Post-Quantum Cryptography

Gauthier Umana, Valérie ; Knudsen, Lars Ramkilde; Leander, Gregor

Publication date:
2011

Document Version
Publisher's PDF, also known as Version of record

Technical University of Denmark

Department of Mathematics

PhD Thesis

Post-Quantum Cryptography

Supervisors: **Professor Lars Ramkilde Knudsen**
Associate Professor Gregor Leander

Valérie Gauthier Umaña

October 2011

Post-Quantum Cryptography

Author:

Valérie Gauthier Umaña
Technical University of Denmark
Department of Mathematics
Building 303S, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253008
www.mat.dtu.dk

Supervisors:

Professor Lars Ramkilde Knudsen and Associate Professor Gregor Leander
Technical University of Denmark
Department of Mathematics
Building 303S, DK-2800 Kongens Lyngby, Denmark

Censors:

Anne Canteaut, INRIA Paris-Rocquencour, France

Thomas Johansson, Lund University, Sweden

Peter Beelen, Technical University of Denmark

Summary

The security of almost all the public-key cryptosystems used in practice depends on the fact that the prime factorization of a number and the discrete logarithm are hard problems to solve. In 1994, Peter Shor found a polynomial-time algorithm which solves these two problems using quantum computers. The public key cryptosystems that can resist these emerging attacks are called *quantum resistant* or *post-quantum cryptosystems*. There are mainly four classes of public-key cryptography that are believed to resist classical and quantum attacks: *code-based cryptography, hash-based cryptography, lattice-based cryptography* and *multivariate public-key cryptography*.

In this thesis, we focus on the first two classes. In the first part, we introduce coding theory and give an overview of code-based cryptography. The main contribution is an attack on two promising variants of McEliece's cryptosystem, based on quasi-cyclic alternant codes and quasi-dyadic codes (joint work with Gregor Leander). We also present a deterministic polynomial-time algorithm to solve the Goppa Code Distinguisher problem for high rate codes (joint work with Jean-Charles Faugère, Ayoub Otmani, Ludovic Perret and Jean-Pierre Tillich).

In the second part, we first give an overview of hash based signature schemes. Their security is based on the collision resistance of a hash function and is a good quantum resistant alternative to the used signature schemes. We show that several existing proposals of how to make multiple-time signature schemes are not any better than using existing one-time signature schemes a multiple number of times. We propose a new variant of the classical one-time signature schemes based on (near-)collisions resulting in two-time signature schemes. We also give a new, simple and efficient algorithm for traversing a tree in tree-based signature schemes (joint work with Lars R. Knudsen and Søren S. Thomsen).

Resumé

Sikkerheden i de fleste public-key kryptosystemer, der bruges i praksis, afhænger af, at faktoriseringsproblemet eller det diskrete logaritme-problem er svære at løse. I 1994 fandt Peter Shor en algoritme med polynomiel tidskompleksitet, som løser disse to problemer ved hjælp af en kvantecomputer. Public-key kryptosystemer, som bevarer deres sikkerhed over for en kvantecomputer, kaldes *post-kvante kryptosystemer*. Der findes overordnet set fire typer af public-key kryptografi, som menes af være sikre mod en kvantecomputer: kode-baseret kryptografi, hash-baseret kryptografi, lattice-baseret kryptografi samt kryptografi baseret på multivariate andengradspolynomier.

I denne afhandling fokuserer vi på de første to typer. I den første del introduceres fejlrettende koder, og der gives et overblik over kode-baseret kryptografi. Det væsentligste bidrag er et angreb på to lovende varianter af McEliece kryptosystemet baseret på kvasicykliske alternerende koder og på kvasidyadiske koder (samarbejde med Gregor Leander). Der præsenteres desuden en deterministisk polynomieltidsalgoritme, som løser det såkaldte "Goppa Code Distinguisher" problem for koder med høj rate (samarbejde med Jean-Charles Faugère, Ayoub Otmani, Ludovic Perret og Jean-Pierre Tillich).

I afhandlingens anden del gives først et overblik over hash-baserede digitale signatursystemer. Sikkerheden i disse er baseret på kollisionssikkerheden af en haskfunktion, og de er gode post-kvante alternativer til de signatursystemer, der bruges i praksis i dag. Det vises, at mange eksisterende forslag til hash-baserede digitale signatursystemer, som kan underskrive flere beskeder for hver nøgle, hvad angår nøglestørrelse ikke er bedre end ældre signatursystemer, der kun kan underskrive en enkelt besked per nøgle. Der foreslås desuden en ny variant af et klassisk engangs-signatursystem: en variant der er baseret på (nær-)kollisioner i en hashfunktion, og som kan underskrive to beskeder per nøgle. Slutteligt beskrives en ny, simpel og hurtig algoritme til beregning af knuderne i et træ i træ-baserede signatursystemer (samarbejde med Lars R. Knudsen og Søren S. Thomsen).

Preface

This thesis is submitted in partial fulfillment of the requirements for obtaining the PhD degree at the Technical University of Denmark. Two thirds of the PhD project were funded by a grant from Villum Kann Rasmussen Fonden (VKR 021031) "Ny Kryptologi", and the other third by the Technical University of Denmark. It was carried out at the Department of Mathematics of the Technical University of Denmark from November 2008 to October 2011. The supervisors of the project were Professor Lars Ramkilde Knudsen and Associate Professor Gregor Leander. The thesis is based on the following three papers:

- Valérie Gauthier Umaña and Gregor Leander. Practical Key Recovery Attacks On Two McEliece Variants. Presented at SCC 2010 and submitted to: Special Issue of Mathematics in Computer Science on Symbolic Computation and Cryptography II. Guest Editors: Carlos Cid (RHUL) and Jean-Charles Faugère (INRIA-UPMC)

- Jean-Charles Faugère, Valérie Gauthier Umaña, Ayoub Otmani, Ludovic Perret and Jean-Pierre Tillich. A Distinguisher for High Rate McEliece Cryptosystems. In IEEE Information Theory Workshop (ITW 2011), pages 1-5, October 2011.

- Lars R. Knudsen, Søren S. Thomsen and Valérie Gauthier Umaña. On hash based digital signatures. Submitted to the journal "Designs, Codes and Cryptography".

Acknowledgements

I would like to thank my two supervisors Lars R. Knudsen and Gregor Leander, I am very happy that I had the opportunity to work with them. I would like to thank them for giving me the opportunity to join the crypto group at DTU, for always being available when I needed to ask any questions, to supporting and helping me enormously during the all process. They were a fundamental guide in the making of this thesis.

I would like to thank all my coauthors: Ayoub Otmani, Gregor Leander, Jean-Charles Faugère, Jean-Pierre Tillich, Lars R. Knudsen, Ludovic Perret and Søren S. Thomsen. It was a great pleasure for me to have the opportunity to work with all of them.

I also would like to thank Tom Høholdt, for taking care of all the PhD students, for being concerned about me and my adaptation to Denmark. For being a great professor and always being open to answer any questions and giving me great advice.

To Søren, who was always ready to work with me, proof-reading my thesis and giving me very useful advice and comments.

I am very grateful with all the discrete math group, for welcoming to Denmark, and always being open to answer any question. I had a great time during my PhD and that was because of the the great work environment. Thank you Lars, Gregor, Tom, Peter, Carsten, Søren, Julia, Christiane, Safia, Mohamed, Hoda, Johan, Fernando, Erik, Krystian, Praveen, Kristian, Diego and Charlotte.

During my PhD studies I had the opportunity to visit the group SECRET at INRIA Paris-Rocquencourt. I want to thank Jean-Pierre Tillich and Ayoub Otmani for inviting me and giving me such a nice welcome. It was a pleasure to have been there and getting to work with them. I will also like to thank all the people in the SECRET group that welcomed me in a very friendly way.

To Christiane, for proof-reading the thesis, it was very nice to share the office (even for a short time).

I would like to thank all the people at DTU Mathematics, specially the PhD students (some of them that have already left), we shared a lot of good moments together. It was a pleasure to share so many coffee breaks, lunches, trips, cakes, football table games, beer clubs, Christmas dinners, etc. I really enjoyed and appreciated their company.

To Julia, who was not only my PhD mentor and office mate, but a great friend. It was a pleasure for me to share so many chats and beers with her. A lot of good moments that we shared inside and outside DTU come to my mind now, thanks a lot.

A Alex, Natalia y M.Angélica que aunque desde lejos, siempre me dieron muchas palabras de aliento y razones para sonreír.

A Yovana, que estoy segura de que no hubiera podido salir tan bien librada en todos estos años sin su gran apoyo y amistad. Mil gracias por las eternas chateadas, los viajes, los sketch, en fin: la incondicionalidad. Mil y mil gracias por ser ese chaleco salvavidas (que aunque te conocí sin mangas) desde hace 10 años sabíamos que lo íbamos a necesitar, pero lo importante es que si estuvo ahí.

A la Abuelita, Peri, Brayan y toda mi familia por estar presentes desde la distancia siempre apoyándome y dándome valor. No es fácil estar lejos de ellos, me hicieron mucha falta y esta tesis, que es uno de los resultados de mi viaje a Europa, es en gran parte suya por que sin tanto apoyo no habría sido capaz de llegar a hasta aquí.

A Cécile, Mamá y Papá: es imposible expresar lo agradecida que estoy por tenerlos en mi vida: mil gracias! este logro no solo es mio sino suyo, gracias por estar siempre ahí, estoy muy feliz de poder compartir este momento con ellos, los quiero mucho.

Valérie Gauthier Umaña
Lyngby, October 2011

Contents

Chapter 1

Introduction

Human beings have tried to hide written information since writing was developed. There are stone inscriptions and papyruses showing that many ancient civilizations like the Egyptians, Hebrews and Assyrians had cryptographic systems. The word *cryptology* is derived from the Greek words "$\kappa\rho\upsilon\pi\tau\acute{o}\varsigma$" (hidden) and "$\lambda o\gamma\acute{\iota}\alpha$" (study). It combines *cryptography* and *cryptanalysis*, i.e., the art of making cryptosystems and the art of analyzing the security of the cryptosystems and trying to break them.

Before the modern era, cryptography was used only to ensure secrecy in communications, i.e., to enable two people to communicate over an insecure channel so that any third party can neither understand nor modify the message. The main idea is to modify the message such that no-body apart from the receiver can understand its meaning; we will call this new message the ciphertext.

Nowadays, cryptography is the cornerstone in data security and is used for many purposes: secrecy of data, to ensure anonymity, to ensure the authenticity of communications, digital signatures, etc. Some example of daily use of cryptography are the electronic commerce, e-banking, ATM cards, computer password, etc.

Cryptography is mainly divided in two types: *symmetric* and *asymmetric* (or *public-key*) cryptography. Let us assume that Alice wants to send a message to Bob through an insecure channel. In the first case, Alice and Bob agree on a secret key. This key is used in the encryption and the decryption process. In the asymmetric case, there exist two different keys (produced by Bob): the public key, used in the encryption process, and the secret key used to decrypt the ciphertext. As we can see in Figure 1.1, Alice chooses a message m, encrypts it using Bob's public key and sends it to Bob. Bob is the only one who is able to find the original message, since he is the only one who knows the secret key. In a public-key cryptosystem (PKC) we need a function that is easy to compute in one way (anybody can encrypt the message) and that is hard to invert unless we have an additional information called a *trapdoor*. Therefore, these functions are called *trapdoor one-way functions*. The idea of using this kind of function was proposed by Diffie and Hellman in 1976, but they didn't give any example. The first public-key cryptosystem (called RSA) was proposed by Rivest, Shamir and Adleman in 1977, after this scheme many other schemes emerged using different kinds of one-way functions. Nowadays, many strong, and standardized, public key encryption schemes are available. Nevertheless, the security of the public-key cryptosystems used in practice depend dangerously on only the two following problems:

- **The factoring problem:** Given $n = pq$, where p and q are different primes, find p and q. This is a hard problem.

- **The discrete logarithm problem:** Given α, m and $\beta = \alpha^a \bmod m$, find a. This is a hard problem if the involved numbers are large.

In both cases one has to be careful with the choice of the values since there are some ease cases.

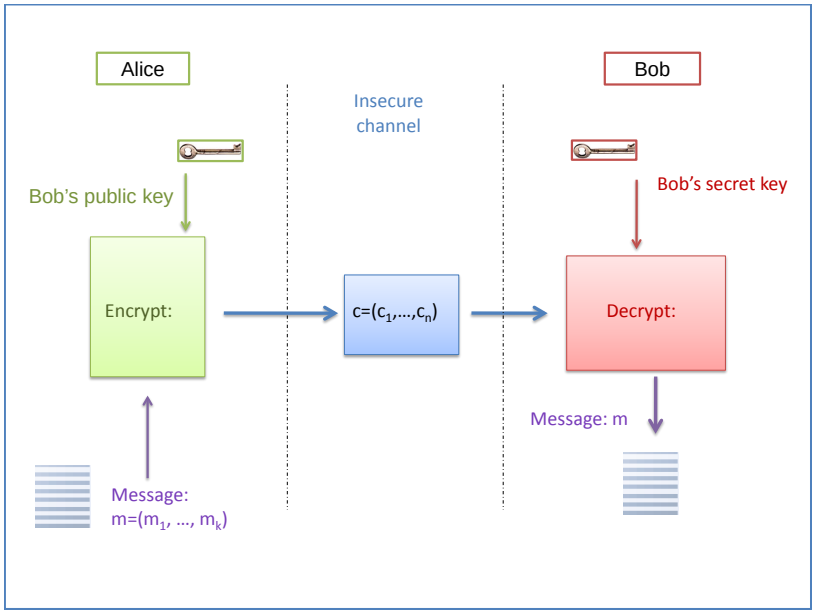

Figure 1.1: Public-key cryptosystem scheme.

In 1994, Peter Shor found a polynomial-time algorithm [105] which solves these two problems using quantum computers. Therefore, public key cryptosystems based on these problems would be broken as soon as quantum computers of an appropriate size could be built. The public key cryptosystems that remain secure even when the adversary has access to a quantum computer are called *quantum resistant* or *post-quantum cryptosystems*. Grover's algorithm [56], is another quantum algorithm that may lead to some attacks, but it is not too dangerous since cryptographers can avoid the attack by a simple change of parameters (the algorithm has exponential complexity).

An other reason to motivate the research of alternative systems is that most of the standard schemes are too expensive to be implemented in very constrained environments like RFID tags or sensor networks.

There are mainly four classes of public-key cryptography that are believed [24] to resist classical and quantum attacks:

- Code-based cryptography
- Hash-based cryptography
- Lattice-based cryptography

- Multivariate public-key cryptography.

In this thesis we will focus on the first two classes. In the first part, we will introduce coding theory (see Chapter 2) and give an overview of code-based cryptography (see Chapter 3). We will introduce the McEliece cryptosystem [78], that is a well-known alternative public-key encryption scheme based on the hardness of decoding random (looking) linear codes. To today's knowledge, it resists quantum computers. Another advantage is that for encryption only elementary binary operations are needed and one might therefore hope that McEliece is suitable for constrained devices, see for example [38, 59]. However, this scheme has a serious drawback: the public and the secret keys have larger magnitudes compared to RSA. Therefore one very reasonable approach is to modify McEliece's original scheme in such a way that it remains secure while the key size is reduced. A lot of papers already followed that line of research, but so far no satisfying answer has been found. Some of these proposals will be introduced in Chapter 4. In Chapter 5 we introduce two attacks on two McEliece variants: one based on quasi-cyclic alternant codes by Berger et al. [11] and the other based on quasi-dyadic matrices by Barreto and Misoczki [85] (introduced in Chapter 4). The first attack is from [52] and is a joint work with Gregor Leander. The second attack is due to Faugère et al. [44].

In 2001, Courtois et al. introduced the Goppa Code Distinguishing (GCD) problem [34]. This is a decision problem that aims at recognizing in polynomial time a generator matrix of a binary Goppa code from a randomly drawn binary matrix. The main motivation for introducing the GCD problem is to formalize the security of the McEliece public-key cryptosytem. In the same paper, Courtois et al. prove that the CFS signature scheme security can be reduced to the syndrome decoding problem and the distinguishability of binary Goppa codes from a random code. In Chapter 6 we present a deterministic polynomial-time distinguisher for high rate codes. This chapter is based on the paper [43], that is a joint work with Jean-Charles Faugère, Ayoub Otmani, Ludovic Perret and Jean-Pierre Tillich. The fact that this distinguisher problem is solved in the range of parameters used in the CFS signature scheme is not an attack on the system, but it invalidates the hypothesis of the security proof.

In the second part we will give an overview of hash based signature schemes. A digital signatures on a message is a special encryption of the message that can easily be verified by third parties. It is used in everyday situations providing authenticity, integrity and non-repudiation of data. The digital signature schemes that are used in practice are able to sign an unlimited number of messages for a given key and are based on trapdoor one-way functions.

We would like to have an alternative to these signature schemes, such that they are secure and efficient now and in the possible presence of quantum computers. The security of hash-based digital signature schemes is based on the collision resistance of a hash function. They are a good candidate for post-quantum alternatives. The first scheme originates from Lamport [68] devised a simple one-time signature scheme based on a one-way (hash) function. Later, Merkle and Winternitz [81, 82] proposed improvements to Lamport's original scheme. We will introduce these one-time signature schemes in Chapter 8.

Multiple-time signature schemes have also been proposed. They allow multiple (but usually still only a few) messages to be signed. We will introduce these in Chapter 9. The drawback of most hash-based signature schemes is that they are only able to sign a limited number of times for each secret/public key pair, and that the signature and the key pair are large. But on the other hand, they are typically efficient on the generation and verification process.

In 1979, Merkle introduced tree-based signature schemes which are build on one-time signatures, but which can be used to sign many messages. There are simple and efficient "chaining" methods to combine these one-time signature schemes. However, these "chaining" methods generally have a negative impact on the signing and verification efficiency, and also on the signature length. This will be explained in Chapter 10. Multiple-time signature schemes can often also be chained together, which would, in principle, allow for a large number of signatures with limited impact on signing, verification times and the length of the signature.

This part is based on joint work with Lars R. Knudsen and Søren S. Thomsen that appears in [64]. The contributions are: In Section 9.5, we show that several existing proposals of how to make multiple-time signature schemes are not any better than using existing one-time signature schemes a multiple number of times. We question whether any such schemes based on so-called cover-free families will be better than the simpler solution. Also, we investigate whether the cover-free families proposed in [73] based on orthogonal arrays would yield better results, but the answer is negative. In Section 9.7, we propose a new variant of the classical one-time signature schemes based on (near-)collisions resulting in two-time signature schemes. The new scheme can be used as the underlying signature schemes in Merkle's tree-based signature schemes. In Section 10.2, we give a new, simple and efficient algorithm for traversing a tree in tree-based signature schemes.

Part I

Code-based Cryptography

Chapter 2

Linear codes

Claude Shannon's paper "A Mathematical Theory of Communication" [103] from 1948 gave birth to the twin disciplines of information theory and coding theory. The main goal of coding theory is to find an efficient and reliable data transmission method. When we send a message (that can be seen as a string of symbols) through a noisy channel it is possible that the message gets modified (e.g., some errors are added to the message by changing some of the symbols). One simple way to find the errors would be to send the message several times and a majority vote method can enable us to recover the correct message. This is however not the best way to do it in order to be efficient. The main idea is to send more information through the channel than what we actually need. If this redundancy has a structure, we will be able to use it to correct possible added errors. In this chapter we will give an overview of the principal definitions and theorems that we will need in the thesis, for more information please refer to [62] and [76].

2.1 Linear codes

One of the largest families of error-correcting codes are *block codes*. Each message is divided in blocks of the same length and each of these blocks is encoded and sent separately. Linear codes are one of the most important subfamilies and are the ones we are interested in. Let $\mathbb{F}$ be a finite field and n and k two natural numbers such that $k < n$. All vectors are row vectors. This notation is going to be used throughout the thesis (unless we write explicitly another definition). An (n, k)-linear code $\mathscr{C}$ is a k-dimensional subspace of the vector space $\mathbb{F}^n$. The block of information that we want to send can be seen as a vector in $\mathbb{F}^k$. Using a linear encoding function we will map this k-vector into a codeword in $\mathbb{F}^n$. Any matrix that corresponds to this linear function is called a *generator matrix of* $\mathscr{C}$ and we denote it by G. We then have that $\mathscr{C} \stackrel{\text{def}}{=} \{uG | u \in \mathbb{F}^k\}$. G is said to be in a *systematic form*, if its first k columns form the identity matrix. Note that the redundancy is added by choosing $k < n$. The code uses n symbols to send k message symbols, the *transmission rate* is $\frac{k}{n}$. We may also define a *parity-check vector*, h, of length n which satisfies

$$Gh^T = 0$$

(where h^T denotes the transpose of h). The parity-check vectors are a subspace of the vector space $\mathbb{F}^n$. We define a *parity-check matrix* H for an (n, k)-linear code over $\mathbb{F}$ by an $(n - k) \times n$ matrix whose rows are linearly independent parity-check vectors. Note that for each generator matrix G we can determine it's kernel, whose basis is

given by H: the corresponding parity-check matrix of G such that $GH^T = 0$, where 0 is a $k \times (n-k)$ matrix of zeros. The *dual code* $\mathscr{C}^\perp$ of $\mathscr{C}$ is the $(n, n-k)$-linear code over $\mathbb{F}$ generated by H.

