `dist` is the highest value so far, store $(\hat{s}_1, \dots, \hat{s}_k)$ and set `dist` $= 0$.
8: **end for**
9: Output $(\hat{s}_1, \dots, \hat{s}_k)$ with the highest value `dist`.

Write $\hat{X}(i) = (\hat{x}_1, \dots, \hat{x}_k, v_{k+1}, \dots, v_n)$. Since $U(\hat{X}(i)) = \hat{z}(i) + e$, we have that

$$\sum_{j=1}^{k} s_j \hat{x}_j + \sum_{j=k+1}^{n} s_j v_j = \hat{z}(i) + e$$

and rewrite it as

$$\sum_{j=1}^{k} (s_j + \hat{s}_j)\hat{x}_j + \sum_{j=k+1}^{n} s_j v_j + e = \sum_{j=1}^{k} \hat{s}_j \hat{x}_j + \hat{z}(i). \tag{3.9}$$

for some value of $(\hat{s}_1, \dots, \hat{s}_k)$. Notice that $W = \sum_{j=k+1}^{n} s_j v_j$ in (3.9) is a fixed binary random variable for all $\hat{X}(i)$'s. We have two cases:

- *Correct hypothesis* $(s_1, \dots, s_k = \hat{s}_1, \dots, \hat{s}_k)$. Here, $\sum_{j=1}^{k}(s_j + \hat{s}_j)\hat{x}_j = 0$ and the probability of the right hand side of (3.9) to be zero is $\Pr(W + e = 0) = 1/2 \pm 2^{w-1}\epsilon^{w}$ depending on whether $W = 0$ or $W = 1$. Then, `num` has a binomial distribution $\mathsf{Bin}(m_i, p)$ with $p = 1/2 \pm 2^{w-1}\epsilon^{w}$.

- *Incorrect hypothesis* $(s_1, \dots, s_k \neq \hat{s}_1, \dots, \hat{s}_k)$. In this case, $\sum_{j=1}^{k}(s_j + \hat{s}_j)\hat{x}_j \neq 0$. This results in `num` having a binomial distribution $\mathsf{Bin}(m_i, p)$ with $p = 1/2$.

In order to distinguish the two distributions, a square distance $(m_i - 2 \cdot \mathtt{num})^2$ is used, which considers the difference of the number of times $\sum_{j=1}^{k} \hat{s}_j \hat{x}_j = \hat{z}(i)$ holds and the number of times it does not hold. Let $m_i = m_1$ for all i. The time complexity of the pre-computation part is $O\left(N^{\lceil w/2 \rceil}\right)$ and $O\left(m m_1 k 2^k\right)$ for the main computation. The authors also present an algorithm which selects a candidate $(\hat{s}_1, \dots, \hat{s}_k)$ and extends it to obtain a set of surviving candidates. This other algorithm employs the same pre-computation and has time complexity $O\left(m m_1 k 2^{nc}\right)$, $c < 1$, for the main computation. Given w, k and m, the success of the attack depends on p; see [JJ00] for further details.

Attack based on list decoding

The attack by Mihaljević et al. [MFI02] is based on *list decoding* [Eli57], where the decoder outputs a list of possible candidate codewords. Decoding is referred to as *list-of-ℓ decoding* when the list is composed of ℓ candidates. The authors perform a partial exhaustive search on the first B information bits and target to decode $D > n - B$ bits.

Given the partial exhaustive search, the first B bits of the initial state are assumed to be known and parity-checks can include any number of those bits. Mihaljević et al. obtain parity-checks as in another attack proposed by the same authors [MFI01]; we do not show that method here. The complexity of finding parity-checks is $O\left(D\binom{N-n}{2}\right)$. If employing memory proportional to $O(N-n)$, the complexity becomes $O(D(N-n))$. The number of parity-checks involving a symbol s_i is expected to be $m = s^{B-n}\binom{N}{2}$.

Algorithm 3.8 Attack based on list decoding by Mihaljević et al.

Input: Keystream $z_1, \dots, z_N$, set of parity-checks, number of bits M for correlation check, correlation threshold T and parameters B and D.
Output: Initial state of the LFSR.

1: Choose a new value $(\hat{s}_1, \dots, \hat{s}_B)$ for the first B bits of the initial state. If no new value can be chosen, terminate.
2: Evaluate the parity-checks for the target bits s_i, $i = B+1, \dots, D$.
3: Create two most-reliable estimators as follows:

- Select the $n - B$ positions corresponding to the bits with the most satisfied parity-checks. Assume they are correct and compute the information bits $\hat{s}_{B+1}, \dots, \hat{s}_n$.
- Select the $n - B$ positions corresponding to the bits with the most unsatisfied parity-checks. Assume they are incorrect and compute the information bits $\hat{s}_{B+1}, \dots, \hat{s}_n$.

4: For each of the estimators above, generate $\hat{s}_1, \dots, \hat{s}_M$ and compute the distance $S = \sum_{i=1}^{M}(\hat{s}_i + z_i)$. If $S \leqslant T$, return $(\hat{s}_1, \dots, \hat{s}_n)$ as the initial state, otherwise, goto Step 1.

Algorithm 3.8 summarises the attack to recover the initial state. The attack employs list-of-2^{B+1} decoding to get a list of candidates and the distance $S = \sum_{i=1}^{M}(\hat{s}_i + z_i)$ to select the true candidate. Mihaljević et al. employ parity-checks of weight 3. Let m be the number of parity-checks and M a parameter specifying the number of bits needed for correlation check. The time complexity is $O\left(2^B((D-B)m + (M-n)\mathrm{wt}(g))\right)$.

Improving the search for parity-checks and their evaluation

Chose et al. [CJM02] presented algorithmic improvements for fast correlation attacks. Particularly, a new method for finding parity-checks and an application of the Fourier-Hadamard transform to efficiently evaluate them with the keystream bits. As in the attack by Mihaljević et al. above, the first B bits of the LFSR's initial state are recovered by exhaustive search and the rest by decoding some $D > n - B$ target bits.

The weight-w parity-checks associated to a symbol s_i contain other $w-1$ output bits and a combination of the B guessed bits; they are written as follows:

$$s_i = s_{i_1} + \cdots + s_{i_{w-1}} + \sum_{j=1}^{B} c_{i,I,j} s_j,$$

where $I = [i_1, \ldots, i_{w-1}]$. Let $s = (s_1, \ldots, s_n)$, $A(s) = \sum_{j=1}^{n} a_j s_j$ for some fixed constants a_j, and $w' \in \{w, w-1\}$. Parity-checks are obtained using Algorithm 3.9. This match-and-sort algorithm finds equations of the form

$$A(s) = s_{i_1} + \cdots + s_{i_{w'}} + \sum_{j=1}^{B} c_{i,I,j} s_j.$$

In the first two loops, we compute the formal sum of l_2 bits and l_4 bits in terms of the initial state, and store those expressions in tables U and V, respectively. Let S be a subset of the $n - B$ unknown bits (i.e., initial state bits not recovered by brute force). We find matching indices u from U and formal sums of $A(s)$ and l_1 bits (in terms of the initial state), and store them in table C. These matches are required to be equal to a value s' in the subset S. Similarly, we search for matches in the subset S between indices v from V and formal sums of l_3 bits. For each match, we find collisions within indices c from table C, this time on the full set of $n - B$ unknown bits. The authors suggest to use $\frac{w'}{4}\log_2 N$ bits for S to maximise memory usage. When w is odd, $w' = w - 1$, $A(s)$ represents one of the target bits s_i and the algorithm is executed for each of the D target bits. When w is even, $w' = w$, $A(s) = 0$ and one application of the algorithm yields the parity-checks for all bits, not only the targeted ones. The expected number of parity-checks for a symbol s_i is $m \approx 2^{B-n}\binom{N}{w-1}$. The time complexity is $O\left(N^{\lceil w/2 \rceil}\log N\right)$ and the space complexity is $O\left(N^{\lfloor (w+1)/4 \rfloor}\right)$.

Now, let $B = B_1 + B_2$. Then, all parity-checks involving a given z_i can be rewritten as

$$z_i = \underbrace{z_{i_1} + \cdots + z_{i_{w-1}} + \sum_{j=1}^{B_1} c_{i,I,j} s_j}_{t^1_{i,I}} + \underbrace{\sum_{j=B_1+1}^{B_1+B_2} c_{i,I,j} s_j}_{t^2_{i,I}}.$$

(Since keystream bits appear in the equation above, equality holds with some probability.) Let $c_2 = (c_{2,1}, \ldots, c_{2,B_2})$ be a vector of length B_2 and group the parity-checks

Algorithm 3.9 Finding parity-checks - Chose et al.

Input: $g(x)$, weight w' and $A(s)$.
Output: Parity-checks of weight w' for $A(s)$.

Evenly split w' as $w' = l_1 + l_2 + l_3 + l_4$ with $l_1 \geqslant l_2$ and $l_3 \geqslant l_4$.
for all possible values of l_2 bits $(j_1, \ldots, j_{l_2})$ **do**
  Formally compute $s_{j_1} + \cdots + s_{j_{l_2}} = \sum_{k=1}^{n} u_k s_k$ and let $u = (u_1, \ldots, u_n)$.
  Set $U[u] = \{j_1, \ldots, j_{l_2}\}$.
end for
for all possible values of l_4 bits $(m_1, \ldots, m_{l_4})$ **do**
  Formally compute $s_{m_1} + \cdots + s_{m_{l_4}} = \sum_{k=1}^{n} v_k s_k$ and let $v = (v_1, \ldots, v_n)$.
  Set $V[v] = \{m_1, \ldots, m_{l_4}\}$.
end for
Let S be a subset of the $n - B$ unknown bits and π_S be the projection on these bits.
for all possible values s' of the bits in S **do**
  for all possible values of l_1 bits $(i_1, \ldots, i_{l_1})$ **do**
    Formally compute $A(s) + s_{i_1} + \cdots + s_{i_{l_1}} = \sum_{k=1}^{n} c_k s_k$ and let $c = (c_1, \ldots, c_n)$.
    Search for u in U such that $\pi_S(u + c) = s'$.
    Set $C[u + c] = \{i_1, \ldots, i_{l_1}, j_1, \ldots, j_{l_2}\}$.
  end for
  for all possible values of l_3 bits $(k_1, \ldots, k_{l_3})$ **do**
    Formally compute $s_{k_1} + \cdots + s_{k_{l_3}} = \sum_{k=1}^{n} d_k s_k$ and let $d = (d_1, \ldots, d_n)$.
    Search for v in V such that $\pi_S(v + d) = s'$ and let $t = v + d$.
    Search for c in C such that $\pi_{n-B}(c + t) = 0$.
    Output $\{A(s), i_1, \ldots, i_{l_1}, j_1, \ldots, j_{l_2}, k_1, \ldots, k_{l_3}, m_1, \ldots, m_{l_4}, c + t\}$.
  end for
end for

into sets

$$M_i(c_2) = \{I \mid c_{i,I,B_1+j} = c_{2,j}, j = 1, \ldots, B_2\},$$

i.e., $M_i(c_2)$ contains parity-checks that depend on the same last B_2 guessed variables. If X_1 is a fixed guessed value for the first B_1 bits of the initial state of the LFSR, $t^1_{i,I}$ can be computed. Define the function $f_i(c_2)$ as

$$f_i(c_2) = \sum_{I \in M_i(c_2)} (-1)^{t^1_{i,I}}.$$

The Fourier-Hadamard transform of $f_i(c_2)$ is

$$\begin{aligned} \widehat{f_i}(X_2) &= \sum_{c_2} f_i(c_2)(-1)^{c_2 \cdot X_2} = \sum_{c_2} \left(\sum_{I \in M_i(c_2)} (-1)^{t^1_{i,I}} \right) (-1)^{c_2 \cdot X_2} \\ &= \sum_{I} (-1)^{t^1_{i,I} + t^2_{i,I}}. \end{aligned}$$

Hence, $\widehat{f_i}(X_2)$ gives the difference between the predicted number of zeros and ones for z_i when the B guessed bits have value (X_1, X_2). Therefore, a single computation of the Fourier-Hadamard transform evaluates the difference for all possible values X_2 of the B_2 bits. Choosing $B_2 = \log_2 m$, the complexity of evaluating the parity-checks is $O(2^B D \log_2 m)$. Meanwhile, the straightforward approach has complexity $O(2^B D m)$.

Chose et al. employ a variation of the decoding procedure used by Mihaljević et al. above. The time complexity of decoding is

$$O\left(2^B D\log_2 m + (1+p_e(2^B-1))\binom{L-B-\delta}{\delta}\frac{1}{(2p-1)^2}\right),$$

where p_e is the probability of accepting a wrong solution and δ is a parameter. We refer to the original paper for in-depth details on this attack.

Attack using low rate codes

Molland et al. [MMH03] introduce a technique for finding parity-checks based on the generalised birthday problem presented by Wagner [Wag02]. They also propose a *quick metric* decoding technique which is able to use a very big number m of parity-checks.

Using the same $n\times N$ matrix G_{LFSR} as in (3.6) and for a given B, $1 \leqslant B \leqslant n$, Molland et al. search for weight-w parity-checks

$$c_1s_1 + c_2s_2 + \cdots + c_Bs_B = s_{i_1} + s_{i_2} + \cdots + s_{i_w}. \tag{3.10}$$

The authors present a method with complexity $O(N^{w-1}\log N)$ for finding all such parity-checks. However, they state that not all parity-checks are required for the attack to succeed. Hence, they employ a more efficient method fixing $w = 4$ to obtain only a subset of all the parity-checks. Let the columns of the matrix G_{LFSR} be $g_1,\ldots,g_N$. Algorithm 3.10 shows the method by Molland et al. The matrix G_2 has $N_2 = \frac{N^2}{2^{n-B+1}}$ columns. The time complexity is $O(N_2\log N_2)$.

Algorithm 3.10 Finding parity-checks - Molland et al.

Input: G_{LFSR}, B and $B_4 < B$.
Output: Parity-checks of weight $w = 4$.
1: Sort the $n \times N$ matrix G_{LFSR} according to the last $n - B$ positions.
2: Find all pairs g_{i_1}, g_{i_2} of columns of G_{LFSR} such that $f = g_{i_1} + g_{i_2}$ is zero in the last $n - B$ positions. Add column f to a matrix G_2 and store the indices i_1, i_2.
3: Sort the $n \times N_2$ matrix G_2 according to the last $n - B_4$ positions.
4: Find all pairs f_{j_1}, f_{j_2} of columns of G_2 such that $f_{j_1} + f_{j_2} = g_{i_1} + g_{i_2} + g_{i'_1} + g_{i'_2}$ is zero in the last $n - B_4$ positions. The first B positions of $f_{j_1} + f_{j_2}$ correspond to $c_1,\ldots,c_B$. Output $c_1,\ldots,c_B$ and i_1, i_2, i_3, i_4.

For the decoding stage, Molland et al. use a method similar to that in the attack by Johansson and Jönsson based on convolutional codes [JJ99b]. The authors improve the storage of the parity-checks and their evaluation to compute the metrics for the Viterbi decoder. Notice that for all m parity-checks (3.10) there are only 2^B different versions for the left hand side. When $m \gg 2^B$, many equations will have the same left hand side *type* defined by $c_1,\ldots,c_B$. Let E and sum be tables with 2^B entries. As each parity-check is found, let

$$e = c_1 + 2c_2 + 2^2c_3 + \cdots + 2^{B-1}c_B,$$
$$s = (z_{i_1} + \cdots + z_{i_w}) \bmod 2$$

and update $E(e) = E(e) + 1$ and $\mathsf{sum}(e) = \mathsf{sum}(e) + s$. After having found all parity-checks, $E(e)$ is the number of equations of type e and $\mathsf{sum}(e)$ is the number of those equations whose right hand side sum to 1. Let $\hat{s} = (\hat{s}_1, \ldots, \hat{s}_B)$ be a guess for the first B bits of the initial state. If $c_1\hat{s}_1 + \cdots + c_B\hat{s}_B = 1$ for a type e equation, the number of equations that hold is $\mathsf{sum}(e)$ and the metric for $\hat{s}$ is updated by $\mathsf{sum}(e)$. When $c_1\hat{s}_1 + \cdots + c_B\hat{s}_B = 0$, the number of equations that hold is $E(e) - \mathsf{sum}(e)$ and the metric is updated by $E(e) - \mathsf{sum}(e)$. This way, the equations are tested in one step instead of $E(e)$. The complexity of the whole decoding stage employing the Viterbi decoder is $O\left(Tm + T2^{2B}\right)$.

Attack exploiting the knowledge of the filtering function

Leveiller et al. [Lev+03] introduce an attack which considers the characteristics of the LFSR and the filtering function to compute *a posteriori* probabilities (APP) for decoding. The authors present two algorithms to compute these probabilities: SOJA-1 and SOJA-2. Decoding is done by using one of their proposed algorithms: SOJA-Gallager, a modified version of Gallager's iterative algorithm, or SOJA-threshold, a threshold decoder.

Let a weight-w parity-check be written as

$$s_{i_1} + \cdots + s_{i_w} = 0$$

and let $\mathcal{E}^b(i)$ be the set of parity-checks containing s_i. Recall that parity-checks hold for shifted versions of the LFSR sequence as well. Following the notation by Leveiller et al., let $X(i) = (X_1(i), \ldots, X_\ell(i))$ be the input vector to the filtering function f at time i. Then, a *vectorial* parity-check is a set E of vectors $X(i_1), \ldots, X(i_w)$ such that their sum is zero. The set of vectorial parity-checks containing $X(i)$ will be denoted by $\mathcal{E}^v(i)$. The equations in $\mathcal{E}^b(i)$ are computed using the method in [CT00] (see algorithm 3.5). The vectorial parity-checks are then computed from $\mathcal{E}^b(i)$. Notice that $|\mathcal{E}^v(i)| \approx \frac{|\mathcal{E}^b(i)|}{\ell}$.

Let $E = \{X(i_1), \ldots, X(i_w)\}$ and $J_E = \{j : X(j) \in E\}$. Define

$$z^E = \{z_j : j \in J_E\} \quad \text{and} \quad z^{\mathcal{E}} = \{z_j : j \in J_E \text{ for all } E \in \mathcal{E}^v(i)\}.$$

We abuse notation in the definitions above and we mean the set of bits z_j, not the actual values 0 or 1. The APP computed by SOJA-1 is

$$\Pr(X(i) = x \mid z^{\mathcal{E}}, f) = \frac{\prod_{E \in \mathcal{E}^v(i)} \Gamma^i(x)}{\sum_y \prod_{E \in \mathcal{E}^v(i)} \Gamma^i(y)},$$

where

$$\Gamma^i(x) = \Pr(X(i) = x \mid z^E, f).$$

Let ϕ_u be a linear function given by $\phi_u : X(i) \to u \cdot X(i)$, where $u \in \mathbb{F}_2^\ell$ and $u \cdot X(i) = \sum_{j=1}^{\ell} u_j X_j(i)$. Then, we can also get the APP

$$\Pr(u \cdot X(i) = 1 \mid z, f) = \sum_{x : \phi_u(x) = 1} \frac{\prod_{E \in \mathcal{E}^v(i)} \Gamma^i(x)}{\sum_y \prod_{E \in \mathcal{E}^v(i)} \Gamma^i(y)}.$$

Now, in SOJA-2, consider vectorial parity-checks $X(i_1) + \cdots + X(i_w) = 0$, the idea is to enumerate the input vectors that satisfy the parity-checks, then evaluate the proportions of 1s and 0s corresponding to each component thus obtaining the APPs $\Pr(s_i = 1)$ on the input bits. Let $m = |\mathcal{E}^b(i)|$, then the complexity of SOJA-1 is $O\left(\ell 2^{2\ell+1} + \frac{m}{\ell} N 2^{\ell-1}\right)$ while the complexity of SOJA-2 is $O\left(\ell 2^{\ell} + Nm\right)$. Leveiller et al. show how to compute $\Gamma^i(x)$ and the APP for SOJA-2; we refer to the original paper [Lev+03] for the details.

Let $\Lambda(s_i)$ be the *a posteriori* probability assigned to s_i. SOJA-Gallager initialises $\mathsf{APP}^{(0)}(s_i)$ and $\mathsf{Obs}(s_i)$ to $\Lambda(s_i)$ for all $i = 1, \dots, N$. Then, for a fixed number θ of iterations, $\mathsf{APP}^{(k)}(s_i)$ is updated for all i as

$$\mathsf{APP}^{(k)}(s_i) \approx \mathsf{Obs}(s_i) \times \prod_{e \in \mathcal{E}^b(t)} \frac{1 - \prod_{s_j \in e : j \neq i}(1 - 2 \cdot \mathsf{APP}^{(k-1)}(s_i))}{2}.$$

Finally, if $\mathsf{APP}^{(\theta)}(s_i) > 0.5$, z_i is decoded as 1, and 0 otherwise. SOJA-threshold considers the K bits with the most reliable *a posteriori* probabilities $\Lambda(s_i)$, i.e., if $|\Lambda(s_i) - 0.5|$ is close to 0 or 1. For each of the K bits, z_i is decoded as 1 when $\Lambda(s_i) > 0.5$ and decoded as 0 when $\Lambda(s_i) < 0.5$.

Algorithm 3.11 shows the general attack method in [Lev+03].

Algorithm 3.11 SOJA attack by Leveiller et al.

Input: Keystream $z_1, \dots, z_N$, set of parity-checks.
Output: Initial state of the LFSR.

1: Using SOJA-1 or SOJA-2, compute the *a posteriori* probabilities $\mathsf{APP}(s_i)$ associated to the LFSR sequence bits.
2: Compute the initial state by decoding with SOJA-Gallager or SOJA-threshold.
3: Output the initial state of the LFSR.

Attack based on finding zero inputs of the filtering function

The attack by Didier [Did07] is related to the one above by Leveiller et al. since the former also exploits the knowledge of the function f. The main idea is to identify when the filtering function f has as input the zero vector. The ℓ bits involved in those zero vectors are equal to zero. By expressing these bits in terms of the initial state, we construct a system of linear equations which is used to recover the initial state.

Let a vectorial parity-check of weight $w + 1$ be

$$X_i + X_{i_1} + \cdots + X_{i_w} = 0$$

and define

$$P_X = \Pr\left(f(X_{i_1}) + \cdots + f(X_{i_w}) = 0 \mid \sum_{j=1}^{w} X_{i_j} = X \right),$$

i.e., the probability that $z_{i_1} + \cdots + z_{i_w} = 0$ knowing that $X_i = X$. Didier shows that for even w, (i) $P_0 > 0.5$ and $P_0 \geqslant P_X$, for all $X \neq 0$, and (ii) there is always a gap between P_0 and the other P_X's if f has a good autocorrelation property. The idea is to use many parity-checks to compute a good approximation of P_{X_i} associated to a position i. If the gap between P_0 and the other P_X is large enough, the indices i for which $X_i = 0$ are detected. Parity-checks of weight $w + 1 = 2w' + 1$ are used; they are computed using the method in [CJM02] (see Algorithm 3.9) with complexity $O\left(2^{N/2+(w'-1)\ell+1}\right)$. The main stage of the attack has complexity $O\left(\lceil N/\ell \rceil 2^{\ell+2+2\ell(w'-1)}\right)$. Algorithm 3.12 shows the attack by Didier.

Algorithm 3.12 Attack by Didier

Input: Keystream $z_1, \ldots, z_N$, set of vectorial parity-checks.
Output: Initial state of the LFSR.

1: Approximate P_{X_i} for the first $L = \lceil \frac{N}{\ell} \rceil 2^{\ell}$ keystream bits by counting how many parity-checks are satisfied by the keystream bits. Among the L bits, only the ones for which $z_i = f(0)$ have to be considered.
2: Assume that the $\lceil N/\ell \rceil$ bits with the highest P_{X_i} correspond to positions where $X_i = 0$. Construct a system of linear equations with those bits expressed in term of the initial state.
3: Solve the system of equations and output the initial state of the LFSR.

3.2.3 Attack by Todo et al.

Let $s_t = s_{t_1} + \cdots + s_{t_{w-1}}$ be a parity-check. Since every s_i can be expressed as a linear combination of the initial state S_1, a parity-check may be written as

$$s_t = \langle S_1, a_{t_1} \rangle + \cdots + \left\langle S_1, a_{t_{w-1}} \right\rangle = \langle S_1, a_t \rangle \,,$$

where $a_{t_1}, \ldots, a_{t_{w-1}} \in \mathbb{F}_2^n$, $a_t = \sum_{i=1}^{w-1} a_{t_i}$ and $\langle \cdot, \cdot \rangle$ denotes the dot product. Let e_t be the error generated by the filtering function. Then $z_t = s_t + e_t$. So, Todo et al. [Tod+18] write parity-checks as

$$e_t = \langle S_1, a_t \rangle + z_t.$$

Let $p = \Pr(e_t = 1)$, then the correlation is defined as $c = 1 - 2p$.

The error e_t may not be highly biased. However, high correlation may be observed by summing optimally chosen linear masks $\Gamma_i \in \mathbb{F}_2^n$. Assume that

$$e'_t = \sum_{i \in \mathcal{T}_s} \langle S_{t+i}, \Gamma_i \rangle + \sum_{i \in \mathcal{T}_z} z_{t+i}$$

may be highly biased for some $\mathcal{T}_s$, $\mathcal{T}_z$ and Γ_i, where S_{t+i} is the LFSR state at time $t + i$. Using equation (2.2), we get

$$e'_t = \left\langle S_1, \Gamma \times M^{t-1} \right\rangle + \sum_{i \in \mathcal{T}_z} z_{t+i}, \tag{3.11}$$

where $\Gamma = \sum_{i \in \mathcal{T}_s} (\Gamma_i \times M^i)$ and M is the matrix implementing the LFSR (see Section 2.4). Then, parity-checks are given by (3.11), $p = \Pr(e'_t = 1)$ and the correlation c is redefined from this value of p. Assuming that N parity-checks are available, we guess the initial state, compute $s_1, \ldots, s_N$ from that guess and evaluate $\sum_{t=1}^{N} (-1)^{e'_t}$. When the correct initial state is guessed, the sum follows a normal distribution $\mathbf{N}(Nc, N)$.

Otherwise, we assume that the sum behaves at random and it follows a normal distribution $\mathbf{N}(0, N)$. We have

$$\begin{aligned}\sum_{t=1}^{N}(-1)^{e'_t} &= \sum_{t=1}^{N}(-1)^{\langle S,\Gamma\times M^{t-1}\rangle+\sum_{i\in\mathcal{T}_z} z_{t+i}}\\ &= \sum_{x\in\mathbb{F}_2^n}\left(\sum_{t\in\{1,\ldots,N\mid\Gamma\times M^{t-1}=x\}}(-1)^{\sum_{i\in\mathcal{T}_z} z_{t+i}}\right)(-1)^{\langle S,x\rangle}.\end{aligned}$$

Then, in a similar way as Chose et al. [CJM02], from

$$w(x) = \sum_{t\in\{1,\ldots,N\mid\Gamma\times M^{t-1}=x\}}(-1)^{\sum_{i\in\mathcal{T}_z} z_{t+i}}$$

we compute $\widehat{w}$ using the fast Fourier-Hadamard transform (FFT); $\widehat{w}(S)$ corresponds to the value of the sum when S is guessed.

Let $\mathbb{F}_{2^n} = \mathbb{F}_2[x]/(g)$ and $\alpha \in \mathbb{F}_{2^n}$ such that $g(\alpha) = 0$ and it is a primitive element of $\mathbb{F}_{2^n}$. Also, let $A_i \in \mathbb{F}_2^n$ denote the first row of M^i, $i \geqslant 0$. The vector A_i is represented by $\alpha^i \in \mathbb{F}_{2^n}$. Let $\Gamma \in \mathbb{F}_2^n$ be represented by $\gamma \in \mathbb{F}_{2^n}$. The important remark is that the vector $\Gamma \times M$ is represented by $\gamma\alpha$ and $\Gamma \times M^i$ by $\gamma\alpha^i$. Now, let M_γ be an $n \times n$ matrix over $\mathbb{F}_2$ such that its j-th row is the vector representation of $\gamma\alpha^{j-1}$. Then, $\alpha^i\gamma$ is the representation of $A_i \times M_\gamma$. Since $\gamma\alpha^i = \alpha^i\gamma$, we have that $\Gamma \times M^i = A_i \times M_\gamma$. Given this "commutative" feature,

$$\left\langle S_1, \Gamma \times M^{t-1}\right\rangle = \left\langle S_1, A_{t-1} \times M_\gamma\right\rangle = \left\langle S_1 \times M_\gamma^T, A_{t-1}\right\rangle$$

and equation (3.11) is equivalent to

$$e'_t = \left\langle S_1 \times M_\gamma^T, A_{t-1}\right\rangle + \sum_{i\in\mathcal{T}_z} z_{t+i}.$$

Assume high correlation is observed when guessing S_1 and parity-checks are generated from $\Gamma \times M^{t-1}$. Then, the same high correlation is observed when guessing $S_1 \times M_\gamma^T$ and parity-checks are generated from A_{t-1}.