Property 2.1. *[62, page 4] Given the parity-check matrix in the* canonical form $H = [A|I_{n-k}]$, *we have that* $G = [I_k| - A^T]$. *Here* I_j *is the* $j \times j$ *identity matrix.*

Assume that we want to send a message $u \in \mathbb{F}^k$ through a noisy channel, the first step is to encode it, and we will obtain the codeword $c = uG$. The receiver will get $y = c + e$ where $e \in \mathbb{F}^n$ is an error vector. We assume that the amount of errors added is not too big. In Figure 2.1 we can see the encoding scheme.

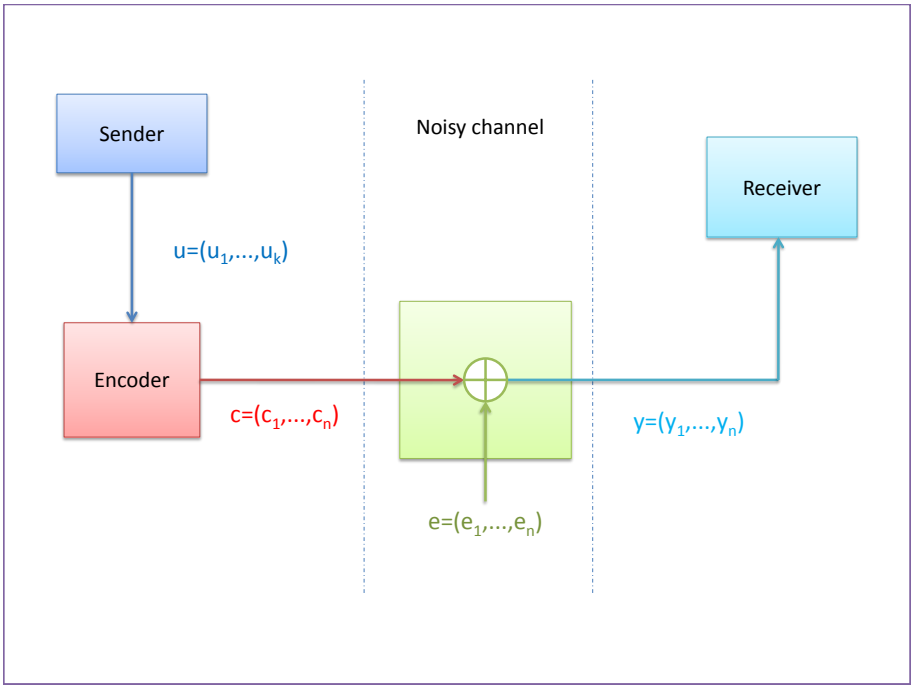

Figure 2.1: Encoding scheme.

2.2 Decoding problems

Keeping the same notations, our goal is to recover c from y. Once we have it, it will be easy to recover u. In Figure 2.2 we can see the complete communication system. We denote u' the vector that the receiver gets, it is assumed to be equal to the original message u (but it may be different if the receiver does not recover c from y but an other codeword).

As we assume that the amount of errors added is not too big, we try to find the "closest" codeword of y, and we claim that this codeword is c. *Decode* y is the process of recovering the codeword that is the "closest" to y. To define properly the "closest" notions we introduce the Hamming distance.

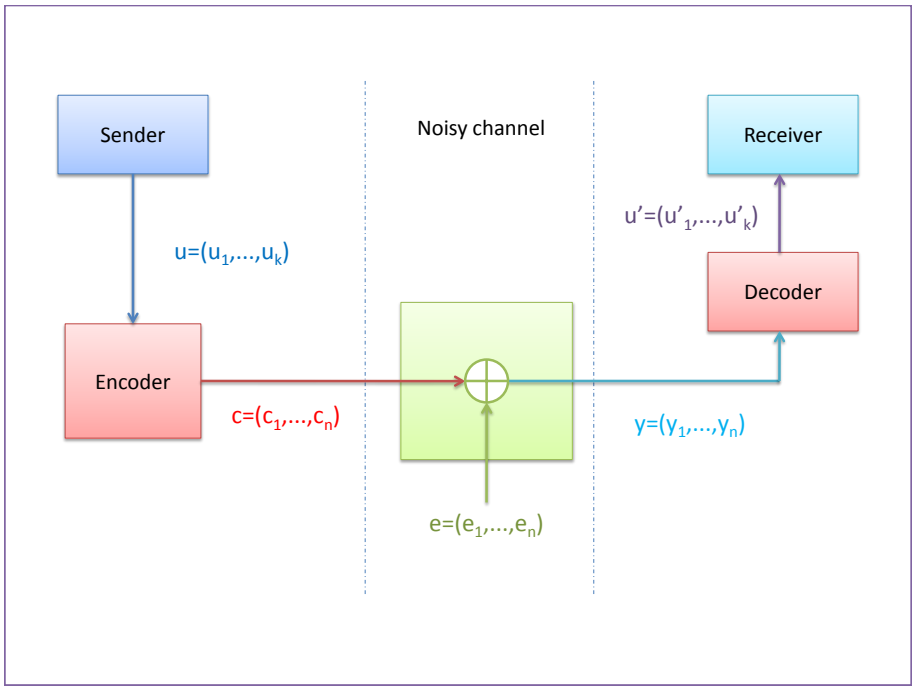

Figure 2.2: The communication system.

Definition 2.1. *[62, page 4,5] The Hamming distance between two words x and y in $\mathbb{F}^n$ ($dist(x,y)$) is defined to be the number of coordinates in which x and y differ. The Hamming weight $wt(x)$ of x is the number of non-zero coordinates of x.*

Definition 2.2. *[62, page 5] Let $\mathscr{C}$ denote an (n,k)-linear code over $\mathbb{F}$. The minimum distance d of $\mathscr{C}$ is the smallest Hamming distance between distinct codewords. We add this information in the description of the code, and we say that it is a (n,k,d)-linear code.*

For a linear code the minimum distance is the minimal weight of a non-zero codeword.

Theorem 2.1 (The Singleton bound)**.** *[62, page 49] Let $\mathscr{C}$ be a (n,k)-linear code with minimum distance d. Then $d \leq n-k+1$.*

To prove this theorem we need the following two lemmas.

Lemma 2.1. *Let $\mathscr{C}$ be an (n,k,d)-linear code with parity-check matrix H. If j columns are linearly dependent, $\mathscr{C}$ contains a codeword with nonzero elements in some of the corresponding positions. If $\mathscr{C}$ contains a word of weight j, then there exist j linearly dependent columns of H.*

This follows immediately from the fact that the codewords are defined by the vectors c such that $Hc^T = 0$.

Lemma 2.2. *Let $\mathscr{C}$ be an (n, k, d)-linear code with parity-check matrix H. The minimum distance d is equal to the minimal number of linearly dependent columns of H.*

Theorem 2.1 follows from the fact that the rank of a parity-check matrix H is $n-k$, we have then that any $n-k+1$ columns of H are linearly dependent. Therefore the minimal number of dependent columns (that we know from Lemma 2.2, that is equal to d) must be smaller or equal to $n-k+1$.

Definition 2.3. *Given a code $\mathscr{C}$, a vector $y \in \mathbb{F}_q^n$ and a positive integer w a (w-error-correcting) decoding algorithm for $\mathscr{C}$ is an algorithm that gives the set of all elements $c \in \mathscr{C}$ such that $dist(x, y) \leq w$; the set is empty if there is no such c.*

Unique decoding algorithm: In order to have a unique decoding solution, the maximal number of errors that a linear code with minimum distance d can correct is the *error correcting capability* $t \stackrel{\text{def}}{=} \lfloor \frac{d-1}{2} \rfloor$: the biggest integers such that it is strictly smaller than $d/2$. If we decode until this amount of errors, we are sure to have a unique decoding solution. In fact, if we assume that there exists a codeword c with $dist(y, c) \leq t$, if there exist another codeword, named c' for example, such that the $dist(c', y) \leq t$, we will have that $dist(c, c') < d$ and this will contradict the definition of d. If there is no a codeword c such that $dist(c, y) \leq t$, we say that y cannot be decoded. In Figure 2.3 we can visualize the problem, if y is in the blue area it cannot be decoded. In an ideal case the circles centered in the codewords and of radius t will represent a partition of the space and all the received words will be able to be uniquely decoded.

List decoding algorithm: Given the received word y and a parameter $w > \lfloor \frac{d-1}{2} \rfloor$, the idea is to give the list of codewords $c_i \in \mathscr{C}$ such that the distance between c_i and y is less than w. This list may be empty, but may also contain more than one codeword. The main purpose is to allow a greater number of errors than in the unique decoding technique. We are only interested in the parameters w such that the problem has a unique solution with high probability.

Syndrome decoding: Keeping the previous notations, we define the syndrome by $syn(y) \stackrel{\text{def}}{=} Hy^T$. Then $syn(y)$ is a column vector of length $r = n - k$. Note that if the received word is $y = c + e$, then $syn(y) = H(c + e)^T = He^T$.

Theorem 2.2. *[76, page 17] For a binary code, the syndrome is equal to the sum of the columns of H where the errors occurred.*

This is why it is called the "syndrome", because it gives the symptoms of the errors. Define the syndrome mapping (related to H) by

$$\begin{array}{rccc} S_H: & \{0,1\}^n & \to & \{0,1\}^r \\ & y & \mapsto & (Hy^T)^T \end{array}$$

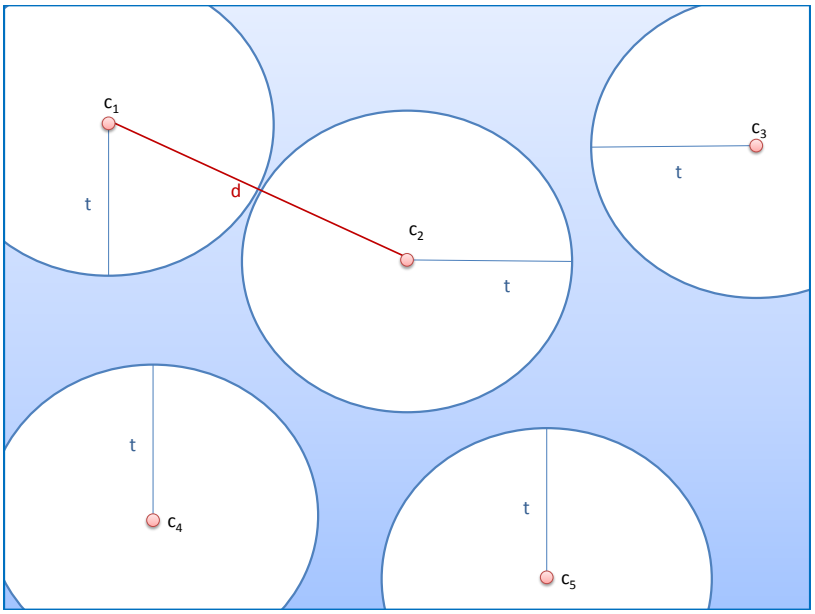

Figure 2.3: The codewords surrounded by circles of radius t, that is the biggest integers such that it is strictly smaller than $d/2$.

And we denote $S_H^{-1}(s)$ the set of words that have syndrome s^T:

$$S_H^{-1}(s) \stackrel{\text{def}}{=} \{y \in \{0,1\}^n | (Hy^T)^T = s\}.$$

Definition 2.4. *Let $\mathscr{C}$ be an (n,k)-linear code and $a \in \mathbb{F}^n$. The coset containing a is the set $a + \mathscr{C} = \{a + c | c \in \mathscr{C}\}$.*

If two words x and y are in the same coset, there exist two codewords c_1 and c_2 such that $x = a + c_1$ and $y = a + c_2$. We have then that $Hx^T = H(a + c_1)^T = Ha^T = H(a + c_2)^T = Hy^T$ so the two words have the same syndrome. On the other hand if two words x and y have the same syndrome, then $Hx^T = Hy^T$ and therefore $H(x - y)^T = 0$. This is the case only if $x - y$ is a codeword and therefore x and y are in the same coset. Therefore we have that

Lemma 2.3. *Two words are in the same coset if and only if they have the same syndrome.*

The cosets form a partition of the space into classes.

Proposition 2.1. *[93, page 106] For any $s \in \{0,1\}^r$ we have*

$$S_H^{-1}(s) = y + \mathscr{C} = \{y + c | c \in \mathscr{C}\},$$

where y is any word of $\{0,1\}^n$ of syndrome s^T. Moreover, finding such a word y from s (and H) can be achieved in polynomial time.

Given the parity-check matrix H, a vector y and its syndrome S, we would like to find the error e of minimum weight such that $He^T = S$. This is called *syndrome-decoding* problem. We also define the following problem:

Problem 2.1 (Computational Syndrome Decoding (CSD)). *Given a binary $r \times n$ parity-check matrix H, a word $s \in \{0,1\}^r$ and an integer $w > 0$, find a word $e \in S_H^{-1}(s)$ of Hamming weight $\leq w$.*

Berlekamp, McEliece and van Tilborg showed in 1978 [13] that the associated decision problem is $\mathcal{NP}$-complete.

Finding Low-Weight Codewords: Canteaut and Chabaud [26] showed that decoding a linear code can be reduced to the problem of finding small codewords in a related linear code. Keeping the previous notation, we have that $\mathscr{C}$ is a linear code of minimum distance d and generator matrix G. We fix $c \in \mathscr{C}$, $e \in \mathbb{F}_2^n$ of weight less than half the minimum distance d, and $y = c + e$ be the received word. Then we define $\mathscr{C}' =< \mathscr{C}, y >$, the linear code spanned by $\mathscr{C}$ and y. i.e., the code with generator matrix

$$\begin{bmatrix} G \\ y \end{bmatrix}.$$

As $dist(y, c) = wt(e) < d/2$, by definition of the minimum distance for every $c' \in \mathscr{C}$, $dist(y, c') \geq d/2$ and therefore $e = y - c$ is the codeword of $\mathscr{C}'$ of minimum weight. Thus, if we find the codeword of minimum weight of $\mathscr{C}'$ (i.e., e), we can decode y.

Problem 2.2 (Decoding problem). *Let $\mathscr{C}$ be an (n, k, d) linear code over $\mathbb{F}$, $y \in \mathbb{F}^n$ and $S = syn(y)$. The general decoding problem for linear codes is defined as solving one of the following equivalent problems.*

1. *Find $x \in \mathscr{C}$ where the Hamming distance between x and y is minimal.*
2. *Find an error vector $e \in y + \mathscr{C}$ of minimal Hamming weight.*
3. *Find an error vector $e \in S_H^{-1}(S^T)$ of minimal Hamming weight.*

The problem of decoding a random code is a long-standing problem, the most effective algorithms [16, 18, 27, 47, 71, 72, 109] have an exponential time complexity. These algorithms are based on information set decoding. We will give an overview in Section 3.2.1.

2.3 Special codes

In this section we are going to define some of the codes that we will use in the following chapters, for more information see [76]. Before getting to the proper definitions we will introduce some special type of matrices that we will need.

Definition 2.5. *[85] Let $X = (x_1, \ldots, x_n) \in \mathbb{F}^n$ and $r > 0$. The Vandermonde matrix with parameters r and X is defined as*

$$Vdm(r, X) = \begin{pmatrix} 1 & \cdots & 1 \\ x_1 & \cdots & x_n \\ \vdots & & \vdots \\ x_1^{r-1} & \cdots & x_n^{r-1} \end{pmatrix}$$

Definition 2.6. *[85] Given $r, n > 0$ and two disjoint sequences $z = (z_0, \ldots, z_{r-1}) \in \mathbb{F}^r$ and $L = (L_0, \ldots, L_{n-1}) \in \mathbb{F}^n$ of distinct elements. The Cauchy matrix $C(z, L)$ is the $r \times n$ matrix with elements $C_{ij} = 1/(z_i - L_j)$ i.e.,*

$$C(z, L) = \begin{pmatrix} \frac{1}{z_0 - L_0} & \cdots & \frac{1}{z_0 - L_{n-1}} \\ \vdots & \ddots & \vdots \\ \frac{1}{z_{r-1} - L_0} & \cdots & \frac{1}{z_{r-1} - L_{n-1}} \end{pmatrix}$$

Let $X = (x_1, \ldots, x_n) \in \mathbb{F}^n$, the *diagonal matrix* $Diag(X)$ is the square matrix whose diagonal is given by the entries of X and all the other entries are zeros.

$$Diag(X) = \begin{pmatrix} x_1 & 0 & \ldots & 0 \\ 0 & x_2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & x_n \end{pmatrix}$$

Cyclic codes

Cyclic codes form one of the most important class of linear codes.

Definition 2.7. *[62, page 63][Cyclic codes] An (n, k)-linear code $\mathscr{C}$ over $\mathbb{F}$ is called cyclic if any cyclic shift of a codeword is again a codeword i.e., if*

$$c = (c_0, c_1, \ldots, c_{n-1}) \in \mathscr{C} \Rightarrow c' = (c_{n-1}, c_0, \ldots, c_{n-2}) \in \mathscr{C}.$$

It is easier to understand the properties of these codes if we use the following algebraic description: we associate to each vector $c = (c_0, c_1, \ldots, c_{n-1}) \in \mathbb{F}^n$ the polynomial $c(x) = c_0 + c_1 x + \cdots + c_{n-1} x^{n-1}$ in $\mathbb{F}[x]$.

Theorem 2.3. *[62, page 64] Let $\mathscr{C}$ be a cyclic (n, k) code over $\mathbb{F}$ and $g(x)$ be a monic polynomial of minimal degree in $\mathscr{C} \setminus \{0\}$. Then*

1. *$g(x)$ divides $c(x)$ for every $c(x) \in \mathscr{C}$.*
2. *$g(x)$ divides $x^n - 1$ in $\mathbb{F}[x]$.*
3. *$k = n - deg(g(x))$.*

Note that $g(x)$ is unique. It is called the *generator polynomial* of the cyclic code $\mathscr{C}$.

Theorem 2.4. *[62, page 64] Suppose $g(x) \in \mathbb{F}[x]$ is monic and divide $x^n - 1$. Then*

$$\mathscr{C} = \{i(x)g(x) | i(x) \in \mathbb{F}[x], deg(i(x)) < n - deg(g(x))\}$$

is the cyclic code with generator polynomial $g(x)$.

Definition 2.8. *[76, page 506][Quasi-cyclic codes] A code of length n is called quasi-cyclic of order s (for n a multiple of s), if every cyclic shift of a codeword by s coordinates is again a codeword.*

Reed-Solomon and Generalized Reed-Solomon codes

Introduced in 1959 by Reed and Solomon [122], they are one of the most important class of error-correcting codes, having a wide application range, from encoding CDs and DVDs to satellite communications. Let q be an integer, we denote $\mathbb{F}_q$ the finite field with q elements and $\mathbb{P}_k$ the set of polynomials in $\mathbb{F}_q[x]$ of degree less than k.

Definition 2.9. *Let n and k two integers such that $k \leq n \leq q$ and $x_1, \ldots, x_n$ be different elements of $\mathbb{F}_q$. A Reed-Solomon code consists of the codewords*

$$(f(x_1), f(x_2), \ldots, f(x_n)) \text{ where } f \in \mathbb{P}_k.$$

The parity-check and the generator matrix of a Reed-Solomon code of length n and dimension k can be written as $H = Vdm(n-k, X) \times X^T$ and $G = Vdm(k, X)$, where $X = (x_1, \ldots, x_n) \in {\mathbb{F}_q}^n$ (proof in [62, page 56]). Since we must have $n \leq q$ we are not interested in the binary case, however in the majority of the practical cases we have $q = 2^m$. A polynomial of degree less than k can have at most $k-1$ zeros, so the weight of a codeword is at least $n - k + 1$. From Theorem 2.1 we can conclude that $d = n - k + 1$, i.e., the code corrects the maximal number of errors for the given parameters. There are several polynomial time methods to decode $\lfloor \frac{n-k}{2} \rfloor$ errors in a Reed Solomon code. An overview can be found in [62, Chapter 5].

Definition 2.10. *[76, page 303][General Reed-Solomon codes $GRS_r(X, D)$] Let $X = (x_1, \ldots, x_n)$ such that the x_i are pairwise different elements of $\mathbb{F}_{q^m}$ and $D = (D_1, \ldots, D_n)$ where D_i are non-zero elements of $\mathbb{F}_{q^m}$. For $k \leq n$ consider the set $\mathbb{P}_k$ of polynomials in $\mathbb{F}_{q^m}[x]$ of degree less than k. Let $r = n - k$, a General Reed-Solomon code $GRS_r(X, D)$ consists of the codewords*

$$(D_1 f(x_1), D_2 f(x_2), \ldots, D_n f(x_n)) \text{ where } f \in \mathbb{P}_k$$

$GRS_r(X, D)$ is an $(n, k, r+1)$-linear code over $\mathbb{F}_{q^m}$, and has a parity-check matrix $H = Vdm(r, X) \times Diag(D)$.

Alternant codes

Alternant codes form a very large class of codes. We will define them and some of their subclasses. Figure 2.4 (taken from [76, page 333]) shows the relationship between these subclasses (it is not drawn to scale). Any linear subspace of $\mathscr{C}$ is said to be a subcode of $\mathscr{C}$.

Definition 2.11 (Subfield-subcode)**.** *If $\mathscr{C}$ is a code over $\mathbb{F}$ and $\mathbb{F}_{SUB}$ is a subfield of $\mathbb{F}$, then the $\mathbb{F}_{SUB}$-subfield subcode of $\mathscr{C}$ is the code consisting of all words of $\mathscr{C}$, which have only entries in $\mathbb{F}_{SUB}$.*

A $\mathbb{F}_{SUB}$-subfield subcode is a $\mathbb{F}_{SUB}$-linear code. Next we discus how to derive the parity check matrix of the subfield subcode (see [76, page 207]):

Let $\mathbb{F} = \mathbb{F}_{q^m}$, $\mathbb{F}_{SUB} = \mathbb{F}_q$ and $H = (h_{ij})$ the parity-check matrix of the code $\mathscr{C}$ defined over $\mathbb{F}_{q^m}$, where $h_{ij} \in \mathbb{F}_{q^m}$ for $1 \leq i \leq r$ and $1 \leq j \leq n$. Pick a basis $(\alpha_1, \ldots, \alpha_m)$ for $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$ and write

$$h_{ij} = \sum_{l=1}^{m} h_{ijl}\alpha_l, \text{ where } h_{ijl} \in \mathbb{F}_q.$$

The parity-check matrix H' of the subfield subcode can be obtained by replacing each element of H by the corresponding column vector $(h_{ij1}, \ldots h_{ijm})^T$ of length m from $\mathbb{F}_q$. Thus

$$H' = \begin{pmatrix} h_{111} & h_{121} & \ldots & h_{1n1} \\ h_{112} & h_{122} & \ldots & h_{1n2} \\ \vdots & \vdots & \ddots & \vdots \\ h_{11m} & h_{12m} & \ldots & h_{1nm} \\ h_{211} & h_{221} & \ldots & h_{2n1} \\ \vdots & \vdots & \ddots & \vdots \\ h_{r1m} & h_{r2m} & \ldots & h_{rnm} \end{pmatrix}.$$

Definition 2.12 (Alternant code)**.** *Let $X = (x_1, \ldots, x_n)$ such that the x_i are pairwise different elements of $\mathbb{F}_{q^m}$ and $D = (D_1, \ldots, D_n)$ where D_i are nonzero elements of $\mathbb{F}_{q^m}$. The Alternant code of order r, denoted $\mathscr{A}_r(X, D)$, is the subfield subcode of the Generalized Reed-Solomon code $GRS_r(X, D)$.*

This means that $\mathscr{A}_r(X, D)$ is the restriction of $GRS_r(X, D)$ to $\mathbb{F}_q$, therefore

$$\mathscr{A}_r(X, D) = \{c \in \mathbb{F}_q^n | Vdm(r, X) \times Diag(D)c^T = 0\}.$$

Theorem 2.5. *[76, page 334] $\mathscr{A}_r(X, D)$ is an (n, k, d)-linear code over $\mathbb{F}_q$ with $n - mr \leq k \leq n - r$ and minimum distance $d \geq r + 1$.*

Fact 1. *[76, page 365] There exists a polynomial time algorithm decoding all errors of Hamming weight at most $\frac{r}{2}$ for an alternant code $\mathscr{A}_r(X, D)$ of order r once a parity-check matrix of the form $H = Vdm(r, X) \times Diag(D)$ is given for it.*

Goppa codes

A subfamily of alternant codes was introduced by Goppa in 1970 [55].

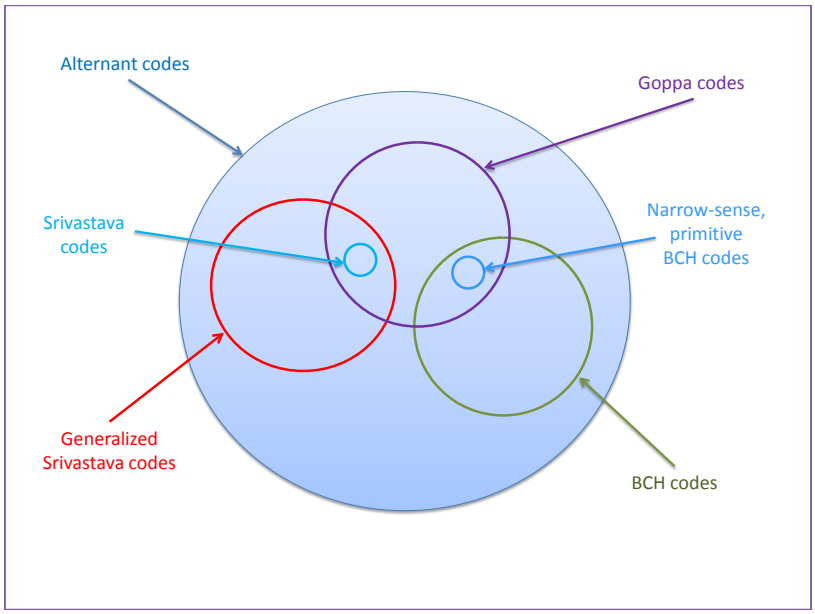

Figure 2.4: Relationship between various subclasses of alternant codes (taken from [76, page 333]).

Definition 2.13 (Goppa code)**.** *Given a sequence* $L = (L_0, \ldots, L_{n-1}) \in \mathbb{F}_{q^m}^n$ *of distinct elements and* $g(x)$ *a polynomial in* $\mathbb{F}_{q^m}[x]$ *of degree* t*, such that* $g(L_i) \neq 0$ *for* $0 \leq i < n$*. The Goppa code* $\Gamma(L, g)$ *over* $\mathbb{F}_q$ *is* $\mathscr{A}_t(L, D)$*, the alternant code over* $\mathbb{F}_q$ *that correspond to the* $GRS_t(L, D)$*, where* $D = (g(L_o)^{-1}, \ldots, g(L_{n-1})^{-1})$*.*

Goppa codes $\Gamma(L, g)$ have dimension $k \geq n - mt$ and minimum distance $d \geq t + 1$ (with t the degree of $g(x)$ and $n = |L|$). Their parity-check matrix is $Vdm(t, L) \times diag(D)$ i.e.,

$$H = \begin{pmatrix} 1 & \ldots & 1 \\ L_1 & \ldots & L_n \\ \vdots & & \vdots \\ L_1^{t-1} & \ldots & L_n^{t-1} \end{pmatrix} \begin{pmatrix} g(L_1)^{-1} & & 0 \\ & \ddots & \\ 0 & & g(L_n)^{-1} \end{pmatrix}.$$

Theorem 2.6. *[115] The Goppa code generated by a monic polynomial* $g(x) = \prod_{i=0}^{t-1}(x - z_i)$ *without multiple zeros admits a parity-check matrix of the form*

$$H = \begin{pmatrix} \frac{1}{z_0 - L_0} & \cdots & \frac{1}{z_0 - L_{n-1}} \\ \vdots & \ddots & \vdots \\ \frac{1}{z_{t-1} - L_0} & \cdots & \frac{1}{z_{t-1} - L_{n-1}} \end{pmatrix}.$$

This means that the Goppa code $\Gamma(L, g)$ consists of all elements $c = (c_0, \ldots, c_{n-1}) \in \mathbb{F}_q^n$ such that for all $j \in \{0, \ldots, t-1\}$

$$\sum_{i=0}^{n-1} \frac{c_i}{z_j - L_i} \equiv 0 \mod g(x). \tag{2.3.1}$$

Goppa codes are alternant codes. So by Fact 1 there exists a decoding algorithm that can correct up to $\frac{t}{2}$ errors. In the case of *binary* Goppa codes, we can correct twice as many errors. This follows from the following theorem [76, page 341].

Theorem 2.7. *A binary Goppa code $\Gamma(L, g)$ associated to a Goppa polynomial $g(x)$ of degree t without multiple roots is equal to the alternant code $\mathscr{A}_{2t}(L, D)$, with $D_i = g(L_i)^{-2}$.*

Fact 2. *There exists a polynomial time algorithm decoding all errors of Hamming weight at most t in a Goppa code $\Gamma(L, g)$ when g has degree t and has no multiple roots, if L and g are known. This algorithm is due to Patterson [94].*

BCH codes

Definition 2.14 (BCH-codes)**.** *A cyclic code of length n over $\mathbb{F}_q$ with generator polynomial $g(x)$ is a BCH code of designed distance δ if, for some integer $b \geq 0$, $g(x)$ is the monic polynomial of lowest degree over $\mathbb{F}_q$ having $\alpha^b, \alpha^{b+1}, \ldots, \alpha^{b+\delta-2}$ as zeros.*

Its parity-check matrix is

$$H = \begin{pmatrix} 1 & \alpha^b & \alpha^{2b} & \ldots & \alpha^{b(n-1)} \\ 1 & \alpha^{b+1} & \alpha^{2(b+1)} & \ldots & \alpha^{(n-1)(b+1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{b+\delta-2} & \alpha^{2(b+\delta-2)} & \ldots & \alpha^{(n-1)(b+\delta-2)} \end{pmatrix}$$

Note that taking $r = \delta - 1$, $D = (1, \alpha^b, \ldots, \alpha^{b(n-1)})$ and $X = (1, \alpha, \alpha^2 \ldots, \alpha^{(n-1)})$,

$$H = Vdm(r, X) \times Diag(D).$$

If $b = 1$ the code is called *narrow sense BCH* code.

Generalized Srivastava codes

Definition 2.15 (Generalized Srivastava codes)**.** *Given a prime power q and $m, s, n, t \in \mathbb{N}$, let $\alpha = (\alpha_1, \ldots, \alpha_n)$, $w = (w_1, \ldots, w_s)$ be $n + s$ distinct elements of $\mathbb{F}_{q^m}$ and $(z_1, \ldots, z_n)$ be non zero elements of $\mathbb{F}_{q^m}$. The Generalized Srivastava code of order st and length n is defined by a parity-check matrix of the form*

$$H = \begin{pmatrix} H_1 \\ H_2 \\ \vdots \\ H_s \end{pmatrix} \tag{2.3.2}$$

where each block is

$$H_i = \begin{pmatrix} \frac{z_1}{\alpha_1 - w_i} & \cdots & \frac{z_n}{\alpha_n - w_i} \\ \frac{z_1}{(\alpha_1 - w_i)^2} & \cdots & \frac{z_n}{(\alpha_n - w_i)^2} \\ \vdots & \vdots & \vdots \\ \frac{z_1}{(\alpha_1 - w_i)^t} & \cdots & \frac{z_n}{(\alpha_n - w_i)^t} \end{pmatrix}. \tag{2.3.3}$$

The Generalized Srivastava codes have length $n \leq q^m - s$, dimension $k \geq n - mst$, minimum distance $d \geq st + 1$ and are alternant codes. The original Srivastava codes are the case $t = 1$ and $z_i = \alpha_i^\mu$ for some μ.

Chapter 3

Code-based cryptography

In 1978 McEliece introduced a public-key cryptosystem [78] informally based on the hardness of decoding random linear codes. In this chapter we will describe it and one of its variants proposed by Niederreiter in 1986 [87]. We will see that for the text-book versions of these PKC there are protocol-based attacks, but they can be avoided by using the CCA2-secure scheme presented by Kobara and Imai [67]. In Section 3.2 we will present the known attacks on McEliece PKC, none of them presents a serious threat (apart for some parameters). In the last section we will introduce a signature scheme based on these cryptosystems.

3.1 McEliece and Niederreiter PKC

3.1.1 McEliece PKC

The main idea of the McEliece cryptosystem is to use a code that has an efficient decoding algorithm and to disguise it, so that it looks like a general instance of the decoding problem. In the original version [78] McEliece uses a (1024, 524, 101)-binary Goppa code. The user picks $u, t \in \mathbb{N}$, such that $n = 2^u$ and $t \lll n$. Then randomly selects an irreducible polynomial g of degree t over $\mathbb{F}_{2^u}$ and chooses a Goppa code $\Gamma(L, g)$ of length n, dimension $k \geq n - ut$ and minimum distance $d \geq t + 1$. We call γ the polynomial time decoding algorithm that can correct up to t errors, and G_0 a $k \times n$ generator matrix of $\Gamma(L, g)$. The user also picks a random $n \times n$ permutation matrix P and a $k \times k$ non-singular matrix S, then computes $G \stackrel{\text{def}}{=} SG_0P$. The public key is (G, t) and the secret key is (S, G_0, P, γ). The encryption and decryption are presented in Algorithm 1 and the PKC is illustrated in Figure 3.1.

In fact $c \times P^{-1} = m \times (SG_0P)P^{-1} + e \times P^{-1} = (m \times S) \times G_0 + e \times P^{-1}$. As P^{-1} does not affect the weight of e, we can apply the decoding algorithm γ and recover mSG_0, then by linear algebra we recover m. In the case of the McEliece's original proposal Canteaut and Chabaud [26] state that *"the row scrambler S has no cryptographic function; it only assures for McEliece's system that the public matrix is not systematic otherwise most of the bits of the plaintext would be revealed"*. This statement is not valid for all the variants, for example in the case of the CCA2-secure scheme presented in [67]. The matrix P is indispensable to hide the algebraic structure of the code.

Algorithm 1 McEliece's encryption and decryption algorithm

- **Encryption:**
  - **Input:** (G, t) and $m \in \mathbb{F}_2^k$.
  - Randomly pick e in $\mathbb{F}_2^n$ of weight t.
  - **Output:** $c = m \times G + e$.
- **Decryption:**
  - **Input:** (S, G_0, P, γ) and $c \in \mathbb{F}_2^n$.
    - Compute $\gamma(c \times P^{-1}) = mSG_0$.
    - Use linear algebra to recover m.
  - **Output:** m.

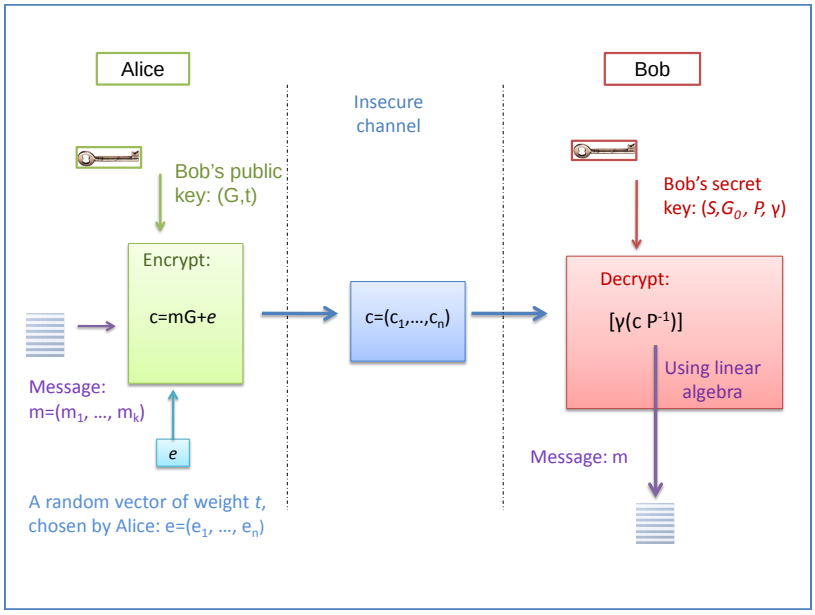

Figure 3.1: Alice sends a message to Bob using McEliece PKC.

3.1.2 Niederreiter PKC

In 1986, Niederreiter proposed a different code-based encryption scheme [87] using GRS codes. The main difference is that instead of representing the message as a codeword, Niederreiter proposed to encode it in the error vector. This cryptosystem describes the code through the parity-check matrix and hides its structure by scrambling and permuting transformations. The encryption algorithm takes as input words of weight t (the number of errors that can be decoded). In 1992 Sidelnikov and Shestakov showed that this proposal is insecure [108]. Nevertheless if we substitute the GRS codes by the Goppa codes we can see that it is the dual variant of the McEliece PKC and it remains a secure cryptosystem. In the following, by "Niederreiter PKC" we refer to the version that uses Goppa codes.

As in the McEliece PKC, the user picks $u, t \in \mathbb{N}$, such that $n = 2^u$ and $t \lll n$. Then she/he randomly selects an irreducible polynomial g of degree t over $\mathbb{F}_{2^u}$ and chooses a Goppa code $\Gamma(L, g)$ of length n, dimension $k \geq n - ut$ and minimum distance $d \geq t + 1$. We call γ the polynomial time syndrome decoding algorithm, and H_0 the $(n-k) \times n$ parity-check matrix of $\Gamma(L, g)$. The user also picks a random $n \times n$ permutation matrix P and a $(n-k) \times (n-k)$ non-singular matrix S, then she/he computes $H \stackrel{\text{def}}{=} SH_0P$. The public key is (H, t) and the secret-key is (S, H_0, P, γ). The encryption and decryption are presented in Algorithm 2.

Algorithm 2 Niederreiter's encryption and decryption algorithm

- **Encryption:**
  - **Input:** (H, t) and $m \in \mathbb{F}_2^n$ of weight less or equal than t.
  - **Output:** $c = H \times m^T$.
- **Decryption:**
  - **Input:** (S, G_0, P, γ) and $c \in \mathbb{F}_2^{n-k}$.
  - **Output:** $m = P^{-1} \times [\gamma(S^{-1} \times c)]$.

In fact $S^{-1} \times c = S^{-1}(SH_0P) \times m^T = H_0(P \times m^T)$. As P does not affect the weight of m, we can apply the syndrome decoding algorithm γ and recover $P \times m^T$. Then by multiplying with P^{-1} we recover m^T.

The disadvantage is that the message has to be encoded into an error vector by a function $\phi_{n,t} : \{0,1\}^l \to W_{2,n,t}$ where $l = \lfloor log_2 \binom{n}{t} \rfloor$ and $W_{2,n,t}$ denotes the words of $\mathbb{F}_2^n$ of weight t. In the Algorithm 3 we will see how to build $\phi_{n,t}$, we present a corrected version of the algorithm presented in [93, page 99].

The inverse is easy to define (assuming $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ where $n \geq k$ and 0 otherwise):

$$\phi_{n,t}^{-1}(e) = \sum_{i=1}^{n} \left(e_i \times \binom{i-1}{\sum_{j=1}^{i} e_j} \right) + 1.$$

Algorithm 3 $\phi_{n,t} : \{0,1\}^l \to W_{q,n,t}$

Input: $x \in \{0,1\}^l$, n and t.
Output: a word $e = (e_1, e_2, \cdots, e_n)$ of weight t and length n.

1. $c \leftarrow \binom{n}{t}$, $c' \leftarrow 0$, $j \leftarrow n$, $t' \leftarrow t$ and $i \leftarrow$ the natural number that represent x in decimal base.
2. **while** $j > 0$ do
   - $c' \leftarrow c \times \frac{j-t'}{j}$
     - **if** $i \leq c'$ then $e_j \leftarrow 0$ and $c \leftarrow c'$
     - **else** $e_j \leftarrow 1$, $i \leftarrow i - c'$, $c \leftarrow c \times \frac{t'}{j}$ and $t' \leftarrow t' - 1$
   - $j \leftarrow j - 1$

This algorithm is quite inefficient and has complexity $\mathcal{O}(n^2 log_2 n)$, in [93] Sendrier and Overbeck discuss more efficient alternatives.