A linear mask may equivalently be represented by $\Gamma \in \mathbb{F}_2^n$ or $\gamma \in \mathbb{F}_{2^n}$. Linear masks which yield high correlation are referred to as highly biased linear masks. Let parity-checks be generated from $\Gamma \times M^{t-1}$ and assume we guess an incorrect initial state $S'_1 = S_1 \times M_{\gamma'}^T$. Then

$$\left\langle S'_1, \Gamma \times M^{t-1}\right\rangle = \left\langle S_1 \times M_{\gamma'}^T, A_{t-1} \times M_\gamma\right\rangle = \left\langle S_1, A_{t-1} \times M_{\gamma\gamma'}\right\rangle,$$

i.e., it is equivalent to using the linear mask $\gamma\gamma'$ instead of γ. If γ and $\gamma\gamma'$ are highly biased linear masks, guessing $S_1 \times M_{\gamma'}^T$ also yields high correlation. Based on this, Todo et al. introduce a new wrong-key hypothesis: *Assume that there are m highly biased linear masks $\gamma_1, \ldots, \gamma_m$ and parity-checks are generated from A_t (with the same $\mathcal{T}_z$). Then, we observe high correlation when $S_1 \times M_{\gamma_i}^T$ is guessed for any $i \in \{1, \ldots, m\}$. Otherwise, we assume the correlation is 0.*

Let $\gamma_1, \ldots, \gamma_m$ be highly biased linear masks. The attack by Todo et al. is presented in Algorithm 3.13. It consists of three steps: constructing parity-checks, computing the FFT and removing the linear masks. Parity-checks are constructed from A_t and

$\sum_{i\in\mathcal{T}_z} z_{t+i}$. The FFT is used to evaluate $\langle S, A_t\rangle + \sum_{i\in\mathcal{T}_z} z_{t+i}$ and those S with correlation higher than a threshold th are chosen. Applying the FFT to all possible values of S has high time complexity $n2^n$. Todo et al. bypass β bits (i.e., set those bits to a constant value) and guess only $n-\beta$ bits. The chosen solutions have the form $S_1 \times M^T_{\gamma_i}$. For each solution, we try all γ_i to remove $M^T_{\gamma_i}$ from S. This method recovers the correct initial state for some γ_i and an incorrect one for other masks. If the expected number of times the correct S_1 appears is greater than that for incorrect ones, S_1 is uniquely determined. The parameters β and th are chosen such that this is the case. We refer to the original paper [Tod+18] for a more detailed description and analysis.

Algorithm 3.13 Attack by Todo et al.

Input: Keystream $z_1, \ldots, z_N$, number β of bypassed bits, correlation threshold th.
Output: Initial state of the LFSR.

1: Construct parity-check equations from A_t and $\sum_{i\in\mathcal{T}_z} z_{t+i}$.
2: Apply the FFT to evaluate $\sum_{t=1}^{N}(-1)^{\langle S, A_{t-1}\rangle + \sum_{i\in\mathcal{T}_z} z_{t+i}}$, where the β bypassed bits of S are fixed to a constant (i.e, only $n-\beta$ bits are guessed).
3: Pick those solutions S with correlation greater than th. Each solution has the form $S = S_1 \times M^T_{\gamma_i}$. Remove $M^T_{\gamma_i}$ by exhaustively guessing γ_i and recover S_1.

Let m_p be the number of masks with positive correlation and m_m be the number of masks with negative correlation. The total number of masks is $m = m_p + m_m$. The time complexity of the attack is estimated [Tod+18] as $N + (n-\beta)2^{n-\beta} + m2^{n-\beta}\epsilon_1 + (m_p^2 + m_m^2)2^{-\beta}\epsilon_2$, where $\epsilon_1 = \Pr(\mathbf{N}(0, N) > th)$ and $\epsilon_2 = \Pr(\mathbf{N}(Nc, N) > th)$. The term $(m_p^2 + m_m^2)2^{-\beta}\epsilon_2$ is negligible and considering $N = (n-\beta)2^{n-\beta} = m2^{n-\beta}\epsilon_1$, the time complexity is $3(n-\beta)2^{n-\beta}$ and the required number of parity-checks is $N = (n-\beta)2^{n-\beta}$.

3.2.4 Summary of fast correlation attacks and some results

Table 3.1 shows a summary of the time complexity of the attacks described above. Recall that $p = \Pr(s_i = z_i) = 1/2 + \epsilon$ and m denotes the number of weight-w parity-check equations used. In [JJ99b] and [JJ99a], the parameter $B < n$ is the memory (of the convolutional code) and $J = n + 10B$. In [CJS01], $k < n$ is a complexity parameter. In [JJ00], m is the number of samples, m_1 is the cardinality of the subset of parity-checks used and k is a parameter. In [MFI02], B is the number of bits in the partial brute force and D, M are parameters. In [CJM02], B is the number of bits in the partial brute force, p_e is the probability of accepting a wrong solution, and $D > n - B$ and a small δ are parameters. In [MMH03], $B < n$ is the memory, $N_2 = \frac{N^2}{2^{n-B+1}}$ and $T \approx n$ is a parameter. In [Lev+03], θ is a parameter for the number of iterations while decoding. In [Did07], $w = 2w' + 1$. The parameter β in [Tod+18] is the number of bypassed bits.

Recall that when modelled as a decoding problem, the crossover probability of the associated BSC is $1-p$. Table 3.2 contains some numerical results of the fast correlations attacks above. The table shows only some of the best reported results for the given values. Entries with * in the second column are theoretical results only. Many of these attacks use the same degree-40 polynomial of weight 17 from [JJ99b]. For the other experiments, most authors employ randomly chosen feedback polynomials. The symbol ? in the third column indicates that neither g is explicitly presented nor its weight is reported. The authors in [Lev+03] used coprime spacings for the inputs

Attack	Complexity	
	Pre-computation	Computation
[JJ99b]	$O\left(\binom{N-J}{w-1}\right)$	$O\left(Jm2^B\right)$
[JJ99a]	$O\left(M\binom{N-J}{w-1}\right)$	$O\left(6MJm2^B\right)$
[CJS01]	$O\left(N^2\right)$	$O\left(2^k k \frac{2}{(2\epsilon)^{2w}}\right)$
[CT00]	$O\left(\binom{N-1}{w-2}\right)$	$O\left(5(w-1)mN\right)$
[JJ00]	$O\left(N^{\lceil w/2 \rceil}\right)$	$O\left(mm_1k2^k\right)$ or $O\left(mm_1k2^{nc}\right), c < 1$
[MFI02]	$O\left(D\binom{N-n}{2}\right)$ or $O\left(D(N-n)\right)$	$O\left(2^B((D-B)m + (M-n)wt(g))\right)$
[CJM02]	$O\left(N^{\lceil w/2 \rceil}\log N\right)$	$O\left(2^B D \log_2 m + (1+p_e(2^B-1))\binom{L-B-\delta}{\delta}\frac{1}{(2p-1)^2}\right)$
[MMH03]	$O\left(N_2 \log N_2\right)$	$O\left(Tm + T2^{2B}\right)$
[Lev+03]	same as [CT00]	$O\left(\ell 2^{2\ell+1} + \frac{m}{\ell}N2^{\ell-1}\right) + \text{dec}$ or $O\left(\ell 2^\ell + Nm\right) + \text{dec}$, $\text{dec} = O\left(\theta N m\right)$ or $O(N)$
[Did07]	$O\left(2^{N/2+(w'-1)\ell+1}\right)$	$O\left(\lceil N/\ell \rceil 2^{\ell+2+2\ell(w'-1)}\right)$
[Tod+18]	$O\left((n-\beta)2^{n-\beta}\right)$	$O\left(2(n-\beta)2^{n-\beta}\right)$

Table 3.1. Summary of the time complexity of fast correlation attacks.

to f, i.e., $\gcd(\gamma_1, \dots, \gamma_{\ell-1}) = 1$. For more details, we refer to the original sources. The attack in [Tod+18] was not applied to the filter generator, but to ciphers in the Grain family [Hel+08]. Due to this, those results are omitted in Table 3.2. Ciphers in the Grain family contain an LFSR, an NFSR and an output function. Section 4.8 presents more details and the results in [Tod+18].

Attack	deg(g)	wt(g)	w	$1-p$	N
[JJ99b]	40	17	2	0.260	$4 \cdot 10^4$
	40	17	2	0.400	$4 \cdot 10^5$
[JJ99a]	40	17	2	0.300	$4 \cdot 10^4$
	40	17	2	0.410	$4 \cdot 10^5$
[CJS01]	60	?	3	0.300	$6.3 \cdot 10^4$
	60	?	3	0.400	$6 \cdot 10^5$
	70	?	3	0.350	$1.12 \cdot 10^6$
[CT00]	40	17	4	0.440	$4 \cdot 10^5$
	40	17	5	0.482	$3.6 \cdot 10^5$
[JJ00]	40	17	2	0.450	$4 \cdot 10^5$
	60	13	3	0.320	$1.5 \cdot 10^5$
	60	13	2	0.430	$4 \cdot 10^7$
[MFI02]	40	17	3	0.469	$4 \cdot 10^5$
	40	17	3	0.490	$3.6 \cdot 10^5$
	*89	?	3	0.469	$\approx 2.5 \cdot 10^{11}$
	*89	?	3	0.478	$\approx 10^{12}$
	*89	?	3	0.480	$\approx 4 \cdot 10^{12}$
[CJM02]	40	17	4	0.469	$8 \cdot 10^4$
	*40	17	4	0.490	$8 \cdot 10^4$
	*89	?	4	0.469	2^{28}
[MMH03]	60	?	4	0.430	$1.5 \cdot 10^7$
	60	?	4	0.470	$1 \cdot 10^8$
[Lev+03]	40	17	5	0.375	$1.7 \cdot 10^4$
	100	3	3	0.4375	$3 \cdot 10^4$
[Did07]	53	?	5	0.4375	$\approx 4 \cdot 10^5$
	59	?	5	0.4531	$\approx 1.45 \cdot 10^6$
	61	?	5	0.4531	$\approx 2.1 \cdot 10^6$

Table 3.2. Some numerical results of fast correlation attacks.

3.3 Deterministic attacks

This class of attacks exploit information about the filter generator, such as the feedback polynomial, the filtering function and its inputs from the LFSR. The efficiency of deterministic attacks can be diminished by designing the filter generator with certain characteristics. Additionally to the requirements in the previous section, it is recommended to choose $k_1, \ldots, k_\ell$ such that the input spacings to the filtering function f are coprime and its *memory* is as close to n as possible. The memory is defined [Lev+01] as

$$\Gamma = 1 + \frac{\sum_{i=1}^{\ell-1} \gamma_i}{\gcd(\gamma_1, \ldots, \gamma_{\ell-1})} = 1 + \frac{k_\ell - k_1}{\gcd(\gamma_1, \ldots, \gamma_{\ell-1})},$$

where $\gamma_i = k_{i+1} - k_i$ are the input spacings. Golić presents in [Gol96b] an extensive list of design criteria to make the filter generator resistant to various attacks.

3.3.1 Some deterministic attacks

Initial direction by Anderson

The work by Anderson [And95] is among the first ones in this class. It is based on the *augmented function* of f, which captures dependencies between the keystream bits and the input bits that generated them.

Let m be a parameter and let $F_m : \mathbb{F}_2^{m+k_\ell} \to \mathbb{F}_2^m$ be defined as

$$F_m : (x_1, \ldots, x_{m+k_\ell}) \mapsto (f(x_{1+k_1}, \ldots, x_{1+k_\ell}), f(x_{2+k_1}, \ldots, x_{2+k_\ell}), \ldots, f(x_{m+k_1}, \ldots, x_{m+k_\ell})),$$

where $k_1, \ldots, k_\ell$ are the input spacings to the filtering function f. Let

$$s_t^{(m)} = (s_t, \ldots, s_{t+m-1}) \quad \text{and} \quad z_t^{(m)} = (z_t, \ldots, z_{t+m-1})$$

denote m consecutive LFSR and keystream bits, respectively, at time t. Then, we have that $z_t^{(m)} = F_m\left(s_t^{(m+k_\ell)}\right)$. Anderson considers the case $k_{i+1} = k_i + 1$ for $i = 1, \ldots, \ell-1$, i.e., k_i are consecutive integers. The function F_ℓ is called the augmented function of f and $z_t^{(\ell)} = F_\ell\left(s_t^{(2\ell-1)}\right)$.

The idea in Anderson's attack is to analyse the dependence between a keystream bit z_t and the corresponding ℓ input bits s_{t+k_i}, and also how each input bit influences ℓ different keystream bits. In total, $2\ell - 1$ input bits will affect ℓ keystream bits. The attack analyses the truth table of F_ℓ, which is constructed from that of f. The objective is to find, for each output value of F_ℓ, whether there are constant values for some of the inputs or very high correlations in the inputs. The initial state is then recovered by solving a system of linear equations from the bits that have a fixed value and highest correlations. Anderson concludes that a careful choice of f should be made in order to avoid this attack.

Inversion attacks

Golić presented the inversion attack in [Gol96b] and it has two variants: forward and backward attack. The applicability of these variants depends on whether the filtering function f is linear in the first or last input variable, respectively. Being linear in the first or last variable means that

$$f(x_1, \ldots, x_\ell) = x_1 + f_1(x_2, \ldots, x_\ell) \quad \text{or} \quad f(x_1, \ldots, x_\ell) = x_\ell + f_2(x_1, \ldots, x_{\ell-1}),$$

for some Boolean functions f_1 or f_2, respectively. Assuming linearity in the first variable, we have that

$$s_{t+k_1} = z_t + f_1(s_{t+k_2}, \ldots, s_{t+k_\ell}). \tag{3.12}$$

Assume $\gcd(\gamma_i) = 1$, then $\Gamma = k_\ell - k_1 + 1$. The (forward) attack consists in guessing the initial value of the $\Gamma - 1$ *memory* bits $s_{k_1+1}, \ldots, s_{k_\ell+1}$, then compute the value of the remaining $n - \Gamma + 1$ initial state bits using the given keystream and (3.12). Therefore, the sequence bits $\{s_t\}_{t=n+1}^{N}$ can be obtained by the definition of the LFSR. Finally, a new keystream sequence is computed and compared against the original one, thus finding the initial state when both coincide. The time complexity is $O(2^{\Gamma-1})$. Górska and Górski [GG02] propose guessing $n - m$ bits instead of $\Gamma - 1$, where m denotes the largest gap between cells of the LFSR which have taps to the filtering function or connection polynomial.

The generalised inversion attack by Golić et al. [GCD00] works if f is not linear in x_1 and x_ℓ. It employs trees to recover the initial state and critical branching processes [Har63; AN72] for the probabilistic analysis. Again, assume $\gcd(\gamma_i) = 1$. The attack represents the value of the $\Gamma - 1$ memory bits $s_{k_1+1}, \ldots, s_{k_\ell+1}$ as the root of a tree with maximum depth $n - \Gamma + 1$. Each node in this tree represents a memory state of $\Gamma - 1$ bits. The main idea is to expand a tree to level t according to the solutions of $z_t = f(s_{t+k_1}, \ldots, s_{t+k_\ell})$. The value of z_t is known and the values for $s_{t+k_1}, \ldots, s_{t+k_\ell}$ are determined by the current node. The number of solutions (one, two or none) indicates the number of new nodes to add from the current one. When a tree reaches the maximum depth, the keystream is recomputed and compared against the given keystream to check whether the correct initial state was found. In the worst case, all possible $2^{\Gamma-1}$ different values for the memory bits are checked, i.e, $2^{\Gamma-1}$ trees are processed. Golić et al. show that the number of survivor nodes at the last level of the trees is linear in n. Let $M = \Gamma - 1$, the time complexity of the attack is $O\left(q_{n-M}^{-1} 2^M\right)$, where $q_{n-M} \approx 1 - \left(1 - \frac{2}{p(n-M)}\right)^{2^M}$ and p depends on f.

$\{0, 1\}$-metric Viterbi decoding technique

The technique by Leveiller et al. [Lev+01] is another deterministic attack. It is based on a trellis that is derived from the function f and the output bits z_i. Let Γ be the memory of the filter generator and the function f to be balanced. Each section of the trellis consists of 2^Γ states/vectors representing the LFSR state bits which contain the inputs to f. Each vector on one section of the trellis is connected to two vectors on the other section as follows: let $v = (v_1, v_2, \ldots, v_\Gamma)$, then it is connected to the vectors $(v_2, \ldots, v_\Gamma, 0)$ and $(v_2, \ldots, v_\Gamma, 1)$ on the other section. The latter vectors are the successors of v. The mapping on the trellis transitions (i.e, the edges connecting the vectors) are labelled with the value of f applied to the successor state. Algorithm 3.14 shows the basic version of the attack. The time complexity is $O\left(2^\Gamma\right)$.

Leveiller et al. present a generalisation of the basic attack in which they check not only the left-most bit of the states in the trellis, but a linear combination of the bits

Algorithm 3.14 $\{0, 1\}$-metric Viterbi decoding by Leveiller et al.

Input: Keystream $z_1, \ldots, z_N$.
Output: Initial state of the LFSR.

Initialise $N_{dec} = 0$. Half of the states in the trellis correspond to z_1, discard all the invalid states and store the survivor ones.
for $t = 2, 3, \ldots$ **do**
  According to the bit z_t, the survivor states at time $t - 1$ and the mapping on the transitions, store the new valid states matching z_t along with the label of the mappings.
  if the left-most bit of all survivors is a constant equal to b **then**
    Set $s_t = b$ and increment N_{dec} by 1.
    if $N_{dec} \geqslant n$ and the set of decoded bits contain n independent bits **then**
      Terminate the iteration.
    end if
  end if
end for
Solve the system of equations and output the initial state of the LFSR.

of the survivor states. This generalisation requires the computation of the Fourier-Hadamard transform on vectors of length Γ, hence the complexity of the attack increases by a factor of $O(\Gamma 2^{\Gamma})$. This, however, allows the recovery of the initial state using less keystream bits. Both, the basic and generalised attack also have forward and backward variants. By combining both variants, more information is obtained at a given time t, which reduces even more the required length of the keystream. We refer to the original paper [Lev+01] for further details.

3.3.2 Summary of deterministic attacks and some results

Table 3.3 contains a summary of the time complexity and some results of the deterministic attacks above. The symbol ? in the fifth column indicates that the degree of f was not reported. For more details, we refer to the original sources.

Attack	$\deg(g)$	$\mathrm{wt}(g)$	ℓ	$\deg(f)$	memory Γ	N	Complexity
[GCD00]	100	5	5	?	5	100	$O\left(q_{n-M}^{-1} 2^M\right)$, where $M = k_\ell - k_1 = \Gamma - 1$, $q_{n-M} \approx 1 - \left(1 - \frac{2}{p(n-M)}\right)^{2^M}$ and p depends on f
	100	5	5	?	16	100	
	100	5	10	?	10	100	
	100	5	10	?	16	100	
[Lev+01]	100	5	5	3	5	185	$O\left(2^{\Gamma}\right)$
	100	5	5	3	9	182	
	100	5	8	4	8	268	

Table 3.3. Summary and some results of deterministic attacks.

3.4 Algebraic attacks

This type of attacks model the cipher as a system of multivariate equations. Following the notation in [CM03], L denotes the transition function, which corresponds to the action of the matrix M here, i.e., $S_i = L(S_{i-1}) = L^{i-1}(S_1)$. Then, the keystream is given by

$$\begin{cases} z_1 = f(s_0, \dots, s_{n-1}) \\ z_2 = f(L(s_0, \dots, s_{n-1})) \\ z_3 = f(L^2(s_0, \dots, s_{n-1})) \\ \qquad \vdots \end{cases}$$

and the initial state can be recovered by solving the equations above. The complexity of these attacks is greatly influenced by the degree of the algebraic system.

The original idea in [CM03] is to solve the system of equations for a subset of keystream bits z_i using low-degree multiples of f. Let g, h be multivariate polynomials of low degree such that $f(x)g(x) = h(x)$. Then, for each keystream bit we get $f(S_i) \cdot g(S_i) = h(S_i)$. The attack finds relations like this for sufficiently many keystream bits to get an overdetermined system of multivariate equations of low degree. Finding the relations can be seen as a pre-computation step in algebraic attacks; solving the system of equations is then the online step.

Fast algebraic attacks were introduced in [Cou03] as an improvement to the original algebraic attacks. One key difference is that the fast version employs relations involving several keystream bits, not only one. This idea is similar to the augmented function defined in [And95]. As the number m of keystream bits considered in the relations increases, finding the relations becomes harder. Curtois proposed in [Cou03] a general and a fast method for the pre-computation step. In [Arm04; HR04], the authors present further improvements on the pre-computation step. In [Can06], Canteaut describes algebraic attacks and presents some open questions regarding their complexity and cryptographic properties of Boolean functions under these attacks. Over $\mathbb{F}_2$, the existence of low-degree relations is closely related to the existence of low-degree *annihilators* of f or $(f+1)$ [MPC04]. An annihilator of the Boolean function f is another Boolean function g such that $f(x)g(x) = 0$.

CHAPTER 4

New cryptanalysis of LFSR-based stream ciphers

In this chapter we present a new cryptanalytic method which may be used against LFSR-based stream ciphers. Particularly, we focus on the filter generator and Grain-v1 [HJM07]. This is the result of joint work with Semaev [CS21].

Filter generators are described in Section 3.1. Let S_i be the LFSR state at time i as a column vector and M be the matrix implementing the LFSR (see Section 2.4), i.e.,

$$S_i = \begin{pmatrix} s_i \\ s_{i+1} \\ \vdots \\ s_{i+n-1} \end{pmatrix} \quad \text{and} \quad M = \begin{pmatrix} 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \\ c_n & c_{n-1} & \cdots & c_1 \end{pmatrix}.$$

Then, $S_i = M^{i-1} S_1$ for all $i \geqslant 1$. Let Λ be the $\ell \times n$ matrix that "selects" the inputs to f, i.e.,

$$\begin{pmatrix} s_{i+k_1} \\ \vdots \\ s_{i+k_\ell} \end{pmatrix} = \Lambda S_i, \quad \text{where } \Lambda = \begin{pmatrix} e_{k_1+1} \\ \vdots \\ e_{k_\ell+1} \end{pmatrix}$$

and $e_j = (0 \ldots 010 \ldots 0)$, $j = 1, \ldots, n$, and the only 1 is in position j from the left. Let

$$A_i = \Lambda M^{i-1}.$$

Then, the keystream bit at time i is $z_i = f(s_{i+k_1}, \ldots, s_{i+k_\ell}) = f(A_i X)$, where $X = S_1$. We assign a uniform probability distribution on the pre-image of z_i (i.e., the set of all possible values $a \in \mathbb{F}_2^\ell$ such that $z_i = f(a)$) and all other values get probability 0. That defines a probability distribution for a random variable X_i on the values of $A_i X$. We assume X to be uniformly distributed on $\mathbb{F}_2^n$ and X_i to be independent. Let N bits of the keystream be available. Then, the key recovery attack on the filter generator is to find the value $X = x$ with maximum probability under the condition that

$$A_i X = X_i,\ i = 1, \ldots, N. \tag{4.1}$$

With the description above, the key recovery attack is a particular case of the problem of finding solutions to systems of linear equations with associated probability distributions on the set of right hand sides. This problem is of a general nature not relevant to LFSRs and is stated in Section 4.1. We first solve this problem with the multivariate correlation attack in Section 4.2, which is a generalisation of the correlation attack by Siegenthaler [Sie85]. The multivariate correlation attack, however, has high time complexity. In Section 4.3, a more efficient method is presented. This novel method requires the computation of *relations modulo* B (see Section 4.4), where B is a matrix over a finite field, and a set of probability distributions induced by these relations. Relations modulo B can be seen as a generalisation of parity-checks used in fast correlation attacks. Section 4.5 presents different techniques for computing the probability distributions induced by relations modulo B. The analysis of the new method is in Section 4.6. The experimental results of applying our new technique to some hard instances of the filter generator are reported in Section 4.7. Section 4.8 focuses on a practical application against a toy Grain-like cipher and a theoretical application against Grain-v1. The idea of the method and theoretical results in the first six sections are due to Semaev.

4.1 The problem to solve

Let A_i, $i = 1, \ldots, N$, be matrices of size $\ell_i \times n$ and rank ℓ_i over a finite field $\mathbb{F}_q$, where ℓ_i are small compared to n. Let X be a vectorial random variable with values in $\mathbb{F}_q^n$ and X_i be vectorial random variables with values in $\mathbb{F}_q^{\ell_i}$, $i = 1, \ldots, N$. We assume that X is uniformly distributed. Also, let $\Pr(X_i = a) = P_i(a)$ for some probability distributions P_i on $\mathbb{F}_q^{\ell_i}$. We consider a system of equations

$$A_1 X = X_1, \ldots, A_N X = X_N. \tag{4.2}$$

The task is to find $X = x$ with the largest conditional probability

$$\Pr\left(X = x \,|\, A_1 X = X_1, \ldots, A_N X = X_N\right);$$

such x is called a solution to (4.2). It is equivalent to maximising the likelihood $\Pr\left(X_1 = A_1 x, \ldots, X_N = A_N x\right)$. If X_i are independent, we may maximise

$$\sum_{i=1}^{N} \ln \Pr(X_i = A_i x) = \sum_{i=1}^{N} \ln P_i(A_i x),$$

for $P_i(A_i x) \neq 0$. The variables $X, X_1, \ldots, X_N$ will be assumed to be independent, unless otherwise stated.

A particular case of this problem is the equations (4.1) from the key recovery attack against the filter generator. Multiple right hand side equation systems introduced in [RS08] are also a particular case of the problem.

4.2 Multivariate correlation attack

Our task is to find a solution given $A_1, \ldots, A_N$ and $P_1, \ldots, P_N$. For each $x \in \mathbb{F}_q^n$, we decide whether $x_i = A_i x$ were taken from the distributions P_i or from the uniform distributions on $\mathbb{F}_q^{\ell_i}$. Let us consider the statistic $S(x) = \sum_{i=1}^{N} \ln P_i(x_i)$ and let β be a prescribed success probability. When x_i are taken from the distributions P_i, a threshold c such that $\Pr(S(x) \geqslant c) = \beta$ is computed. Then, x *survives* if

$$P_i(x_i) \neq 0, \quad i = 1, \ldots, N, \tag{4.3}$$

$$S(x) = \sum_{i=1}^{N} \ln P_i(x_i) \geqslant c \tag{4.4}$$

simultaneously hold. We now define asymptotic distributions of the statistic $S(x)$ in two cases. Assume that x_i are taken independently in both cases. The two hypothesis are:

H0. When x_i are taken from the distributions P_i, $i = 1, \ldots, N$,

$$\mu_{0,i} = \sum_{y \in \mathbb{F}_q^{\ell_i}} P_i(y) \ln P_i(y) \quad \text{and} \quad \sigma_{0,i}^2 = \sum_{y \in \mathbb{F}_q^{\ell_i}} P_i(y) \ln^2 P_i(y) - \mu_{0,i}^2$$

are the expectation and the variance of $\ln P_i(x_i)$, respectively. Then,

$$\mu_0 = \sum_{i=1}^{N} \mu_{0,i} \quad \text{and} \quad \sigma_0^2 = \sum_{i=1}^{N} \sigma_{0,i}^2$$

are the expectation and variance of $S(x)$, respectively. Let P_i be close to the uniform distributions on their support. Then, the Lyapunov condition is satisfied for $S(x)$. For large enough N, the distribution of $S(x)$ approximately follows the normal distribution $\mathbf{N}(\mu_0, \sigma_0^2)$ by the Lyapunov Central Limit Theorem (see Section 2.2.3).

H1. Let K_i denote the size of the support of P_i. When x_i are taken from the uniform distributions on $\mathbb{F}_q^{\ell_i}$, $i = 1 \ldots, N$,

$$\mu_{1,i} = \sum_{y \in \mathbb{F}_q^{\ell_i}, P_i(y) \neq 0} \frac{\ln P_i(y)}{K_i} \quad \text{and} \quad \sigma_{1,i}^2 = \sum_{y \in \mathbb{F}_q^{\ell_i}, P_i(y) \neq 0} \frac{\ln^2 P_i(y)}{K_i} - \mu_{1,i}^2$$

are the expectation and the variance of $\ln P_i(x_i)$, respectively. Then

$$\mu_1 = \sum_{i=1}^{N} \mu_{1,i} \quad \text{and} \quad \sigma_1^2 = \sum_{i=1}^{N} \sigma_{1,i}^2$$

are the expectation and variance of $S(x)$, respectively. Under the condition that $P_i(x_i) \neq 0$, $i = 1, \ldots, N$, the distribution of $S(x)$ approximately follows the normal distribution $\mathbf{N}(\mu_1, \sigma_1^2)$ by the Lyapunov Central Limit Theorem.

The threshold c is computed from

$$\beta = \Pr(\mathbf{N}(\mu_0, \sigma_0^2) \geqslant c).$$

The probability of an incorrect x passing the tests (4.3) and (4.4) is

$$\alpha = \left(\prod_{i=1}^{N} \frac{K_i}{q^{\ell_i}}\right) \Pr(\mathbf{N}(\mu_1, \sigma_1^2) \geqslant c).$$

The number of incorrect survivors is αq^n on the average. We may get multiple candidate solutions (i.e., survivors), however, the solution is unique for large enough N. The time complexity of this straightforward attack is $O(Nq^n)$ operations.