The security of the McEliece and the Niederreiter PKC is equivalent. An attacker who can break one is able to break the other [74]. But for given parameters, the Niederreiter cipher presents many advantages. First of all the public key can be in a systematic form (reducing the key size, since it is sufficient to store the redundant part of the matrix H) without any cost of security whereas this would reveal a part of the plaintext in the McEliece system. The public key in the Niederreiter system is then $n/(n-k)$ times smaller than in the McEliece version (since the public key in the Niederreiter system has $(n-k) \times k$ bits and in the McEliece system it has $n \times k$). The systematic form of the public matrix H and the low-weight of vector m significantly reduce the computational cost involved in the encryption in Niederreiter's version.

3.1.3 Protocol-based attacks

All the attacks presented in this section require either additional information, such as partial knowledge on the target plaintexts, or a decryption oracle which can decrypt arbitrarily given ciphertexts except the challenged one.

Known-partial-plaintext attacks

The partial knowledge of the target plaintext reduces the computational cost of the attacks on the McEliece PKC [28], [66]. For example, let m_l denote the first k_l bits of m and m_r the last bits, where $k = k_l + k_r$ and $m = (m_l \| m_r)$. We suppose that the adversary knows m_r, we have that

$$c = mG + e, \text{ then}$$

$$c = m_l G_l + m_r G_r + e$$

where G_l and G_r are the the upper k_l rows and the remaining k_r rows of G respectively, then

$$c + m_r G_r = m_l G_l + e.$$

If k_l is a small enough to use the information-set-decoding attacks (see Section 3.2.1) in polynomial time, the computational cost to recover the missing values of m is polynomial on n.

Related-message attack

Let m_1 and m_2 be two plaintexts, $c_1 = m_1 G + e_1$, $c_2 = m_2 G + e_2$ and $e_1 \neq e_2$. In this attack we assume that the adversary knows $\delta m = m_1 + m_2$ [19]. Note that

$$\delta m G + c_1 + c_2 = (m_1 + m_2)G + (m_1 G + e_1) + (m_2 G + e_2) = e_1 + e_2.$$

The adversary chooses k coordinates whose values are 0 in $(\delta m G + c_1 + c_2)$ (in these coordinates e_1 and e_2 have high probability to be 0) and apply information-set-decoding attacks (see Section 3.2.1) to either c_1 or c_2, it is very likely that he can recover m_1 and m_2. If the same message is encrypted twice (or more), the difference between e_1 and e_2 is $c_1 + c_2$, this case is referred as *message-resend attack* [19].

Reaction attack

In this attack [57], the adversary flips one or a small number of bits of the target ciphertext c, we denote c' the flipped bit. He sends c' to the receiver that has the private key and observes his reaction. They are two possible reactions:

- Reaction A: he repeats the request to the adversary due to an uncorrectable error or due to the meaningless plaintext.
- Reaction B: he returns an acknowledgment or does nothing since the proper plaintext m is decrypted.

The reaction B will happen if there are still less than t errors in c', otherwise we will have the reaction A. Repeating this process a polynomial number of times on n the adversary can obtain the error vector.

Adaptive chosen-ciphertext attack (CCA2)

The attacker knows c and wants to find m such that $c = mG + e$. She/he has access to a decryption oracle which provide her/him with ciphertext-plaintext pairs of her/his choice (except for c). We say that a cryptosystem is secure against *adaptive ciphertext attack (CCA2 secure)* if such attacker has no advantage in deciphering a given ciphertext. In the McEliece case, the attacker can generate a new ciphertext $c' = c + m'G = (m + m')G + e$, ask the oracle to give her/him back $c' = m + m'$ and then she/he is able to recover m.

We can see that McEliece PKC is not secure against adaptive chosen-ciphertext attacks. However, Imai and Kobara [67] proposed in 2001 a CCA2-secure version that prevent all the attacks described in this section. An overview can be found in [39].

3.2 Attacks on McEliece Cryptosystem

There are mainly two guidelines to attack McEliece cryptosystem:

1. Decoding attacks: decode the public code which has no visible structure. Attack a single ciphertext using a generic decoding algorithm (independent od the inner code).

2. Structural attacks: recover the original structure of the secret code from the public key.

There are also some side-channel attacks approaches [7,106,112], but we will focus on the first two kinds of attack. We use the same notation as before, G is a $k \times n$ generator matrix of a (n, k, d)-linear code, $c = mG + e$ is the encrypted word and t is the error correcting capability.

3.2.1 Decoding attacks

Assume that the trapdoor is sufficiently well hidden. We want to correct errors in a linear code for which we only know the generator (or the parity-check) matrix.

The most effective attacks known against the McEliece and Niederreiter cryptosystems are derived from Information-Set Decoding. The idea was proposed by McEliece in his original paper [78] and there are many variants. An information set of an (n, k, d)-linear code, with generator matrix G is the set I of k elements of $\{1, 2, \ldots, n\}$ such that the set of columns of G indexed by I form a $k \times k$ invertible submatrix of G, denoted G_I.

Let us choose an information set I and the columns of c and e restricted to I (denoted by c_I and e_I), we then have that $c_I = mG_I + e_I$. The main idea of the attack is that if $e_I = 0$ as G_I is non-singular, m can be recovered by Gaussian elimination. This is called *plain information-set-decoding*.

In 1988 Lee and Brickell proposed to allow a very small number of errors ($0 \leq p \leq t$) in the selected e_I [71]. In Algorithm 4 we can see the main idea of the attack.

We call *generalized information-set-decoding attack* when $e_I \neq 0$. Leon [72] proposed an improvement by looking for codewords containing zeros in a windows of size s. In 1989 Stern proposed [109] to divide the information set in two parts, allowing to speed-up the search for codewords with zeros in the window by a birthday attack technique. Other improvements have been proposed, see for example the following papers: Canteaut and Chabaud [27], Bernstein et al. [16], Finiasz and Sendrier [47] and Bernstein et al. [18]. All variants look for specific error patterns as shown in Figure 3.2 (taken from [93] and from [18]). In the figure we find the distribution of the non-zero elements in the error vector and the number inside the boxes is the Hamming weight of the corresponding segment.

Algorithm 4 Information set decoding (for parameter p)

- **Input:** a $k \times n$ matrix G, the received word c, the error correcting capability t and a parameter p such that $0 \leq p \leq t$.
- **Output:** a vector $\epsilon \in \mathbb{F}_2^n$, with $wt(\epsilon) \leq t$ such that $c - \epsilon \in \mathscr{C}$ if such ϵ exists.
- 1. Choose an information set I and compute $E \stackrel{\text{def}}{=} c + c_I G_I^{-1} G$
  2. List all the possible errors e'_j of Hamming weight $\leq p$ in the se I. For each of this vectors compute
  $$\epsilon \stackrel{\text{def}}{=} E + e'_j G_I^{-1} G$$
     - **IF** $wt(\epsilon) \leq t$, RETURN ϵ and STOP.
     - **ELSE** try another vector of the list.
  3. Go back to step 1.

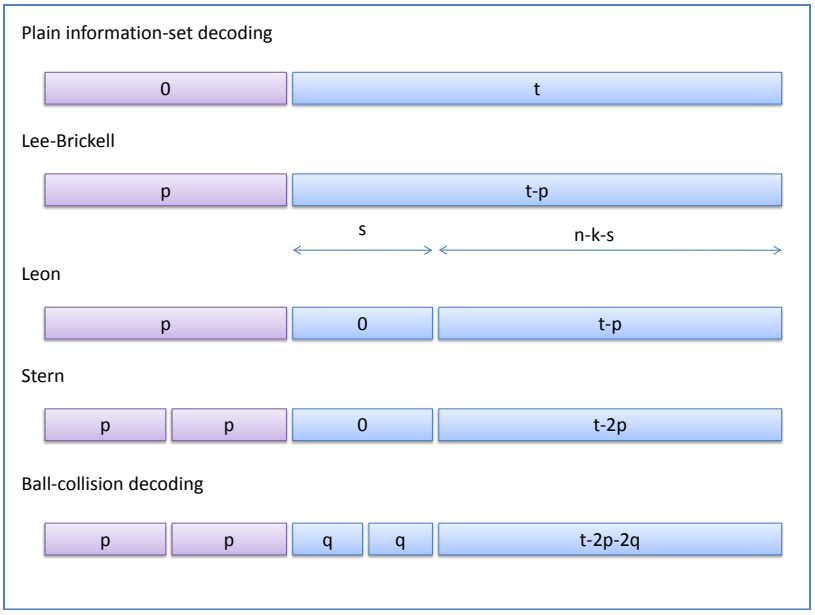

Figure 3.2: Distribution of the non-zero elements in the error vector (the number inside the boxes is the Hamming weight of the corresponding segment).

McEliece's original parameters $(n, k, t) = (1024, 524, 50)$ are not secure any more, in Table 3.1 we can see the year, the method and the work factor of some attacks based on information set decoding for this parameter. Ball-collision decoding [18] is asymptotically faster than the algorithm proposed by Finiasz and Sendrier [47].

Table 3.1: Year, method and work factor of the different attacks on McEliece original parameters $(n, k, t) = (1024, 524, 50)$.

Year	Method	Work factor
1998	Canteaut and Chabaud [27]	$2^{64.2}$
2008	Bernstein, Lange and Peters [16]	$2^{60.5}$
2009	Finiasz and Sendrier [47]	$2^{59.9}$

This kind of attacks does not destroy the McEliece cryptosystem but may help in the choice of secure parameters. We consider a parameter choice b-bit secure if it takes at least 2^b bit operations to decrypt a single ciphertext using information-set decoding techniques. In Table 3.2 we can find some parameters for various security levels in CCA2-secure variants of the McEliece's cryptosystem taken from [16] and the public key size of the RSA cryptosystem for those security levels [1].

Table 3.2: Parameters for various security levels in CCA2-secure variants of the McEliece's cryptosystem. Where (n, k) are the length and dimension of the Goppa codes and t is the number of errors than can be corrected. And the public key size for McEliece (PK-McEliece) and RSA (PK-RSA) cryptosystems.

Security level	(n, k)	t	PK-McEliece	PK-RSA
80-bit	$(2048, 1751)$	27	520047	1248
128-bit	$(2960, 2288)$	56	1537536	3248
256-bit	$(6624, 5129)$	115	7667855	15424

3.2.2 Structural attacks

In the original paper McEliece proposed to choose the code amongst the irreducible binary Goppa codes. With this choice (changing the parameters) no efficient algorithm has been discovered yet for decomposing G into (S, G_0, P). Considering for the secret key $\Gamma(L, g)$ a t-error correcting binary irreducible Goppa codes of length $n = 2^u$ over $\mathbb{F}_{2^u}$, it is composed by

- a *generator*, a monic irreducible polynomial $g(z)$ of degree t over $\mathbb{F}_{2^u}$ and
- a *support*, a vector $L \in \mathbb{F}_{2^u}^n$ with distinct coordinates.

If we know one of the two components we can find the other in a polynomial time from the public key G:

1. If we know the support L, we can obtain $g(z)$ using some codewords from G and Equation 2.3.1 on page 17. After computing a few gcd's (usually one is enough) the generator polynomial is obtained.

2. If we have the generator polynomial, we can construct a generator matrix G' of the Goppa code $\Gamma(L_0, g(z))$ where L_0 is fixed and chosen arbitrarily. We can then obtain L by applying to G' and G the support splitting algorithm due to Sendrier [101].

In both cases, we need an exhaustive search attack, either by enumerating the irreducible polynomials or the permutations. The first case is the best known structural attack on McEliece's encryption scheme due to Loidreau and Sendrier [75]. They give a structural attack that reveals part of the structure of a "weak" G which is generated from a "binary" Goppa polynomial. This attack can be avoided if we do not use such weak public keys (this implies G_0 should not be a BCH code). The second case is Gibson's attack: find an equivalent Goppa code of G (which is not necessarily G_0) such that we know a decoding algorithm for it. In [2] and [53] is shown that the probability of this case is negligibly small.

In Chapter 5 we will present two structural attacks on two variants of the McElice's cryptosystem.

3.3 CFS signature scheme

The size of the keys in the McEliece PKC is one of the reasons to prefer the RSA PKC. Another main disadvantage was the belief that McEliece could not be used to sign. A digital signature is a small size piece of information, that depends on the message and the signer. It needs an algorithm to compute a signature for any message (such that the desired person is the only one that is able to sign) and a fast public verification algorithm. Consider the following public-key cryptosystem: Let $\mathcal{X}$ be the set of plain texts, $\mathcal{Y}$ the set of ciphertexts, $e : \mathcal{X} \to \mathcal{Y}$ the encryption function, $d : \mathcal{Y} \to \mathcal{X}$ the decryption function (such that $d \circ e = id$) and $\mathcal{M}$ the set of messages. Now let $h : \mathcal{M} \to \mathcal{Y}$ a one-way public collision-resistant hash function (i.e., it is hard to find m and m' in $\mathcal{M}$ such that $h(m) = h(m')$). A signature scheme can be built from these PKC:

- The signature of the message $m \in \mathcal{M}$ is $\sigma = d(h(m))$.
- For the verification, we just have to apply the encryption function to σ and check if $e(\sigma) = h(m)$.

If we assume that h is independent from e and d, then the computation of $d(h(m))$ is as hard as the computation of $d(y)$ for any $y \in \mathcal{Y}$. If we assume now that the PKC is based on error correcting codes, there is a problem in the computation of $d(h(m))$, in fact it is very likely that $h(m)$ is not a codeword unless it is explicitly produced as an output of the encryption function, i.e., for any $m \in \mathcal{M}$, $h(m) = e(g(m))$ for a function g that should be secret. In this case $g = d \circ h$ and thus h is not independent from e and d.

In 1990, Xinmei Wang proposed the first digital signature scheme based on error-correcting codes [121], two years later Harn and Wang [58], Alabbadi and Wicker [3]

and van Tilburg [116] showed that it was not a secure scheme. They also proposed some modifications, but all of them have been attacked [4–6, 117]. In 1993 Stern proposed an identification scheme based on the syndrome decoding problem [110]; a dual version (using generator matrix) was proposed by Véron [119]. In 1997 Kabatianskii et al. proposed a signature scheme [63]. Its security and the efficiency have been studied in [29, 88]. The first practical code-based signature scheme [34] came out almost twenty years after McEliece's proposal. It is due to Courtois, Finiasz and Sendrier and is called CFS signature scheme. The difference between encryption and this signature scheme lies in the choice of the parameters of the binary Goppa codes. For signature the Goppa codes have to correct very few errors i.e., they have a very high rate.

Let e and d be the encryption and decryption functions of Niederreiter cryptosystem, with a binary irreducible $[n = 2^u, k = n - ut, 2t + 1]$-Goppa code. And let h be a collision resistant hash function, from an arbitrarily long binary sequence to a value in $\{0,1\}^{n-k}$. The algorithm to produce the CFS signature is described in Algorithm 5.

Algorithm 5 CFS signature scheme

Input: The encryption (e) and the decryption (d) functions, the message m and the hash function h.

Output: σ, the signature of m.

1. $\forall i \geq 0$, let $y_i = h(m||i) \in \{0,1\}^{n-k}$.
2. Let i_0 the smaller integer such that y_{i_0} can be decoded, we denote $x_{i_0} = d(y_{i_0})$.
3. $\sigma = (x_{i_0}, i_0)$.

Verification: Check if $e(x_{i_0})$ and $h(m||i_0)$ are equals.

Overbeck and Sendrier state in [93, page 101] that " The average number of attempts needed to reach a decodable syndrome can be estimated by comparing the total number of syndromes to the number of efficiently correctable syndromes:

$$\frac{\sum_{i=0}^{t} \binom{n}{t}}{2^{n-k}} \simeq \frac{n^t/t!}{n^t} = \frac{1}{t!}."$$

Since by the definitions of the parameters of the Goppa code, $2^{n-k} = 2^{ut} = n^t$ and as $t \lll n$, we have $\sum_{i=0}^{t} \binom{n}{t} \simeq \binom{n}{t} \simeq \frac{n^t}{t!}$.

In Table 3.3 (from [34]) we can see the characteristics of the signature scheme based on a $(n = 2^u, k = n - tu, d \geq 2t + 1)$-binary Goppa code.

As proven in [34] CFS signature scheme security can be reduced to the syndrome decoding problem and the distinguishability of binary Goppa codes from a random code. However in Chapter 6, we will see that in joint work with Jean-Charles Faugère, Ayoub Otmani, Ludovic Perret and Jean-Pierre Tillich [43], we solve the second problem in the range of parameters used in the CFS signature scheme. This is not an attack in the system, but it invalidates the hypotheses of the security proof.

Table 3.3: Characteristics of the signature scheme based on a $(n = 2^u, k = n - tu, d \geq 2t + 1)$ binary Goppa code

Signature cost	$t!t^2u^3$
Signature length	$(t-1)u + log_2 t$
Verification cost	t^2u
Public key size	$tu2^u$

An attack on this signature scheme due to Bleichenbacher is explained in [47] and consists of forging a valid signature for a chosen message. It is based in the General Birthday Attack and made the original CFS parameters insecure. Finiasz state in [46] that " for a given security level, resisting Bleichenbacher's attack only requires a small parameter increase, but this small increase can have a heavy impact on the signature time or the public key size". In 2010 [46] Finiasz proposed a modification to the CFS signature scheme, called *parallel CFS*, that produce more than one CFS signature in parallel for the same message and that resists Bleichenbacher's attack. There are other recent proposals based on quasi-dyadic codes, like Barreto's et al. [10] and Kobara's [65].

We can also construct other cryptographic primitives, such as random numbers generators, hash functions and identification schemes based on coding theory. We are not going to introduce them but an overview can be found in [93].

Chapter 4

McEliece Variants

The main drawback of McEliece PKC is the huge size of the public and the secret keys (several hundred thousand bits in general). The secret key may be reduced by using a pseudo-random number generator, save only the seed and recompute everything each time. In this chapter we will see some examples of McEliece variants, using other codes than the Goppa codes, that have been proposed to reduce the key size. Some of the codes that will mention are defined in Section 2.3; the others are defined in Appendix B.

In the last section we will introduce two McEliece variants: one is based on quasi-cyclic alternant codes proposed by Berger et al. [11] and the other variant is based on quasi-dyadic matrices proposed by Barreto and Misoczki [85]. We will see that both papers follow a very similar approach and that the reduction (in comparison to classical Goppa codes) in the key size of both schemes is impressive. In Chapter 5 we will present two independent attacks [44, 52] of these two variants.

In Chapter 3 we saw that the main idea of the McEliece cryptosystem is to choose a code (given by a generator matrix G) that is easy to decode, and modify G in order to hide its structure. In the original scheme McEliece proposed to set $G' \stackrel{\text{def}}{=} SGP$, but there are other ways to do this modification. In [93, page 100] Overbeck and Sendrier regroup the main strategies used to hide the structure of a code. We may combine these strategies but this have to be done carefully since it may lead to an insecure cryptosystem.

4.1 Use other families of linear codes

If we change the family of linear codes used in the McEliece PKC we may find a shorter key. If we for example use Reed-Solomon codes, we have to only remember the generator polynomial and we can recreate the generator matrix from it. However, such modifications can make the McElice PKC vulnerable to structural attacks. The family of linear codes have to fulfill the following requirements:

1. Avoid enumeration: one should avoid the attack that consists in enumerating all the codes in the family until a code equivalent to the public code is found. This can be done by using Sendrier's support-splitting algorithm [101]. This algorithm determines if two generator matrices correspond to equivalent codes and then recovers the permutation.

2. Neither the generator nor the parity-check matrix of a permutation equivalent code should give any information about the structure of the secret code.

The second item dismisses many families of codes, for example the GRS, the concatenated and the product codes.

Generalized Reed-Solomon codes (GRS): The structure of a permuted GRS code can be easily recovered solving a linear system. The attack was presented by Sidelnikov and Shestakov [108] in the cryptographic context, but the main idea was previously introduced in [99]. The attack is based on the following property that relate the parity-check matrix of a GRS code with its systematic form.