Siegenthaler's attack [Sie85] is a particular case for $q = 2$, $\ell_i = 1$ and there are only two different distributions among P_i. In that case, only (4.4) works to test the candidate solutions. If the distributions P_i are uniform on their supports, the statistic $S(x)$ is a constant and only (4.3) works to test the candidate solutions. An example is the equations (4.1) for the filter generator in Section 4.7. The method is then reduced to brute force on the LFSR's initial state.

4.2.1 The number of equations

Let the distributions $P_1, \ldots, P_N$ be permutations of the same distribution. Given a desired success probability β and the number of survivors αq^n, we can estimate the number of necessary equations N and define the threshold c. Since

$$\mu_0 = N\mu_{0,1},\ \sigma_0^2 = N\sigma_{0,1}^2, \quad \text{and} \quad \mu_1 = N\mu_{1,1},\ \sigma_1^2 = N\sigma_{1,1}^2,$$

we can find c and N from the equations

$$\alpha \prod_{i=1}^{N} \frac{q^{\ell_i}}{K_i} = \Pr(\mathbf{N}(N\mu_{1,1}, N\sigma_{1,1}^2) \geqslant c) \quad \text{and}$$
$$\beta = \Pr(\mathbf{N}(N\mu_{0,1}, N\sigma_{0,1}^2) \geqslant c).$$

4.2.2 Improved complexity

Let every probability distribution P_i be close to uniform such that $P_i(y) = q^{-\ell_i} + o(q^{-\ell_i})$, and ξ be a primitive q-th root of unity. The Fourier spectrum of P_i is given by the values

$$W_{i,a} = \sum_{y \in \mathbb{F}_q^{\ell_i}} P_i(y)\xi^{-a\cdot y},$$

where $a \in \mathbb{F}_q^{\ell_i}$ and $a \cdot y$ denotes the dot-product of a and y. By the inverse of the Fourier transform we have that

$$\begin{aligned}
P_i(y) &= q^{-\ell_i} \sum_{a \in \mathbb{F}_q^{\ell_i}} W_{i,a} \xi^{a \cdot y} \\
&= q^{-\ell_i} \left(W_{i,0} \xi^{0 \cdot y} + \sum_{a \in \mathbb{F}_q^{\ell_i} : a \neq 0} W_{i,a} \xi^{a \cdot y} \right) \\
&= q^{-\ell_i} \left(1 + \sum_{a \in \mathbb{F}_q^{\ell_i} : a \neq 0} W_{i,a} \xi^{a \cdot y} \right).
\end{aligned}$$

By assumption, $P_i(y)$ are close to $q^{-\ell_i}$, so $\sum_{a \neq 0} W_{i,a} \xi^{a \cdot y}$ are small. Since $\ln(1+\varepsilon) \approx \varepsilon$ for small ε, we have

$$\ln P_i(y) = \ln \left(1 + \sum_{a \in \mathbb{F}_q^{\ell_i} : a \neq 0} W_{i,a} \xi^{a \cdot y} \right) - \ell_i \ln q \approx \sum_{a \in \mathbb{F}_q^{\ell_i} : a \neq 0} W_{i,a} \xi^{a \cdot y} - \ell_i \ln q.$$

Therefore,

$$\sum_{i=1}^{N} \ln P_i(A_i x) \approx \sum_{i=1}^{N} \sum_{a \in \mathbb{F}_q^{\ell_i} : a \neq 0} W_{i,a} \xi^{a \cdot A_i x} - \sum_{i=1}^{N} \ell_i \ln q = \sum_{b \in \mathbb{F}_q^n} C(b) \xi^{b \cdot x} - \ln q \sum_{i=1}^{N} \ell_i,$$

where $C(b) = \sum_{i=1}^{N} \sum_{a \in \mathbb{F}_q^{\ell_i} : a \neq 0, aA_i = b} W_{i,a}$.

For each P_i, its Fourier spectrum is computed with $O(\ell_i q^{\ell_i})$ operations using the fast Fourier transform (FFT). All values $C(b)$ are then computed with $O(\sum_{i=1}^{N} q^{\ell_i})$ operations. Finally, $\sum_b C(b) \xi^{b \cdot x}$ for all $x \in \mathbb{F}_q^n$ are computed with $O(nq^n)$ operations using the FFT again. We have to keep all values $C(b)$ in order to apply the FFT. Therefore, the space complexity is q^n. Overall, the time complexity of the attack is $O(\sum_{i=1}^{N} \ell_i q^{\ell_i} + nq^n)$ operations. This can be seen as a multivariate extension of the method by Chose et al. [CJM02].

4.3 Test-and-extend algorithm

Here we present a new method for finding solutions to equations (4.2). Let $\langle V \rangle$ denote the linear space spanned by the rows of a matrix V.

Definition 4.3.1. Let B_r be a matrix over $\mathbb{F}_q$ of size $r \times n$ and rank r, where $1 \leqslant r \leqslant n$. A set of indices $I \subseteq \{1, \ldots, N\}$ such that

$$\langle A_i, i \in I \rangle \cap \langle B_r \rangle \neq \langle 0 \rangle \tag{4.5}$$

is called a *relation modulo* B_r and $|I|$ is called the *weight* of the relation. If the weight is small, the relation is said to be *short*.

Let $t_{r,I} > 0$ be the dimension of the space (4.5). This space is spanned by the rows of a matrix $T_{r,I}B_r$, where $T_{r,I}$ is a matrix of size $t_{r,I} \times r$ and rank $t_{r,I}$. If $|I|$ is small, we may efficiently compute a conditional probability distribution $p_{r,I}$ as

$$p_{r,I}(v) = \Pr\left((T_{r,I}B_r)X = v \,|\, A_iX = X_i, i \in I\right), \quad v \in \mathbb{F}_q^{t_{r,I}}. \tag{4.6}$$

Let $Y = B_rX$ and Y_I denote a random variable on $\mathbb{F}_q^{t_{r,I}}$ with the distribution $p_{r,I}$. Also, let $\mathcal{I}_r$ be a set of relations modulo B_r. Then,

$$T_{r,I}Y = Y_I, \quad I \in \mathcal{I}_r,$$

is a system of equations of the same type as (4.2), but with smaller dimension $r \leqslant n$. Since X is uniformly distributed on $\mathbb{F}_q^n$, the random variable Y is uniformly distributed on $\mathbb{F}_q^r$. The multivariate correlation method in Section 4.2 is applied to solve the new system. That is, $b_r = B_rX$ is tested with

$$p_{r,I}(b_{r,I}) \neq 0, \quad I \in \mathcal{I}_r, \tag{4.7}$$

$$S_r(b_r) = \sum_{I \in \mathcal{I}_r} \ln p_{r,I}(b_{r,I}) \geqslant c_r, \tag{4.8}$$

where $b_{r,I} = T_{r,I}b_r$ and c_r is a threshold defined by the success probability β. We may use the FFT to compute the values of the statistic S_r if the probabilities $p_{r,I}(v)$ are close to $q^{-t_{r,I}}$. For matrices B_r of large rank r, we need to run over q^r vectors b_r, which might still be inefficient. To overcome this, we use a test-and-extend algorithm.

The new method comprises two stages: pre-computation (Section 4.3.1) and main computation (Section 4.3.2). The latter has two variants: a simple tree search and a hybrid variant combining the FFT and a tree search. The success probability of the new method, and its time and data complexity are shown in Section 4.6.

4.3.1 Pre-computation

First, we choose a sequence of matrices $B_1, \ldots, B_n$. Each matrix B_r has size $r \times n$ and rank r. Also, these matrices have the property that for $r = 1, \ldots, n-1$, B_r is the submatrix of B_{r+1} comprising its first r rows, i.e.,

$$B_{r+1} = \begin{pmatrix} B_r \\ * \end{pmatrix}.$$

For each matrix B_r, we obtain a set $\mathcal{I}_r$ of relations modulo B_r with small weight $\leqslant d$. Then, we compute their probability distributions $p_{r,I}$. In Section 4.4 we show how to obtain relations modulo B_r. In Section 4.5 we show how to compute the distributions (4.6).

Finally, we compute a set of thresholds c_r, $r = 1, \ldots, n$, such that the correct solution is found with the desired success probability β. That defines the statistical tests (4.7) and (4.8) for $r = 1, \ldots, n$. The computation of these thresholds is shown in Section 4.6.1.

4.3.2 Main computation

In this stage, we may apply two variants of the test-and-extend algorithm: Tree search and Hybrid variant.

Tree search

Let $b = (a_1, a_2, \dots, a_n) \in \mathbb{F}_q^n$. For $r = 1, \dots, n$, we denote $b_r = (a_1, a_2, \dots, a_r)$, so that $b_n = b$. We will use a predicate R_r. We say $R_r(b_r) = 1$ if both conditions (4.7) and (4.8) are satisfied, and $R_r(b_r) = 0$ otherwise. The task is to find b such that

$$R_1(b_1) = 1, \dots, R_n(b_n) = 1. \tag{4.9}$$

We do this by traversing a tree in depth-first search, where b_r is tested at level r. If $R_r(b_r) = 0$, that branch is not explored and the search backtracks. If $R_r(b_r) = 1$, then b_r is extended to b_{r+1}, the value $R_{r+1}(b_{r+1})$ is checked to either backtrack or extend again, and the search continues in that fashion. Whenever $R_n(b_n) = 1$, the value of b_n is a solution to (4.9). In this way, we can find all the solutions to (4.9). Generally, the tree search finds candidate solutions to (4.2).

Hybrid variant

First, we choose a parameter r_0, such that $1 \leqslant r_0 \leqslant n$, and we compute the values of the statistic $S_{r_0}(b_{r_0})$ for all $b_{r_0} \in \mathbb{F}_q^{r_0}$. Ideally, this is done with the FFT as in Section 4.2.2. Then, the candidates b_{r_0} are ranked (i.e., sorted) according to the values $S_{r_0}(b_{r_0})$. Finally, we perform the tree search starting at level r_0. That is, as in the variant above, the candidates b_{r_0} are tested and extended to candidate solutions b_{r_0+1}, which in turn are tested, further extended and so on. The tree search is done following the ranking of the candidates b_{r_0}.

4.4 Relations modulo B_r

Let B_r be an $r \times n$ matrix of rank r and $I = \{i_1, \dots, i_d\}$ be a short relation modulo B_r of weight d. We present two methods to find short relations.

4.4.1 Brute force

Given a relation I, (4.5) is equivalent to the system of homogeneous linear equations

$$\sum_{i \in I} \nu_i A_i = \nu B_r, \tag{4.10}$$

where the variables are vectors $\nu_i \in \mathbb{F}_q^{\ell_i}$, $i \in I$, and $\nu \in \mathbb{F}_q^r$ such that $\nu \neq 0$. The system incorporates n equations in $\sum_{i \in I} \ell_i + r$ variables from $\mathbb{F}_q$. We have to solve $\binom{N}{d}$ such systems to find all relations of weight $\leqslant d$ modulo B_r.

Let $\ell_i = \ell$ for $i = 1, \dots, N$. We may expect to find at least one relation if $N > (d/e)\, q^{\frac{n-d\ell-r+1}{d}}$. There are $q^{\ell d} - 1$ non-zero vectors in the left hand sides of (4.10) for every I if dependencies between the rows of A_i, $i \in I$, are neglected. The probability that one random vector hits the space $\langle B_r \rangle$ is q^{r-n}. If a vector belongs to $\langle B_r \rangle$, then its multiples by non-zero constants belong to $\langle B_r \rangle$ too. For $\ell d + r < n$, the probability that two non-collinear vectors for the same I hit $\langle B_r \rangle$ is negligible. The average number of relations (4.10) is around

$$\frac{\binom{N}{d}(q^{\ell d} - 1)}{q^{n-r}(q-1)}. \tag{4.11}$$

For small d and large N, we have $\binom{N}{d} \approx \frac{N^d}{d!}$. That implies the bound for N.

4.4.2 Lattice reduction

Assume that q is a small prime number. Let A be a vertical concatenation of the matrices $A_1, \ldots, A_N$. Thus, A is a matrix with $m = \sum_{i=1}^{N} \ell_i$ rows, n columns and integer entries. Let L denote a lattice of all integer vectors v of length m such that $vA \in \langle B_r \rangle$ modulo q. Clearly, if (4.10) holds, then

$$(0, \ldots, 0, v_{i_1}, 0 \ldots, 0, v_{i_d}, 0 \ldots, 0) \in L.$$

That is a relatively short vector in the lattice since its norm is $\leqslant \frac{q}{2}(\sum_{i \in I} \ell_i)^{1/2}$.

The rank of the lattice L is m and the volume is q^{n-r}, the basis is easy to construct. A reduction algorithm (e.g., LLL [LLL82]) is applied to compute the reduced basis. Then, we extract short vectors and check whether short relations are found. Since we may want many short relations, the initial basis of L is modified and the reduction algorithm is applied again.

4.5 Computing the distributions $p_{r,I}$

We now present four different methods to compute the probabilities (4.6). To simplify notation, let $I = \{1, \ldots, d\}$ and $\mathcal{C}$ denote the event $A_i X = X_i, i \in I$. Let V be a matrix of size $t \times n$ and rank t such that the rows of V are in the space generated by the rows of $A_1, \ldots, A_d$. Then

$$p_{r,I}(v) = \Pr(VX = v \,|\, \mathcal{C}),$$

where $V = T_{r,I} B_r$. The results are summarised in Table 4.1, where $R = \sum_{i=1}^{d} q^{\ell_i}$ and $\ell_i = \operatorname{rank}(A_i)$. The term R appears in all methods because the corresponding computations depend on all $\sum_{i=1}^{d} q^{\ell_i}$ probability values.

Method	Formula	Complexity	Comments
Section 4.5.1	(4.12)	$dq^n + R$	-
Section 4.5.2	(4.13)	$dq^{\operatorname{rank}(A)} + R$	$A = (A_1, \ldots, A_d)$
Section 4.5.3	(4.15)	$dq^{\operatorname{rank}(W)} + R$	$\langle A_1 \rangle, \ldots, \langle A_d \rangle$ lin. indep. mod $\langle W \rangle$ and $\langle V \rangle \subseteq \langle W \rangle$
Section 4.5.4	(4.16)	$dq^{2 \cdot \operatorname{rank}(V)} + R$	$A_1, \ldots, A_d$ lin. indep.

Table 4.1. Summary of the methods for computing $\Pr(VX = v \,|\, \mathcal{C})$.

The first three methods are universal and the third one is the fastest of the three. The convolution method in Section 4.5.4 may be even faster, and it works if the rows of $A_1, \ldots, A_d$ are linearly independent. Remark that, even if $A_1, \ldots, A_d$ are linearly independent, the matrix W of smallest rank such that $\langle A_1 \rangle, \ldots, \langle A_d \rangle$ are linearly independent modulo $\langle W \rangle$ and $\langle V \rangle \subseteq \langle W \rangle$ may be $A = (A_1, \ldots, A_d)$. For instance, let A_1, A_2, A_3 be linearly independent rows ($\ell_1 = \ell_2 = \ell_3 = 1$) and $V = A_1 + A_2 + A_3$. Then $W = (A_1, A_2, A_3)$ and $\operatorname{rank}(W) = 3$. So, the method from Section 4.5.4 is faster in that case.

4.5.1 Basic formula

By the conditional probability formula,

$$p_{r,I}(v) = \frac{\Pr(VX = v, \mathcal{C})}{\Pr(\mathcal{C})}.$$

Since $X, X_1, \ldots, X_d$ are independent and X is uniformly distributed on $\mathbb{F}_q^n$, we have

$$\begin{aligned} \Pr(VX = v, \mathcal{C}) &= \sum_{x:Vx=v} \Pr(X = x, X_1 = A_1 x, \ldots, X_d = A_d x) \\ &= \frac{1}{q^n} \sum_{x:Vx=v} \prod_{j=1}^{d} P_j(A_j x), \end{aligned} \tag{4.12}$$

where the sum is over $x \in \mathbb{F}_q^n$ such that $Vx = v$. In order to compute $p_{r,I}(v)$, it is enough to compute only $\Pr(VX = v, \mathcal{C})$ for each $v \in \mathbb{F}_q^t$ since $\Pr(\mathcal{C}) = \sum_v \Pr(VX = v, \mathcal{C})$. The whole computation takes dq^n operations.

4.5.2 Change of variables

Let $k = \dim_{\mathbb{F}_q}\langle A_1, \ldots, A_d\rangle$ and let U be a matrix of size $k \times n$ constructed with linearly independent rows of $A_1, \ldots, A_d$. Then $A_j = A_j' U$ and $V = V'U$ for some matrices A_j' and V'. Let $Z = UX$. So, $A_j X = A_j' Z$ and $VX = V'Z$ are uniformly distributed as well and (4.12) implies

$$\Pr(VX = v, \mathcal{C}) = \frac{1}{q^k} \sum_{z:V'z=v} \prod_{j=1}^{d} P_j(A_j' z), \tag{4.13}$$

where the sum is over $z \in \mathbb{F}_q^k$ such that $V'z = v$. There are at most q^k terms in the sums (4.13) for all v and each term is a product of d numbers. Therefore, the cost of computing $p_{r,I}$ is dq^k operations.

4.5.3 Independence in $A_1, \ldots, A_d$ modulo $\langle W\rangle \supseteq \langle V\rangle$

This method may be efficient even if $k = \dim_{\mathbb{F}_q}\langle A_1, \ldots, A_d\rangle$ is large. Let W be a matrix of size $l \times n$ over $\mathbb{F}_q$ and of rank l. The linear spaces

$$\langle A_1\rangle, \ldots, \langle A_d\rangle \tag{4.14}$$

are called linearly independent modulo $\langle W\rangle$ if $\sum_{i=1}^{d} a_i \in \langle W\rangle$ and $a_i \in \langle A_i\rangle$ imply $a_i \in \langle W\rangle$. We will show how to construct a matrix W of lowest rank such that $\langle V\rangle \subseteq \langle W\rangle$ and (4.14) are linearly independent modulo $\langle W\rangle$. Then, we will give a formula to compute

$$\Pr(WX = w, \mathcal{C})$$

for every $w \in \mathbb{F}_q^l$. The probabilities $\Pr(VX = v, \mathcal{C})$ are then easy to deduce. The complexity of the computation is $dq^{\operatorname{rank}(W)}$ operations.

Let U be a linear space of tuples $(b_1, \ldots, b_d)$, where $b_i \in \langle A_i\rangle$, such that

$$b_1 + \cdots + b_d \in \langle V\rangle.$$

Let $\overline{U}$ be a space generated by all $b_1, \ldots, b_d$ such that $(b_1, \ldots, b_d) \in U$. Then W is a matrix whose rows are a basis of $\overline{U}$. Let us prove that (4.14) are linearly independent modulo such W. Let $\sum_{i=1}^{d} a_i \in \langle W \rangle$ and $a_i \in \langle A_i \rangle$. We need to show that $a_i \in \langle W \rangle$. We have $\sum_{i=1}^{d} a_i \in \sum_{i=1}^{d} b_i + \langle V \rangle$, for some $b_i \in \langle A_i \rangle \cap \langle W \rangle$ by the definition of W. Then $\sum_{i=1}^{d}(a_i - b_i) \in \langle V \rangle$ and therefore $(a_i - b_i) \in \langle W \rangle$. Hence, $a_i \in \langle W \rangle$ for $i = 1, \ldots, d$. The spaces (4.14) are linearly independent. The rank of W is the lowest by construction. The following statement is then true.

Lemma 4.5.1. *W is a lowest rank matrix such that $\langle V \rangle \subseteq \langle W \rangle$ and (4.14) are linearly independent.*

We now show how to construct a basis of U by solving a system of linear equations. Let $b_{i1}, \ldots, b_{it_i}$ be a basis for $\langle A_i \rangle / \langle V \rangle$, $i = 1, \ldots, d$. So $b_i = \sum_{j=1}^{t_i} \gamma_{ij} b_{ij} \in \langle A_i \rangle / \langle V \rangle$ for $\gamma_{i1}, \ldots, \gamma_{it_i} \in \mathbb{F}_q$. Therefore $(b_1, \ldots, b_d) \in U$ if and only if

$$\sum_{i=1}^{d} \sum_{j=1}^{t_i} \gamma_{ij} b_{ij} \in \langle V \rangle.$$

We take a set of linearly independent solutions. Each solution results in $(b_1, \ldots, b_d)$ and such b_i with the rows of V generate the space $\overline{U}$. We thus construct the matrix W.

We now show how to compute $\Pr(WX = w, \mathcal{C})$. Let $l = \operatorname{rank}(W)$ and K be a matrix of size $n \times (n - l)$ and of rank $n - l$ such that $WK = 0$. Then $Wx = w$ if and only if $x = x_0 + Ky$, where y is a column vector of length $n - l$ and $Wx_0 = w$. Let V_i be the linear space spanned by the columns of $A_i K$ and

$$\phi : \mathbb{F}_q^{n-l} \to V_1 \times \ldots \times V_d$$

be a linear mapping defined by $\phi(y) = (y_1, \ldots, y_d)$, where $y_i = A_i K y$.

Lemma 4.5.2. *The mapping ϕ is surjective and*

$$\begin{aligned} \Pr(WX = w, \mathcal{C}) &= \Pr(WX = w, A_1 X = X_1, \ldots, A_d X = X_d) \\ &= \frac{|Ker\ \phi|}{q^n} \prod_{i=1}^{d} \sum_{y_i \in V_i} P_i(w_i + y_i), \end{aligned} \tag{4.15}$$

where $w_i = A_i x_0$.

Proof. Let's prove that ϕ is surjective. If not, then the values of ϕ belong to a proper subspace of $V_1 \times \ldots \times V_d$. So there are $v_i \in \mathbb{F}_q^{\ell_i}$ such that $\sum_i v_i A_i K y = 0$ for every $y \in \mathbb{F}_q^{n-l}$ and there are non-zero vectors among $v_1 A_1 K, \ldots, v_d A_d K$. The equality $\sum_i v_i A_i K y = 0$ holds for any y if and only if $(\sum_i v_i A_i) K = 0$, and so $\sum_i v_i A_i \in \langle W \rangle$. By the definition of W, the latter implies $v_i A_i \in \langle W \rangle$. Hence $v_1 A_1 K = \cdots = v_d A_d K = 0$, which is a contradiction. Therefore ϕ is surjective.

By (4.12),

$$\Pr(WX = w, \mathcal{C}) = \frac{1}{q^n} \sum_{x : Wx = w} \prod_{i=1}^{d} P_i(A_i x) = \frac{1}{q^n} \sum_{y : x = x_0 + Ky} \prod_{i=1}^{d} P_i(A_i x_0 + A_i K y),$$

where the first sum is over $x \in \mathbb{F}_q^n$ such that $Wx = w$ and over $y \in \mathbb{F}_q^{n-\ell}$ in the second sum, and where $x = x_0 + Ky$. Hence,

$$\Pr(WX = w, \mathcal{C}) = \frac{|\mathrm{Ker}\,\phi|}{q^n} \sum_{y_1,\dots,y_d} \prod_{i=1}^{d} P_i(w_i + y_i) = \frac{|\mathrm{Ker}\,\phi|}{q^n} \prod_{i=1}^{d} \sum_{y_i} P_i(w_i + y_i),$$

where the sums are over $y_i \in V_i$ for $i = 1, \dots, d$. ■

Let r be the rank of the system of linear equations $\phi(y) = (0, \dots, 0)$. So $|\mathrm{Ker}\,\phi| = q^{n-l-r}$. The values $\sum_{y_i \in V_i} P_i(w_i + y_i)$ may be pre-computed for any i and $w_i \in \mathbb{F}_q^{\ell_i}$. It takes at most $\sum_{i=1}^{d} q^{\ell_i}$ operations. After that, the cost is dq^l operations. Then, the overall cost is $dq^l + \sum_{i=1}^{d} q^{\ell_i}$ operations. Recall that $l = \mathrm{rank}(W) \leqslant k = \dim_{\mathbb{F}_q}\langle A_1, \dots, A_d\rangle$. If $l < k$, this method is more efficient than that in Section 4.5.2.

4.5.4 Convolution formula

Let the rows in $A_1, \dots, A_d$ be linearly independent. Since $\langle V\rangle \subseteq \langle A_1, \dots, A_d\rangle$, we can represent $V = \sum_{i=1}^{d} V_i A_i$, where V_i are matrices of size $(t \times \ell_i)$. This representation is unique and may be found by solving a system of linear equations.

Lemma 4.5.3.

$$\Pr(VX = v \,|\, A_1X = X_1, \dots, A_dX = X_d) = \Pr\left(\sum_{i=1}^{d} V_iX_i = v\right). \tag{4.16}$$

Proof. Since the rows in $A_1, \dots, A_d$ are linearly independent, $A_1X, \dots, A_dX$ are independent uniformly distributed random variables. By the conditional probability formula,

$$\begin{aligned}\Pr(VX = v \,|\, A_1X = X_1, \dots, A_dX = X_d) &= \frac{\Pr\left(\sum_{i=1}^{d} V_iA_iX = v, A_1X = X_1, \dots, A_dX = X_d\right)}{\Pr(A_1X = X_1, \dots, A_dX = X_d)} \\ &= \frac{\Pr\left(\sum_{i=1}^{d} V_iX_i = v, A_1X = X_1, \dots, A_dX = X_d\right)}{\Pr(A_1X = X_1, \dots, A_dX = X_d)},\end{aligned}$$

where

$$\Pr(A_1X = X_1, \dots, A_dX = X_d) = \prod_{i=1}^{d} \Pr(A_iX = X_i) = \prod_{i=1}^{d} 1/q^{\ell_i} = q^{-\sum_{i=1}^{d} \ell_i}$$

and

$$\begin{aligned}\Pr\left(\sum_{i=1}^{d} V_iX_i = v, A_1X = X_1, \dots, A_dX = X_d\right) &= \sum_{\substack{v_1,\dots,v_d:\\ \sum_{i=1}^{d} V_iv_i = v}} \prod_{i=1}^{d} \Pr(A_iX = v_i) \prod_{i=1}^{d} P_i(v_i) \\ &= q^{-\sum_{i=1}^{d} \ell_i} \sum_{\substack{v_1,\dots,v_d:\\ \sum_{i=1}^{d} V_iv_i = v}} \prod_{i=1}^{d} P_i(v_i) \\ &= q^{-\sum_{i=1}^{d} \ell_i} \Pr\left(\sum_{i=1}^{d} V_iX_i = v\right).\end{aligned}$$

The sum is over $v_1, \ldots, v_d$ such that $\sum_{i=1}^{d} V_i v_i = v$. Therefore,

$$\Pr(VX = v \mid A_1 X = X_1, \ldots, A_d X = X_d) = \Pr\left(\sum_{i=1}^{d} V_i X_i = v\right).$$

■

It takes q^{ℓ_i} linear algebra operations to compute the distribution of $V_i X_i$. Then, $\Pr\left(\sum_{i=1}^{d} V_i X_i = v\right)$ may be computed iteratively by a convolution type formula because $V_i X_i$ are independent. That takes dq^{2t} operations. The overall cost of computing the distribution $\Pr(VX = v \mid \mathcal{C})$ is $\sum_{i=1}^{d} q^{\ell_i} + dq^{2\cdot\mathrm{rank}(V)}$. According to Section 4.5.3, $\langle W \rangle = \langle V_1 A_1, \ldots, V_d A_d \rangle$ since the rows in $A_1, \ldots, A_d$ are linearly independent. The cost to compute the conditional distribution $\Pr(WX = w \mid A_1 X = X_1, \ldots, A_d X = X_d)$ is $\sum_{i=1}^{d} q^{\ell_i} + dq^{\mathrm{rank}(W)}$. The conditional distribution on VX may be computed within the same cost since $\langle V \rangle \subseteq \langle W \rangle$. So, the convolution method is preferable if the rows $A_1, \ldots, A_d$ are linearly independent and $\mathrm{rank}(V) < \mathrm{rank}(W)/2$.

4.6 Analysis of the test-and-extend algorithm

To simplify notation, we assume that $\mathcal{I}_1 = \mathcal{I}_2 = \cdots = \mathcal{I}_n = \mathcal{I}$. Given the construction of the matrices $B_1, \ldots, B_n$ and the definition of a relation, every relation I for B_r is a relation for B_{r+1}. So, $\mathcal{I}_r \subseteq \mathcal{I}_{r+1}$. However, a relation I modulo B_{r+1} may not be a relation modulo B_r. In the latter case, we can still consider such $I \in \mathcal{I}_{r+1}$ as a *trivial* relation for B_r, i.e., it spans $\langle 0 \rangle$ in (4.5). Then, $t_{r,I} = 0$ and $p_{r,I}(0) = 1$ for such I. Thus, we can formally augment the set $\mathcal{I}_r$ with $I \in \mathcal{I}_{r+1} \setminus \mathcal{I}_r$ and get $\mathcal{I}_r = \mathcal{I}_{r+1}$.