Proposition 4.1. *Given $Y = (y_1, \ldots, y_n)$ a sequence of distinct elements in $\mathbb{F}_q^n$ and $\alpha = (\alpha_1, \ldots, \alpha_n)$ a sequence of non-zero elements in $\mathbb{F}_q^n$, the systematic form of the parity-check matrix ($H \stackrel{def}{=} Vdm(n, Y) \times diag(\alpha)$) is*

$$H' = \left(\begin{array}{ccc|ccc} 1 & & & R_{1,r+1} & \ldots & R_{1,n} \\ & \ddots & & \vdots & \ddots & \vdots \\ & & 1 & R_{r,r+1} & \ldots & R_{r,n} \end{array}\right)$$

with

$$R_{i,j} = \frac{y_i}{y_j} \prod_{\substack{l=1 \\ l \neq i}}^{r} \frac{\alpha_j - \alpha_l}{\alpha_i - \alpha_l} \text{ for } 1 \leq i \leq r \text{ and } r \leq j \leq n.$$

Reducing the public-key matrix and using the fact that the systematic form is unique we can find a system of equations such that the unknown variables are y_i and α_i. The symmetries that we find in the expressions of $R_{i,j}$ allow us to write a linear system which can be solved in polynomial time.

Concatenated codes: These codes seemed to be a good candidate for replacing Goppa codes, since they have an algorithm to decode really fast. In practice they can decode (i.e., with a probability near to 1) many more errors than the half of the minimum distance. Unfortunately there is a structural attack against this family of codes presented in [100, 102]. The attack is based on the existence of codewords of small weight in the dual of the concatenated code. We can then find in a polynomial time a concatenated structure equivalent to the initial one. This is not enough to decode $\mathscr{C}$ but gives a lot of information about the structure of the code.

Product codes: These codes have the same property as the previous codes (there exist a lot of codewords of small weights in the dual code). The same attack (as for concatenated codes) can be adapted to this family of codes.

Quasi-cyclic codes: This family of codes has the advantage of having a very simple and compact description. Gaborit [51] first proposed to use them, Baldi and Chiaraluce did another proposal [9], but both have been attacked [89]. In the following section we will introduce a variant introduced by Berger et al. [11] using quasi-cyclic alternant codes. However, it has also been attacked. In Chapter 5 we will present two independent attacks [44, 52].

Reed Muller codes: These codes are one of the oldest and most studied families of codes. In 1994 Sidelnikov proposed to use them for cryptography [107]. And was attacked in 2007 by Minder and Shokrollahi [84].

Rank metric (Gabidulin) codes: These codes are a subclass of Srivastava codes, their minimum distance is $d = n - k + 1$ and there exists an efficient decoding algorithm [50]. They were proposed to be used in the McEliece PKC instead of the Goppa codes in several proposal like [49, 50, 90], however all these variants proved to be insecure [91, 92].

There are some codes that may still be used (for some parameters):

Generalized concatenated codes: In this case it is possible to build codes such that the dual distance is big enough, so the previous attack is not efficient in this case. The construction of these codes is possible, but they will not have the same property that the non generalized codes, and so they will not give the same advantage as the others. It may be interesting to use this kind of codes in cryptography.

Algebraic geometry codes: Proposed by Janwa and Moreno [61], they are broken by generalizing the attack of Sidelnikov and Shestakov [45, 77, 83]. The status is unknown for the algebraic geometry codes with a subfield subcode construction.

LDPC: Another idea is to use very sparse matrices. In [104] Shokrollahi et al. proposed to use Low Density Parity-Check (LDPC) codes, but they showed that it is not a secure solution. In 2007 Baldi and Chiaraluce proposed to use quasi-cyclic LDPC codes [9], but Otmani et al. developed an attack [89] that is able to recover the secret key with very high probability. In 2008 Baldi et al. proposed a new version of the cryptosystem that resists this attack [8].

Quasi-dyadic codes: In 2009 Barreto and Misoczki proposed to use quasi-dyadic codes [85], will be explain this variant in the next section. It has been attacked for almost all proposed parameters. In Chapter 5 we will describe two independent attacks [44, 52].

Srivastava: Persichetti [95] proposed a very similar variant to the one in [85] in 2011, which uses Srivastava codes. The attack presented in [52] cannot be applicable

in this case. In the paper, the author says that this variant is secure against the attack presented in [44].

Wild Goppa codes: Bernstein et al. propose to take a subclass of Goppa codes which can correct more errors than the classical case for large fields [17].

4.2 Quasi-cyclic and quasi-dyadic variants of McEliece PKC

In this section we present two McEliece variants: one uses quasi-cyclic alternant codes by Berger et al. [11] and the other uses quasi-dyadic matrices by Barreto and Misoczki [85]. In the following description the notation will differ from the one in [11, 85]. This is an inconvenience necessary in order to unify the description and to be able to apply the attack (that we will introduce in Chapter 5).

4.2.1 Notation

Let r, m be integers and let $q = 2^r$. We denote by $\mathbb{F}_q$ the finite field with q elements and by $\mathbb{F}_{q^m}$ its extension of degree m. In most of the cases we will consider the case $m = 2$ and we stick to this until otherwise stated. For an element $x \in \mathbb{F}_{q^2}$ we denote its conjugate x^q by $\overline{x}$. Given an $\mathbb{F}_q$ basis $(1, \omega)$ of $\mathbb{F}_{q^2}$ we denote by $\psi : \mathbb{F}_{q^2} \to \mathbb{F}_q^2$ the vector space isomorphism such that $\psi(x) = \psi(x_0 + \omega x_1) = \binom{x_1}{x_0}$. Note that, without loss of generality, we can choose θ such that $\psi(x) = \binom{\phi(x)}{\phi(\theta x)}$ where $\phi(x) = x + \overline{x}$ with $\overline{x} = x^q$. Note that we have the identity

$$\phi(\overline{x}) \quad = \quad \phi(x). \tag{4.2.1}$$

A fact that we will use at several instances later is that given $a = \phi(\alpha x)$ and $b = \phi(\beta x)$ for some $\alpha, \beta, x \in \mathbb{F}_{q^2}$ we can recover x as linear combination of a and b (as long as α, β form an $\mathbb{F}_q$ basis of $\mathbb{F}_{q^2}$). More precisely it holds that

$$x \quad = \quad \frac{\overline{\alpha}}{\beta\overline{\alpha} + \overline{\beta}\alpha} b + \frac{\overline{\beta}}{\beta\overline{\alpha} + \overline{\beta}\alpha} a \tag{4.2.2}$$

All vectors are row vectors and they are right multiplied by matrices. The i.th component of a vector x is denote by $x^{(i)}$. Let x_i, c_i two sets of elements in $\mathbb{F}_{q^2}$ of size n and $t \in \mathbb{N}$. Both variants have a secret key parity-check matrix of the form:

$$H = \begin{pmatrix} \phi(c_0) & \phi(c_1) & \dots & \phi(c_{n-1}) \\ \phi(\theta c_0) & \phi(\theta c_1) & \dots & \phi(\theta c_{n-1}) \\ \vdots & \vdots & & \vdots \\ \phi(c_0 x_0^{t-1}) & \phi(c_1 x_1^{t-1}) & \dots & \phi(c_{n-1} x_{n-1}^{t-1}) \\ \phi(\theta c_0 x_0^{t-1}) & \phi(\theta c_1 x_1^{t-1}) & \dots & \phi(\theta c_{n-1} x_{n-1}^{t-1}) \end{pmatrix} = \begin{pmatrix} \mathrm{sk}_0 \\ \vdots \\ \mathrm{sk}_{2t-1} \end{pmatrix} \tag{4.2.3}$$

To simplify the notation later we denote by sk_i the i.th row of H. The public key in both variants is

$$\textbf{(public key)} \quad P \stackrel{\text{def}}{=} SH, \tag{4.2.4}$$

where S is a secret invertible $2t \times 2t$ matrix. Actually, in both schemes P is defined to be the systematic form of H, which leads to a special choice of S. As we do not make use of this fact for the attacks one might as well consider S as a random invertible matrix. In both cases, without loss of generality c_0 and x_0 can be supposed to be 1. In fact, given that the public key H is not uniquely defined, we can always include the corresponding divisions needed for this normalization into the matrix S. The main difference between the two proposals is the choice of the constants c_i and the points x_i. In order to reduce the public key and the secret key size, those $2n$ values are chosen in a highly structured way. Both schemes use random block-shortening of very large private codes (exploiting the $\mathcal{NP}$-completeness of distinguishing punctured codes [120]) and the subfield subcode construction (to resist the classical attack of Sidelnikov and Shestakov, see [108]). In [11, 85] the authors analyze the security of their schemes and demonstrate that none of the known attacks can be applied. They also prove that the decoding of an arbitrary quasi-cyclic (resp. an arbitrary quasi-dyadic) code is $\mathcal{NP}$-complete. For the subfield subcode construction, both schemes allow in principle any subfield to be used. However the most interesting case in terms of key size and performance is the case when the subfield is of index 2 (i.e., $m = 2$) and we focus on this case only. Both schemes use a block based description of the secret codes. They take b blocks of ℓ columns and t rows. The subfield subcode operation will transform each block into a $2t \times \ell$ matrix and the secret parity-check matrix H is the concatenation of the b blocks. Thus, one obtains a code of length ℓb.

4.2.2 The quasi-cyclic variant

Berger et al. propose [11] to use quasi-cyclic alternant codes over a small non-binary field. Let α be a primitive element of $\mathbb{F}_{q^m}$ and $\beta \in \mathbb{F}_{q^m}$ an element of order ℓ (those are public values). The secret key consists of b different values y_j and a_j in $\mathbb{F}_{q^m}$ where b is small, i.e., $b \leq 15$ for the proposed parameters. The constants c_i and points x_i are then defined by

$$c_{\ell j+i} := \beta^{is} a_j \qquad \text{and} \qquad x_{\ell j+i} := \beta^i y_j \tag{4.2.5}$$

for all $0 \leq i \leq \ell - 1$ and $0 \leq j \leq b - 1$. Here $1 \leq s \leq \ell - 1$ is a secret value. Table 4.1 lists the parameters proposed in [11]. Note that in [11] cyclic shifts (modulo ℓ) of the columns are applied. This does not change the structure of the matrix (since β has order ℓ) and that is why we can omit this from our analysis.

4.2.3 The quasi-dyadic variant

Barreto and Misoczki propose [85] to use binary Goppa codes in dyadic form. They consider (quasi) dyadic Cauchy matrices as the parity-check matrix for their code.

Table 4.1: Parameters proposed in [11].

	q	q^m	ℓ	t	b	Public key size (bits)
I			51	100	9	8160
II			51	100	10	9792
III	2^8	2^{16}	51	100	12	13056
IV			51	100	15	20400
V			75	112	6	6750
VI	2^{10}	2^{20}	93	126	6	8370
VII			93	108	8	14880

However, it is well known that Cauchy matrices define generalized Reed Solomon codes in field of characteristic 2 [85] and thus, up to a multiplication by an invertible matrix which we consider to be incorporated in the secret matrix S, the scheme has a parity-check matrix of the form (4.2.3).

Again, the only detail to be described here is how the constants c_i and points x_i are chosen. First we choose $\ell = t$ a power of two. Next, choose $v = [\mathbb{F}_{q^m} : \mathbb{F}_2] = mr$ elements in $\mathbb{F}_{q^m}$: $y_0, y_1, y_2, y_4, \cdots, y_{2^v}$. For each $j = \sum_{k=0}^{v} j_k 2^k$ such that $j_k \in \{0, 1\}$ (i.e., the binary representation of j) we define

$$y_j = \sum_{k=0}^{v} j_k y_{2^k} + (wt(j) + 1)y_0 \tag{4.2.6}$$

for $0 \leq j \leq \#\mathbb{F}q^m - 1$ and $wt(j)$ is the Hamming weight of j. Moreover, choose b different elements k_i with $0 \leq i \leq \#\mathbb{F}_{q^m} - 1$, b different elements $a_i \in \mathbb{F}_{q^m}$ and define

$$x_{\ell i+j} := y_{k_i \oplus j} \qquad \text{and} \qquad c_{\ell i+j} := a_i \tag{4.2.7}$$

for all $0 \leq j \leq \ell - 1$ and $0 \leq i \leq b - 1$. This choice implies the following identity. For $j = \sum_{f=0}^{u-1} j_f 2^f$, where $u = \log_2(\ell)$ it holds that

$$x_{\ell i+j} = \sum_{f=0}^{u-1} j_f x_{\ell i+2^f} + (wt(j) + 1)x_{li}. \tag{4.2.8}$$

Note that in [85] dyadic permutations are applied. However, this does not change the structure of the matrix and that is why we can omit this from our analysis. Table 4.2 lists the parameters proposed in [85, Table 5].

Table 4.2: Sample parameters from [85].

q	q^m	ℓ	t	b	public key size (bits)
2^8	2^{16}	128	128	4	4096
		128	128	5	6144
		128	128	6	8192
		256	256	5	12288
		256	256	6	16384

Chapter 5

Attacks on two McEliece variants

In this chapter, we will present two independent attacks on the two McEliece variants proposed in [11] and [85] (described in Section 4.2). The first four sections give the attack presented by Gauthier-Umaña and Leander in [52], and the last section introduces the independent attack proposed by Faugère et al. in [44].

5.1 General framework of the attack

The starting observation for our analysis and attacks is the following interpretation of the entries in the public key P.

Proposition 5.1. *Let H be the $2t \times n$ parity-check matrix defined as in Equation (4.2.3). Multiplying H by a $2t \times 2t$ matrix S we obtain a $2t \times n$ matrix P of the form*

$$P = SH = \begin{pmatrix} \phi(c_0 g_0(x_0)) & \phi(c_1 g_0(x_1)) & \dots & \phi(c_{n-1} g_0(x_{n-1})) \\ \phi(c_0 g_1(x_0)) & \phi(c_1 g_1(x_1)) & \dots & \phi(c_{n-1} g_1(x_{n-1})) \\ \vdots & \vdots & & \vdots \\ \phi(c_0 g_{2t-1}(x_0)) & \phi(c_1 g_{2t-1}(x_1)) & \dots & \phi(c_{n-1} g_{2t-1}(x_{n-1})) \end{pmatrix}$$

where g_i are polynomials with coefficients in $\mathbb{F}_{q^2}$ of degree less than t. Moreover, if S correspond to a bijective mapping, the polynomials g_i form an $\mathbb{F}_q$ basis of all polynomials of degree at most $t-1$.

Proof of Proposition 5.1. It is enough to consider the effect of multiplying a vector $s \in \mathbb{F}_q^t$ by H. For convenience we label the coordinates of s as

$$s = (\alpha_0, \beta_0, \alpha_1, \beta_1, \dots, \alpha_{t-1}, \beta_{t-1})$$

We compute

$$\begin{aligned} sH &= s \begin{pmatrix} \phi(\theta c_0) & \dots & \phi(\theta c_{n-1}) \\ \phi(c_0) & \dots & \phi(c_{n-1}) \\ \vdots & & \vdots \\ \phi(\theta c_0 x_0^{t-1}) & \dots & \phi(\theta c_{n-1} x_{n-1}^{t-1}) \\ \phi(c_0 x_0^{t-1}) & \dots & \phi(c_{n-1} x_{n-1}^{t-1}) \end{pmatrix} \\ &= \left(\sum_{i=0}^{t-1} \alpha_i \phi(\theta c_0 x_0^i) + \sum_{i=0}^{t-1} \beta_i \phi(c_0 x_0^i), \dots, \sum_{i=0}^{t-1} \alpha_i \phi(\theta c_{n-1} x_{n-1}^i) + \sum_{i=0}^{t-1} \beta_i \phi(c_{n-1} x_{n-1}^i) \right) \\ &= \left(\phi(c_0 \sum_{i=0}^{t-1} (\theta \alpha_i + \beta_i) x_0^i), \dots, \phi(c_{n-1} \sum_{i=0}^{t-1} (\theta \alpha_i + \beta_i) x_{n-1}^i) \right) \\ &= (\phi(c_0 g(x_0)), \dots, \phi(c_{n-1} g(x_{n-1}))) \end{aligned}$$

where $g(x) = \sum_{i=0}^{t-1}(\theta\alpha_i + \beta_i)x^i$.

□

This observation allows us to carry some of the spirit of the attack of Sidelnikov and Shestakov (see [108]) on McEliece variants based on GRS codes. The basic idea is that multiplying the public key P by a vector results (roughly speaking) in the evaluation of a polynomial at the secret points x_i. More precisely the following holds.

Proposition 5.2. *Continuing the notation from above, multiplying the public parity-check matrix P with a vector $\gamma \in \mathbb{F}_q^{2t}$ results in*

$$\gamma P \quad = \quad (\phi(c_0 g_\gamma(x_0)), \ldots, \phi(c_{n-1} g_\gamma(x_{n-1}))) \tag{5.1.1}$$

where $g_\gamma(x) = \sum_{i=0}^{2t-1} \gamma_i g_i(x)$.

As the values γ, g_γ and γP are extensively used below we summarize their relation in Table 5.1.

Table 5.1: The relation among the values γ, g_γ and γP. The polynomials g_i are defined in Proposition 5.1

γ	A vector in $\mathbb{F}_q^{2t}$
g_γ	The polynomial defined by $g_\gamma(x) = \sum_{i=0}^{2t-1} \gamma_i g_i(x)$.
γP	A vector in $\mathbb{F}_q^n$ whose entries are given by $\phi(c_i g_\gamma(x_i))$.

If we would have the possibility to control the polynomial g_γ (even though we do not know the polynomials g_i) then γP reveals, hopefully, useful information on the secret key. While in general, controlling g_γ seems difficult, it becomes feasible in the case where the secret points x_i and the constants c_i are not chosen independently, but rather satisfy (linear) relations. The attack procedure can be split into three phases.

Isolate: The first step of the attack consists in choosing polynomials g_γ that we want to use in the attack. The main obstacle here is that we have to choose g_γ such that the redundancy allows us to efficiently recover the corresponding γ. As we will see later, it is usually not possible to isolate a single polynomial g_γ but rather to isolate a vector space of polynomials (or, equivalently, of vectors γ) of sufficiently small dimension.

Collect: After the choice of a set of polynomials and the recovery of the corresponding vectors γ, the next step of the attack consists in evaluating those polynomials at the secret points x_i. In the light of Proposition 5.2 this is simply done by multiplying the vectors γ with the public parity-check matrix P.

Solve: Given the information collected in the second step of the attack, we now have to extract the secret key, i.e., the values x_i and c_i. This corresponds to solving a system of equations. Depending on the type of collected information this is done simply by solving linear equations, by first guessing parts of the key and then verifying

by solving linear equations, or by solving non-linear equations with the help of Gröbner basis techniques (see Appendix C). The advantage of the first two possibilities is that one can easily determine the running time in general while this is not true for the last one. However, the use of Gröbner basis techniques allows us to attack specific parameters very efficiently.

The isolate phase and the collect Phase in detail

The redundancy in the choice of the points x_i and the constants c_i will allow us to identify sets of polynomials or more precisely vector spaces of polynomials. In this section we elaborate a bit more on this on a general level. Assume that we are able to identify a subspace $\Gamma \subseteq \mathbb{F}_q^{2t}$ such that for each $\gamma \in \Gamma$ we know that g_γ is of the form

$$g_\gamma(x) = \alpha_1 x^{d_1} + \alpha_2 x^{d_2} + \cdots + \alpha_r x^{d_r}$$

for some $\alpha_i \in \mathbb{F}_{q^2}$ and $d_i < t$. Equation (5.1.1) states that multiplying γ with the public key yields

$$\gamma P = (\phi(c_0 g_\gamma(x_0)), \ldots, \phi(c_{n-1} g_\gamma(x_{n-1}))) .$$

Using the assumed form of g_γ, and writing $\alpha_i = \alpha_{i,1} + \alpha_{i,2}\theta$ with $\alpha_{i,1}, \alpha_{i,2} \in \mathbb{F}_q$, we can rewrite $\phi(cg_\gamma(x))$ as

$$\begin{aligned} \phi(cg_\gamma(x)) &= \phi(c(\alpha_1 x^{d_1} + \alpha_2 x^{d_2} + \cdots + \alpha_r x^{d_r})) \\ &= \alpha_{1,1}\phi(cx^{d_1}) + \alpha_{1,2}\phi(\theta c x^{d_1}) + \cdots + \alpha_{r,1}\phi(cx^{d_r}) + \alpha_{r,2}\phi(\theta c x^{d_r}). \end{aligned}$$

Recalling that we denote by sk_i the i.th row of the secret key (cf. Equation 4.2.3), we conclude that

$$\gamma P = \alpha_{1,1}\,\mathrm{sk}_{2d_1} + \alpha_{1,2}\,\mathrm{sk}_{2d_1+1} + \alpha_{2,1}\,\mathrm{sk}_{2d_2} + \alpha_{2,2}\,\mathrm{sk}_{2d_2+1} + \cdots + \alpha_{r,2}\,\mathrm{sk}_{2d_r+1} .$$

Now, if the dimension of Γ is $2r$ this implies that there is a one to one correspondence between the elements $\gamma \in \Gamma$ and the coefficient vector $(\alpha_1, \ldots, \alpha_r)$. Stated differently, there exists an invertible $2r \times 2r$ matrix M such that for a basis $\gamma_1, \ldots, \gamma_{2r}$ of Γ we have

$$\begin{pmatrix} \gamma_1 \\ \vdots \\ \gamma_{2r} \end{pmatrix} P = M \begin{pmatrix} \mathrm{sk}_{2d_1} \\ \vdots \\ \mathrm{sk}_{2d_r+1} \end{pmatrix}, \tag{5.1.2}$$

where we now know all the values on the left side of the equation. This has to be compared to the initial problem (cf Equation 4.2.4) we are facing when trying to recover the secret key given the public key, where S is an invertible $2t \times 2t$ matrix. In this sense, the first step of the attack allows us to break the initial problem into (eventually much) smaller subproblems. Depending on the size of r (which will vary between 1 and $\log_2 t$ in the actual attacks) and the specific exponents d_i involved, this approach will allow us to efficiently reconstruct the secret key.