4.6.1 Success probability of the algorithm

The execution of the algorithm is successful if $b_r = B_r X$ is not rejected for every $r = 1, \ldots, n$, where X is the correct solution. The success probability is defined by

$$\beta = \Pr(S_r(b_r) \geqslant c_r,\ 1 \leqslant r \leqslant n \mid A_1 X = X_1, \ldots, A_N X = X_N).$$

We will show how to compute the thresholds $c_1, \ldots, c_n$ given β. Let

$$\mathcal{S} = \begin{pmatrix} S_1(b_1) \\ \vdots \\ S_n(b_n) \end{pmatrix} = \sum_{I \in \mathcal{I}} \mathcal{S}_I, \quad \mathcal{S}_I = \begin{pmatrix} \ln p_{1,I}(b_{1,I}) \\ \vdots \\ \ln p_{n,I}(b_{n,I}) \end{pmatrix}, \quad c = \begin{pmatrix} c_1 \\ \vdots \\ c_n \end{pmatrix},$$

where $b_{r,I} = T_{r,I} b_r$ by the definition of S_r in (4.8). The inequalities $S_r(b_r) \geqslant c_r$ may be written $\mathcal{S} \geqslant c$ by considering entry-wise comparison. Then

$$\beta = \Pr(\mathcal{S} \geqslant c \mid A_i X = X_i, i = 1, \ldots, N).$$

Since B_r is a submatrix of B_{r+1} in its first r rows, we can choose the matrices $T_{r,I}$ such that $T_{r,I}$ is a submatrix of $T_{r+1,I}$ in its first $t_{r,I}$ rows and first r columns:

$$T_{r+1,I} = \begin{pmatrix} T_{r,I} & * \\ * & * \end{pmatrix}.$$

Then, $b_{r,I} = T_{r,I} b_r$ is a subvector of $b_{r+1,I} = T_{r+1,I} b_{r+1}$ in its first $t_{r,I}$ entries. The mean of $\mathcal{S}_I$ is

$$\mu_I = \begin{pmatrix} \mu_{1,I} \\ \mu_{2,I} \\ \vdots \\ \mu_{n,I} \end{pmatrix}, \quad \mu_{r,I} = \sum_{\nu} p_{r,I}(\nu) \ln p_{r,I}(\nu),$$

where ν runs over $\mathbb{F}_q^{t_{r,I}}$. The mean of $\mathcal{S}$ is therefore $\mu = \sum_{I \in \mathcal{J}} \mu_I$. Let Q_I be the covariance matrix of $\mathcal{S}_I$. The entry in row i and column j of Q_I, $j \geqslant i$, is

$$\sum_{\nu} p_{j,I}(\nu) \ln p_{i,I}(\nu_i) \ln p_{j,I}(\nu) - \mu_{i,I} \mu_{j,I},$$

where ν runs over $\mathbb{F}_q^{t_{j,I}}$ and ν_i is the vector in the first $t_{i,I}$ entries of ν. This is because $j \geqslant i$ and $b_{i,I} = T_{i,I} b_i$ is the vector in the first $t_{i,I}$ coordinates of $b_{j,I} = T_{j,I} b_j$.

The distribution of $\mathcal{S}_I$ only depends on the distribution of X_i, $i \in I$. If any distinct $I_1, I_2 \in \mathcal{J}$ are disjoint, then $\mathcal{S}_{I_j}$, $I_j \in \mathcal{J}$, are independent and the covariance matrix of $\mathcal{S}$ is $Q = \sum_{I \in \mathcal{J}} Q_I$. In practice, the sets I are small (of size at most d) randomly looking subsets of $\{1, \ldots, N\}$. They are mostly pairwise disjoint. For the same reason, for large enough $|\mathcal{J}|$, the sum $\mathcal{S} = \sum_{I \in \mathcal{J}} \mathcal{S}_I$ approximately follows the multivariate normal distribution $\mathbf{N}(\mu, Q)$. Given β, we can compute the threshold c such that $\Pr(\mathbf{N}(\mu, Q) \geqslant c) = \beta$.

4.6.2 Number of nodes in the tree

The complexity of the algorithm is defined by the number of nodes visited during the tree search. At level r a current node b_r is tested with (4.7) and (4.8). The number of nodes b_r to test at level r is the number of survivors b_{r-1} times q. We now show how to compute the number of incorrect survivors b_r.

Let X be taken from the uniform distribution on $\mathbb{F}_q^n$. Therefore, $b_r = B_r X$ is uniformly distributed on $\mathbb{F}_q^r$ and $b_{r,I} = T_{r,I} b_r$ is uniformly distributed on $\mathbb{F}_q^{t_{r,I}}$. Let $\mathcal{E}_{r,I}$ denote the event $p_{r,I}(b_{r,I}) \neq 0$. Also, let $K_{r,I}$ denote the size of the support of $p_{r,I}$, i.e., the number of $\nu \in \mathbb{F}_q^{t_{r,I}}$ such that $p_{r,I}(\nu) \neq 0$. Clearly, $\Pr(\mathcal{E}_{r,I}) = K_{r,I}/q^{t_{r,I}}$. Let $\mathcal{E}_r$ be the joint event $\{\mathcal{E}_{r,I}, I \in \mathcal{J}\}$. If any distinct $I_1, I_2 \in \mathcal{J}$ are disjoint, the events $\mathcal{E}_{r,I}$ are independent. In practice this is likely to happen, so we may assume

$$\varepsilon_r = \Pr(\mathcal{E}_r) = \prod_{I \in \mathcal{J}} K_{r,I}/q^{t_{r,I}}.$$

Let

$$\mathcal{S}(r) = \begin{pmatrix} \mathcal{S}_1(b_1) \\ \vdots \\ \mathcal{S}_r(b_r) \end{pmatrix} = \sum_{I \in \mathcal{J}} \mathcal{S}_I(r), \quad \mathcal{S}_I(r) = \begin{pmatrix} \ln p_{1,I}(b_{1,I}) \\ \vdots \\ \ln p_{r,I}(b_{r,I}) \end{pmatrix}, \quad c(r) = \begin{pmatrix} c_1 \\ \vdots \\ c_r \end{pmatrix},$$

where $b_{r,I} = T_{r,I} b_r$ and c_i are found from $\Pr(\mathbf{N}(\mu, Q) \geqslant c) = \beta$ as in Section 4.6.1. The current b_r passes the tests up to level r if and only if $\mathcal{E}_r$ holds and $\mathcal{S}(r) > c(r)$. The probability of this event is

$$\Pr(\mathcal{S}(r) > c(r), \mathcal{E}_r) = \Pr(\mathcal{E}_r) \cdot \Pr(\mathcal{S}(r) > c(r) \mid \mathcal{E}_r).$$

We show how to compute $\alpha(r) = \Pr(\mathcal{S}(r) > c(r) \mid \mathcal{E}_r)$. Let $\mu_{r,I} = \begin{pmatrix} \mu_{1,I} \\ \vdots \\ \mu_{r,I} \end{pmatrix}$ and $Q_{r,I}$ be the mean vector and the covariance matrix of $\mathcal{S}_I(r)$, respectively. Since $b_{j,I}$ is a vector in the first $t_{j,I}$ entries of $b_{r,I}$, then

$$\mu_{j,I} = \frac{\sum_{\nu_r} \ln p_{j,I}(\nu_j)}{K_{r,I}},$$

where the sum is over all $\nu_r \in \mathbb{F}_q^{t_{r,I}}$ such that $p_{r,I}(\nu_r) \neq 0$ and ν_j is the vector comprising the first $t_{j,I}$ entries of ν_r. Notice that $p_{r,I}(\nu_r) \neq 0$ implies $p_{j,I}(\nu_j) \neq 0$. The entry in row i and column j of the covariance matrix $Q_{r,I}$ is

$$\sum_{\nu_r} \frac{\ln p_{i,I}(\nu_i) \ln p_{j,I}(\nu_j)}{K_{r,I}} - \mu_{i,I}\mu_{j,I},$$

where the sum is over all ν_r such that $p_{r,I}(\nu_r) \neq 0$. For large $|\mathcal{I}|$, the random variable $\mathcal{S}(r) = \sum_{I \in \mathcal{I}} \mathcal{S}_I(r)$ approximately follows a multivariate normal distribution $\mathbf{N}(\mu_r, Q_r)$, where $\mu_r = \sum_{I \in \mathcal{I}} \mu_{r,I}$ and $Q_r = \sum_{I \in \mathcal{I}} Q_{r,I}$. Therefore

$$\alpha(r) \approx \Pr(\mathbf{N}(\mu_r, Q_r) > c(r)).$$

Hence, the number of incorrect b_r which pass the test at level r is approximately

$$\varepsilon_r \cdot \alpha(r) \cdot q^r.$$

4.6.3 Time and space complexity

Pre-computation

The worst-case time complexity for computing relations of weight d is given by the brute force method. It is $\binom{N}{d}$ linear algebra operations since it requires solving $\binom{N}{d}$ systems of linear equations. The search for relations is fully parallelisable.

For small d, the distributions $p_{r,I}$ are relatively easy to compute (see Table 4.1) and more likely to be non-uniform. We do not expect many useful relations if N is moderate and r is small. For larger r, we can obtain a great number of useful relations. On the other hand, for larger d, the time complexity to compute the distributions $p_{r,I}$ increases and the distributions tend to be uniform. We do not know beforehand the best technique to compute each of the distributions $p_{r,I}$. For all $I \in \mathcal{I}$, we have that $\operatorname{rank}(A_1, \ldots, A_d) \leqslant \sum_{i=1}^{d} \ell_i$, where $\ell_i = \operatorname{rank}(A_i)$. Let $\ell_i = \ell$ for $1 \leqslant i \leqslant N$. With the method in Section 4.5.2, we may (very) roughly estimate the worst-case time complexity for computing the distributions as $O(|\mathcal{I}|(dq^{d\ell} + dq^{\ell}))$ operations. For each relation I, we need to keep at most $q^{t_{r,I}}$ probability values for a given r. The highest number is when $r = n$. Let $t_{n,I} = t_n$ for all $I \in \mathcal{I}$. Then, the worst-case space complexity for storing the distributions is about $O(|\mathcal{I}|q^{t_n})$. The computation of the distributions $p_{r,I}$ is fully parallelisable as well.

Tree search

We need $|\mathcal{J}_r|$ arithmetic operations to compute the statistic S_r in (4.8) for each visited node at level r. So, the complexity of the tree search is

$$\sum_{r=0}^{n-1} \varepsilon_r \cdot \alpha(r) \cdot q^{r+1} \cdot |\mathcal{J}_{r+1}|$$

arithmetic operations, where we set $\varepsilon_0 = 1, \alpha(0) = 1$. We do not require additional space for the tree search.

Hybrid variant

This variation requires space of order q^{r_0} to execute the FFT. We can use a large number of relations in $\mathcal{J}_{r_0}$, up to q^{r_0}, within the cost of one application of the FFT. In our experiments, that reduces the time complexity of the tree search. However, since the number of relations $\mathcal{J}_{r_0}$ is large, there may be dependencies between the summands in the definition (4.8) of the statistic for $r = r_0$. Therefore, a normal approximation to the distribution of S_{r_0}, as in Section 4.6.1, may not be accurate and the time complexity of this variation is generally difficult to estimate.

4.7 Application to the filter generator

In this section we apply our test-and-extend algorithm in Section 4.3 to some instances of the filter generator.

Let N bits of the keystream be available. At the beginning of this chapter, the matrices

$$A_i = \Lambda M^{i-1},$$

$i = 1, \dots, N$, were constructed. The keystream bits are written as $z_i = f(A_i X)$, where $X = S_1$. Let $f^{-1}(z_i)$ denote the pre-image of z_i under f (the set of all possible values $a \in \mathbb{F}_2^\ell$ such that $z_i = f(a)$). The probability distribution

$$P_i(a) = \begin{cases} \frac{1}{|f^{-1}(z_i)|} & \text{if } a \in f^{-1}(z_i), \\ 0 & \text{otherwise,} \end{cases}$$

is defined for a random variable X_i on the values of $A_i X$. We assume X to be uniformly distributed on $\mathbb{F}_2^n$ and X_i to be independent. Then, the key recovery attack on the filter generator is to find the value $X = x$ with maximum probability under the condition that

$$A_i X = X_i,\ i = 1, \dots, N.$$

For the experiments in the following sections, we used the statistical software R [R C21] and the package mvtnorm [Gen+21; GB09] to get the vector c (see Section 4.6.1) which defines the tests (4.8), and the probabilities $\alpha(r)$ (see Section 4.6.2).

4.7.1 Matrices B_r and relations in the experiments

We generate the $n \times n$ matrix B_n of rank n and the matrices B_r, $r = 1, \dots, n-1$, are just the corresponding sub-matrices of B_n consisting of the first r rows. We generate B_n by randomly taking linearly independent vectors from aA_i, $i = 1, \dots, N$, where $a = (a_1, \dots, a_\ell)$ and the linear Boolean function $a_1 x_1 + \cdots + a_\ell x_\ell$ is one of the best linear approximations to the filtering function f. The vector aA_i belongs to the space generated by the rows of A_i. So, there are at least r relations modulo B_r of weight 1, thus providing with a few good distributions $p_{r,I}$. This choice proved to be successful for the attack.

The complexity of the tree search, for both variations, is influenced by the number of relations in each set $\mathcal{I}_r$ (see Section 4.6.3). Therefore, we can afford using only a bounded number of relations in practice. The relations $I \in \mathcal{I}_r$ can be ranked according to the size of the support and the entropy of their distributions $p_{r,I}$ on $\mathbb{F}_q^{t_{r,I}}$, and filter out the inferior ones. Recall that the size of the support of $p_{r,I}$ is denoted by $K_{r,I}$. The normalised q-ary entropy is

$$H(p_{r,I}) = -\sum_{v \in \mathbb{F}_q^{t_{r,I}}} p_{r,I}(v) \log_q p_{r,I}(v) - t_{r,I},$$

Let $I, J \in \mathcal{I}_r$. We say that I is a better distinguisher than J (i.e., further away from being uniform) if

$$\frac{K_{r,I}}{q^{t_{r,I}}} < \frac{K_{r,J}}{q^{t_{r,J}}}$$

or if

$$H(p_{r,I}) < H(p_{r,J}) \quad \text{when} \quad \frac{K_{r,I}}{q^{t_{r,I}}} = \frac{K_{r,J}}{q^{t_{r,J}}}.$$

For each r, the best m_r relations are kept in $\mathcal{I}_r$, where m_r are parameters. For each $I \in \mathcal{I}_r$, the entropy of $p_{r,I}$ is computed with $O(q^{t_{r,I}})$ operations. Ranking (i.e., sorting) the relations in $\mathcal{I}_r$ has complexity $O(m_r \log m_r)$. Let $t_{r,I} = t_r$ for all $I \in \mathcal{I}_r$. Then, choosing the best relations in $\mathcal{I}_r$ has complexity $O(m_r q^{t_r} + m_r \log m_r)$. The computation of the entropy for all $I \in \mathcal{I}_r$ is fully parallelisable.

Let I be a relation modulo B_r. Then, the index $j \in I$ is called *irrelevant* modulo B_r if $v_j = 0$ for every solution v_i, $i \in I$, and $v \neq 0$ to (4.10). That means that the distribution $p_{r,I}$ does not depend on X_j, even if $j \in I$. The other indices in I are called *relevant* modulo B_r. Two relations are *equivalent* modulo B_r if their set of relevant indices modulo B_r are equal.

The sets $\mathcal{I}_r$, $r = 1, \dots, n$, are constructed as follows. We obtain a large set $\mathcal{I}$ of relations from a fixed matrix B_{r_0}. For each B_r, $r = 1, \dots, n$, we get the classes of equivalent relations modulo B_r, apply the ranking criteria above to those equivalence classes and choose a suitable number of them to create $\mathcal{I}_r$. We try to choose the relations such that $I \cap J = \emptyset$ for distinct $I, J \in \mathcal{I}_r$, i.e., pairwise disjoint relations. In that case, the distribution of the statistic $\mathcal{S}_r$ may be well approximated with the Central Limit Theorem. In practice, not all relations in $\mathcal{I}_r$ are pairwise disjoint. However, our experimental results show that the approximation is still good in that case. Also, the sets $\mathcal{I}_r$ are chosen to be disjoint. Hence, the tests (4.7) and (4.8) may be considered independent for $r = 1, \dots, n$. In particular, the statistics $\mathcal{S}_r$, $r = 1, \dots, n$, are independent and the covariance matrix Q for their joint distribution is diagonal. That allows our experimental results to be as close as possible to the theoretical analysis based on the normal approximation to the distribution of $\mathcal{S}_r$.

4.7.2 Detailed toy example

Let the keystream $z_1, \dots, z_{11} = 1,0,1,0,0,0,0,0,0,1,0$ be produced by the following filter generator:

- $g(x) = x^7 + x^6 + x^5 + x^2 + 1$,
- $f(x_1, x_2, x_3) = x_1 + x_1x_2 + x_2x_3$,
- $(k_1, k_2, k_3) = (0, 2, 5)$.

We decided not to have contiguous indices k_i and we chose $k_3 = 5$ instead of 6 to maximise the memory Γ (see Section 3.3).

We compute the matrices $A_i = \Lambda M^{i-1}$, $i = 1, \dots, 11$, with

$$M = \begin{pmatrix} 0&1&0&0&0&0&0\\ 0&0&1&0&0&0&0\\ 0&0&0&1&0&0&0\\ 0&0&0&0&1&0&0\\ 0&0&0&0&0&1&0\\ 0&0&0&0&0&0&1\\ 1&1&1&0&0&1&0 \end{pmatrix} \quad \text{and} \quad \Lambda = \begin{pmatrix} e_1\\ e_3\\ e_6 \end{pmatrix} = \begin{pmatrix} 1&0&0&0&0&0&0\\ 0&0&1&0&0&0&0\\ 0&0&0&0&0&1&0 \end{pmatrix}:$$

$$A_1 = \begin{pmatrix} 1&0&0&0&0&0&0\\ 0&0&1&0&0&0&0\\ 0&0&0&0&0&1&0 \end{pmatrix}, \quad A_2 = \begin{pmatrix} 0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0\\ 0&0&0&0&0&0&1 \end{pmatrix}, \quad A_3 = \begin{pmatrix} 0&0&1&0&0&0&0\\ 0&0&0&0&1&0&0\\ 1&1&1&0&0&1&0 \end{pmatrix},$$

$$A_4 = \begin{pmatrix} 0&0&0&1&0&0&0\\ 0&0&0&0&0&1&0\\ 0&1&1&1&0&0&1 \end{pmatrix}, \quad A_5 = \begin{pmatrix} 0&0&0&0&1&0&0\\ 0&0&0&0&0&0&1\\ 1&1&0&1&1&1&0 \end{pmatrix}, \quad A_6 = \begin{pmatrix} 0&0&0&0&0&1&0\\ 1&1&1&0&0&1&0\\ 0&1&1&0&1&1&1 \end{pmatrix},$$

$$A_7 = \begin{pmatrix} 0&0&0&0&0&0&1\\ 0&1&1&1&0&0&1\\ 1&1&0&1&0&0&1 \end{pmatrix}, \quad A_8 = \begin{pmatrix} 1&1&1&0&0&1&0\\ 1&1&0&1&1&1&0\\ 1&0&0&0&1&1&0 \end{pmatrix}, \quad A_9 = \begin{pmatrix} 0&1&1&1&0&0&1\\ 0&1&1&0&1&1&1\\ 0&1&0&0&0&1&1 \end{pmatrix},$$

$$A_{10} = \begin{pmatrix} 1&1&0&1&1&1&0\\ 1&1&0&1&0&0&1\\ 1&1&0&0&0&1&1 \end{pmatrix}, \quad A_{11} = \begin{pmatrix} 0&1&1&0&1&1&1\\ 1&0&0&0&1&1&0\\ 1&0&0&0&0&1&1 \end{pmatrix}.$$

Next, we assign the probability distributions to the random variables X_i. Table 4.2 shows the truth table of f. Following the same order of vectors as in Table 4.2, the distributions $P_{(0)} = (1/4, 1/4, 1/4, 0, 0, 0, 1/4, 0)$ and $P_{(1)} = (0, 0, 0, 1/4, 1/4, 1/4, 0, 1/4)$ correspond to $f(a) = 0$ and $f(a) = 1$, respectively, where $a \in \mathbb{F}_2^3$. Hence, the distributions of X_i are $P_i = P_{(0)}$, for $i = 2, 4, 5, 6, 7, 8, 9, 11$, and $P_i = P_{(1)}$, for $i = 1, 3, 10$.

a	$(0,0,0)$	$(0,0,1)$	$(0,1,0)$	$(0,1,1)$	$(1,0,0)$	$(1,0,1)$	$(1,1,0)$	$(1,1,1)$
$f(a)$	0	0	0	1	1	1	0	1

Table 4.2. Truth table of $f(x_1, x_2, x_3) = x_1 + x_1x_2 + x_2x_3$.

We now get the matrices $B_1, \dots, B_7$. B_7 is constructed by randomly taking linearly independent vectors from $(1, 1, 0)A_i$, i.e., $x_1 + x_2$ is used as the linear approximation to f. We obtained

$$B_7 = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \end{pmatrix}.$$

We used brute force with B_3 for finding relations of weight $d = 2$ and we obtained 45 relations:

$$\mathcal{I} = \left\{ \begin{array}{l} \{1,2\},\{1,3\},\{1,5\},\{1,6\},\{1,7\},\{1,8\},\{1,9\},\{1,10\},\{1,11\}, \\ \{2,3\},\{2,4\},\{2,5\},\{2,6\},\{2,7\},\{2,9\},\{2,10\},\{2,11\},\{3,4\}, \\ \{3,6\},\{3,7\},\{3,9\},\{3,10\},\{3,11\},\{4,5\},\{4,6\},\{4,8\},\{4,10\}, \\ \{4,11\},\{5,6\},\{5,7\},\{5,8\},\{5,9\},\{5,11\},\{6,7\},\{6,10\},\{7,8\}, \\ \{7,9\},\{7,10\},\{7,11\},\{8,9\},\{8,10\},\{8,11\},\{9,10\},\{9,11\},\{10,11\} \end{array} \right\}.$$

Let us show how the set of relevant indices change for different B_r. Take the relations $I_1 = \{1,2\}$ and $I_6 = \{1,8\}$. The corresponding matrices A_i involved in these relations are

$$A_1 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}, \quad A_2 = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \quad A_8 = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}.$$

For the first three levels, we have

- Level 1:

$$\begin{aligned} \begin{pmatrix} 0 \end{pmatrix} B_1 &= \begin{pmatrix} 0 & 0 & 0 \end{pmatrix} A_1 + \begin{pmatrix} 0 & 0 & 0 \end{pmatrix} A_2, \\ \begin{pmatrix} 0 \end{pmatrix} B_1 &= \begin{pmatrix} 0 & 0 & 0 \end{pmatrix} A_1 + \begin{pmatrix} 0 & 0 & 0 \end{pmatrix} A_8; \end{aligned}$$

- Level 2:

$$\begin{aligned} \begin{pmatrix} 0 & 1 \end{pmatrix} B_2 &= \begin{pmatrix} 1 & 1 & 0 \end{pmatrix} A_1 + \begin{pmatrix} 0 & 0 & 0 \end{pmatrix} A_2, \\ \begin{pmatrix} 0 & 1 \end{pmatrix} B_2 &= \begin{pmatrix} 1 & 1 & 0 \end{pmatrix} A_1 + \begin{pmatrix} 0 & 0 & 0 \end{pmatrix} A_8; \end{aligned}$$

- Level 3:

$$\begin{aligned} \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} B_3 &= \begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} A_1 + \begin{pmatrix} 0 & 0 & 0 \\ 1 & 1 & 0 \end{pmatrix} A_2, \\ \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} B_3 &= \begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} A_1 + \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \end{pmatrix} A_8. \end{aligned}$$

That is, I_1 and I_6 are trivial relations for level 1 (i.e., they span $\langle 0 \rangle$). The only relevant index of I_1 and I_6 at level 2 is 1, hence they are equivalent modulo B_2. Finally, they are no longer equivalent at level 3 since their set of relevant indices with B_3 are distinct; actually, they are not equivalent in all subsequent levels.

The probability distributions (4.6) for the relations in $\mathcal{J}$ above were computed using B_7. After ranking the relations in the sets $\mathcal{J}$ modulo B_r, we heuristically created

$$
\begin{aligned}
\mathcal{J}_1 &= \{\{5,7\},\{1,5\}\}\,,\\
\mathcal{J}_2 &= \{\{1,6\},\{1,7\},\{2,5\}\}\,,\\
\mathcal{J}_3 &= \{\{8,10\},\{2,11\},\{3,7\},\{4,8\}\}\,,\\
\mathcal{J}_4 &= \{\{2,4\},\{4,6\},\{2,6\},\{2,7\},\{2,10\}\}\,,\\
\mathcal{J}_5 &= \{\{5,11\},\{1,2\},\{2,9\},\{7,11\},\{6,7\},\{1,11\},\{4,5\},\{10,11\},\{1,8\},\{5,8\}\}\,,\\
\mathcal{J}_6 &= \{\{1,10\},\{3,4\},\{5,9\},\{8,11\},\{4,10\},\{3,6\},\{3,9\}\}\,,\\
\mathcal{J}_7 &= \{\{8,9\}\}\,.
\end{aligned}
$$

Then, we computed the covariance matrix and mean vector for the multivariate distributions as in Sections 4.6.1 and 4.6.2. Using $\beta = 0.9$, we obtained the vector of thresholds

$$c = (-2.8344, -8.8069, -15.4057, -17.0976, -39.5219, -28.2609, -4.0000).$$

The tree search found a unique candidate solution $b = (0,0,1,0,0,1,1)^T$. Figure 4.1 depicts the tree traversal for this example. Then, $B_7X = b$ and solving this linear system yields $X = (1,0,1,1,0,1,0)^T$, which is the correct initial state. Figure 4.2 shows the theoretical and experimental number of survivors at each level of the tree search. According to Section 4.6.3, the theoretical time complexity is $O(2^{5.80735})$ while the experimental one is $O(2^{5.12928})$. Since this example is very small, we computed these complexities following the exact equation in Section 4.6.3. With bigger instances, however, the complexity is determined by the level r with the highest number of survivors. This is because in our experiments such level r also has the maximum value of $|\mathcal{J}_r|$.

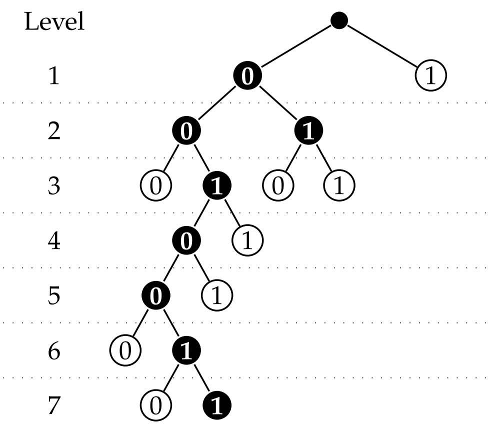


Figure 4.1. Tree traversal for the toy example. Filled nodes represent the survivor nodes. Non-filled nodes represent rejected nodes whose branch is not traversed.

We also applied the hybrid variant with $r_0 = 3$ using all 45 relations modulo B_3. That is, we computed the value of the statistic $S_3(b_3)$ for all $b_3 \in \mathbb{F}_2^3$ using the FFT, sorted the candidates at that level and applied the tree search in that order. The result of ordering according to the value of the statistic was

$$(0,1,0),\quad (1,1,0),\quad (1,1,1),\quad (0,1,1),\quad (0,0,1),\quad (1,0,0),\quad (0,0,0),\quad (1,0,1).$$

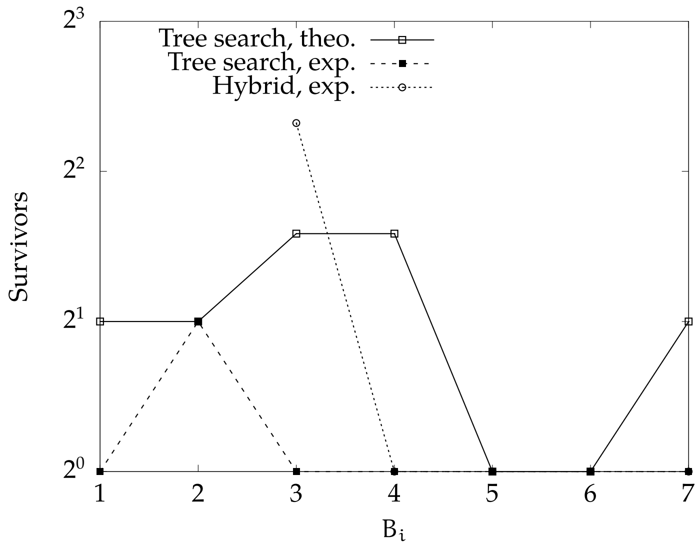


Figure 4.2. Number of survivors for the toy example.

Neither of the first four candidates survived at level 4. The fifth candidate is the one corresponding to the correct initial state, which we recovered, and the tree search stopped at this point. Figure 4.2 also shows the number of survivors following this hybrid method (FFT + tree search). For most of the relations, we have that $\ell_i = 2$. The complexity of the FFT is $O\left(\sum_{i=1}^{45} \ell_i \cdot 2^{\ell_i} + 3 \cdot 2^3\right) \approx O\left(2^{8.58496}\right)$ (see Section 4.2.2). The complexity of the tree search part is $O(2^{5.42626})$. Hence, the complexity of the hybrid variant is given by the FFT. Due to the size of this example, the simple tree search is more efficient. For bigger instances, however, the hybrid variant yields the best results.

4.7.3 Experimental results

We now present results of the new method applied to four instances of the filter generator. The method requires a significantly lower number of keystream bits compared to fast correlation attacks, methods based on the Berlekamp-Massey algorithm [Mas69] and Fast Algebraic Attacks [CM03; Cou03]. So, we compare the efficiency of the new method with brute force. The latter requires $2^n - 1$ trials of the LFSR initial state. For each candidate, we clock the LFSR and generate N bits of the keystream. Therefore, brute force takes essentially $N2^n$ operations.

In the first two experiments, $n = 40$ and the filtering functions f depend on $\ell = 5$ and 7 variables, respectively. We were able to explicitly recover the LFSR initial state with $N = 5\,000$ keystream bits and significantly faster than brute force. The best complexity was achieved with the hybrid variant: $2^{32.06}$ and $2^{35.19}$ additions of reals, respectively, to compute the values of the statistics. The results closely fit the theoretical prediction. In the last two experiments, $n = 64$ and 80, respectively, $\ell = 5$ and $N = 10\,000$. The tree search was executed up to some intermediate level. The complexity was then extrapolated to the whole tree. Again, the best result was achieved with the hybrid variant: $2^{57.39}$ and $2^{70.95}$ additions of reals, respectively.

In the experiments below, we used instances of the filter generator which employ “components” from the existing literature, such as the degree-40 feedback polynomial in [JJ99b] and the filtering Boolean function from Grain-v1 [HJM07] (see Section 4.8.3). In the first three experiments, we used feedback polynomials with high weight and input indices k_i that maximise the memory (see Section 3.3). In the last experiment, we follow closely the definition of Grain-v1, but maximise the memory when choosing the last input to the filtering function. Under various criteria (for example [Gol96b]), the devices are hard instances of the filter generator.

Experiment 1

We used $N = 5\,000$ keystream bits generated by the following device:

- $g(x) = x^{40}+x^{38}+x^{33}+x^{32}+x^{29}+x^{27}+x^{25}+x^{21}+x^{19}+x^{17}+x^{12}+x^{11}+x^{9}+x^{5}+x^{3}+x+1,$
- $f(x_1,\dots,x_5) = x_2+x_5+x_1x_4+x_3x_4+x_4x_5+x_1x_2x_3+x_1x_3x_4+x_1x_3x_5+x_2x_3x_5+x_3x_4x_5,$
- $(k_1,\dots,k_5) = (0,7,15,26,39).$

The polynomial g is a common choice in the literature. The filtering function f is the one used in Grain-v1. Notice that the input spacings to f are coprime and span the whole register.

For this experiment, we used $x_1 + x_3 + x_4$ as the linear approximation to f. We used brute force with B_{10} for finding relations of weight $d = 3$. By equation (4.11), the expected number of relations is $2^{19.27730}$ and we obtained $571\,986 \approx 2^{19.12562}$. Next, we heuristically created $\mathcal{J}$ by selecting 15 000 relations and computed their probability distributions (4.6) with B_{40}. After sorting the relations, we heuristically chose to create $\mathcal{J}_r$ such that $|\mathcal{J}_r| = 50$ for $r = 1,\dots,10$, $|\mathcal{J}_r| = 150$ for $r = 11,\dots,20$ and $|\mathcal{J}_r| = 300$ for $r = 21,\dots,40$. We then computed the covariance matrix and mean vector for the multivariate distributions to get the vector c of thresholds with $\beta = 0.9$.

The simple tree search found a unique solution corresponding to the correct initial state. Figure 4.3 shows the number of theoretical and experimental survivors from the tree search. The maximum of theoretical survivors is $2^{28.58805}$ at B_{30}. For the experimental survivors, it is $2^{28.34194}$ at B_{30}. Since $|\mathcal{J}_{30}| = 300 \approx 2^{8.22881}$, the theoretical complexity is $O(2^{36.81686})$ and in practice it was $O(2^{36.57075})$.

We applied the hybrid variant with $r_0 = 20$ using $2\,269 \approx 2^{11.14784}$ relations. These are all relations in $\mathcal{J}$ mod B_{20} whose support have a non-uniform probability distribution (at level 20). The correct initial state was recovered after executing the tree search on $32\,603 \approx 2^{14.99271}$ candidate solutions. Figure 4.3 shows the number of survivors with the hybrid variant. We have that $\ell_i = 1$ for almost all relations. Then, the complexity of the FFT is $O\left(\sum_{i=1}^{2269} \ell_i \cdot 2^{\ell_i} + 20 \cdot 2^{20}\right) \approx O\left(2^{24.32224}\right)$. The maximum number of survivors is $2^{23.83588}$ at B_{30}. Since $|\mathcal{J}_{30}| = 300 \approx 2^{8.22881}$, the complexity of the tree search is $O(2^{32.06469})$. Hence, the complexity of the hybrid variant is given by the tree search part. Notice that the hybrid variant performs better in this case compared to the simple tree search.

Experiment 2

We used $N = 5\,000$ keystream bits generated by the following device:

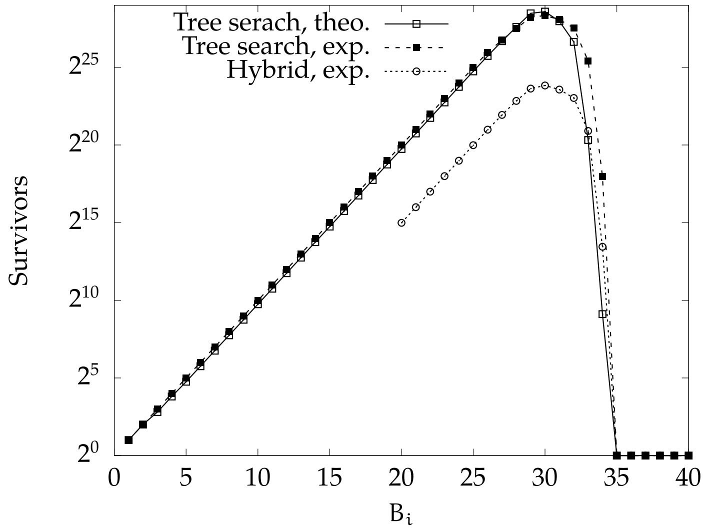


Figure 4.3. Number of survivors for experiment 1.

- $g(x) = x^{40}+x^{38}+x^{33}+x^{32}+x^{29}+x^{27}+x^{25}+x^{21}+x^{19}+x^{17}+x^{12}+x^{11}+x^{9}+x^{5}+x^{3}+x+1$,
- $f(x_1,\ldots,x_7) = 1+x_1+x_2+x_3+x_4+x_5+x_6+x_1x_7+x_2(x_3+x_7)+x_1x_2(x_3+x_6+x_7)$,
- $(k_1,\ldots,k_7) = (0,3,8,15,26,31,39)$.

This device is taken from [Lev+03]. The authors did not specify the input spacings to f, but they reported that $k_1,\ldots,k_7$ are taken to be coprime. In our case, they are coprime and span the whole register.

For this experiment, we used $x_1+x_4+x_5+x_6+x_7$ as the linear approximation to f. We used brute force with B_5 for finding relations of weight $d = 3$. The expected number of relations is $2^{20.27730}$ and we obtained $1\,185\,783 \approx 2^{20.17740}$. Next, we heuristically created $\mathcal{J}$ by selecting 15 000 relations and computed their probability distributions (4.6) with B_{35}. After sorting the relations, we heuristically chose to create $\mathcal{J}_r$ such that $|\mathcal{J}_r| = 50$ for $r = 1,\ldots,10$, $|\mathcal{J}_r| = 150$ for $r = 11,\ldots,20$ and $|\mathcal{J}_r| = 300$ for $r = 21,\ldots,40$. We then computed the covariance matrix and mean vector for the multivariate distributions to get the vector c of thresholds with $\beta = 0.9$.

The simple tree search found 14 solutions which included the one corresponding to the correct initial state. Figure 4.4 shows the number of theoretical and experimental survivors from the tree search. The maximum of theoretical survivors is $2^{30.30912}$ at B_{32}. For the experimental survivors, it is $2^{27.83228}$ at B_{32}. Since $|\mathcal{J}_{32}| = 300 \approx 2^{8.22881}$, the theoretical complexity is $O(2^{38.53793})$ and in practice it was $O(2^{36.06109})$.

We applied the hybrid variant with $r_0 = 20$ using $261 \approx 2^{8.02790}$ relations. These are all relations in $\mathcal{J}$ mod B_{20} whose support have a non-uniform probability distribution (at level 20). The correct initial state was recovered after executing the tree search on $259\,039 \approx 2^{17.98280}$ candidate solutions. Figure 4.4 shows the number of survivors with the hybrid variant. As in the first experiment, the complexity of the FFT is negligible compared to the tree search part. The maximum number of survivors is $2^{26.95707}$ at B_{32}.

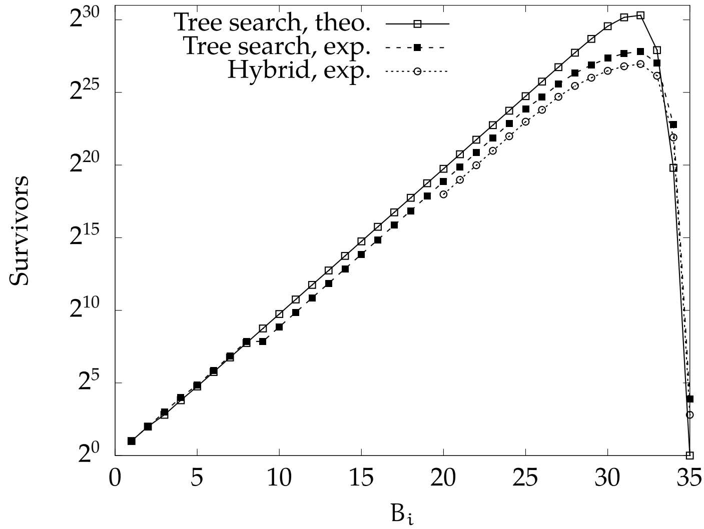


Figure 4.4. Number of survivors for experiment 2.

Since $|\mathcal{I}_{30}| = 300 \approx 2^{8.22881}$, the complexity is $O(2^{35.18588})$. Hence, the hybrid variant also performs better in this application compared to the simple tree search.

Experiment 3

We used $N = 10\,000$ keystream bits generated by the following device:

- $g(x) = x^{64}+x^{62}+x^{55}+x^{49}+x^{44}+x^{42}+x^{37}+x^{24}+x^{23}+x^{20}+x^{16}+x^{15}+x^{10}+x^{8}+x^{6}+x^{2}+1$,
- $f(x_1,\ldots,x_5) = x_2+x_5+x_1x_4+x_3x_4+x_4x_5+x_1x_2x_3+x_1x_3x_4+x_1x_3x_5+x_2x_3x_5+x_3x_4x_5$,
- $(k_1,\ldots,k_5) = (0, 22, 43, 61, 63)$.

The polynomial g was chosen at random with high weight. The function f is the one used in Grain-v1. Notice that the input spacings to f are coprime and span the whole register.

For this experiment, we used $x_4 + x_5$ as the linear approximation to f. We used brute force with B_{32} for finding relations of weight $d = 3$. The expected number of relations is $2^{20.27774}$ and we obtained $1\,172\,961 \approx 2^{20.16172}$. Next, we heuristically created $\mathcal{I}$ by selecting 100 000 relations and computed their probability distributions (4.6) with B_{64}. After sorting the relations, we heuristically chose to create $\mathcal{I}_r$ such that $|\mathcal{I}_r| = 100$ for $r = 1,\ldots,20$, $|\mathcal{I}_r| = 250$ for $r = 21,\ldots,30$, $|\mathcal{I}_r| = 400$ for $r = 31,\ldots,50$ and $|\mathcal{I}_r| = 500$ for $r = 51,\ldots,64$. We then computed the covariance matrix and mean vector for the multivariate distributions to get the vector c of thresholds with $\beta = 0.9$.

We first estimated the theoretical complexity of the tree search and, given the number of expected survivors, we decided to execute it up to level 36 only. Figure 4.5 shows the number of theoretical and partial experimental survivors from the tree search. The maximum of theoretical survivors is $2^{50.66962}$ at B_{52}. Since $|\mathcal{I}_{52}| = 500 \approx 2^{8.96578}$, the theoretical complexity is $O(2^{59.63540})$. At level 36, we got $2^{35.75151}$ survivors experimentally and $2^{35.72974}$ survivors theoretically. Let $\delta = 35.75151 - 35.72974 = 0.02177$. Since the number of experimental survivors follows very closely the theoretical curve, we expect the experimental complexity to be about $O(2^{59.63540+\delta}) = O(2^{59.65717})$.

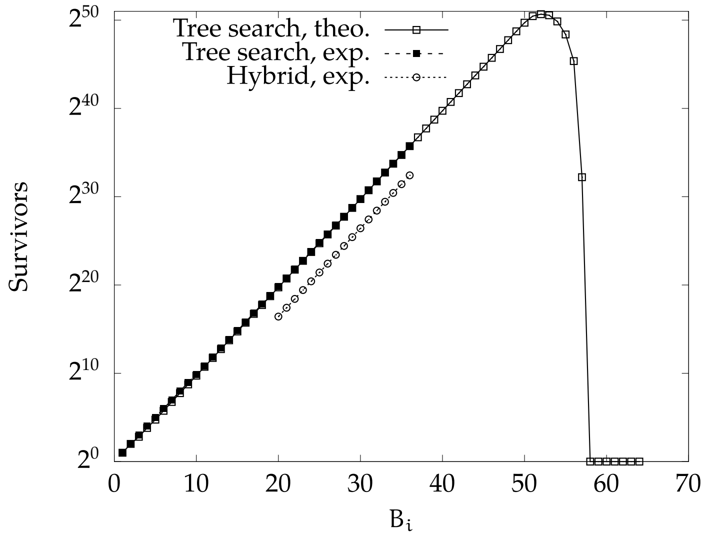


Figure 4.5. Number of survivors for experiment 3.

We applied the hybrid variant with $r_0 = 20$ using $1\,115 \approx 2^{10.12282}$ relations. These are all relations in $\mathcal{J} \bmod B_{20}$ whose support have a non-uniform probability distribution (at level 20). Due to the potential high number of survivors, we executed the tree search part up to level 36 as well. For this partial experiment, however, we knew in advance the candidate at level 20 corresponding to the correct initial state. Otherwise, we would not have been able to know where to stop the tree search and it would be equivalent to brute force all candidates at level 20. Figure 4.5 shows the number of survivors with the hybrid variant. We got $2^{32.42058}$ survivors at level 36. Hence, the hybrid variant also performs better than the simple tree search. As in the previous experiments, the complexity of the FFT is negligible compared to the tree search part. Let us assume that the number of survivors for the hybrid method follows the behaviour of the simple tree search, as in the previous experiments. In the worst case, the tree search part of the hybrid variant will not reject any candidates up to level 52, i.e., $2^{48.42058}$ survivors. Since $|\mathcal{J}_{52}| = 500 \approx 2^{8.96578}$, the worst case complexity is about $O(2^{57.38636})$.

Experiment 4

We used $N = 10\,000$ keystream bits generated by the following device:

- $g(x) = x^{80} + x^{62} + x^{51} + x^{38} + x^{23} + x^{13} + 1$,
- $f(x_1, \dots, x_5) = x_2 + x_5 + x_1x_4 + x_3x_4 + x_4x_5 + x_1x_2x_3 + x_1x_3x_4 + x_1x_3x_5 + x_2x_3x_5 + x_3x_4x_5$,
- $(k_1, \dots, k_5) = (3, 25, 46, 64, 79)$.

The polynomial g, the function f and the indices k_i are taken from the definition of Grain-v1. In that cipher, the last input to f comes from the NFSR; here we wired that input to the last cell of the LFSR ($k_5 = 79$) to maximise the memory.

For this experiment, we used $x_4 + x_5$ as the linear approximation to f. We used brute force with B_{40} for finding relations of weight $d = 3$. The expected number of relations is $2^{12.27774}$ and we obtained $49\,657 \approx 2^{15.59971}$. Next, we heuristically created $\mathcal{I}$ by selecting all the 49 657 relations and computed their probability distributions (4.6) with B_{80}. After sorting the relations, we heuristically chose to create $\mathcal{I}_r$ such that $|\mathcal{I}_r| = 50$ for $r = 1, \ldots, 20$, $|\mathcal{I}_r| = 200$ for $r = 21, \ldots, 45$, $|\mathcal{I}_r| = 400$ for $r = 46, \ldots, 60$ and $|\mathcal{I}_r| = 500$ for $r = 61, \ldots, 80$. We then computed the covariance matrix and mean vector for the multivariate distributions to get the vector c of thresholds with $\beta = 0.9$.

We estimated the theoretical complexity of the tree search first and, given the number of expected survivors, we decided to execute it up to level 36 only. Figure 4.6 shows the number of theoretical and partial experimental survivors from the tree search. The maximum of theoretical survivors is $2^{63.89885}$ at B_{65}. Since $|\mathcal{I}_{65}| = 500 \approx 2^{8.96578}$, the theoretical complexity is $O(2^{72.86463})$. At level 36, we got $2^{34.72253}$ survivors experimentally and $2^{35.72505}$ survivors theoretically. Let $\delta = 34.72253 - 35.72505 = -1.00252$. Since the number of experimental survivors follows very closely the theoretical curve, we can expect the experimental complexity to be about $O(2^{72.86463+\delta}) = O(2^{71.86211})$.

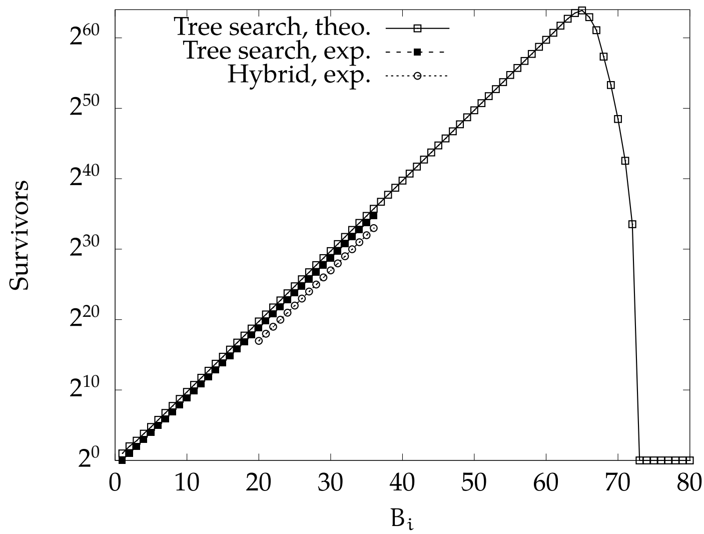


Figure 4.6. Number of survivors for experiment 4.

We applied the hybrid variant with $r_0 = 20$ using $9\,845 \approx 2^{13.26517}$ relations. These are all relations in $\mathcal{I}$ mod B_{20} whose support have a non-uniform probability distribution (at level 20). Due to the potential high number of survivors, we executed the tree search up to level 36 as well. As in the third experiment above, we knew in advance the candidate at level 20 corresponding to the correct initial state. Figure 4.6 shows the number of survivors with the hybrid variant. We got $2^{32.98229}$ survivors at level 36. Hence, the hybrid variant also performs better than the simple tree search. As in the previous experiments, the complexity of the FFT is negligible compared to the tree search part. Let us assume that the number of survivors for the hybrid method follows the behaviour of the simple tree search, as in the previous experiments. In the worst case, the tree search part of the hybrid method will not reject any candidates up to level 65, i.e., $2^{61.98229}$ survivors. Since $|\mathcal{J}_{65}| = 500 \approx 2^{8.96578}$, the worst case complexity is about $O(2^{70.94807})$.

Analysis of experimental results

The relations (4.5) may be seen as a generalisation of the parity-checks used in FCAs. Some of these attacks perform a partial brute force on a subset Ω of the LFSR's initial state, e.g. [CJM02]. We call the parity-checks used in [CJM02] parity-checks modulo Ω. According to [CJM02], the expected number of weight-d parity-checks modulo Ω given the length-N keystream is about $2^{|\Omega|-n}\binom{N}{d-1}$. With the same weight, the set of relations modulo B_r, for an appropriate matrix B_r, incorporates parity-checks modulo Ω, where $|\Omega| = r$. However, for the same number of keystream bits, the expected number (4.11) of relations modulo B_r is significantly higher. Table 4.3 compares these numbers for some explicit parameters.

n	d	N	$r = \|\Omega\|$	# parity-checks mod Ω
40	3	$5 \cdot 10^3$	0	0
			15	0
			25	$2^{8.58}$
40	3	$8 \cdot 10^4$	0	0
			15	$2^{6.58}$
			25	$2^{16.58}$
89	3	2^{28}	32	0

(**a**) Expected number of parity-checks mod Ω of weight d.

n	d	N	ℓ	r	# relations mod B_r
40	3	$5 \cdot 10^3$	5	0	$2^{9.28}$
			5	15	$2^{24.28}$
			5	25	$2^{34.28}$
40	3	$5 \cdot 10^3$	7	0	$2^{15.28}$
			7	15	$2^{30.28}$
			7	25	$2^{40.28}$
40	3	$8 \cdot 10^4$	5	15	$2^{36.29}$
			7	15	$2^{42.29}$
89	3	2^{28}	5	32	$2^{13.42}$
			7	32	$2^{45.42}$

(**b**) Expected number of relations mod B_r of weight d.

Table 4.3. Comparison of the expected number of parity-checks and relations.

That explains why our method requires less keystream bits to recover the LFSR initial state compared to fast correlation attacks while still faster than brute force. In the first two experiments we successfully applied our method with $N = 5 \cdot 10^3$ keystream bits, while the attack in [Lev+03] requires $1.7 \cdot 10^4$ bits; see row 9 in Table 3.2. This is the best comparison for those experiments since we have that $1 - p = 0.375$ for both filtering functions. The third experiment can be compared to the attacks in [CJS01] and [JJ00]. Even though the parameters n and $1 - p$ are not the same, we can notice that our method requires less keystream bits when compared to the entries with $n = 60$ in rows 3 and 5 of Table 3.2. The closest comparison for the fourth experiment may be the result with $n = 70$ from [CJS01]; see row 3 in Table 3.2. In our experiment, the length of the LFSR is larger and our method requires less keystream bits.

The complexity of the deterministic attack by Golić et al. [GCD00] is $O\left(q_{n-r}^{-1} 2^r\right)$, where q_{n-r} is a correction factor. This factor is practically equal to 1 in our experiments and the complexity is $O\left(2^r\right)$. The results in [GCD00] were obtained using a degree-100 weight-5 feedback polynomial and filtering functions with 5 and 10 variables. The authors used input spacings not spanning the whole register and with rather low memory. The hardest instance corresponds to an input spacing yielding memory 15. The attack in [Lev+01] has complexity $O\left(2^r\right)$ as well. The experimental results correspond to devices using the same feedback polynomial as Golić et al. but different filtering functions. The memory of the devices is rather low as well; the hardest instance reported has memory 9. The devices in our first three experiments would render these attacks equivalent to brute force since their memory is equal to the length of the LFSR. For the fourth experiment, the complexity of these attacks would be 2^{77} while our simple tree search and hybrid variants have complexity $2^{71.86211}$ and $2^{70.94807}$, respectively.

4.8 Application to Grain ciphers

Ciphers in the Grain family [Hel+08] are designed to be small and easy to implement in hardware. The family comprises the Grain-v1 [HJM07; Hel+08] and Grain-128a [Ågr+11] ciphers. They are bit oriented synchronous stream ciphers. Their main components are an LFSR, an NFSR and an output function. The output function consists of a nonlinear Boolean function h and linear terms added to h. A general overview is given in Figure 4.7. Before generating the keystream, the cipher is initialised with the key and the initialisation vector (IV), and clocked in a special configuration without producing any keystream. Grain-128a is a new version of Grain-128 [Hel+06; Hel+08] that is resistant against the initial attacks on the latter. Grain-128a also adds support for optional authenticated encryption. Grain-v1 is part of the eSTREAM portfolio [Eur] and Grain-128a is standardised by ISO/IEC [Tec15]. Grain-v1 supports an 80-bit key and a 64-bit IV. Grain-128a and Grain-128 support a 128-bit key and a 96-bit IV. With the introduction of Grain-128a, the use of Grain-128 is discouraged.

In [Tod+18], Todo et al. present a new key recovery attack (see Section 3.2.3) and apply it to Grain-v1, Grain-128 and Grain-128a (in stream cipher mode only). This attack is more efficient than previous attacks against Grain-v1 (e.g., [Zha+14; ZXM18]). Against Grain-128, this is the first attack targeting the keystream generator. Previous attacks [DS11; Din+11] targeted the initialisation of the cipher. For full Grain-128a (i.e., no reduced number of clocks in the initialisation), this is the first cryptanalysis reported.

In this section we show how to construct a system of equations (4.2) for the toy Grain-like cipher described in [Tod+18] and Grain-v1. For both ciphers, we also find

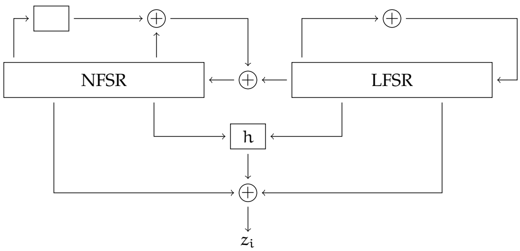


Figure 4.7. Overview of the components in the Grain family of ciphers.

linear combinations of LFSR bits with higher correlation than that indicated by Todo et al. The FFT is used to find such linear combinations. First, we present a method to compute the FFT with long input vectors. Then, we report our results on the ciphers. Particularly, for Grain-v1, the FFT is applied to a vector of length 2^{37} and we successfully recover the LFSR's initial state using the multivariate correlation attack (see Section 4.2) requiring $N = 2^{53.5}$ keystream bits. That is significantly lower than the required $2^{75.11}$ bits in [Tod+18]. However, the time complexity of our attack is higher. The test-and-extend algorithm was not applied in this case.

4.8.1 Computing the Fourier transform

Let $f : \mathbb{F}_2^n \to \mathbb{Z}$, where $n = n_1 + n_2$ with $n_1 > 0$ and $n_2 > 0$. Then, by the definition of the Fourier-Hadamard transform

$$\begin{aligned}\widehat{f}(u) &= \sum_{x \in \mathbb{F}_2^n} f(x)(-1)^{u \cdot x} \\ &= \sum_{x_1 \in \mathbb{F}_2^{n_1}} \sum_{x_2 \in \mathbb{F}_2^{n_2}} (-1)^{(u_1 u_2)\cdot(x_1 x_2)} f(x_1 x_2) \\ &= \sum_{x_1 \in \mathbb{F}_2^{n_1}} \sum_{x_2 \in \mathbb{F}_2^{n_2}} (-1)^{u_1 \cdot x_1} (-1)^{u_2 \cdot x_2} f(x_1 x_2)\end{aligned}$$

where $u_1 \in \mathbb{F}_2^{n_1}$, $u_2 \in \mathbb{F}_2^{n_2}$ and $u_1 u_2$, $x_1 x_2$ denote concatenation of vectors. Let

$$g_{u_1}(x_2) = \sum_{x_1 \in \mathbb{F}_2^{n_1}} (-1)^{u_1 \cdot x_1} f(x_1 x_2),$$

then

$$\begin{aligned}\widehat{f}(u) &= \sum_{x_2 \in \mathbb{F}_2^{n_2}} (-1)^{u_2 \cdot x_2} \sum_{x_1 \in \mathbb{F}_2^{n_1}} (-1)^{u_1 \cdot x_1} f(x_1 x_2) \\ &= \sum_{x_2 \in \mathbb{F}_2^{n_2}} (-1)^{u_2 \cdot x_2} g_{u_1}(x_2) = \widehat{g_{u_1}}(u_2).\end{aligned} \tag{4.17}$$

That is, we obtain the Fourier-Hadamard spectrum of f by computing the Fourier-Hadamard spectrum of g_{u_1} for all 2^{n_1} possible values of u_1. For a fixed value of u_1, we obtain the Fourier-Hadamard spectrum of g_{u_1} by evaluating g_{u_1} in all possible values for x_2 (2^n operations) and then applying the FFT on a vector of length 2^{n_2} ($n_2 2^{n_2}$

operations). Hence, the complexity for $\widehat{f}$ is $2^{n_1}(2^n + n_2 2^{n_2}) = 2^n(2^{n_1} + n_2)$ operations. Similarly, let $g_{u_2}(x_1) = \sum_{x_2 \in \mathbb{F}_2^{n_2}} (-1)^{u_2 \cdot x_2} f(x_1 x_2)$, then

$$\begin{aligned} \widehat{f}(u) &= \sum_{x_1 \in \mathbb{F}_2^{n_1}} (-1)^{u_1 \cdot x_1} \sum_{x_2 \in \mathbb{F}_2^{n_2}} (-1)^{u_2 \cdot x_2} f(x_1 x_2) \\ &= \sum_{x_1 \in \mathbb{F}_2^{n_1}} (-1)^{u_1 \cdot x_1} g_{u_2}(x_1) = \widehat{g_{u_2}}(u_1) \end{aligned} \tag{4.18}$$

and a similar analysis as above shows that the complexity is $2^n(2^{n_2} + n_1)$ operations.