Note that we are actually not really interested in the matrix M, but rather in the values x_i and c_i. Therefore, a description of the result of the isolate and collect phase that is often more useful for actually solving for those unknowns is given by

$$M^{-1} \begin{pmatrix} \gamma_1 \\ \vdots \\ \gamma_{2r} \end{pmatrix} P = \begin{pmatrix} \mathrm{sk}_{2d_1} \\ \vdots \\ \mathrm{sk}_{2d_r+1} \end{pmatrix}. \tag{5.1.3}$$

The advantage of this description is that the equations are considerably simpler (in particular linear in the entries of M^{-1}) as we will see when attacking specific parameters.

5.2 Applying the framework to the quasi-cyclic variant

In this section we show how the framework described above applies to the McEliece variant presented in [11] (defined in Section 4.2.2). In particular we are going to make use of Equation (4.2.5). Recall that β is an element of order ℓ in $\mathbb{F}_{q^2}$. If ℓ is a divisor of $q-1$, such an element is in the subfield $\mathbb{F}_q$. This is the case for all the parameters in Table 4.1 except the parameter set V. We describe first an attack that works for parameters I-IV,VI and VII. Furthermore, for parameters VI and VII we describe attacks that allow us to recover the secret key within a few seconds. And Finally we will see the case that β is not in the subfield. In Table 5.2 we can see running complexity of our attacks and the average running time for the different parameters proposed in [11] (see Table 4.1). Each column corresponds to the three subsections announced above.

Table 5.2: Parameters proposed in [11], the running complexity of our attacks and the average running time. The attacks were carried on a PC with an Intel Core2 Duo with 2.2 GHz and 3 GB memory running MAGMA version V2.15 − 12. Times are averaged over 100 runs.

	Assumed security	Complexity of the attacks ($\log_2$)	Average running time (sec)	Average running time (sec)
I	80	74.9	–	–
II	90	75.1	–	–
III	100	75.3	–	–
IV	120	75.6	–	–
V	80	–	–	47
VI	90	87.3	62	–
VII	100	86.0	75	–

The case $\beta \in \mathbb{F}_q$ (parameters I-IV,VI and VII)

In this part we describe an attack that works essentially whenever β is in the subfield. The attack has a complexity of roughly $q^6 \times (n_d b)(4n_d + b)^2(\log_2 q^2)^3$ (where $n_d =$

$\lfloor \log_2(t-\ell) \rfloor$) which is the best decoding attacks known so far (they are more efficient results using Gröbner basis techniques [44]). Moreover, the attack is a key recovery attack, thus running the attack once allows an attacker to efficiently decrypt any ciphertext. However, these attacks are far from being practical (cf. Table 5.2, first column, for actual values).

In the attack we apply the general framework twice. The first part will reduce the number of possible constants c_i to q^6 values. In the second part, for each of those possibilities, we try to find the points x_i by solving an over defined system of linear equations. This system will be solvable for the correct constants and in this case reveal the secret points x_i. The secret value s (cf. Equation (4.2.5)) can be recovered very efficiently, we assume it to be known from now on, and we will see later how to find it.

Recovering the Constants c_j:

Isolate: We start by considering the simplest possible candidate for g_γ, namely $g_\gamma(x) = 1$. The task now is to compute the corresponding vector γ. Multiplying the desired vector γ with the public key P we expect (cf. Equation (5.1.1)) to obtain the following

$$\gamma P = (\phi(c_0 g_\gamma(x_0)), \ldots, \phi(c_{n-1} g_\gamma(x_{n-1}))) = (\phi(c_0), \phi(c_1), \ldots, \phi(c_{n-1})).$$

Now, taking Equation (4.2.5) into account, this becomes

$$\begin{aligned} \gamma P \;=\; & \Big(\phi(a_0), \phi(\beta^s a_0), \phi(\beta^{2s} a_0), \ldots, \phi(\beta^{(\ell-1)s} a_0), \\ & \phi(a_1), \phi(\beta^s a_1), \phi(\beta^{2s} a_1), \ldots, \phi(\beta^{(\ell-1)s} a_1), \\ & \quad \vdots \\ & \phi(a_{b-1}), \phi(\beta^s a_{b-1}), \phi(\beta^{2s} a_{b-1}), \ldots, \phi(\beta^{(\ell-1)s} a_{b-1})\Big). \end{aligned}$$

Since β is in the subfield we have $\phi(\beta x) = \beta\phi(x)$ for any $x \in \mathbb{F}_{q^2}$. Using this identity we see that γ corresponding to the constant polynomial g_γ satisfies

$$\gamma P \;=\; \phi(a_0)v_0 + \phi(a_1)v_1 + \cdots + \phi(a_{b-1})v_{b-1}$$

where

$$v_i = (\underbrace{0, \ldots, 0}_{i\ell}, 1, \beta^s, \beta^{2s}, \ldots, \beta^{(\ell-1)s}, \underbrace{0, \ldots 0}_{((b-1)-i)\ell}) \quad \text{for } 0 \le i \le b-1.$$

In other words, the γ we are looking for is such that γP is contained in the space U spanned by v_0 up to v_{b-1}, i.e., $\gamma P \in U = \langle v_0, \ldots, v_{b-1} \rangle$. Thus to compute candidates for γ we have to compute a basis for the space $\Gamma_0 = \{\gamma \mid \gamma P \in U\}$. We computed this space for many randomly generated public keys and observed the following.

Fact 3. *The dimension of the space Γ_0 is always 4.*

We do not prove this, but the next lemma explains why the dimension is at least 4.

Lemma 5.1. *Let γ be a vector such that $g_\gamma(x) = \alpha_0 + \alpha_1 x^\ell$. Then $\gamma \in \Gamma_0$.*

Proof. To show that γ is in Γ_0 we have to show that γP is a linear combination of the vectors v_i. To see this, it suffices to note that $g_\gamma(\beta x) = \alpha_0 + \alpha_1(\beta x)^\ell = \alpha_0 + \alpha_1 x^\ell = g_\gamma(x)$ as $\beta^\ell = 1$. As the points x_i satisfy Equation (4.2.5) we conclude $\gamma P = \phi(a_0 g_\gamma(y_0))v_0 + \phi(a_1 g_\gamma(y_1))v_1 + \cdots + \phi(a_{b-1} g_\gamma(y_{b-1}))v_{b-1}$. □

As, due to Observation 3, $\dim(\Gamma_0) = 4$ we conclude that

$$\{g_\gamma \mid \gamma \in \Gamma_0\} = \{\alpha_0 + \alpha_1 x^\ell \mid \alpha_0, \alpha_1 \in \mathbb{F}_{q^2}\}.$$

Collect Phase: Denote by $\gamma_1, \ldots, \gamma_4$ a basis of the four dimensional space Γ_0. Referring to Equation (5.1.3) we get

$$M^{-1} \begin{pmatrix} \gamma_1 \\ \gamma_2 \\ \gamma_3 \\ \gamma_4 \end{pmatrix} P \;=\; \begin{pmatrix} \mathrm{sk}_0 \\ \mathrm{sk}_1 \\ \mathrm{sk}_{2\ell} \\ \mathrm{sk}_{2\ell+1} \end{pmatrix}. \tag{5.2.1}$$

for an (unknown) 4×4 matrix M^{-1} with coefficients in $\mathbb{F}_q$.

Solve Phase: We denote the entries of M^{-1} by (β_{ij}). The i.th component of the first two rows of Equation (5.2.1) can be rewritten as

$$\begin{aligned} \beta_{00}(\gamma_1 P)^{(i)} + \beta_{01}(\gamma_2 P)^{(i)} + \beta_{02}(\gamma_3 P)^{(i)} + \beta_{03}(\gamma_4 P)^{(i)} = \mathrm{sk}_0^{(i)} = \phi(c_i) &= c_i + \overline{c_i} \\ \beta_{10}(\gamma_1 P)^{(i)} + \beta_{11}(\gamma_2 P)^{(i)} + \beta_{12}(\gamma_3 P)^{(i)} + \beta_{13}(\gamma_4 P)^{(i)} = \mathrm{sk}_1^{(i)} = \phi(\theta c_i) &= \theta c_i + \overline{\theta c_i}. \end{aligned}$$

Dividing the second equation by $\overline{\theta}$ and adding them, we get

$$\delta_0(\gamma_1 P)^{(i)} + \delta_1(\gamma_2 P)^{(i)} + \delta_2(\gamma_3 P)^{(i)} + \delta_3(\gamma_4 P)^{(i)} \;=\; \left(\frac{\theta}{\overline{\theta}} + 1\right) c_i, \tag{5.2.2}$$

where

$$\delta_i = \left(\beta_{0i} + \frac{\beta_{1i}}{\overline{\theta}}\right) \in \mathbb{F}_{q^2}.$$

Assume without loss of generality that $c_0 = 1$. Then, for each possible choice of δ_0, δ_1 and δ_2 we can compute δ_3 (using $c_0 = 1$) and subsequently candidates for all constants c_i. We conclude that there are $(q^2)^3$ possible choices for the constants c_i (and thus in particular for the b constants $a_0 = c_0, \ldots, a_{b-1} = c_{(b-1)\ell}$). We will have to repeat the following step for each of those choices.

Recovering Points x_i: Given one out of the q^6 possible guesses for the constants c_i we now explain how to recover the secret values x_i by solving an (over defined) system of linear equations. Most of the procedure is very similar to what was done to (partially) recover the constants.

Isolate: Here we make use of polynomials $g_\gamma = x^d$ for $d \leq t-1$. The case $g_\gamma = 1$ is thus a special case $d = 0$. Following the same computations as above, we see that for the vector γ corresponding to $g_\gamma = 1$ it holds that $\gamma P \in U_d$ where

$$U_d \quad = \quad \langle v_{(d)0}, \ldots, v_{(d)b-1} \rangle \tag{5.2.3}$$

and

$$v_{(d)i} = (\underbrace{0, \ldots, 0}_{i\ell}, 1, \beta^{s+d}, \beta^{2(s+d)}, \ldots, \beta^{(\ell-1)(s+d)}, \underbrace{0, \ldots 0}_{((b-1)-i)\ell}) \quad \text{for } 0 \leq i \leq b-1.$$

As before we define $\Gamma_d = \{\gamma \mid \gamma P \in U_d\}$, and, based on many randomly generated public keys we state the following.

Fact 4. *For $d \leq 2t - \ell$ the dimension of the space Γ_d is always 4.*

Similar as above, the next lemma, which can be proven similar as Lemma 5.1, explains why the dimension of Γ_d is at least 4.

Lemma 5.2. *Let γ be a vector such that $g_\gamma(x) = \alpha_0 x^d + \alpha_1 x^{d+\ell}$. Then $\gamma \in \Gamma_d$.*

As, due to Observation 4, $\dim(\Gamma_d) = 4$ we conclude that

$$\{g_\gamma \mid \gamma \in \Gamma_d\} = \{\alpha_0 x^d + \alpha_1 x^{d+\ell} \mid \alpha_0, \alpha_1 \in \mathbb{F}_{q^2}\}.$$

Collect Phase: Denote by $\gamma_{(d)1}, \ldots \gamma_{(d)4}$ a basis of the four dimensional space Γ_d. Referring to Equation (5.1.3) we get

$$M_d^{-1} \begin{pmatrix} \gamma_{(d)1} \\ \gamma_{(d)2} \\ \gamma_{(d)3} \\ \gamma_{(d)4} \end{pmatrix} P \quad = \quad \begin{pmatrix} \mathrm{sk}_{2d} \\ \mathrm{sk}_{2d+1} \\ \mathrm{sk}_{2(\ell+d)} \\ \mathrm{sk}_{2(\ell+d)+1} \end{pmatrix}$$

for an (unknown) 4×4 matrix M_d^{-1} with coefficients in $\mathbb{F}_q$ from which we learn (similar to Equation (5.2.2))

$$(\frac{\theta}{\overline{\theta}} + 1) c_i x_i^d = \delta_{(d)0}(\gamma_{(d)1} P)^{(i)} + \delta_{(d)1}(\gamma_{(d)2} P)^{(i)} + \delta_{(d)2}(\gamma_{(d)3} P)^{(i)} + \delta_{(d)3}(\gamma_{(d)4} P)^{(i)} \tag{5.2.4}$$

for unknowns $\delta_{(d)i} \in \mathbb{F}_{q^2}$ (and unknowns x_i). How to solve such a system? Here, the freedom of choice in d allows us to choose $1 \leq d \leq t - \ell$ as a power of two. In this case, Equations (5.2.4) become *linear* in the bits of x_i when viewed as binary equations for a fixed guess for c_i. Let n_d be the number of possible choices for d, i.e., $n_d = \lfloor \log_2(t - \ell) \rfloor$. We get a linear system with $(\log_2 q^2)(4n_d + b)$ unknowns ($4n_d$ for the unknowns $\delta_{(d)i}$ and b unknowns for the points $x_{\ell j} = y_j$) and $(\log_2 q^2) n_d b$ equations ($\log_2 q^2$ equation for each d and each component $i = j\ell$). Thus whenever $b > 4$ and $n_d \geq 2$ (i.e., $t \geq 4$) this system is likely to be over defined and thus reveals the secret values x_i. We verified the behavior of the system and observed the following.

Fact 5. *Only for the right guess for the constants c_i the system is solvable. When we fix wlog $x_0 = 1$, for the right constants there is a unique solution for the values x_i.*

As there are q^6 possibilities for the constants and it takes roughly $(n_d b)(4n_d + b)^2(\log_2 q^2)^3$ binary operations to solve the system, the overall running time of this attack is $q^6 \times (n_d b)(4n_d + b)^2(\log_2 q^2)^3$. For the concrete parameters the attack complexity is summarized in Table 5.2 (first column).

Recovering s: Now that we have all the notation that we need, we can find s: let $d' = d + s$ then for $0 \leq i \leq b - 1$

$$\begin{aligned} v_{(d)i} &= (\underbrace{0, \ldots, 0}_{i\ell}, 1, \beta^{s+d}, \beta^{2(s+d)}, \ldots, \beta^{(\ell-1)(s+d)}, \underbrace{0, \ldots 0}_{((b-1)-i)\ell}) \\ &= (\underbrace{0, \ldots, 0}_{i\ell}, 1, \beta^{d'}, \beta^{2d'}, \ldots, \beta^{(\ell-1)d'}, \underbrace{0, \ldots 0}_{((b-1)-i)\ell}). \end{aligned}$$

Then $\Gamma_d = \Gamma_{d'-s}$, as the dimension of $\Gamma_d = 4$ if $0 \leq d \leq 2t - \ell$, we have that the dimension of $\Gamma_{d'-s} = 4$ if $0 \leq d' - s \leq 2t - \ell$ i.e., if $s \leq d' \leq 2t - \ell + s$. The idea is to check the dimension of $\Gamma_{d'-s}$ for each d' from 0 to $2t - 1$ and s will be the first d' such that $\dim(\Gamma_{d'-1}) = 2$ and $\dim(\Gamma_{d'}) = 4$.

Practical attacks for parameter sets VI and VII

In this part we describe how, using Gröbner basis techniques, we can recover the secret key for the parameter sets VI and VII of Table 4.1 within a few seconds on a standard PC. The attack resembles in large parts the attack described above. The main difference in the solve phase is that we are not going to guess the constants to get linear equations for the points, but instead solve a non-linear system with the help of Gröbner basis techniques.

Isolate: Again, we make use of polynomials $g_\gamma = x^d$ but this time with the restriction $t - \ell \leq d < \ell$. To recover the corresponding vectors γ we make use of the space U_d defined by Equation (5.2.3). Now, with the given restriction on d it turns out that the situation, from an attacker's point of view, is nicer as for $\Gamma_d = \{\gamma \mid \gamma P \in U_d\}$, we obtain

Fact 6. *For $t - \ell \leq d < \ell$ the dimension of the space Γ_d is always 2.*

Thus, we isolated the polynomials $g(x) = \alpha_d x^d$ in this case. In other words

$$\{g_\gamma \mid \gamma \in \Gamma_d\} = \{\alpha x^d \mid \alpha \in \mathbb{F}_{q^2}\}.$$

The reason why we did not get the second term, i.e., $x^{d+\ell}$ in this case, is that the degree of g_γ is bounded by $t - 1$ and $d + \ell$ exceeds this bound.

Collect Phase: Denote by $\gamma_{(d)1}, \gamma_{(d)2}$ a basis of the two dimensional space Γ_d. Referring to Equation (5.1.3) we get

$$M_d^{-1} \begin{pmatrix} \gamma_{(d)1} \\ \gamma_{(d)2} \end{pmatrix} P \;=\; \begin{pmatrix} \mathrm{sk}_{2d} \\ \mathrm{sk}_{2d+1} \end{pmatrix},$$

for an (unknown) 2×2 matrix M_d^{-1} with coefficients in $\mathbb{F}_q$.

Solve Phase: We denote the entries of M_d^{-1} by (β_{ij}). The i.th component of the first row can be rewritten as

$$\beta_{00}(\gamma_{(d)1}P)^{(i)} + \beta_{01}(\gamma_{(d)2}P)^{(i)} \;=\; c_i x_i^d + \overline{c_i x_i^d} \tag{5.2.5}$$

Again, we can assume $x_0 = c_0 = 1$. This (for $i = 0$) reveals $\beta_{00}(\gamma_{(d)1}P)^{(0)} + \beta_{01}(\gamma_{(d)2}P)^{(0)} = 0$ and thus $\beta_{01} = \frac{\beta_{00}(\gamma_{(d)1}P)^{(0)}}{(\gamma_{(d)2}P)^{(0)}}$. Substituting back into Equation (5.2.5) we get

$$\beta_{00}\left((\gamma_{(d)1}P)^{(i)} + \frac{(\gamma_{(d)1}P)^{(0)}}{(\gamma_{(d)2}P)^{(0)}}(\gamma_{(d)2}P)^{(i)}\right) = c_i x_i^d + \overline{c_i x_i^d}.$$

For parameter sets VI and VII we successfully solved this set of equations within seconds on a standard PC using MAGMA [23]. For parameters VI, d ranges from 33 to 92 and for parameters VII from 15 to 92. Thus in both cases we can expect to get a highly overdefined system. This allows us to treat $\overline{c_i}$ and $\overline{x_i^d}$ as independent variables, speeding up the task of computing the Gröbner basis by a large factor. The average running times are summarized in Table 5.2 (second column).

This attack does not immediately apply to parameters I to IV as here the range of d satisfying $t - \ell \leq d < \ell$ is too small (namely $d \in \{49, 50\}$) which does not result in sufficiently many equations. However, we anticipate that using Gröbner basis techniques might speed up the attack for those parameters as well.

A practical attack for parameter set V

Recall that β is an element of order ℓ in $\mathbb{F}_{q^2}$, we focus on the case that β is not in the subfield $\mathbb{F}_q$. In the case things are a little different.