The time complexity using equations (4.17) or (4.18) is higher compared to using directly the FFT ($n2^n$ operations). However, the variant above may be of interest when space (i.e., memory/storage) is constrained. The FFT operates on a vector with 2^n elements. In some cases we might not have access to that amount of space. For example, to compute the Fourier-Hadamard spectrum on 2^{34} elements, each one requiring $2^6 = 64$ bits, the total requirement is 2^{40} bits (1 TiB). If we are interested in certain points u (e.g., where $\widehat{f}(u) \neq 0$ or $|\widehat{f}(u)| \geqslant t$ for a threshold t), we can choose n_1 and n_2 such that the computations of $\widehat{g_{u_1}}$ (or $\widehat{g_{u_2}}$) can be done with the available space and discard the irrelevant data. We can also parallelise the computation. It is easy to distribute the computation of all possible values of u_1 (or u_2) among all available processors. This is additional to the parallelisation in the implementation of the FFT used to compute $\widehat{g_{u_1}}$ (or $\widehat{g_{u_2}}$).

4.8.2 Grain toy cipher

The cipher consists of an LFSR and an NFSR of size 24 bits. Let $(s_t, \dots, s_{t+23})$ and $(b_t, \dots, b_{t+23})$ represent the state of the LFSR and NFSR, respectively, at time t. The LFSR and NFSR feedbacks are given by

$$\begin{aligned} s_{t+24} &= s_t + s_{t+1} + s_{t+2} + s_{t+7}, \\ b_{t+24} &= s_t + b_t + b_{t+5} + b_{t+14} + b_{t+20}b_{t+21} + b_{t+11}b_{t+13}b_{t+15}, \end{aligned}$$

respectively. The keystream bit is

$$z_t = h(s_{t+3}, s_{t+7}, s_{t+15}, s_{t+19}, b_{t+17}) + \sum_{j \in \mathcal{A}} b_{t+j}, \tag{4.19}$$

where $\mathcal{A} = \{1, 3, 8\}$ and

$$\begin{aligned} h(s_{t+3}, s_{t+7}, s_{t+15}, s_{t+19}, b_{t+17}) = {} & s_{t+7} + b_{t+17} + s_{t+3}s_{t+19} + s_{t+15}s_{t+19} + s_{t+19}b_{t+17} + \\ & s_{t+3}s_{t+7}s_{t+15} + s_{t+3}s_{t+15}s_{t+19} + s_{t+3}s_{t+15}b_{t+17} + \\ & s_{t+7}s_{t+15}b_{t+17} + s_{t+15}s_{t+19}b_{t+17}. \end{aligned}$$

Assume the N-bit keystream $z_1, \dots, z_N$ is given. Let $X = (s_1, s_2, \dots, s_{24})^T$ be the unknown LFSR initial state and

$$\begin{aligned} A_t X = (& s_{t+3}, s_{t+7}, s_{t+15}, s_{t+19}, s_{t+8}, s_{t+12}, s_{t+20}, s_{t+24}, s_{t+17}, s_{t+21}, \\ & s_{t+29}, s_{t+33}, s_{t+27}, s_{t+31}, s_{t+39}, s_{t+43}, s_{t+1} + s_{t+3} + s_{t+8})^T, \end{aligned}$$

where A_t is a 17×24-matrix. That is, A_tX incorporates 17 linear functions in X. We will construct a multivariate probability distribution on A_tX for a random variable X_t and get a system of equations $A_tX = X_t$, $t = 1, \ldots, N$ as in Section 4.1.

Let $\mathcal{T} = \{0, 5, 14, 24\}$ as in [Tod+18]. We may have two distributions on A_tX depending on the bit $Z_t = \sum_{i \in \mathcal{T}} z_{t+i}$. From the definition of the NFSR feedback

$$\sum_{i \in \mathcal{T}, j \in \mathcal{A}} b_{t+j+i} = \sum_{j \in \mathcal{A}} s_{t+j} + \sum_{j \in \mathcal{A}} (b_{t+20+j} b_{t+21+j} + b_{t+11+j} b_{t+13+j} b_{t+15+j}).$$

So, (4.19) implies

$$\begin{aligned} Z_t + \sum_{i \in \mathcal{T}} h(s_{t+3+i}, s_{t+7+i}, s_{t+15+i}, s_{t+19+i}, b_{t+17+i}) + \sum_{j \in \mathcal{A}} s_{t+j} = \\ \sum_{j \in \mathcal{A}} (b_{t+20+j} b_{t+21+j} + b_{t+11+j} b_{t+13+j} b_{t+15+j}) \end{aligned} \tag{4.20}$$

and

$$s_{t+17} + \sum_{i \in \mathcal{T}} b_{t+17+i} = b_{t+37} b_{t+38} + b_{t+28} b_{t+30} b_{t+32}. \tag{4.21}$$

The distribution of X_t on A_tX is then computed as a uniform distribution conditioned by the relations (4.20) and (4.21). The distribution is non-uniform. To be specific, let $a = (a_1, \ldots, a_{17})$ be a 17-bit vector, we want to compute $\Pr(A_tX = a)$. By $A_tX = a$, (4.20) and (4.21) the following 22 bits of

$$u = (Z_t, a_1, \ldots, a_{17}, b_{t+17}, b_{t+22}, b_{t+31}, b_{t+41}) \tag{4.22}$$

uniquely define 3 bits of

$$v = (b_{t+22}, \sum_{j \in \mathcal{A}} (b_{t+20+j} b_{t+21+j} + b_{t+11+j} b_{t+13+j} b_{t+15+j}), b_{t+37} b_{t+38} + b_{t+28} b_{t+30} b_{t+32}). \tag{4.23}$$

So, $\phi(u) = v$ for a 22-bit to 3-bit mapping ϕ. Each v has the same number 2^{19} of pre-images u under ϕ. The distribution p_v on (4.23) is pre-computed by running over 15 variables involved in the right hand side. This induces a distribution $2^{-19} p_{\phi(u)}$ on (4.22). Under the condition that Z_t is fixed by $\varepsilon = 0$ or 1 we have

$$\Pr(X_t = a \mid Z_t = \varepsilon) = 2^{-18} \sum_{b_{t+17}, b_{t+22}, b_{t+31}, b_{t+41}, Z_t = \varepsilon} p_{\phi(u)},$$

where the sum is over all values of $b_{t+17}, b_{t+22}, b_{t+31}, b_{t+41}$ and $Z_t = \varepsilon$. Therefore $A_tX = X_t$, $t = 1, \ldots, N$.

We apply the FFT-based method in Section 4.2.2 to recover X. By Section 4.2, we find the parameters of the limit distributions as

$$\mu_{0,1} = -11.782815, \quad \sigma^2_{0,1} = 0.00137196$$

and

$$\mu_{1,1} = -11.784191, \quad \sigma^2_{1,1} = 0.00138229.$$

By formulae in Section 4.2.1, for $c = -358\,013.3911$ and $N = 30\,382 \approx 2^{14.89}$, the number of incorrect survivors is < 1 on the average and the success probability $\beta = 0.9999$.

The condition (4.3) is fulfilled. The FFT is used to compute the values of the statistic in (4.4), thus recovering X. The complexity of the attack is proportional to $2^{17}N+24\cdot 2^{24} \approx 2^{31.89}$ operations; this was verified experimentally. The internal state of the cipher is 48 bits long. According to [Tod+18], with $N = 2^{23.25}$ the whole state may be recovered with time complexity (number of operations) and space complexity of order N.

Let p_0 and p_1 be a probability distribution, then $\delta = p_0 - p_1$ is its correlation. With the Fourier-Hadamard transform we find all linear combinations of the entries of A_tX with non-zero correlations. Table 4.4 shows the absolute value of the non-zero correlations δ and the number of linear combinations N_δ with the same δ. The data does not depend on Z_t. In [Tod+18], it is stated that there are 1024 linear combinations with highest absolute value of the correlation $2^{-10.41503}$. However, that is not correct according to Table 4.4. There are linear combinations with even a higher correlation. The reason for the discrepancy is the relation (4.21) which was ignored in [Tod+18].

δ	N_δ
$\frac{9437184}{2^{33}} = 2^{-9.83007...}$	128
$\frac{6291456}{2^{33}} = 2^{-10.41503...}$	768
$\frac{4718592}{2^{33}} = 2^{-10.83007...}$	512
$\frac{3145728}{2^{33}} = 2^{-11.41503...}$	3968
$\frac{1572864}{2^{33}} = 2^{-12.41503...}$	3584

Table 4.4. Correlations for toy Grain-like cipher.

For instance, the absolute value of the correlation of

$$s_{t+7} + s_{t+19} + s_{t+12} + s_{t+24} + s_{t+17} + s_{t+21} + s_{t+31} + s_{t+43} + s_{t+1} + s_{t+3} + s_{t+8} \tag{4.24}$$

is $2^{-9.83007}$. We verified experimentally the correlation value as follows. Let s denote (4.24) with $t = 0$. We randomly chose 2^{30} different initial states for the cipher (i.e, LFSR and NFSR initial states). For each initial state, we computed $Z_0 = z_0 + z_5 + z_{14} + z_{24}$, and when $Z_0 = 0$, we computed s and kept track of the number of times $s = 0$. We got that $Z_0 = 0$ occurred $536\,879\,412 \approx 2^{29.000022}$ times and among those, $s = 0$ occurred $268\,737\,466 = 2^{28.001622}$ times. With this, $p_0 = 2^{28.001622}/2^{29.000022}$ and we obtained $\delta = 2^{-9.816232}$ as experimental correlation.

Using the FFT for computing the statistic

We employ $N = 30\,382$ keystream bits. All probability distributions on A_tX, $t = 1, \ldots, N$, are permutations of the same distribution P. The latter is a distribution on $\mathbb{F}_2^{17}$ and its support is the whole vector space. The different probability values of P and their frequencies are in Table 4.5. We have that $2^{-17} \approx 7.6294\cdot 10^{-6}$. According to Table 4.5, we may consider P to be close to uniform.

With the values $\mu_{0,1}$, $\sigma_{0,1}^2$ and $\mu_{1,1}$, $\sigma_{1,1}^2$ above, the parameters of the distribution of $S(x)$ under the hypotheses H0 and H1 are

$$\mu_0 = N\mu_{0,1} = -357\,985.47, \quad \sigma_0^2 = N\sigma_{0,1}^2 = 41.68$$

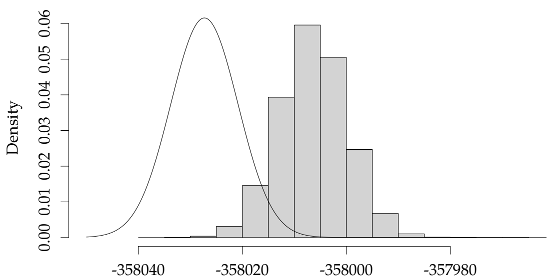
Prob. value	$29/4194304 \approx 6.9141 \cdot 10^{-6}$	$61/8388608 \approx 7.2718 \cdot 10^{-6}$	$1/131072 \approx 7.6294 \cdot 10^{-6}$	$67/8388608 \approx 7.9870 \cdot 10^{-6}$	$35/4194304 \approx 8.3447 \cdot 10^{-6}$
Frequency	8192	8192	98304	8192	8192

Table 4.5. Probability values and frequencies of the distributions for the toy Grain-like cipher.

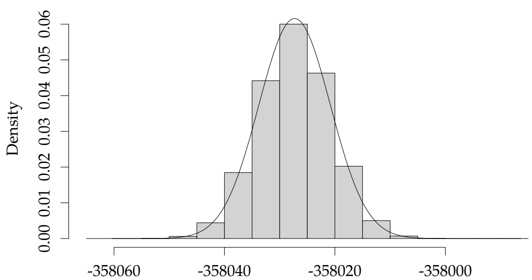
and

$$\mu_1 = N\mu_{1,1} = -358\,027.29, \quad \sigma_1^2 = N\sigma_{1,1}^2 = 41.99,$$

respectively (see Section 4.2.1). In our cryptanalysis above, we computed, for all $x \in \mathbb{F}_2^{24}$, an approximate to $S(x)$ with the FFT (as in Section 4.2.2), denoted by $\bar{S}(x)$. This computation corresponds to hypothesis H1, because we use the same keystream and "choose" different initial states of the LFSR, all of them incorrect except one. Figure 4.8a shows the probability density of $\mathbf{N}(\mu_1, \sigma_1^2)$ and the histogram with the density of the computed values of $\bar{S}(x)$. Figure 4.8b shows the corresponding histogram for the exact values of $S(x)$ (i.e., using equation (4.4)) along with the probability density of $\mathbf{N}(\mu_1, \sigma_1^2)$.



(**a**) Probability density of $\bar{S}(x)$.



(**b**) Probability density of $S(x)$.

Figure 4.8. Probability density of the values of the statistics $S(x)$ and $\bar{S}(x)$ under hypothesis H1.

The distribution of $S(x)$ practically follows $\mathbf{N}(\mu_1, \sigma_1^2)$. The distribution of the estimate $\bar{S}(x)$, however, is not that close to $\mathbf{N}(\mu_1, \sigma_1^2)$. This is due to the error introduced by using $\ln(1+\epsilon) \approx \epsilon$ for small ϵ in the computation of $\bar{S}(x)$ (see Section 4.2.2). Since we use $S(x)$ to rank the candidates, we analyse whether this difference affects the outcome for our application. Let x_0 be the correct initial state of the LFSR. When computing the exact values of $S(x)$, we get

$$S(x_0) = -357\,986.31 \quad \text{and} \quad |\{x \in \mathbb{F}_2^{24} : S(x) \geqslant S(x_0)\}| = 1.$$

That is, the correct initial state is ranked first; this agrees with the number of incorrect survivors being < 1 in the cryptanalysis above. When using the FFT, we get

$$\bar{S}(x_0) = -357\,964.92 \quad \text{and} \quad |\{x \in \mathbb{F}_2^{24} : \bar{S}(x) \geqslant \bar{S}(x_0)\}| = 1.$$

So, even though using the FFT does not produce the exact values of $S(x)$, we get the same "quality" for the values $\bar{S}(x)$, at least for this application. We attribute this to the probability distribution P actually being close to uniform.

4.8.3 Grain-v1

We apply a similar method to Grain-v1. The cipher consists of an LFSR and an NFSR of size 80 bits. Let $(s_t, \dots, s_{t+79})$ and $(b_t, \dots, b_{t+79})$ represent the state of the LFSR and NFSR, respectively, at time t. The LFSR and NFSR feedbacks are given by

$$\begin{aligned}
s_{t+80} &= s_t + s_{t+13} + s_{t+23} + s_{t+38} + s_{t+51} + s_{t+62},\\
b_{t+80} &= s_t + b_{t+62} + b_{t+60} + b_{t+52} + b_{t+45} + b_{t+37} + b_{t+33} + b_{t+28} + b_{t+21} + b_{t+14} +\\
&\quad b_{t+9} + b_t + b_{t+63}b_{t+60} + b_{t+37}b_{t+33} + b_{t+15}b_{t+9} + b_{t+60}b_{t+52}b_{t+45} +\\
&\quad b_{t+33}b_{t+28}b_{t+21} + b_{t+63}b_{t+45}b_{t+28}b_{t+9} + b_{t+60}b_{t+52}b_{t+37}b_{t+33} +\\
&\quad b_{t+63}b_{t+60}b_{t+21}b_{t+15} + b_{t+63}b_{t+60}b_{t+52}b_{t+45}b_{t+37} +\\
&\quad b_{t+33}b_{t+28}b_{t+21}b_{t+15}b_{t+9} + b_{t+52}b_{t+45}b_{t+37}b_{t+33}b_{t+28}b_{t+21},
\end{aligned}$$

respectively. The keystream bit is

$$z_t = h(s_{t+3}, s_{t+25}, s_{t+46}, s_{t+64}, b_{t+63}) + \sum_{j \in \mathcal{A}} b_{t+j}, \tag{4.25}$$

where $\mathcal{A} = \{1, 2, 4, 10, 31, 43, 56\}$ and

$$\begin{aligned}
h(s_{t+3}, s_{t+25}, s_{t+46}, s_{t+64}, b_{t+63}) &= s_{t+25} + b_{t+63} + s_{t+3}s_{t+64} + s_{t+46}s_{t+64} + s_{t+64}b_{t+63} +\\
&\quad s_{t+3}s_{t+25}s_{t+46} + s_{t+3}s_{t+46}s_{t+64} + s_{t+3}s_{t+46}b_{t+63} +\\
&\quad s_{t+25}s_{t+46}b_{t+63} + s_{t+46}s_{t+64}b_{t+63}.
\end{aligned}$$

Let $X = (s_1, s_2, \dots, s_{80})^T$ be the unknown LFSR initial state and

$$\begin{aligned}
A_tX = (&s_{t+3}, s_{t+25}, s_{t+46}, s_{t+64}, s_{t+17}, s_{t+39}, s_{t+60}, s_{t+78}, s_{t+24}, s_{t+67}, s_{t+85}, s_{t+31},\\
&s_{t+53}, s_{t+74}, s_{t+92}, s_{t+40}, s_{t+62}, s_{t+83}, s_{t+101}, s_{t+48}, s_{t+70}, s_{t+91}, s_{t+109}, s_{t+55},\\
&s_{t+77}, s_{t+98}, s_{t+116}, s_{t+63}, s_{t+106}, s_{t+124}, s_{t+65}, s_{t+87}, s_{t+108}, s_{t+126}, s_{t+105}, s_{t+144},\\
&s_{t+1} + s_{t+2} + s_{t+4} + s_{t+10} + s_{t+31} + s_{t+43} + s_{t+56}),
\end{aligned}$$

where A_t is a 37×80-matrix. That is, A_tX incorporates 37 linear functions in X. We will construct a multivariate probability distribution on A_tX for a random variable X_t and get a system of equations $A_tX = X_t$, $t = 1, \dots, N$, as in Section 4.1.

Following [Tod+18], let $\mathcal{T} = \{0, 14, 21, 28, 37, 45, 52, 60, 62, 80\}$. We may have two distributions on A_tX depending on $Z_t = \sum_{i\in\mathcal{T}} z_{t+i}$. From the definition of the NFSR

$$\sum_{i\in\mathcal{T}, j\in\mathcal{A}} b_{t+i+j} = \sum_{j\in\mathcal{A}} s_{t+j} + \sum_{j\in\mathcal{A}} g'(b^{(t+j)}), \tag{4.26}$$

where

$$\begin{aligned} g'(b^{(t)}) = {} & b_{t+33} + b_{t+9} + b_{t+63}b_{t+60} + b_{t+37}b_{t+33} + b_{t+15}b_{t+9} + b_{t+60}b_{t+52}b_{t+45} + \\ & b_{t+33}b_{t+28}b_{t+21} + b_{t+63}b_{t+45}b_{t+28}b_{t+9} + b_{t+60}b_{t+52}b_{t+37}b_{t+33} + \\ & b_{t+63}b_{t+60}b_{t+21}b_{t+15} + b_{t+63}b_{t+60}b_{t+52}b_{t+45}b_{t+37} + \\ & b_{t+33}b_{t+28}b_{t+21}b_{t+15}b_{t+9} + b_{t+52}b_{t+45}b_{t+37}b_{t+33}b_{t+28}b_{t+21}. \end{aligned}$$

So (4.25) and (4.26) imply

$$Z_t + \sum_{i\in\mathcal{T}} h(s_{t+3+i}, s_{t+25+i}, s_{t+46+i}, s_{t+64+i}, b_{t+63+i}) + \sum_{j\in\mathcal{A}} s_{t+j} = \sum_{j\in\mathcal{A}} g'(b^{(t+j)}) \tag{4.27}$$

and

$$s_{t+63} + \sum_{i\in\mathcal{T}} b_{t+63+i} = g'(b^{(t+63)}). \tag{4.28}$$

Let $a = (a_1, \dots, a_{37})$ be a 37-bit vector, we want to compute $\Pr(A_tX = a)$. By (4.27) and (4.28), the following 48 bits of

$$\begin{aligned} u = (& Z_t, a_1, \dots, a_{37}, b_{t+63}, b_{t+77}, b_{t+84}, b_{t+91}, b_{t+100}, b_{t+108}, b_{t+115}, b_{t+123}, \\ & b_{t+125}, b_{t+143}) \end{aligned} \tag{4.29}$$

uniquely define 9 bits of

$$v = (b_{t+77}, b_{t+84}, b_{t+91}, b_{t+100}, b_{t+108}, b_{t+115}, b_{t+123}, \sum_{j\in\mathcal{A}} g'(b^{(t+j)}), g'(b^{(t+63)})). \tag{4.30}$$

So, $\phi(u) = v$ for a 48-bit to 9-bit mapping ϕ. Each v has 2^{39} pre-images u under ϕ. The distribution p_v on (4.30) incorporating 64 variables is pre-computed. This induces a distribution $2^{-39}p_{\phi(u)}$ on (4.29). The last entry in A_tX above incorporates 6 different variables (s_{t+31} appears in position 12 as well). Hence, under the condition that Z_t is fixed by $\epsilon = 0$ or 1, we have

$$\Pr(X_t = a \mid Z_t = \epsilon) = 2^{-38} \sum_{\substack{b_{t+63}, b_{t+77}, b_{t+84}, b_{t+91} \\ b_{t+100}, b_{t+108}, b_{t+115}, b_{t+123} \\ b_{t+125}, b_{t+143}, Z_t = \epsilon}} p_{\phi(u)}.$$

The distribution p_v was pre-computed as follows. The expression for (4.30) incorporates 64 variables. Some of the variables are fixed by constants, then $\sum_{j\in A} g'(b^{(t+j)})$ and $g'(b^{(t+63)})$ are represented as sums of "independent" polynomials with fewer variables. Independence means that each of the rest variables appears in one polynomial only. The distributions relevant to the independent polynomials are computed separately. Finally, they are combined to get p_v. We computed p_v by fixing

$b_{t+38}, b_{t+46}, b_{t+64}, b_{t+65}, b_{t+71}$ and b_{t+91}. The largest computation corresponded with a polynomial in 23 variables.

By Section 4.2, we find the parameters of the limit distributions as

$$\mu_{0,1} = -25.646445680717974846, \quad \sigma^2_{0,1} = 3.204164923186231 \cdot 10^{-15}$$

and

$$\mu_{1,1} = -25.646445680717978051, \quad \sigma^2_{1,1} = 3.204164923189462 \cdot 10^{-15}.$$

By formulae in Section 4.2.1, for $c = -326687075514236406.749337$ and $N = 2^{53.5}$, the number of incorrect survivors is < 1 on the average and the success probability is $\beta = 0.9991$. The condition (4.3) is fulfilled. The FFT is used to compute the values of the statistic in (4.4), thus recovering X with time complexity proportional to $2^{37}N + 80 \cdot 2^{80} \approx 2^{90.5}$ operations. Then, we can guess some bits of the NFSR's initial state and employ equation (4.25) to recover the whole 160-bit initial state. The key for Grain-v1 is 80-bit long, therefore, the complexity of this attack is higher than brute force. On the other hand, the number of keystream bits required to obtain the LFSR's initial state is low. According to [Tod+18], with $N = 2^{75.11}$ the whole state may be recovered with time complexity and space complexity of order N.

Applying the Fourier-Hadamard transform to $f(v) = p_v$, we find linear combinations of the entries of A_tX with non-zero correlations. A direct application of the FFT would require space for 2^{37} elements. Due to this, we adopt the strategy in Section 4.8.1. For this application, we chose $n_2 = 9$ and we parallelised the computation of the 2^9 possible values for u_2. Each computation of the FFT is therefore applied to a vector of length 2^{28}. Since each element is stored on a 64-bit precision floating-point number, the total memory requirement is around 2^{34} bits. For each value of u_2, we only kept the values of u_1 such that $|\widehat{f}(u)| > 2^{-36}$, where $u = (u_1, u_2)$. In other words, we only kept the linear combinations of A_tX given by u whose correlation's absolute value is greater than 2^{-36}. The authors in [Tod+18] found 442 368 such linear combinations, however, we found 443 264. As in the toy cipher above, we attribute this discrepancy to the omission of (4.28) in [Tod+18]. There are 171 different correlation values among the 443 264 linear combinations we found. Table 4.6 shows some of the highest and lowest values. As an example,

$$\begin{aligned}
&s_{t+3} + s_{t+25} + s_{t+64} + s_{t+39} + s_{t+60} + s_{t+78} + s_{t+24} + s_{t+85} + s_{t+53} + s_{t+92} + s_{t+62} + \\
&s_{t+83} + s_{t+101} + s_{t+70} + s_{t+109} + s_{t+77} + s_{t+116} + s_{t+63} + s_{t+106} + s_{t+124} + s_{t+87} + \\
&s_{t+126} + s_{t+105} + s_{t+144} + s_{t+1} + s_{t+2} + s_{t+4} + s_{t+10} + s_{t+31} + s_{t+43} + s_{t+56}
\end{aligned}$$

has the highest correlation $2^{-35.46890046}$.

Summary and future work

New methods for cryptanalysis of LFSR-based stream ciphers were presented. The cryptanalysis is modelled as a more general problem of finding solutions to systems of linear equations with associated probability distributions on the set of right hand sides. First, we introduced the multivariate correlation attack, which is a generalisation of the correlation attack by Siegenthaler [Sie85]. Then, the test-and-extend algorithm was shown. This novel method has lower time complexity and comprises two stages, pre-computation and main computation. In the pre-computation stage,

δ	N_δ
$2^{-35.46890046...}$	64
$2^{-35.50019546...}$	64
$2^{-35.54760452...}$	128
$2^{-35.55461504...}$	640
$2^{-35.57682560...}$	64

(**a**) Highest correlations $> 2^{-36}$ for Grain-v1.

δ	N_δ
$2^{-35.98275923...}$	128
$2^{-35.98646706...}$	1280
$2^{-35.99121484...}$	256
$2^{-35.99186310...}$	640
$2^{-35.99726377...}$	640

(**b**) Lowest correlations $> 2^{-36}$ for Grain-v1.

Table 4.6. Some values of the correlations for Grain-v1.

we find relations modulo B, which can be seen as a generalisation of parity-checks used in fast correlation attacks, and compute the probability distributions induced by these relations. For the second stage, there are two variants: tree search and hybrid variant. The first one finds the initial state of the LFSR (in general, candidate solutions to the systems of linear equations) by traversing a tree along with a statistical test to decide which branches to discard. The second variant also traverses a tree, however, the tree search is started at a further level on the tree following the ranking given by the statistic associated to the nodes.

We applied the test-and-extend algorithm to a variety of hard instances of the filter generator. In the first two experiments, we successfully recovered the initial state of the LFSR used to generate the given keystream. For the other experiments, our cryptanalytic results are theoretical only since the time complexity is high. In all cases, the hybrid variant outperformed the simple tree search. This new method allows successful recovery of the initial state requiring a lower number of keystream bits compared to other published attacks. We also applied the multivariate correlation method to the toy Grain-like cipher from [Tod+18] and Grain-v1. Compared to the method in [Tod+18], we successfully recovered the LFSR's initial state for the toy cipher using less keystream bits. On the other hand, the time complexity to recover the whole cipher state (LFSR and NFSR states) is higher using our method. The results are similar for Grain-v1. Our method requires $2^{53.5}$ keystream bits to recover the LFSR's initial state, sensibly less than $2^{75.11}$ bits for the attack in [Tod+18]. However, the time complexity to recover the whole state of the cipher is higher. In the case of Grain-v1, our results are theoretical. Additionally, for both ciphers, we found linear combinations of LFSR sequence bits with high correlation. The correlations are higher than those reported in [Tod+18]. The results on that paper could be improved using the correlation we found, but we do not follow that direction here. The correlations for Grain-v1 were obtained by computing the FFT on a large input vector; we used a simple method to parallelise this computation (see Section 4.8.1), which, to our knowledge, has not been reported.

Immediate future directions are to use the test-and-extend algorithm for Grain-v1 as well as to apply our methods to other LFSR-based stream ciphers, e.g., other ciphers in the Grain family. Also, these new methods may be used for cryptanalysis of block ciphers. As for Grain-v1, we need to get the corresponding matrices A_i and construct a multivariate probability distribution on the system of equations A_iX, where X is the initial state of the cipher. Also, it would be interesting to analyse the performance of the new methods for solving random systems of equations, i.e., systems not describing a cipher.

CHAPTER 5

Decoders for p-ary QC-MDPC codes

In this chapter, we present decoders for p-ary quasi-cyclic moderate density parity-check (QC-MDPC) codes with low decoding failure rate. These decoders are the result of joint work with Guo and Johansson [CGJ19]. p-ary MDPC codes [GJ16] are an extension of binary MDPC codes [Mis+13] to fields of characteristic p. Here, we focus on decoding for a particular quasi-cyclic instance of the p-ary MDPC scheme, which is similar to that in Ouroboros-E [Den+18]. Our new decoding algorithm is a bit-flipping decoder and the performance is improved by varying thresholds for the different iterations. The new decoder is a general decoding method for p-ary MDPC-like schemes. Thus, it can be used to improve Ouroboros-E or future p-ary MDPC-based primitives.

In Section 5.1, we present the basics on MDPC schemes. A bit-flipping decoder for an instance of the p-ary QC-MDPC scheme is presented in Section 5.2. We improve this decoder in Section 5.3, where we present the essential idea of varying thresholds and two methods for computing them. Section 5.4 shows the results of the application of the new decoder to some parameters for the chosen p-ary QC-MDPC instance.

5.1 Preliminaries

5.1.1 McEliece cryptosystem and MDPC variants

Code-based cryptography is believed to be resistant against attacks using a quantum computer. McEliece proposed in [McE78] one of the first code-based cryptosystems, which uses Gopppa codes [Ber73; MS81]. The security of the McEliece cryptosystem relies on the hardness of the general decoding problem for linear codes, which is NP-complete [BMT78], and the indistinguishability of the code family. The latter is the weakest problem and depends on the chosen family of codes. Let $n = 2^\ell$ and let $\mathcal{F}$ denote a family of binary irreducible length-n Goppa codes of dimension $k \geqslant n - t\ell$

capable of correcting up to t errors. The McEliece cryptosystem is defined as follows:

- Key generation
  - Select randomly and uniformly a code C from $\mathcal{F}$. Let G_0 be a $k \times n$ generator matrix of C in reduced row echelon form and H be a corresponding parity-check matrix.
  - Select a random dense $k \times k$ non-singular matrix S and a random $n \times n$ permutation matrix P. Compute $G = SG_0P$.
  - Return the public key G and private key H.
- Encryption

  Let $m \in \mathbb{F}_2^k$ be the plaintext.
  - Randomly generate $e \in \mathbb{F}_2^n$ with Hamming weight t.
  - Compute $c = mG + e$.
  - Return the ciphertext c.
- Decryption

  Let $c \in \mathbb{F}_2^n$ be the ciphertext and Ψ be a decoding algorithm for C.
  - Compute $c' = \Psi(H, cP^{-1}) = mS$ and $m = c'S^{-1}$.
  - Return the plaintext m.

The McEliece cryptosystem has efficient encryption and decryption procedures. The main disadvantage is the large key sizes. The use of quasi-cyclic (QC) codes has allowed to reduce the size of the keys [Gab05; Ber+09; MB09]. However, algebraic attacks are successful due to the algebraic structure of these codes [Fau+10]. One way to overcome this is to increase the chosen value of the parameters since algebraic attacks present exponential complexity. Another approach is to use codes with no algebraic structure.

Low-density parity-check (LDPC) codes (see Section 2.5.2) are codes with no algebraic structure that admit a sparse parity-check matrix. The main problem of employing LDPC codes in the McEliece scheme is that the low-weight parity-check rows can be seen as low-weight codewords in the dual of the public code. An effective attack consists in building a sparse parity-check matrix by finding dual low-weight codewords [MRS00]. The authors in [BC07] propose fixes to avoid the attack, however, another successful attack was presented in [OTD10]. An improved variant of LDPC-McEliece is suggested in [BBC08].

Definition 5.1.1. An $[n, k]$-linear code admitting a sparse parity-check matrix with constant row weight w is an $[n, k, w]$-*moderate density parity-check* (MDPC) code.

LDPC and MDPC codes are different only in the weight w of the rows in the parity-check matrix. LDPC codes have small values for w (e.g., less than 10).

Misoczki et al. [Mis+13] propose the use of MDPC codes in the McEliece scheme. The authors observed that moderately increasing the length and the row weight of the secret sparse parity-check matrix is enough to avoid message and key recovery attacks. Misoczki et al. consider MDPC codes to have row weights which scale in $O(\sqrt{n \log n})$, where n is the code length. The codes in [Mis+13] are constructed as follows:

- [n, k, w]-MDPC CODE

  Let $r = n - k$. Generate a parity-check matrix $H \in \mathbb{F}_2^{r \times n}$ by randomly selecting vectors $h_i \in \mathbb{F}_2^n$, $i = 1, \ldots, r$, and setting the i-th row of H to be h_i. With very high probability H has full rank and the rightmost $r \times r$ block is invertible, after swapping some columns if needed.

- [n, k, w]-QC-MDPC CODE

  Let $r = n - k$ and $n = n_0 r$. Then, the parity-check matrix has the form

  $$H = (H_0 \mid H_1 \mid \cdots \mid H_{n_0-1}),$$

  where H_i is an $r \times r$ circulant block. The first row of H is defined by randomly selecting a vector $h \in \mathbb{F}_2^n$ with weight w. The other rows of H are obtained from the $r - 1$ quasi-cyclic shifts of h. Each block H_i has row weight w_i, such that $w = \sum_{i=0}^{n_0-1} w_i$. Assuming that H_{n_0-1} is non-singular, the generator matrix is constructed as

  $$G = \left(\begin{array}{c|c} & (H_{n_0-1}^{-1} \cdot H_0)^T \\ I & \vdots \\ & (H_{n_0-1}^{-1} \cdot H_{n_0-2})^T \end{array} \right).$$

  The algebra of $r \times r$ binary circulant matrices is isomorphic to $\mathbb{F}_2[X]/\langle X^r - 1 \rangle$. The matrix/vector operations can then be treated as operations in this polynomial ring.

The McEliece variant in [Mis+13] uses either of the codes above and is defined as follows:

- KEY GENERATION

  – Generate a parity-check matrix H for a t-error-correcting [n, k, w]-MDPC or [n, k, w]-QC-MDPC code.
  – Generate a corresponding generator matrix G in reduced row echelon form.
  – Return the public key G and private key H.

- ENCRYPTION

  Let $m \in \mathbb{F}_2^k$ be the plaintext.

  – Randomly generate $e \in \mathbb{F}_2^n$ with Hamming weight $\leqslant t$.
  – Compute $c = mG + e$.
  – Return the ciphertext c.

- DECRYPTION

  Let $c \in \mathbb{F}_2^n$ be the ciphertext and Ψ be a decoding algorithm for the code in KEY GENERATION.

  – Compute $c' = \Psi(H, c) = mG$ and recover m from the first k positions of c'.
  – Return the plaintext m.

5.1.2 p-ary MDPC variant

In [GJ16], Guo and Johansson extend the MDPC scheme from a binary field to a field of characteristic p. This allows to further reduce the size of the keys compared to the binary MDPC. The main structure of the p-ary scheme is similar to the binary one. However, in the p-ary scheme, the error vectors e are ideally taken from a discrete Gaussian distribution. Also, the Euclidean metric is used instead of the Hamming weight. The length-n codes of dimension k in [GJ16] are constructed as follows:

- p-ARY MDPC CODE

  Let $r = n - k$. Randomly select vectors $h_i \in \mathbb{F}_p^n$, $i = 1, \ldots, r$, with w_{sig} significant entries, such that w_1 entries are chosen from $\{-1, 1\}$, w_2 entries are chosen from $\{-2, 2\}$ and the others are 0. Construct the parity-check matrix $H \in \mathbb{F}_p^{r \times n}$ by setting the i-th row of H to be h_i.

- p-ARY QC-MDPC CODE

  Let $r = n - k$ and $n = n_0 r$. Then, the parity-check matrix has the form

  $$H = (H_0 \mid H_1 \mid \cdots \mid H_{n_0 - 1}),$$

  where H_i is an $r \times r$ circulant block. The first row of H is defined by randomly selecting a vector $h \in \mathbb{F}_p^n$ with w_{sig} significant entries, such that w_1 entries are chosen from $\{-1, 1\}$, w_2 entries are chosen from $\{-2, 2\}$ and the others are 0. The other rows of H are obtained from the $r - 1$ quasi-cyclic shifts of h. Let $w_{sig,i}$ denote the number of significant entries in each block H_i. Then, $w_{sig} = \sum_{i=0}^{n_0-1} w_{sig,i}$. Each block of H is isomorphic to a polynomial $h_i(X) \in \mathbb{F}_p[X]/\langle X^r - 1 \rangle$.

The p-ary MDPC variant in [GJ16] uses either of the codes above and is defined as follows:

- KEY GENERATION
  - Generate a parity-check matrix H for a p-ary MDPC or p-ary QC-MDPC code.
  - Generate a corresponding generator matrix G in reduced row echelon form. G should be a dense matrix, otherwise, generate a new parity-check matrix H.
  - Return the public key G and private key H.

- ENCRYPTION

  Let $m \in \mathbb{F}_p^k$ be the plaintext.
  - Randomly generate $e \in \mathbb{F}_p^n$ according to a discrete Gaussian distribution. Other distributions are also possible.
  - Compute $c = mG + e$.
  - Return the ciphertext c.

- DECRYPTION

  Let $c \in \mathbb{F}_p^n$ be the ciphertext.

– Compute the syndrome vector $s = cH^T = eH^T$ and use a decoder to extract the noise e. Recover m from the first k positions of mG.

– Return the plaintext m.

5.2 A bit-flipping decoder for p-ary QC-MDPC codes

Misoczki et al. [Mis+13] presented a decoder for their binary MDPC codes. The decoder is based on the bit flipping algorithm for LDPC codes by Gallager [Gal62] (see Section 2.5.2). The error-correction capability of the latter increases linearly with the length of the code and decreases somewhat linearly with the weight of the parity-checks. Hence, there is a degradation going from LDPC to MDPC codes.

Guo and Johansson [GJ16] proposed a new iterative decoder for their p-ary MDPC codes. The p-ary MDPC scheme can be seen as an extension of McEliece MDPC into the Euclidean metric, or as an NTRU-type [HPS98] lattice-based scheme with iterative decoding. A specific instantiation of the p-ary MDPC scheme was used by Deneuville et al. to construct Ouroboros-E [Den+18], a lattice-based key-exchange protocol.

The goal of using iterative decoding techniques in lattice-based cryptography is to reduce the alphabet size while good error performance is maintained. This technique can enhance efficiency and security. On one hand, it allows to propose a smaller public key size; on the other hand, lattice reduction algorithms like sieving and enumeration can be less efficient when the alphabet is small – these attack algorithms are considered to be the main threat nowadays.

We now consider an instance similar to that of Ouroboros-E. The p-ary QC-MDPC code has length n and dimension k. The parity-check matrix has two blocks of size $k \times k$. Other parameter sets can be based on secure parameter suggestions from NTRU-based or Ring-LWE-based [LPR10] cryptosystems, e.g., choosing a non-prime alphabet size (denoted by q instead of p for the general setting), or using the ring $\mathcal{R} = \mathbb{Z}_q[X]/\langle X^k + 1\rangle$ rather than the quasi-cyclic structure. Safe parameters include q a prime, k a power of 2 and $\mathcal{R} = \mathbb{Z}_q[X]/\langle X^k + 1\rangle$, or q a power of 2, k a prime and $\mathcal{R} = \mathbb{Z}_q[X]/\langle X^k - 1\rangle$. Let d be a positive integer, and define $\mathcal{I}_d = \{-d, \dots, 0, \dots, d\}$, $h_0^{(d)} = \lfloor \frac{q}{2d+1} \rceil (1, 0, \dots, 0)$ and $h_1^{(d)} = \lfloor \frac{q}{(2d+1)^2} \rceil (1, 0, \dots, 0)$. The p-ary MDPC instance is defined as:

- Key generation
  - Given a parameter d, compute $h_0 = h_0^{(d)} + \hat{h}_0$ and $h_1 = h_1^{(d)} + \hat{h}_1$, where $\hat{h}_0, \hat{h}_1 \in \mathcal{I}_d^k$ are chosen randomly. We assume that the ring $\mathbb{Z}_q[X]/\langle X^k - 1\rangle$ is employed and require the sum of the coefficients to be 0 for $\hat{h}_0$ and $\hat{h}_1$.
  - Compute $H = (H_0 | H_1)$, where H_0 and H_1 are matrices obtained by performing $k-1$ cyclic shifts of h_0 and h_1, respectively. If H_0 is singular, regenerate h_0 and h_1.
  - Compute $G = (I \,|\, H_0^{-1} H_1)$.
  - Return the public key G and private key H.

- Encryption

  Let $m \in \mathbb{F}_q^k$ be the plaintext.

  - Randomly choose a vector e from $\mathcal{J}_d^n$.
  - Compute $c = mG + e$.
  - Return the ciphertext c.

- Decryption

  Let $c \in \mathbb{F}_q^n$ be the ciphertext.

  - Get mG using a decoder to remove the noise e from c. Recover m from the first k entries of mG.
  - Return the plaintext m.

Algorithm 5.1 shows the decoder for the scheme above with $d = 1$. The decoder is close to the noisy p-ary bit-flipping decoder for Ouroboros-E, with adjustments for the p-ary MDPC. The decoder takes as inputs the ciphertext c, the parity-check matrix H and a parameter $iter$ specifying the number of iterations. The algorithm outputs the error vector e if decoding was successful, or $\perp$ (decoding error) otherwise. The main idea is to recover the error e by iteratively updating the value of each entry e_i according to a decision rule. When the correct value of e is found, we have that $s - He^T = 0$, where $s = Hc^T$. If the computed error e is such that $s - He^T \neq 0$ after $iter$ iterations, decoding was unsuccessful.

Algorithm 5.1 p-ary bit-flipping

Input: The ciphertext c, the private key H and the number of iterations $iter$
Output: The error e if success, $\perp$ otherwise

$e = 0 \in \mathbb{Z}_q^n$
Compute the syndrome $s = Hc^T$
$p = s$
for $i = 1$ **to** $iter$ **do**
  $e' = \text{Decide}(p)$
  $e = e + e'$
  $\text{Transform}(e)$
  $p = s - He^T$
  if $p_j = 0$ for all $j \in \{1, \dots, k\}$ **then**
    return e
  end if
end for
return $\perp$

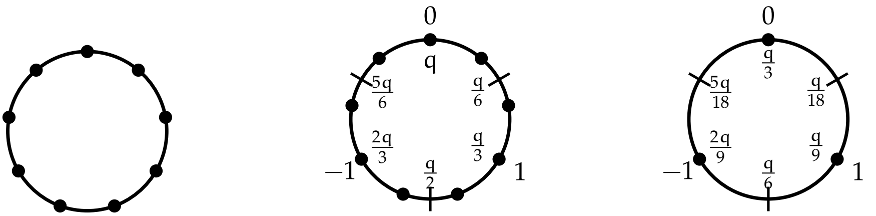
Algorithm 5.2 details the updating decision rule for recovering e. Recall that each parity-check (row) of H has two significant entries, namely, the $h_0^{(1)}$ and $h_1^{(1)}$ parts of h_0 and h_1, respectively. Each of these entries is sampled from $\{-1, 0, 1\}$, and thus we have 9 different signal points to consider, i.e., $\{-1, 0, 1\} \times \{-1, 0, 1\}$. We spread these 9 points throughout $[0, q] \pmod q$ as shown in figure 5.1a. In order to update e for the $h_0^{(1)}$ part, the set $[0, q] \pmod q$ is divided into 3 intervals. These intervals are determined by $\frac{q}{6}$, $\frac{q}{2}$ and $\frac{5q}{6}$, and each one has 3 signal points and an associated update value, as shown in Figure 5.1b. Notice that any of these intervals has "length" $\frac{q}{3}$ and three signal points which represent the possible update values for the $h_1^{(1)}$ part. Updating e for the

$h_1^{(1)}$ part is done in a similar way as for $h_0^{(1)}$. We divide the set $[0, \frac{q}{3}] \pmod{\frac{q}{3}}$ into 3 intervals determined by $\frac{q}{18}$, $\frac{q}{6}$ and $\frac{5q}{18}$, each one having 1 signal point, as shown in Figure 5.1c.

Algorithm 5.2 DECIDE

Input: Vector p
Output: Vector e' resulting from applying the decision rule to p

$e' = 0$
for $j = 1$ **to** k **do**
  if $p_j \in [\lceil \frac{q}{6} \rceil, \lfloor \frac{q}{2} \rfloor]$ **then**
    $e'_j = 1$
  else if $p_j \in [\lceil \frac{q}{2} \rceil, \lfloor \frac{5q}{6} \rfloor]$ **then**
    $e'_j \leftarrow -1$
  end if
  if $(p_j \bmod \lfloor \frac{q}{3} \rceil) \in [\lceil \frac{q}{18} \rceil, \lfloor \frac{q}{6} \rfloor]$ **then**
    $e'_{k+j} \leftarrow 1$
  else if $(p_j \bmod \lfloor \frac{q}{3} \rceil) \in [\lceil \frac{q}{6} \rceil, \lfloor \frac{5q}{18} \rfloor]$ **then**
    $e'_{k+j} \leftarrow -1$
  end if
end for
return e'



(**a**) Signal points across $[0, q] \pmod{q}$.

(**b**) Intervals of $[0, q] \pmod{q}$ with their signal points and associated values for $h_0^{(1)}$.

(**c**) Intervals of $[0, \frac{q}{3}] \pmod{\frac{q}{3}}$ with their signal points and associated values for $h_1^{(1)}$.

Figure 5.1. Graphical representation of the decoding decision rule. When $d = 1$, there are 9 signal points to consider.

The vector e is updated in line 6 of Algorithm 5.1. Notice that the addition of e and e' may lead to coefficients equivalent to 2 or -2. We thus employ the function TRANSFORM to ensure the resulting vector is valid, i.e., from $\mathcal{T}_1^n$. In [Den+18], the function TRANSFORM sets a coefficient to be -1 if it is 2 and 1 if it is -2.

5.3 The new decoders

In this section we show an enhanced version of the decoding algorithm for the p-ary MDPC scheme in Section 5.2. The performance is improved using varying thresholds for different iterations, as the bit-flipping decoders in [Gal62; Mis+13].

5.3.1 The idea behind the new decoders

The decision rule in Algorithm 5.2 keeps the same interval bounds for all iterations in the main loop of Algorithm 5.1. We propose to update these bounds. For this, we introduce a threshold $\mathtt{thr}^{(i)}$ that determines the new interval bounds in iteration i, $i = 1, \dots, \mathtt{iter}$. Figure 5.2 depicts how the intervals might change throughout the main loop. When an entry p_j does not lie within any of the intervals for 1 and -1, we are not able to decide the value for e' at the corresponding position, and we set it to zero. Algorithm 5.3 shows the new decision rule using the proposed thresholds. In Sections 5.3.2 and 5.3.3 we propose new methods for computing $\mathtt{thr}^{(i)}$.

(a) $i = 1$ **(b)** $i = 2$ **(c)** $i = \mathtt{iter} - 1$ **(d)** $i = \mathtt{iter}$

Figure 5.2. Intervals used for iterations $i = 1, \dots, \mathtt{iter}$ in the new decoding procedure.

Algorithm 5.3 DecideThr(p)

Input: Vector p
Output: Vector e' resulting from applying the decision rule to p

$e' = 0$
Compute $\mathtt{thr}^{(i)}$
for $j = 1$ **to** k **do**
  if $p_j \in [\lceil \frac{q}{6} \rceil + \mathtt{thr}^{(i)}, \lfloor \frac{q}{2} \rfloor - \mathtt{thr}^{(i)}]$ **then**
    $e'_j = 1$
  else if $p_j \in [\lceil \frac{q}{2} \rceil + \mathtt{thr}^{(i)}, \lfloor \frac{5q}{6} \rfloor - \mathtt{thr}^{(i)}]$ **then**
    $e'_j = -1$
  end if
  if $(p_j \bmod \lfloor \frac{q}{3} \rceil) \in [\lceil \frac{q}{18} \rceil + \mathtt{thr}^{(i)}, \lfloor \frac{q}{6} \rfloor - \mathtt{thr}^{(i)}]$ **then**
    $e'_{k+j} = 1$
  else if $(p_j \bmod \lfloor \frac{q}{3} \rceil) \in [\lceil \frac{q}{6} \rceil + \mathtt{thr}^{(i)}, \lfloor \frac{5q}{18} \rfloor - \mathtt{thr}^{(i)}]$ **then**
    $e'_{k+j} = -1$
  end if
end for
return e'

Note that this new proposal is a generalisation of the previous decoder: when $\mathtt{thr}^{(i)} = 0$, it is equivalent to using the interval thresholds as in Algorithm 5.2. Moreover, Algorithm 5.3 can be further generalised: at iteration i, instead of choosing $\mathtt{thr}^{(i)}$ only, it is possible to choose different thresholds for the upper and lower bounds of each of the intervals.

5.3.2 Gallager-B type decoder

Here we estimate the error probability and derive a series of theoretical values for the decoding thresholds. The analogue in the Hamming metric is the tree-based analysis in [Gal62].

First, we assume that the noise random variable in each iteration is Gaussian with mean 0 (due to the intuition from the central limit theorem) since it is a sum of many small-noise variables with mean 0. For a fast estimation, we ignore the independence issue that may occur in the decoding process. We track the change of the average error variance in each iteration.

Every entry p_j in p is a sum of the contribution $e_j \lfloor \frac{q}{3} \rceil + e_{k+j} \lfloor \frac{q}{9} \rceil$ and a second part called noise, denoted N_j, where $N_j = (\hat{h}_0^{[j]}, \hat{h}_1^{[j]}) e^T$ and $\hat{h}_i^{[j]}$ means $\hat{h}_i$ cyclically shifted j steps. The noise variance σ_0^2 is initially $\frac{4n}{9} = \frac{8k}{9}$. We now only record the probability that a signal point (e_j, e_{k+j}) is wrongly decoded to its neighbour in the torus, e.g., (0,0) to (0,-1) or (0,1); (1,1) to (-1,-1) or (1,0). Let us denote

$$\pi_{i+1,+} = \operatorname{erfc}\left(\frac{\frac{q}{18} + \mathrm{thr}_{i+1}}{\sqrt{2}\sigma_i} \right),$$

and

$$\pi_{i+1,-} = \operatorname{erfc}\left(\frac{\frac{q}{18} - \mathrm{thr}_{i+1}}{\sqrt{2}\sigma_i} \right).$$

Let e be the original error vector and $\hat{e}^{(i)}$ be the guessed error vector in the i-th iteration. We have $\hat{e}^{(0)} = 0$ and we get

$$p^{(i)} = s - H\hat{e}^{(i)} = H(e - \hat{e}^{(i)}),$$

which is the input to the DecideThr() procedure in the i-th iteration.

We know that for $1 \leqslant j \leqslant k$, e_j and e_{k+j} are distributed uniformly in $\mathcal{I}_1 = \{-1, 0, 1\}$ before the first iteration. If $e_{k+j} = 0$, the coefficient e_j is almost certainly correctly decoded. If the signal point is $(0, 1)$ with probability $1/9$, then the probability to wrongly decode e_j to 1 is $0.5\pi_{1,+}$. If the signal point is $(1, -1)$ with probability $1/9$, then the probability to wrongly decode e_j to 0 is $0.5\pi_{1,-}$. If the signal point is $(1, 1)$ with probability $1/9$, then the probability to wrongly decode e_j to 0 is $0.5(\pi_{1,-} - \pi_{1,+})$ and to -1 is $0.5\pi_{1,+}$. We can compute the error variance by symmetry for the rest signal points. The noise variance introduced by the first part of the error vector can be estimated as $\frac{2}{3} \cdot \frac{2k}{9}(\pi_{1,-} + 2\pi_{1,+})$. Similarly, we estimate the noise variance introduced by the second part of the error vector as $\frac{2}{3} \cdot \frac{6k}{9}(\pi_{1,-} + 2\pi_{1,+})$. We can compute the noise variance σ_1^2 as $\sigma_1^2 = \frac{2}{3} \cdot \frac{8k}{9}(\pi_{1,-} + 2\pi_{1,+})$.

Since the error occurring in the first k positions is much easier to correct than the errors in the last k positions, we track only the noise variance from the last k positions of the error vector, from the second iteration. Let π_i denote the probability that the decision on the position $\hat{e}_{k+j}^{(i)}$ is correct. Thus, we have that

$$\pi_{i+1} = \pi_i(1 - \pi_{i+1,+}) + (1 - \pi_i)(1 - \pi_{i+1,-}),$$

and $\pi_0 = 1/3$.

Finally, we can iteratively estimate the value of the remaining noise in the $(i+1)$-th iteration, σ_{i+1}^2, as

$$\sigma_{i+1}^2 = \frac{2k}{3} \cdot (\pi_i \pi_{i+1,+} + (1 - \pi_i)(\pi_{i+1,-} + 1.5\pi_{i+1,+})), \tag{5.1}$$

for $i \geqslant 1$. We can also alternatively introduce a new parameter[1] $c_\lambda > 1$ to have better performance heuristically, i.e., we compute σ_{i+1}^2 by

$$\sigma_{i+1}^2 = c_\lambda \cdot \frac{2k}{3} \cdot (\pi_i \pi_{i+1,+} + (1-\pi_i)(\pi_{i+1,-} + 1.5\pi_{i+1,+}))\,, \tag{5.2}$$

for $i \geqslant 1$.

We choose the thresholds $\mathrm{thr}^{(i)}$ such that the noise variance σ_i^2 is minimised, which can be solved numerically. We denote by G1 and G2 the decoders using thresholds computed from Equations (5.1) and (5.2), respectively.

5.3.3 Heuristic decoder

Here we present three heuristic decoders called H1, H2 and H3, respectively. The three decoders are similar to their Hamming counterpart from [HP03; Mis+13] for MDPC-McEliece.

Let $\mathcal{A}$ be the set of nine signal points, i.e.,

$$\mathcal{A} := \{0, \lfloor q/9 \rceil, \lfloor 2q/9 \rceil, \ldots, \lfloor 8q/9 \rceil\}.$$

In the i-th iteration of H1, we compute

$$\mathrm{thr}_{\mathrm{max}}^{(i)} = \frac{q}{18} - \min_{\substack{j \in \{1,\ldots,k\}, \\ a \in \mathcal{A} \setminus \{0\}}} |p_j^{(i)} - a|,$$

to make a parity-check equation with its updated syndrome $p_j^{(i)}$ closest to a non-zero signal point corrected (flipped). The threshold in the i-th iteration is

$$\mathrm{thr}^{(i)} = \max(0, \mathrm{thr}_{\mathrm{max}}^{(i)} - \delta), \tag{5.3}$$

where a positive constant δ determined by simulation is used to reduce the required number of iterations in the average case.

The decoder H2 is a variant of H1 and provides comparable (or even better) error performance in some simulations. In the i-th iteration, we compute

$$\mathrm{thr}_{\mathrm{min}}^{(i)} = \min_{\substack{j \in \{1,\ldots,k\}, \\ a \in \mathcal{A}}} |p_j^{(i)} - a|.$$

The threshold in the i-th iteration is

$$\mathrm{thr}^{(i)} = \mathrm{thr}_{\mathrm{min}}^{(i)} + \delta \tag{5.4}$$

for a positive constant δ determined by simulation.

The decoder H3 is also a variant of H1 and provides the best error performance in simulations. We choose $\delta = \delta_0$, for some value δ_0. In the i-th iteration, we compute $\mathrm{thr}_{\mathrm{max}}^{(i)}$ and $\mathrm{thr}^{(i)}$ as in H1. If decoding is unsuccessful, decrease the value of δ by 1 and restart the process. This is repeated until decoding is successful or $\delta = 0$ (decoding failure).

[1] We have $c_\lambda > 1$ since the error contribution of the first k positions is omitted in Equation (5.1).

5.4 Experimental results

Here we give the experimental results using the decoders presented in Sections 5.3.2 and 5.3.3. We compare the proposed decoders with the reference decoder [Den+18]. Note that the values of k and q are only chosen for testing the error performance.