Isolate Phase: Assume that again we would like to isolate the polynomial $g_\gamma(x) = x^d$. Multiplying the vector γ with the public key P yields

$$\begin{aligned} \gamma P \;=\; \Big(&\phi(a_0 y_0), \phi(\beta^{s+d} a_0 y_0), \phi(\beta^{2(s+d)} a_0 y_0) \ldots, \phi(\beta^{(\ell-1)(s+d)} a_0 y_0), \\ &\phi(a_1 y_1), \phi(\beta^{s+d} a_1 y_1), \phi(\beta^{2(s+d)} a_1 y_1) \ldots, \phi(\beta^{(\ell-1)(s+d)} a_1 y_1), \\ &\quad\vdots \\ &\phi(a_{b-1} y_{b-1}), \phi(\beta^{s+d} a_{b-1} y_{b-1}), \ldots, \phi(\beta^{(\ell-1)(s+d)} a_{b-1} y_{b-1})\Big). \end{aligned}$$

However, as β is not in the subfield we cannot continue as before. Instead $(\gamma P)^{(0)}$ and $(\gamma P)^{(1)}$ allow to recover $a_0 y_0$ by means of $(\gamma P)^{(0)} = \phi(a_0 y_0)$ and $(\gamma P)^{(1)} = \phi(\beta^{s+d} a_0 y_0)$ using Equation (4.2.2), which reveals $a_0 y_0$ as

$$a_0 y_0 = \frac{(\gamma P)^{(0)} \overline{\beta^{s+d}} + (\gamma P)^{(1)}}{\beta^{s+d} + 1}.$$

The same argument reveals $a_j y_j$ using $(\gamma P)^{(j\ell)}$ and $(\gamma P)^{(j\ell+1)}$. Therefore, when looking for γ corresponding to x^d we can solve for all γ such that γP satisfies

$$(\gamma P)^{(j\ell+i)} = \phi\left(\beta^{i(s+d)} \frac{(\gamma P)^{(j\ell)} \overline{\beta^{s+d}} + (\gamma P)^{(j\ell+1)}}{\beta^{s+d} + 1}\right) \tag{5.2.6}$$

for $0 \leq j < b$ and $0 \leq i < \ell$. We denote by Γ_d the space of all possible solutions, i.e.,

$$\Gamma_d = \{\gamma \mid \gamma P \text{ satisfies Equation (5.2.6) } \}$$

Fact 7. *The dimension of* Γ_d *is in* $\{4, 6, 8\}$.

We next explain those dimensions.

Lemma 5.3. $\{g_\gamma \mid \gamma \in \Gamma_d\}$ *contains all polynomials*

$$\alpha_0 x^d + \alpha_1^{d+\ell} + \alpha_2 x^r + \alpha_4 x^{r+\ell}$$

of degree at most $t-1$ *where* $r = q(d+s) - s \bmod \ell$.

Proof. In order to prove Lemma 5.3 we claim that any polynomial satisfies either $g(\beta x) = \beta^d g(x)$ or $\beta^s g(\beta x) = \overline{\beta^{d+s}} g(x)$ is in the set. The first condition is obvious and the second follows from the fact that in this case (using Equation (4.2.1))

$$\phi(\beta^s g(\beta x)) = \phi(\overline{\beta^{d+s}} g(x)) = \phi(\beta^{d+s} \overline{g(x)})$$

and

$$\phi(g(x)) = \phi(\overline{g(x)}).$$

If $g(x)$ is a monomial $g(x) = x^r$ we get

$$g(\beta x) = \beta^r g(x).$$

Thus, to satisfy the second equations r has to fulfil.

$$r = q(d+s) - s \bmod \ell$$

□

Clearly, the smaller the dimension of Γ_d is, the better the attack. We pick only those d such that $\dim \Gamma_d = 4$ (avoiding the exponents $d+\ell$ and $r+\ell$). The condition for this is

$$t - \ell \leq d \leq \ell \text{ and } r - \ell \leq d \leq \ell$$

and $\beta^{d+s} \notin \mathbb{F}_q$. In this case

$$\{g_\gamma \mid \gamma \in \Gamma_d\} = \{\alpha_0 x^d + \alpha_1 x^r\}$$

where $r = q(d+s) - s \bmod \ell$. For parameter set V, we ran through all possible values s and verified that in any case the number of suitable exponents d is at least 8.

Collect Phase: The collect phase, is also different in this case. Denote by $\gamma_{(d)1}$, $\gamma_{(d)2}$ two linearly independent elements in Γ_d. Define

$$g_{\gamma_{(d)1}} = \alpha_0 x^d + \alpha_1 x^r$$

and

$$g_{\gamma_{(d)2}} = \alpha_0' x^d + \alpha_1' x^r.$$

We have

$$\begin{aligned}(\gamma_{(d)1}P)^{(i\ell)} &= \phi(a_i g(y_i)) \\ &= \phi(a_i(\alpha_0 y_i^d + \alpha_1 y_i^r)) \\ &= \phi(a_i\alpha_0 y_i^d + \overline{a_i\alpha_1 y_i^r})\end{aligned}$$

and

$$\begin{aligned}(\gamma_{(d)1}P)^{(i\ell+1)} &= \phi(a_i\beta^s g(\beta y_i)) = \phi(a_i\beta^s(\alpha_0\beta^d y_i^d + \alpha_1\beta^r y_i^r)) \\ &= \phi(\beta^{s+d}a_i\alpha_0 y_i^d + \overline{\beta^{s+r}a_i\alpha_1 y_i^r})) \\ &= \phi(\beta^{s+d}(a_i\alpha_0 y_i^d + \overline{a_i\alpha_1 y_i^r}))\end{aligned}$$

where we made use of the identity $\overline{\beta^{s+r}} = \beta^{s+d}$. Thus, given $(\gamma_{(d)1}P)^{(i\ell)}$ and $(\gamma_{(d)1}P)^{(i\ell+1)}$ allows us to compute

$$\eta_i = a_i\alpha_0 y_i^d + \overline{a_i\alpha_1 y_i^r}$$

and similarly

$$\eta_i' = a_i\alpha_0' y_i^d + \overline{a_i\alpha_1' y_i^r}.$$

We obtain vectors $\eta, \eta' \in \mathbb{F}_{q^2}^b$ such that

$$\begin{pmatrix} \eta \\ \eta' \end{pmatrix} = \begin{pmatrix} \alpha_0 & \overline{\alpha_1} \\ \alpha_0' & \overline{\alpha_1'} \end{pmatrix} \begin{pmatrix} a_0 y_0^d, a_1 y_1^d, \ldots, a_{b-1} y_{b-1}^d \\ \overline{a_0 y_0^r}, \overline{a_1 y_1^r}, \ldots, \overline{a_{b-1} y_{b-1}^r} \end{pmatrix}$$

Stated differently, there exist elements $\beta_0, \beta_1, \beta_2, \beta_3$ such that

$$\begin{pmatrix} \beta_0 & \beta_1 \\ \beta_2 & \beta_3 \end{pmatrix} \begin{pmatrix} \eta \\ \eta' \end{pmatrix} = \begin{pmatrix} a_0 y_0^d, a_1 y_1^d, \ldots, a_{b-1} y_{b-1}^d \\ \overline{a_0 y_0^r}, \overline{a_1 y_1^r}, \ldots, \overline{a_{b-1} y_{b-1}^r} \end{pmatrix}. \tag{5.2.7}$$

Solve Phase: We only consider the first row of Equation (5.2.7). In other words

$$\beta_0\eta^{(i)} + \beta_1\eta'^{(i)} = a_i y_i^d.$$

Again, we assume wlog that $a_0 = y_0 = 1$ and this allows us to represent β_1 in terms of the unknown β_0. Thus, we finally get equations

$$\beta_0\eta^{(i)} + \left(\frac{\beta_0\eta^{(0)} + 1}{\eta'^0}\right)\eta'^{(i)} = a_i y_i^d.$$

Using the computer algebra package MAGMA this system of equations can be solved very quickly on a standard PC. We give the running time in Table 5.2 (third column).

5.3 Applying the framework to the dyadic variant

In this section we introduce, in a very similar way as we did in Section 5.2, how to apply the general framework of the attack to the McEliece variant introduced in [85] and described in Section 4.2.3. For $u = \log_2 t$ the attack has a complexity of roughly $q^2 \times (\log_2 q^2)^3 (u^2+3u+b)^2 u(u+b)$ binary operations, which for the parameters given in [85] means that we can recover the secret key within at most a few days with a standard PC (cf. Table 5.3).

Recovering Constants c_j:

Isolate phase: As before we consider $g_\gamma(x) = 1$ and we want to compute the corresponding vector γ. From Equation (5.1.1) we have that

$$\gamma P = (\phi(c_0 g_\gamma(x_0)), \ldots, \phi(c_{n-1} g_\gamma(x_{n-1}))) = (\phi(c_0), \phi(c_1), \ldots, \phi(c_{n-1})).$$

Now, taking Equation (4.2.7) into account, this becomes

$$\begin{aligned} \gamma P \;=\; & (\phi(a_0), \phi(a_0), \phi(a_0) \ldots, \phi(a_0), \\ & \phi(a_1), \phi(a_1), \phi(a_1) \ldots, \phi(a_1), \\ & \quad \vdots \\ & \phi(a_{b-1}), \phi(a_{b-1}), \phi(a_{b-1}) \ldots, \phi(a_{b-1}))\,. \end{aligned}$$

We see that γ corresponding to the constant polynomial g_γ satisfies

$$\gamma P = \phi(a_0)v_0 + \phi(a_1)v_1 + \cdots + \phi(a_{b-1})v_{b-1}$$

where

$$v_i = (\underbrace{0, \ldots, 0}_{i\ell}, 1, 1, 1, \ldots, 1, \underbrace{0, \ldots 0}_{((b-1)-i)\ell}) \quad \text{for } 0 \le i \le b-1.$$

Let U be the space spanned by v_0 up to v_{b-1}. The γ that we are looking for is such that

$$\gamma P \in U = \langle v_0, \ldots, v_{b-1} \rangle.$$

Thus in order to find γ we have to compute a basis for the space $\Gamma_0 = \{\gamma \mid \gamma P \in U\}$. We did this for many randomly generated public keys and observe the following.

Fact 8. *The dimension of the space Γ_0 is always 2.*

The next lemma shows, why the dimension is at least 2.

Lemma 5.4. *Let γ be a vector such that $g_\gamma(x) = \alpha_0$. Then $\gamma \in \Gamma_0$.*

Note that $\dim \Gamma_0 = 2$ is actually the best case we can hope for within our framework.

Collect Phase: Denote by γ_1, γ_2 a basis of the two dimensional space Γ_0. Referring to Equation (5.1.3) we get

$$M^{-1} \begin{pmatrix} \gamma_1 \\ \gamma_2 \end{pmatrix} P = \begin{pmatrix} \mathrm{sk}_0 \\ \mathrm{sk}_1 \end{pmatrix} \tag{5.3.1}$$

for an (unknown) 2×2 matrix M^{-1} with coefficients in $\mathbb{F}_q$.

Solve Phase: We denote the entries of M^{-1} by (β_{ij}). We get

$$\begin{pmatrix} \beta_{00} & \beta_{01} \\ \beta_{10} & \beta_{11} \end{pmatrix} \begin{pmatrix} \gamma 1 \\ \gamma 2 \end{pmatrix} P = \begin{pmatrix} \phi(c_0), & \phi(c_1), & \cdots, & \phi(c_{b-1}) \\ \phi(\theta c_0), & \phi(\theta c_1), & \cdots, & \phi(\theta c_{b-1}) \end{pmatrix}.$$

Assuming wlog that $c_0 = 1$, we can compute β_{01} as a function of β_{00} and β_{11} as a function of β_{10}. Then guessing β_{00} and β_{10} allows us to recover all the constants. We conclude that there are q^2 possible choices for the b constants $a_0, \ldots, a_{b-1}$. We will have to repeat the following step for each of those choices.

Recovering Points x_i***:*** Assuming that we know the constants c_i we explain how to recover the secret values x_i by solving an (over-defined) system of linear equations. If the set of constants that we have chosen in the previous step is not the correct one, the system will not be solvable.

Isolate: We start by considering $g_\gamma(x) = x$, and multiply the desired vector γ with the public key P. We expect (cf. Equation (5.1.1)) to obtain the following:

$$\gamma P = (\phi(c_0 g_\gamma(x_0)), \ldots, \phi(c_{n-1} g_\gamma(x_{n-1})))$$

then

$$\begin{array}{rccccc} \gamma P = & (\phi(a_0 x_0), & \phi(a_0 x_1), & \ldots, & \phi(a_0 x_{\ell-1}), \\ & \phi(a_1 x_\ell), & \phi(a_1 x_{\ell+1}), & \ldots, & \phi(a_1 x_{2\ell-1}), \\ & \vdots & \vdots & & \vdots \\ & \phi(a_{b-1} x_{(b-1)\ell}), & \phi(a_{b-1} x_{(b-1)\ell+1}), & \ldots, & \phi(a_{b-1} x_{b\ell-1})). \end{array} \tag{5.3.2}$$

Recalling Equation (4.2.8) we see that the vector γ we are looking for satisfies

$$(\gamma P)^{(\ell i + j)} = \sum_{f=0}^{u-1} j_f (\gamma P)^{(\ell i + 2^f)} + (1 + W_H(j))(\gamma P)^{(\ell i)} \quad \forall\ 0 \le i < b, 0 \le j < \ell \tag{5.3.3}$$

where $j = \sum_{f=0}^{u-1} j_f 2^f$ is the binary representation of j. Denoting $\Gamma_1 = \{\gamma \in \mathbb{F}_q^{2t} \mid \gamma \text{ satisfies (5.3.3)}\}$ we got the following observation by randomly generating many keys.

Fact 9. *The dimension of the space* Γ_1 *is always* $u + 1$.

Clearly, the dimension is at least $u + 1$ as we are actually only checking if g_γ is $\mathbb{F}_2$ affine and therefore if γ is such that $g_\gamma(x) = \alpha_0 + \alpha_1 x + \alpha_2 x^2 + \cdots + \alpha_u x^{2^{u-1}}$ then $\gamma \in \Gamma_1$.

Collect Phase: A straight-forward application of Equation (5.1.2) would lead to a linear system that becomes only over-defined for a large number of blocks. Thus, in order to avoid this we modify the collect phase as follows. Let $\gamma \in \Gamma_1$ be given. We have

$$\begin{aligned} \gamma P &= (\phi(a_0 g_\gamma(x_0)), \phi(a_0 g_\gamma(x_1)), \ldots, \phi(a_0 g_\gamma(x_{l-1})), \\ &\quad \phi(a_1 g_\gamma(x_\ell)), \phi(a_1 g_\gamma(x_{\ell+1})), \ldots, \phi(a_1 g_\gamma(x_{2\ell-1})), \ldots) \end{aligned}$$

where g_γ is an $\mathbb{F}_2$ affine polynomial. Making use of the identity

$$x_0 + x_i = x_\ell + x_{\ell+i} \quad \forall\ 0 \leq i < \ell$$

allows us to compute $\mu_\gamma^{(i)} = \phi(a_0(g_\gamma(x_0 + x_i) + g_\gamma(0)))$ and $\nu_\gamma^{(i)} = \phi(a_1(g_\gamma(x_0 + x_i) + g_\gamma(0)))$. As we assume we know the constants a_0 and a_1, given $\mu_\gamma^{(i)}$ and $\nu_\gamma^{(i)}$ we can recover (cf. Equation (4.2.2)) $z_\gamma^{(i)} = g_\gamma(x_0 + x_i) + g(0)$ (as long as (a_0, a_1) is an $\mathbb{F}_q$ basis of $\mathbb{F}_{q^2}$). Next, by solving a system of linear equations, we compute a γ' such that

$$z_{\gamma'}^{(i)} = \theta z_\gamma^{(i)}.$$

It turns out that the corresponding polynomial $g_{\gamma'}$ is unique up to adding constants, i.e., $g_{\gamma'} = \theta g_\gamma + c$. Summarizing our findings so far we get

$$\begin{aligned} \gamma P &= (\phi(a_0 g_\gamma(x_0)), \phi(a_0 g_\gamma(x_1)), \ldots, \phi(a_{b-1} g_\gamma(x_{n-1}))) \\ \gamma' P &= (\phi(\theta a_0 g_\gamma(x_0) + a_0 c), \phi(\theta a_0 g_\gamma(x_1) + a_0 c), \ldots, \phi(\theta a_{b-1} g_\gamma(x_{n-1}) + a_{b-1} c)). \end{aligned}$$

This, again using Equation (4.2.2), allows us to compute

$$\delta = (a_0 g_\gamma(x_0), \ldots, a_{b-1} g_\gamma(x_{n-1})) + (a_0 c', \ldots, a_{b-1} c') + (\overline{a_0} c'', \ldots, \overline{a_{b-1}} c'')$$

for (unknown) constants c', c''. Repeating this procedure for different elements $\gamma \in \Gamma_1$ will eventually result in $\delta_1, \ldots, \delta_{u+2}$ that span a space of dimension $u + 2$. The data we collected can thus be written as

$$\begin{pmatrix} \delta_1 \\ \vdots \\ \delta_{u+2} \end{pmatrix} = M \begin{pmatrix} (a_0, a_0, \ldots, a_{b-1}) \\ (\overline{a_0}, \overline{a_0}, \ldots, \overline{a_{b-1}}) \\ (a_0 x_0, a_0 x_1, \ldots, a_{b-1} x_{n-1}) \\ \vdots \quad \vdots \\ (a_0 x_0^{2^{u-1}}, a_0 x_1^{2^{u-1}}, \ldots, a_{b-1} x_{n-1}^{2^{u-1}}) \end{pmatrix} \tag{5.3.4}$$

for an invertible $(u + 2) \times (u + 2)$ matrix M.

Solve Phase: Multiplying Equation (5.3.4) by M^{-1} yields equations that, when viewed as binary equations, are linear in the entries of M^{-1} and the values x_i (as we assume the a_i to be known). The first two rows of M are determined by the (known) values of the constants a_i. Thus we are left with $N_u = \log_2(q^2)(u(u + 2) + (u + b))$ unknowns, i.e., the remaining $u(u + 2)$ entries of M^{-1} and the $u + b$ points

$$x_0, x_1, x_2, x_4, \ldots, x_{2^{u-1}}, x_\ell, x_{2\ell}, x_{3\ell}, \ldots x_{(b-1)\ell}$$

(all other points are given as linear combinations of those). The number of equations is $N_e = \log_2(q^2)(u+b) \times u$. In particular, whenever $b \geq 4$ and $u \geq 4$, i.e., $t \geq 2^4$, we get more equations than unknowns and can hope for a unique solution. We implemented the attack and observed the following.

Fact 10. *Only for the right guess of the constants c_i the system is solvable. In this case the constants x_0 and x_1 could be chosen as arbitrary non-zero elements in $\mathbb{F}_{q^2}$.*

As there are q^2 possibilities for the constants and it takes roughly $(N_e N_u^2)$ binary operations to solve the system, the overall running time of this attack is $q^2 \times (\log_2 q^2)^3(u^2+3u+b)^2 u(u+b)$ binary operations. In Table 5.3 we computed the complexity of the attack for the sample parameters given in [85, Table 5].

Table 5.3: Sample parameters from [85] along with the complexity of our attack. Running time was measured on a PC with an Intel Core2 Duo with 2.2 GHz and 3 GB memory running MAGMA version V2.15 − 12.

q	q^m	ℓ	t	b	Public key size	Assumed security	Complexity of the attack ($\log_2$)	Estimated running time(h)
2^8	2^{16}	128	128	4	4096	80	43.7	36
		128	128	5	6144	112	43.8	41
		128	128	6	8192	128	44.0	47
		256	256	5	12288	192	44.8	107
		256	256	6	16384	256	44.9	125

5.4 The binary case of the dyadic variant

In [85] Barreto and Misoski discusses why in the binary case, for chosen parameters, the cryptosystem is not affected by the attacks presented by Faugère et.al. in [44]. We decide to do the binary case in a separated section, to study it in a more detail way and see for which parameters the attack will be effective.

All the notation is like in the Section 4.2.1, we just do some small modifications to be in the binary case: let m be an integer, we denote by $\mathbb{F}_{2^m}$ the extension of $\mathbb{F}_2$ of degree m. For an element $x \in \mathbb{F}_{2^m}$ we denote its conjugate by $\overline{x}$. Given an $\mathbb{F}_2$ basis $1, \omega_1, \cdots, \omega_{m-1}$ of $\mathbb{F}_{2^m}$ we denote by $\psi : \mathbb{F}_{2^m} \to (\mathbb{F}_2)^m$ the vector space isomorphism such that

$$\psi(x) = \psi(x_0 + \omega_1 x_1 + \cdots + \omega_{m-1} x_{m-1}) = (x_{m-1}, \cdots, x_1, x_0)^T.$$

Note that, without loss of generality, we can choose $(\theta_0, \cdots, \theta_{m-1})$ such that

$$\psi(x) = (\phi(\theta_0 x), \cdots, \phi(\theta_{m-1} x))^T$$

where $\phi(x) = Tr(x) = x + x^2 + \cdots + x^{2^{m-1}}$ is the trace mapping. A fact that we will use at several instances later is that given $\phi(\alpha_0 x), \phi(\alpha_1 x), \cdots, \phi(\alpha_{m-1} x)$ for some $\alpha_0, \alpha_1, \cdots, \alpha_{m-1}, x \in \mathbb{F}_{2^m}$ we can recover x as long as $\alpha_0, \alpha_1, \cdots, \alpha_{m-1}$ form a basis of $\mathbb{F}_{2^m}$.