For G1, we computed the theoretical thresholds as in Section 5.3.2; the thresholds and variances obtained are shown in Table 5.1 when $c_\lambda = 1.00$. For G2, we computed the thresholds varying the value of c_λ from 1.0 to 2.0 by steps of 0.01. We heuristically chose to use for our experiments the values shown in Table 5.1, with $c_\lambda > 1.00$. For both, G1 and G2, let i be the smallest integer such that $\mathtt{thr}^{(i)} = 0$. Then we have that $\mathtt{thr}^{(j)} = 0$ for $j \geqslant i$. However, in the experiments we kept on using the last non-zero threshold up to iteration $\frac{\mathtt{iter}}{2}$, i.e., $\mathtt{thr}^{(j)} = \mathtt{thr}^{(i-1)}$ for $j = i, \dots, \frac{\mathtt{iter}}{2}$ and $\mathtt{thr}^{(j)} = 0$ for $j = \frac{\mathtt{iter}}{2} + 1, \dots, \mathtt{iter}$. If we had not made this, both decoders would have been different to that in [Den+18] in the first $i-1$ iterations only.

k	q	c_λ	$i=1$		$i=2$		$i=3$	
			$\mathtt{thr}^{(i)}$	σ_i^2	$\mathtt{thr}^{(i)}$	σ_i^2	$\mathtt{thr}^{(i)}$	σ_i^2
491	345	1.00	8	288.32	7	123.64	4	31.30
491	345	1.01	8	288.32	7	124.87	5	32.18
491	345	1.24	8	288.32	7	153.31	6	54.80
491	345	1.31	8	288.32	7	161.96	6	62.49
491	360	1.00	8	269.43	7	101.18	4	15.16
491	360	1.17	8	269.43	7	118.38	5	24.94
491	360	1.43	8	269.43	7	144.69	6	43.97
491	360	1.66	8	269.43	7	167.97	6	64.39

k	q	c_λ	$i=4$		$i=5$		$i\geqslant 6$	
			$\mathtt{thr}^{(i)}$	σ_i^2	$\mathtt{thr}^{(i)}$	σ_i^2	$\mathtt{thr}^{(i)}$	σ_i^2
491	345	1.00	2	0.12	0	0.0000	0	0.00
491	345	1.01	2	0.14	0	0.0000	0	0.00
491	345	1.24	3	2.71	0	0.0000	0	0.00
491	345	1.31	4	4.72	1	5.65×10^{-5}	0	0.00
491	360	1.00	1	3.70×10^{-5}	0	0.00	0	0.00
491	360	1.17	2	0.01	0	0.00	0	0.00
491	360	1.43	3	0.65	0	0.00	0	0.00
491	360	1.66	4	4.15	1	4.36×10^{-20}	0	0.00

Table 5.1. Computed theoretical thresholds for the Gallager-like decoders.

For the heuristic decoders, we first executed some moderate size experiments (100 MDPC instances and 1 000 decoding executions per instance) to determine the best values of δ. We used these values to execute the experiments reported here.

Table 5.2 summarises the results obtained from the experiments. One experiment comprises the random generation of 10 000 MDPC instances and 1 000 decoding executions per instance. The entries in the table show the number of decoding errors among 10^7 decoding executions. For decoder G2, columns $c_{\lambda,2}$, $c_{\lambda,3}$ and $c_{\lambda,4}$ show the results when using the thresholds in the 2nd, 3rd and 4th rows of Table 5.1, respectively, for the different values of k and q. In general, our proposals have better performance than the reference decoder. The heuristic decoders present the best decoding failure rate. Decoder H3 presents a very low failure rate with the parameters in table 5.2. We also executed this decoder with $k = 491$, $q = 375$ obtaining 0 decoding errors for $\mathtt{iter} = 20$.

We noted in the experiments that the execution time of the reference decoder was significantly higher than that of the decoders we propose. This difference in performance is due to the number of iterations required to finish the decoding procedure in the average case.

k	q	iter	[Den+18]	G1	G2			H1		H2		H3
					$c_{\lambda,2}$	$c_{\lambda,3}$	$c_{\lambda,4}$	$\delta = 16$	$\delta = 17$	$\delta = 2$	$\delta = 3$	$\delta_0 = \lfloor \frac{q}{18} \rceil$
491	345	100	271	11	14	17	27	4	5	2	6	0
491	345	75	405	19	24	17	22	10	13	5	11	0
491	345	50	992	60	53	49	98	29	23	27	28	0
491	345	25	14543	909	949	911	1309	346	367	402	402	12
491	360	100	5	1	2	3	1	1	1	0	0	0
491	360	75	4	2	2	3	2	4	2	1	1	0
491	360	50	17	10	11	2	11	5	8	1	5	0
491	360	25	267	163	131	122	188	44	32	32	48	2

Table 5.2. Decoding failure rate $(\times 10^{-7})$ of experiments for different parameters. Each experiment comprises the random generation of 10 000 MDPC instances and 1 000 decoding executions per instance.

Summary and future work

We presented novel iterative decoders for the p-ary MDPC scheme by varying the thresholds used in each iteration. These thresholds are determined either by numerically optimising the error level in the next iteration, as was done by Gallager [Gal62], or by applying heuristic methods. We have demonstrated improved decoding performance by simulation. Particularly, our heuristic decoders presented the best error failure rate (see Table 5.2).

We identify two more interesting problems to be further investigated. Firstly, to propose a better worst-case decoder, as was studied in [CS16] for the binary MDPC. Also, to test the heuristic independence assumption made in [GJ16] for proposing parameters with an arbitrarily small decryption error probability. The latter can be crucial to resist the potential reaction attack [GJS16; Fab+17] that is already a threat to the MDPC/LDPC-based cryptosystems.

Bibliography

[Ågr+11] Martin Ågren, Martin Hell, Thomas Johansson and Willi Meier. "Grain-128a: A New Version of Grain-128 with Optional Authentication". In: *International Journal of Wireless and Mobile Computing* 5.1 (Dec. 2011), pp. 48–59.

[Ågr+12] Martin Ågren, Carl Löndahl, Martin Hell and Thomas Johansson. "A survey on fast correlation attacks". In: *Cryptography and Communications* 4.3 (Dec. 2012), pp. 173–202.

[And95] Ross Anderson. "Searching for the optimum correlation attack". In: *Fast Software Encryption. FSE 1994*. Ed. by Bart Preneel. Vol. 1008. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 1995, pp. 137–143.

[Arm04] Frederik Armknecht. "Improving Fast Algebraic Attacks". In: *Fast Software Encryption. FSE 2004*. Ed. by Bimal Roy and Willi Meier. Vol. 3017. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 2004, pp. 65–82.

[AN72] Krishna Athreya and Peter Ney. *Branching Processes*. Vol. 196. Grundlehren der mathematischen Wissenschaften. Springer, Berlin, Heidelberg, 1972.

[Bah+74] Lalit Bahl, John Cocke, Fred Jelinek and Josef Raviv. "Optimal decoding of linear codes for minimizing symbol error rate". In: *IEEE Transactions on Information Theory* 20.2 (Mar. 1974), pp. 284–287.

[BBC08] Marco Baldi, Marco Bodrato and Franco Chiaraluce. "A New Analysis of the McEliece Cryptosystem Based on QC-LDPC Codes". In: *Security and Cryptography for Networks. SCN 2008*. Ed. by Rafail Ostrovsky, Roberto De Prisco and Ivan Visconti. Vol. 5229. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 2008, pp. 246–262.

[BC07] Marco Baldi and Franco Chiaraluce. "Cryptanalysis of a new instance of McEliece cryptosystem based on QC-LDPC Codes". In: *2007 IEEE International Symposium on Information Theory*. IEEE, 2007, pp. 2591–2595.

[Ben+97] Charles Bennett, Ethan Bernstein, Gilles Brassard and Umesh Vazirani. "Strengths and Weaknesses of Quantum Computing". In: *SIAM Journal on Computing* 26.5 (1997), pp. 1510–1523.

[Ber+09] Thierry Berger, Pierre-Louis Cayrel, Philippe Gaborit and Ayoub Otmani. “Reducing Key Length of the McEliece Cryptosystem”. In: *Progress in Cryptology – AFRICACRYPT 2009*. Ed. by Bart Preneel. Vol. 5580. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 2009, pp. 77–97.

[Ber73] Elwyn Berlekamp. “Goppa codes”. In: *IEEE Transactions on Information Theory* 19.5 (Sept. 1973), pp. 590–592.

[BMT78] Elwyn Berlekamp, Robert McEliece and Henk van Tilborg. “On the Inherent Intractability of Certain Coding Problems”. In: *IEEE Transactions on Information Theory* 24.3 (May 1978), pp. 384–386.

[BGT93] Claude Berrou, Alain Glavieux and Punja Thitimajshima. “Near Shannon limit error-correcting coding and decoding: Turbo-codes (1)”. In: *Proceedings of ICC ’93 – IEEE International Conference on Communications*. Vol. 2. IEEE, 1993, pp. 1064–1070.

[Bil95] Patrick Billingsley. *Probability and Measure*. 3rd ed. Wiley Series in Probability and Statistics. John Wiley and Sons, 1995.

[Buc04] Johannes Buchmann. *Introduction to Cryptography*. 2nd ed. Undergraduate Texts in Mathematics. Springer, New York, 2004.

[CGJ19] Isaac A. Canales-Martínez, Qian Guo and Thomas Johansson. “Improved Decoders for p-ary MDPC”. In: *11th International Workshop on Coding and Cryptography. WCC 2019*. 2019.

[CS21] Isaac A. Canales-Martínez and Igor Semaev. “Multivariate Correlation Attacks and the Cryptanalysis of LFSR-based Stream Ciphers”. In preparation. 2021.

[Can06] Anne Canteaut. “Open Problems Related to Algebraic Attacks on Stream Ciphers”. In: *Coding and Cryptography. WCC 2005*. Ed. by Øyvind Ytrehus. Vol. 3969. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 2006, pp. 120–134.

[Can11] Anne Canteaut. “Linear Feedback Shift Register”. In: *Encyclopedia of Cryptography and Security*. Ed. by Henk van Tilborg and Sushil Jajodia. Springer, Boston, MA, 2011, pp. 726–729.

[CT00] Anne Canteaut and Michaël Trabbia. “Improved Fast Correlation Attacks Using Parity-Check Equations of Weight 4 and 5”. In: *Advances in Cryptology – EUROCRYPT 2000*. Ed. by Bart Preneel. Vol. 1807. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 2000, pp. 573–588.

[Car21] Claude Carlet. *Boolean Functions for Cryptography and Coding Theory*. Cambridge University Press, 2021.

[CB02] George Casella and Roger Berger. *Statistical Inference*. 2nd ed. Duxbury Advanced Series. Duxbury, 2002.

[CS16] Julia Chaulet and Nicolas Sendrier. “Worst case QC-MDPC decoder for McEliece cryptosystem”. In: *2016 IEEE International Symposium on Information Theory*. IEEE, 2016, pp. 1366–1370.

[CJS01] Vladimor Chepyzhov, Thomas Johansson and Ben Smeets. “A Simple Algorithm for Fast Correlation Attacks on Stream Ciphers”. In: *Fast Software Encryption. FSE 2000*. Ed. by Gerhard Goos, Juris Hartmanis, Jan van Leeuwen and Bruce Schneier. Vol. 1978. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 2001, pp. 181–195.

[CJM02] Philippe Chose, Antoine Joux and Michel Mitton. “Fast Correlation Attacks: An Algorithmic Point of View”. In: *Advances in Cryptology – EUROCRYPT 2002*. Ed. by Lars Knudsen. Vol. 2332. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 2002, pp. 209–221.

[Cou03] Nicolas Courtois. “Fast Algebraic Attacks on Stream Ciphers with Linear Feedback”. In: *Advances in Cryptology – CRYPTO 2003*. Ed. by Dan Boneh. Vol. 2729. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 2003, pp. 176–194.

[CM03] Nicolas Courtois and Willi Meier. “Algebraic Attacks on Stream Ciphers with Linear Feedback”. In: *Advances in Cryptology – EUROCRYPT 2003*. Ed. by Eli Biham. Vol. 2656. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 2003, pp. 345–359.

[Den+18] Jean-Christophe Deneuville, Philippe Gaborit, Qian Guo and Thomas Johansson. “Ouroboros-E: An efficient Lattice-based Key-Exchange Protocol”. In: *2018 IEEE International Symposium on Information Theory*. IEEE, 2018, pp. 1450–1454.

[Did07] Frédéric Didier. “Attacking the Filter Generator by Finding Zero Inputs of the Filtering Function”. In: *Progress in Cryptology – INDOCRYPT 2007*. Ed. by Kannan Srinathan, Chandrasekaran Pandu Rangan and Moti Yung. Vol. 4859. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 2007, pp. 404–413.

[DH76] Whitfield Diffie and Martin Hellman. “New Directions in Cryptography”. In: *IEEE Transactions on Information Theory* 22.6 (Nov. 1976), pp. 644–654.

[Din+11] Itai Dinur, Tim Güneysu, Christof Paar, Adi Shamir and Ralf Zimmermann. “An Experimentally Verified Attack on Full Grain-128 Using Dedicated Reconfigurable Hardware”. In: *Advances in Cryptology – ASIACRYPT 2011*. Ed. by Dong Hoon Lee and Xiaoyun Wang. Vol. 7073. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 2011, pp. 327–343.

[DS11] Itai Dinur and Adi Shamir. “Breaking Grain-128 with Dynamic Cube Attacks”. In: *Fast Software Encryption. FSE 2011*. Ed. by Antoine Joux. Vol. 6733. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 2011, pp. 167–187.

[ElG85a] Taher ElGamal. “A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms”. In: *IEEE Transactions on Information Theory* 31.4 (July 1985), pp. 469–472.

[ElG85b] Taher ElGamal. “A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms”. In: *Advances in Cryptology. CRYPTO 1984*. Ed. by George Robert Blakley and David Chaum. Vol. 196. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 1985, pp. 10–18.

[Eli57] Peter Elias. “List Decoding for Noisy Channels”. In: *Wescon Convention Record* Part 2 (July 1957), pp. 94–104.

[Eur] European Network of Excellence in Cryptology. *eSTREAM: the ECRYPT Stream Cipher Project*. http://www.ecrypt.eu.org/stream. Accessed in December 2021.

[Fab+17] Tomáš Fabšič, Viliam Hromada, Paul Stankovski, Pavol Zajac, Qian Guo and Thomas Johansson. “A Reaction Attack on the QC-LDPC McEliece Cryptosystem”. In: *Post-Quantum Cryptography. PQCrypto 2017*. Ed. by Tanja Lange and Tsuyoshi Takagi. Vol. 10346. Lecture Notes in Computer Science. Springer, Cham, 2017, pp. 51–68.

[Fau+10] Jean-Charles Faugère, Ayoub Otmani, Ludovic Perret and Jean-Pierre Tillich. “Algebraic Cryptanalysis of McEliece Variants with Compact Keys”. In: *Advances in Cryptology – EUROCRYPT 2010*. Ed. by Henri Gilbert. Vol. 6110. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 2010, pp. 279–298.

[Gab05] Philippe Gaborit. “Shorter keys for code based cryptography”. In: *International Workshop on Coding and Cryptography. WCC 2005*. 2005.

[Gal62] Robert Gallager. “Low-Density Parity-Check Codes”. In: *IRE Transactions on Information Theory* 8.1 (Jan. 1962), pp. 21–28.

[GB09] Alan Genz and Frank Bretz. *Computation of Multivariate Normal and t Probabilities*. Vol. 195. Lecture Notes in Statistics. Springer, Berlin, Heidelberg, 2009.

[Gen+21] Alan Genz, Frank Bretz, Tetsuhisa Miwa, Xuefei Mi, Friedrich Leisch, Fabian Scheipl and Torsten Hothorn. *mvtnorm: Multivariate Normal and t Distributions*. R package version 1.1-3. 2021. URL: https://CRAN.R-project.org/package=mvtnorm.

[GRS95] Oded Goldreich, Ronitt Rubinfeld and Madhu Sudan. “Learning polynomials with queries: The highly noisy case”. In: *Proceedings of IEEE 36th Annual Foundations of Computer Science*. IEEE, 1995, pp. 294–303.

[Gol96a] Jovan Dj. Golić. “Computation of low-weight parity-check polynomials”. In: *Electronics Letters* 32.21 (Oct. 1996), pp. 1981–1982.

[Gol96b] Jovan Dj. Golić. “On the Security of Nonlinear Filter Generators”. In: *Fast Software Encryption. FSE 1996*. Ed. by Dieter Gollmann. Vol. 1039. Lecture Notes Computer Science. Springer, Berlin, Heidelberg, 1996, pp. 173–188.

[GCD00] Jovan Dj. Golić, Andrew Clark and Ed Dwason. “Generalized Inversion Attack on Nonlinear Filter Generators”. In: *IEEE Transactions on Computers* 49.10 (Oct. 2000), pp. 1100–1109.

[Gol17] Solomon Golomb. *Shift Register Sequences*. 3rd ed. World Scientific, 2017.

[GG02] Anna Górska and Karol Górski. “Improved inversion attacks on nonlinear filter generators”. In: *Electronics Letters* 38.16 (Aug. 2002), pp. 870–871.

[Gro96] Lov Grover. “A Fast Quantum Mechanical Algorithm for Database Search”. In: *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*. Association for Computing Machinery, 1996, pp. 212–219.

[GJ16] Qian Guo and Thomas Johansson. "A p-ary MDPC scheme". In: *2016 IEEE International Symposium on Information Theory*. IEEE, 2016, pp. 1356–1360.

[GJS16] Qian Guo, Thomas Johansson and Paul Stankovski. "A Key Recovery Attack on MDPC with CCA Security Using Decoding Errors". In: *Advances in Cryptology – ASIACRYPT 2016*. Ed. by Jung Hee Cheon and Tsuyoshi Takagi. Vol. 10031. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 2016, pp. 789–815.

[Har63] Theodore Edward Harris. *The Theory of Branching Processes*. Grundlehren der mathematischen Wissenschaften. Springer, Berlin, Heidelberg, 1963.

[HR04] Philip Hawkes and Gregory Rose. "Rewriting Variables: The Complexity of Fast Algebraic Attacks on Stream Ciphers". In: *Advances in Cryptology – CRYPTO 2004*. Ed. by Matt Franklin. Vol. 3152. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 2004, pp. 390–406.

[Hel+06] Martin Hell, Thomas Johansson, Alexander Maximov and Willi Meier. "A Stream Cipher Proposal: Grain-128". In: *2006 IEEE International Symposium on Information Theory*. IEEE, 2006, pp. 1614–1618.

[Hel+08] Martin Hell, Thomas Johansson, Alexander Maximov and Willi Meier. "The Grain Family of Stream Ciphers". In: *New Stream Cipher Designs. The eSTREAM Finalists*. Ed. by Matthew Robshaw and Olivier Billet. Vol. 4986. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 2008, pp. 179–190.

[HJM07] Martin Hell, Thomas Johansson and Willi Meier. "Grain: A Stream Cipher for Constrained Environments". In: *International Journal of Wireless and Mobile Computing* 2.1 (May 2007), pp. 86–93.

[HPS98] Jeffrey Hoffstein, Jill Pipher and Joseph Silverman. "NTRU: A ring-based public key cryptosystem". In: *Algorithmic Number Theory. ANTS 1998*. Ed. by Joe Buhler. Vol. 1423. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 1998, pp. 267–288.

[HP03] William Huffman and Vera Pless. *Fundamentals of Error-Correcting Codes*. Cambridge University Press, 2003.

[JZ15] Rolf Johannesson and Kamil Sh. Zigangirov. *Fundamentals of Convolutional Coding*. 2nd. Wiley-IEEE Press, 2015.

[JJ99a] Thomas Johansson and Fredrik Jönsson. "Fast Correlation Attacks Based on Turbo Code Techniques". In: *Advances in Cryptology – CRYPTO '99*. Ed. by Michael Wiener. Vol. 1666. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 1999, pp. 181–197.

[JJ99b] Thomas Johansson and Fredrik Jönsson. "Improved Fast Correlation Attacks on Stream Ciphers via Convolutional Codes". In: *Advances in Cryptology – EUROCRYPT '99*. Ed. by Jacques Stern. Vol. 1592. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 1999, pp. 347–362.

[JJ00] Thomas Johansson and Fredrik Jönsson. "Fast Correlation Attacks through Reconstruction of Linear Polynomials". In: *Advances in Cryptology – CRYPTO 2000*. Ed. by Mihir Bellare. Vol. 1880. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 2000, pp. 300–315.

[KL14] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. 2nd ed. Cryptography and Network Security. CRC Press, 2014.

[Ker83] Auguste Kerckhoffs. “La cryptographie militaire”. In: *Journal des Sciences Militaires* 9 (Jan. 1883), pp. 5–38, 161–191.

[Lan02] Serge Lang. *Algebra*. 3rd ed. Vol. 211. Graduate Texts in Mathematics. Springer, New York, 2002.

[LLL82] Arjen Lenstra, Hendrik Lenstra and László Lovász. “Factoring polynomials with rational coefficients”. In: *Mathematische Annalen* 261.4 (Dec. 1982), pp. 515–534.

[Lev+01] Sabine Leveiller, Joseph Boutros, Philippe Guillot and Gilles Zémor. “Cryptanalysis of Nonlinear Filter Generators with $\{0,1\}$-Metric Viterbi Decoding”. In: *Cryptography and Coding. 8th IMA International Conference*. Ed. by Bahram Honary. Vol. 2260. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 2001, pp. 402–414.

[Lev+03] Sabine Leveiller, Gilles Zémor, Philippe Guillot and Joseph Boutros. “A New Cryptanalytic Attack for PN-generators Filtered by a Boolean Function”. In: *Selected Areas in Cryptography. SAC 2002*. Ed. by Kaisa Nyberg and Howard Heys. Vol. 2595. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 2003, pp. 232–249.

[LN96] Rudolf Lidl and Harald Niederreiter. *Finite Fields*. 2nd ed. Vol. 20. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1996.

[LPR10] Vadim Lyubashevsky, Chris Peikert and Oded Regev. “On Ideal Lattices and Learning with Errors over Rings”. In: *Advances in Cryptology – EUROCRYPT 2010*. Ed. by Henri Gilbert. Vol. 6110. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 2010, pp. 1–23.

[MS81] Florence Jessie MacWilliams and Neil James Alexander Sloane. *The Theory of Error-Correcting Codes*. Vol. 16. North-Holland Mathematical Library. North-holland Publishing Company, 1981.

[Mas69] James Massey. “Shift-Register Synthesis and BCH Decoding”. In: *IEEE Transactions on Information Theory* 15.1 (Jan. 1969), pp. 122–127.

[McE78] Robert McEliece. “A Public-Key Cryptosystem Based On Algebraic Coding Theory”. In: *The Deep Space Network Progress Report* 42-44 (1978), pp. 114–116.

[Mei11] Willi Meier. “Fast Correlation Attacks: Methods and Countermeasures”. In: *Fast Software Encryption. FSE 2011*. Ed. by Antoine Joux. Vol. 6733. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 2011, pp. 55–67.

[MPC04] Willi Meier, Enes Pasalic and Claude Carlet. “Algebraic Attacks and Decomposition of Boolean Functions”. In: *Advances in Cryptology – EUROCRYPT 2004*. Ed. by Christian Cachin and Jan Camenisch. Vol. 3027. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 2004, pp. 474–491.

[MS89] Willi Meier and Othmar Staffelbach. "Fast Correlation Attacks on Certain Stream Ciphers". In: *Journal of Cryptology* 1.3 (Oct. 1989), pp. 159–176.

[MFI01] Miodrag Mihaljević, Marc Fossorier and Hideki Imai. "A Low-Complexity and High-Performance Algorithm for the Fast Correlation Attack". In: *Fast Software Encryption. FSE 2000*. Ed. by Gerhard Goos, Juris Hartmanis, Jan van Leeuwen and Bruce Schneier. Vol. 1978. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 2001, pp. 196–212.

[MFI02] Miodrag Mihaljević, Marc Fossorier and Hideki Imai. "Fast Correlation Attack Algorithm with List Decoding and an Application". In: *Fast Software Encryption. FSE 2001*. Ed. by Mitsuru Matsui. Vol. 2355. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 2002, pp. 196–210.

[MB09] Rafael Misoczki and Paulo Barreto. "Compact McEliece Keys from Goppa Codes". In: *Selected Areas in Cryptography. SAC 2009*. Ed. by Michael Jacobson, Vincent Rijmen and Reihaneh Safavi-Naini. Vol. 5867. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 2009, pp. 376–392.

[Mis+13] Rafael Misoczki, Jean-Pierre Tillich, Nicolas Sendrier and Paulo Barreto. "MDPC-McEliece: New McEliece Variants from Moderate Density Parity-Check Codes". In: *2013 IEEE International Symposium on Information Theory*. IEEE, 2013, pp. 2069–2073.

[MMH03] Håvard Molland, John Erik Mathiassen and Tor Helleseth. "Improved Fast Correlation Attack Using Low Rate Codes". In: *Cryptography and Coding. 9th IMA International Conference*. Ed. by Kenneth Paterson. Vol. 2898. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 2003, pp. 67–81.

[MRS00] Chris Monico, Joachim Rosenthal and Amin Shokrollahi. "Using low density parity check codes in the McEliece cryptosystem". In: *2000 IEEE International Symposium on Information Theory*. IEEE, 2000, p. 215.

[NP33] Jerzy Neyman and Egon Sharpe Pearson. "IX. On the Problem of the most Efficient Tests of Statistical Hypotheses". In: *Philosophical Transactions of the Royal Society of London* 231.694–706 (Feb. 1933), pp. 289–337.

[NIS] NIST Cryptographic Technology Group. *Block Cipher Techniques*. https://csrc.nist.gov/Projects/block-cipher-techniques/BCM. Accessed in December 2021.

[OTD10] Ayoub Otmani, Jean-Pierre Tillich and Léonard Dallot. "Cryptanalysis of Two McEliece Cryptosystems Based on Quasi-Cyclic Codes". In: *Mathematics in Computer Science* 3.2 (Jan. 2010), pp. 129–140.

[R C21] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2021. URL: %5Curl%7Bhttps://www.R-project.org%7D.

[Rab79] Michael Rabin. *Digitalized Signatures and Public-Key Functions as Intractable as Factorization*. Massachusetts Institute of Technology, 1979.

[RS08] Håvard Raddum and Igor Semaev. "Solving Multiple Right Hand Sides linear equations". In: *Designs, Codes and Cryptography* 49.1 (Dec. 2008), pp. 147–160.

[RSA78] Ronald Linn Rivest, Adi Shamir and Leonard Adleman. “A Method for Obtaining Digital Signatures and Public-Key Cryptosystems”. In: *Communications of the ACM* 21.2 (Feb. 1978), pp. 120–126.

[Sho97] Peter Shor. “Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer”. In: *SIAM Journal on Computing* 26.5 (1997), pp. 1484–1509.

[Sie84] Thomas Siegenthaler. “Correlation-Immunity of Nonlinear Combining Functions for Cryptographic Applications”. In: *IEEE Transactions on Information Theory* 30.5 (Sept. 1984), pp. 776–780.

[Sie85] Thomas Siegenthaler. “Decrypting a Class of Stream Ciphers Using Ciphertext Only”. In: *IEEE Transactions on Computers* C-49.1 (Jan. 1985), pp. 81–85.

[SP17] Douglas Stinson and Maura Paterson. *Cryptography Theory and Practice*. 4th ed. Textbooks in Mathematics. CRC Press, 2017.

[Stu91] Study Group VII of the International Telegraph and Telephone Consultative Committee. *Security Architecture for Open Systems Interconnections for CCITT Applications*. Standard. International Telecommunication Union, 1991.

[Tec15] Technical Committee ISO/IEC JTC 1/SC 31. *Information technology – Automatic identification and data capture techniques – Part 13: Crypto suite Grain-128A security services for air interface communications*. Standard. International Organization for Standardization, 2015.

[Tec00] Technical Committee ISO/TC 97. *Information processing systems – Open Systems Interconnection – Basic Reference Model, Part 2: Security Architecture*. Standard. International Organization for Standardization, 2000.

[Tod+18] Yosuke Todo, Takanori Isobe, Willi Meier, Kazumaro Aoki and Bin Zhang. “Fast Correlation Attack Revisited. Cryptanalysis on Full Grain-128a, Grain-128, and Grain-v1”. In: *Advances in Cryptology – CRYPTO 2018*. Ed. by Hovav Shacham and Alexandra Boldyreva. Vol. 10992. Lecture Notes in Computer Science. Springer Cham, 2018, pp. 129–159.

[Vit67] Andrew Viterbi. “Error bounds for convolutional codes and an asymptotically optimum decoding algorithm”. In: *IEEE Transactions on Information Theory* 13.2 (1967), pp. 260–269.

[Wag02] David Wagner. “A Generalized Birthday Problem”. In: *Advances in Cryptology – CRYPTO 2002*. Ed. by Moti Yung. Vol. 2442. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 2002, pp. 288–304.

[Zha+14] Bin Zhang, Zhenqi Li, Dengguo Feng and Dongdai Lin. “Near Collision Attack on the Grain v1 Stream Cipher”. In: *Fast Software Encryption. FSE 2013*. Ed. by Shiho Moriai. Vol. 8424. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 2014, pp. 518–538.

[ZXM18] Bin Zhang, Chao Xu and Willi Meier. “Fast Near Collision Attack on the Grain v1 Stream Cipher”. In: *Advances in Cryptology – EUROCRYPT 2018*. Ed. by Jesper Buus Nielsen and Vincent Rijmen. Vol. 10821. Lecture Notes in Computer Science. Springer Cham, 2018, pp. 771–802.

Graphic design: Communication Division, UiB / Print: Skipnes Kommunikasjon AS

uib.no

ISBN: 9788230857854 (print)
9788230858042 (PDF)