Recovering Constants c_j**:** This can be done exactly in the same way as in the previous section, keeping the same notation, we did the isolate phase for many randomly generated public keys and observed the following.

Fact 11. *The dimension of the space* Γ_0 *is always* m*.*

Lemma 5.4 shows, why the dimension is at least m. Note that $\dim \Gamma_0 = m$ is actually the best case we can hope for within our framework.

Collect Phase: Denote by $(\gamma_0, \cdots, \gamma_{m-1})$ a basis of the m dimensional space Γ_0. Referring to Equation (5.1.3) we get

$$M^{-1} \begin{pmatrix} \gamma_0 \\ \vdots \\ \gamma_{m-1} \end{pmatrix} P \;=\; \begin{pmatrix} \mathrm{sk}_0 \\ \vdots \\ \mathrm{sk}_{m-1} \end{pmatrix} \tag{5.4.1}$$

for an (unknown) $m \times m$ matrix M^{-1} with coefficients in $\mathbb{F}_2$.

Solve Phase: We denote the entries of M^{-1} by (β_{ij}). We get

$$\begin{pmatrix} \beta_{00} & \cdots & \beta_{0(m-1)} \\ \vdots & \cdots & \vdots \\ \beta_{(m-1)0} & \cdots & \beta_{(m-1)(m-1)} \end{pmatrix} \begin{pmatrix} \gamma_0 \\ \vdots \\ \gamma_{m-1} \end{pmatrix} P = \begin{pmatrix} \phi(\theta_0 c_0), & \cdots, & \phi(\theta_0 c_{b-1}) \\ \vdots & \cdots & \vdots \\ \phi(\theta_{m-1} c_0), & \cdots, & \phi(\theta_{m-1} c_{b-1}) \end{pmatrix}.$$

Assuming wlog that $c_0 = 1$, we can compute the first column of M^{-1} as a function of the other columns. Then guessing $(m-1) \times m$ unknowns, allows us to recover all the constants. We conclude that there are $2^{m(m-1)}$ possible choices for the b constants $a_0, \ldots, a_{b-1}$. We will have to repeat the following step for each of those choices.

Recovering Points x_i**:** Doing exactly the same as in the previous section and keeping the same notation, we made the following observation by randomly generating many keys.

Fact 12. *The dimension of the space* Γ_1 *is always* $(u+1)m$*.*

Clearly, the dimension is at least $(u+1)m$ as we are actually only checking if g_γ is $\mathbb{F}_2$ affine and therefore if γ is such that $g_\gamma(x) = \alpha_0 + \alpha_1 x + \alpha_2 x^2 + \cdots + \alpha_u x^{2^{u-1}}$ then $\gamma \in \Gamma_1$.

Collect Phase: A straight-forward application of Equation (5.1.2) would lead to a linear system that becomes only over-defined for a large number of blocks. Thus, in order to avoid this we modify the collect phase as follows. Let $\gamma \in \Gamma_1$ be given. We have

$$\begin{aligned} \gamma P \;=\; & (\phi(a_0 g_\gamma(x_0)), \phi(a_0 g_\gamma(x_1)), \ldots, \phi(a_0 g_\gamma(x_{t-1})), \\ & \phi(a_1 g_\gamma(x_t)), \phi(a_1 g_\gamma(x_{t+1})), \ldots, \phi(a_1 g_\gamma(x_{2t-1})), \ldots) \end{aligned}$$

where g_γ is an $\mathbb{F}_2$ affine polynomial. Making use of the identity

$$x_0 + x_i = x_t + x_{t+i} \quad \forall\ 0 \le i < t$$

allows us to compute $\mu_0^{(i)} = \phi(a_0(g_\gamma(x_0+x_i)+g_\gamma(0)))$, $\mu_1^{(i)} = \phi(a_1(g_\gamma(x_0+x_i)+g_\gamma(0)))$, $\cdots, \mu_{m-1}^{(i)} = \phi(a_{m-1}(g_\gamma(x_0 + x_i) + g_\gamma(0)))$. As we assume we know the constants $a_0, a_1, \cdots, a_{m-1}$, given $\mu_0^{(i)}, \cdots, \mu_{m-1}^{(i)}$ we can recover $z_\gamma^{(i)} = g_\gamma(x_0 + x_i) + g_\gamma(0)$ (as long as $(a_0, a_1, \cdots, a_{m-1})$ is an $\mathbb{F}_2$ basis of $\mathbb{F}_{2^m}$, this happen with probability 0.288 [1]). Next, by solving a system of linear equations, we compute γ_j for $j = 0, 1, \cdots, m-1$, such that

$$z_{\gamma_j}^{(i)} = \theta_j z_\gamma^{(i)}.$$

It turns out that the corresponding polynomial g_{γ_j} is unique up to adding constants. Summarizing our findings so far we get

$$\begin{aligned}
\gamma_0 P &= (\phi(\theta_0 a_0 g_\gamma(x_0)), \phi(\theta_0 a_0 g_\gamma(x_1)), \ldots, \phi(\theta_0 a_{b-1} g_\gamma(x_{n-1}))) \\
\gamma_1 P &= (\phi(\theta_1 a_0 g_\gamma(x_0) + a_0 c_1), \phi(\theta_1 a_0 g_\gamma(x_1) + a_0 c_1), \ldots, \phi(\theta_1 a_{b-1} g_\gamma(x_{n-1}) + a_{b-1} c_1)) \\
&\vdots \\
\gamma_{m-1} P &= (\phi(\theta_{m-1} a_0 g_\gamma(x_0) + a_0 c_{m-1}), \ldots, \phi(\theta_{m-1} a_{b-1} g_\gamma(x_{n-1}) + a_{b-1} c_{m-1})).
\end{aligned}$$

This allows us to compute

$$\begin{aligned}
\delta = {} & (a_0 g_\gamma(x_0), a_0 g_\gamma(x_1), \ldots, a_{b-1} g_\gamma(x_{n-1})) + (a_0 c'_0, a_0 c'_0, \ldots, a_{b-1} c'_0) + \\
& (a_0^2 c'_1, a_0^2 c'_1, \ldots, a_{b-1}^2 c'_1) + \cdots + (a_0^{2^{m-1}} c'_{m-1}, a_0^{2^{m-1}} c'_{m-1}, \ldots, a_{b-1}^{2^{m-1}} c'_{m-1})
\end{aligned}$$

for (unknown) constants $c'_0, c'_1, \cdots, c'_{m-1}$. Repeating this procedure for different elements $\gamma \in \Gamma_1$ will eventually result in $\delta_1, \ldots, \delta_{u+m}$ that span a space of dimension $u + m$. The data we collected can thus be written as

$$\begin{pmatrix} \delta_1 \\ \vdots \\ \delta_{u+m} \end{pmatrix} = M \begin{pmatrix} (a_0, a_0, \ldots, a_{b-1}) \\ \vdots \\ (a_0^{2^{m-1}}, a_0^{2^{m-1}}, \ldots, a_{b-1}^{2^{m-1}}) \\ (a_0 x_0, a_0 x_1, \ldots, a_{b-1} x_{n-1}) \\ \vdots \quad \vdots \\ (a_0 x_0^{2^{u-1}}, a_0 x_1^{2^{u-1}}, \ldots, a_{b-1} x_{n-1}^{2^{u-1}}) \end{pmatrix} \tag{5.4.2}$$

for an invertible $(u + m) \times (u + m)$ matrix M.

Solve Phase: Multiplying Equation (5.4.2) by M^{-1} yields equations that, when viewed as binary equations, are linear in the entries of M^{-1} and the values x_i (as we assume the a_i to be known). The first m rows of M are determined by the (known)

[1] The probability that $(a_0, a_1, \cdots, a_{m-1})$ is an $\mathbb{F}_2$ basis of $\mathbb{F}_{2^m}$ is $\prod_{i=1}^{m-1} \frac{2^m - 2^i}{2^m} \approx 0.288$ for $m \ge 8$.

values of the constants a_i. Thus we are left with $N_u = m(u(u+m)+(u+b))$ unknowns, i.e., the remaining $u(u+m)$ entries of M^{-1} and the $u+b$ points

$$x_0, x_1, x_2, x_4, \ldots, x_{2^{u-1}}, x_t, x_{2t}, x_{3t}, \ldots, x_{(b-1)t}$$

(all other points are given as linear combinations of those). The number of equations is $N_e = mu(u+b)$.

In particular, whenever $b \geq \lceil \frac{um+u}{u-1} \rceil$, we get more equations than unknowns and can hope for a unique solution. We implemented the attack and observed the following.

Fact 13. *Only for the right guess for the constants c_i the system is solvable. In this case the constants x_0 and x_1 could be chosen as arbitrary non-zero elements in $\mathbb{F}_{2^m}$.*

As there are $2^{m(m-1)}$ possibilities for the constants and it takes roughly $(N_e N_u^2)$ binary operations to solve the system, the overall running time of this attack is $2^{m(m-1)} m^3 u(u+b)(u^2+(m+1)u+b)^2$ binary operations. In Table 5.4 we computed the complexity of the attack for different values of m and t, and give the number of blocks needed such that the attack can be applied, i.e., if the code has more blocks that the one in the table, the attack will be effective.

Table 5.4: Complexity of the attack, security level and minimum number of block needed to use the attack for different parameters

m	t	Number of block needed	Complexity of the Attack ($\log_2$)	Assumed security level ($\log_2$)
16	128	61	275	275,4
	256	33	276	300
14	64	112	215	215,6
	128	40	216	217
	256	24	217	231
12	64	55	164	164,78
	128	26	165	171

Note that if we use more blocks that the one in the table 5.4, the security level is bigger, but the complexity of the attack is still the same (since using that many blocks as in the table, we can find the constants, and then find the remaining blocks is not that expensive). This can be observe in table 5.5.

For $m = 8$ and $m = 10$ it is not possible to use the attack. In the first case the minimum number of blocks that we need to applied the attack is too big, and the the code cannot exist. In the second case, the assumed security is always smaller than the complexity of the attack.

Table 5.5: Complexity of the attack, security level

m	t	Number of block needed	Number of block used	Complexity of the Attack ($\log_2$)	Assumed security level ($\log_2$)
16	128	61	62	275	278
			64		284
			70		300
16	256	33	34	276	311
			38		252
			42		389
14	128	40	41	216	226
			46		247
			50		263
14	256	24	25	217	246
			29		301
			32		338
12	128	26	27	165	177
			32		209
			36		230

5.5 An independent attack due to Faugère et al.

Faugère et al. proposed an independent attack in [44] that we will briefly explain in this section. As we showed earlier, given a sequence $X = (X_1, \ldots, X_n) \in \mathbb{F}_{q^m}^n$ of distinct elements and $Y = (Y_1, \ldots, Y_n) \in \mathbb{F}_{q^m}^n$ of non-zero elements, the matrix

$$H = Vdm(r, X) \times Diag(Y)$$

is the parity-check matrix of a Goppa code $\Gamma(X, Y)$. We also know that given the generator matrix G of $\Gamma(X, Y)$ we have that $HG^T = 0$, then if we denote by $g_{i,j}$ the entries of G in the i^{th} row and the j^{th} column, we have that:

$$\Big\{ g_{i,1} Y_1 X_1^e + \cdots + g_{i,n} Y_n X_n^e = 0 | i \in \{1, \ldots, k\}, e \in \{0, \ldots, r-1\} \Big\}. \tag{5.5.1}$$

We also know by Fact 2, that if we are able to find X and Y, we are able to decode $\Gamma(X, Y)$. As McEliece PKC uses Goppa codes, this means that if we are able to solve the system of equation presented in Equation 5.5.1 we are also able to find the secret key in the McEliece PKC. For the original scheme, the system is too large, but for the variants [11] and [85], the structure added in the codes permit to drastically reduce the number of variables; allowing to solve (5.5.1) for a large set of parameters in polynomial-time using dedicated Gröbner bases techniques. This attack allow to recover the key in few seconds for almost all the parameters proposed in [11] and [85], only the binary case of the Quasi-dyadic variant is still not attacked.

Chapter 6

A Distinguisher for high rate McEliece cryptosystem

This chapter investigates the difficulty of the Goppa Code Distinguishing (GCD) problem which first appeared in [34]. This is a decision problem that aims at recognizing in polynomial time a generator matrix of a binary Goppa code from a randomly drawn binary matrix. It is assumed that no polynomial time algorithm exists that distinguishes a generator matrix of a Goppa code from a randomly picked generator matrix.

We present a deterministic polynomial-time distinguisher for high rate codes. It is based on the algebraic attack developed by Faugère et al. against compact variants introduced in [44]. In this approach, the key-recovery problem is transformed into the one of solving an algebraic system (cf. Section 5.5). By using a linearizing technique, we are able to derive a linear system whose rank is different from what one would expect. More precisely, we observe experimentally that this *defect* in the rank is directly related to the type of codes. This chapter is based on the paper [43] that is a joint work with Jean-Charles Faugère, Ayoub Otmani, Ludovic Perret and Jean-Pierre Tillich.

We first define an algebraic distinguisher, then provide explicit formulas that predict the behavior of the distinguisher coming from heavy experimentations. In Section 6.2, we give a proof of its typical behavior in the random case. In Section 6.3 and Section 6.4, we give explanations of the formulas for alternant and binary Goppa codes. And finally, we conclude over the cryptographic implications the distinguisher induces.

6.1 The distinguisher

Keeping the notation from Section 5.5, let $G = (g_{ij})_{\substack{1 \leqslant i \leqslant k \\ 1 \leqslant j \leqslant n}}$ be the generator matrix of the public code. We can assume without loss of generality that G is systematic in its k first positions, such a form can be easily obtained by a Gaussian elimination and by a suitable permutation of the columns. We describe now a simple way of using this particular form for solving (5.5.1). The strategy is as follows: let $P = (p_{ij})_{\substack{1 \leqslant i \leqslant k \\ k+1 \leqslant j \leqslant n}}$ be the submatrix of G formed by its last $n - k = mr$ columns (i.e., $G = (I_k|P)$). For any $i \in \{1, \ldots, k\}$ and $e \in \{0, \ldots, r-1\}$, we can rewrite (5.5.1) as

$$Y_i X_i^e = \sum_{j=k+1}^{n} p_{i,j} Y_j X_j^e. \tag{6.1.1}$$

Because of the trivial identity $Y_iY_iX_i^2 = (Y_iX_i)^2$ and Equation 6.1.1 it follows that

$$\forall i \in \{1,\dots,k\},\ \begin{cases} Y_i & = & \sum_{j=k+1}^{n} p_{i,j}Y_j \\ Y_iX_i & = & \sum_{j=k+1}^{n} p_{i,j}Y_jX_j \\ Y_iX_i^2 & = & \sum_{j=k+1}^{n} p_{i,j}Y_jX_j^2 \end{cases}$$

for all i in $\{1,\dots,k\}$, we get:

$$\sum_{j=k+1}^{n} p_{i,j}Y_j \sum_{j=k+1}^{n} p_{i,j}Y_jX_j^2 = \left(\sum_{j=k+1}^{n} p_{i,j}Y_jX_j\right)^2.$$

It is possible to reorder this to obtain

$$\sum_{j=k+1}^{n-1}\sum_{j'>j}^{n} p_{i,j}p_{i,j'}\left(Y_jY_{j'}X_{j'}^2 + Y_{j'}Y_jX_j^2\right) = 0.$$

We can now linearize this system by letting $Z_{jj'} \stackrel{\text{def}}{=} Y_jY_{j'}X_{j'}^2 + Y_{j'}Y_jX_j^2$. We obtain a system $\mathcal{L}_{\mathbf{P}}$ of k linear equations involving the $Z_{jj'}$'s:

$$\mathcal{L}_{\mathbf{P}} \stackrel{\text{def}}{=} \left\{ \sum_{j=k+1}^{n-1}\sum_{j'>j}^{n} p_{i,j}p_{i,j'}Z_{jj'} = 0 \;\middle|\; i \in \{1,\dots,k\} \right\}. \tag{6.1.2}$$

To solve this system it is necessary that the number of equations is greater than the number of unknowns i.e., $k \geqslant \binom{mr}{2}$ with the hope that the rank of $\mathcal{L}_{\mathbf{P}}$ denoted by $\text{rank}(\mathcal{L}_{\mathbf{P}})$ is almost equal to the number of variables. Observe that the linear systems (6.1.2) have coefficients in $\mathbb{F}_q$ whereas solutions are sought in the extension field $\mathbb{F}_{q^m}$. But the dimension D of the vector space solution of $\mathcal{L}_{\mathbf{P}}$ does not depend on the underlying field because $\mathcal{L}_{\mathbf{P}}$ can always be seen as a system over $\mathbb{F}_{q^m}$. Remark that we obviously have $D = \binom{mr}{2} - \text{rank}(\mathcal{L}_{\mathbf{P}})$.

It appears that D is amazingly large. It even depends on whether or not the code with generator matrix G is chosen as a (generic) alternant code or as a Goppa code. Interestingly enough, when G is chosen at random, $\text{rank}(\mathcal{L}_{\mathbf{P}})$ is equal to $\min\left\{k, \binom{mr}{2}\right\}$ with very high probability. In particular, the dimension of the solution space is typically 0 when k is larger than the number of variables $\binom{mr}{2}$.

Although this *defect* in the rank is an obstacle to break the McEliece cryptosystem, it can be used to distinguish the public generator from a random code. Moreover, since the linear system $\mathcal{L}_{\mathbf{P}}$ is defined over $\mathbb{F}_q$, there exist two vector spaces solution depending on whether the underlying field is $\mathbb{F}_{q^m}$ or $\mathbb{F}_q$. This duality leads to the following definition.

Definition 6.1. *For any integer $r \geqslant 1$ and $m \geqslant 1$, let us denote by $N \stackrel{def}{=} \binom{mr}{2}$ the number of variables in the linear system $\mathcal{L}_{\mathbf{P}}$ as defined in (6.1.2) and D the dimension of the vector space solution of $\mathcal{L}_{\mathbf{P}}$. The* normalized dimension *of $\mathcal{L}_{\mathbf{P}}$ denoted by Δ is defined as:*

$$\Delta \stackrel{def}{=} \frac{D}{m}.$$

We consider three cases corresponding to the possible choices for the entries $p_{i,j}$'s, we denote by Δ_{random} the normalized dimension when the p_{ij}'s are chosen uniformly and independently at random in $\mathbb{F}_q$. When G is chosen as a generator matrix of a random alternant (*resp.* Goppa) code of degree r, we denote the normalized dimension by $\Delta_{\text{alternant}}$ (*resp.* Δ_{Goppa}). Note that in our probabilistic model, a random alternant code is obtained by picking uniformly and independently at random two vectors $(x_1, \ldots, x_n)$ and $(y_1, \ldots, y_n)$ from $(\mathbb{F}_{q^m})^n$ such that the x_i's are all different and the y_i's are all nonzero. A random Goppa code is obtained by also taking in the same way a random vector $(x_1, \ldots, x_n)$ in $(\mathbb{F}_{q^m})^n$ with all the x_i's different and a random *irreducible* polynomial $g(z) = \sum_i \gamma_i z^i$ of degree r.

A thorough experimental study (see Appendix D) through intensive computations with Magma [23] by randomly generating alternant and Goppa codes over the field $\mathbb{F}_q$ with $q \in \{2, 4, 8, 16, 32\}$ for values of r in the range $\{3, \ldots, 50\}$ and several m revealed that the (normalized) dimension of the vector space over $\mathbb{F}_q$ of the solutions of (6.1.2) follows the following formulas. Recall that by definition $N = \binom{mr}{2}$ and $k = n - rm$ where $n \leqslant q^m$.

Experimental Fact 1 (Alternant Case)**.** *As long as $N - m\Delta_{alternant} < k$, with very high probability the normalized dimension $\Delta_{alternant}$ has the following value $T_{alternant}$:*

$$T_{alternant} = \frac{1}{2}(r-1)\left((2e+1)r - 2\frac{q^{e+1}-1}{q-1}\right) \tag{6.1.3}$$

where $e \stackrel{def}{=} \lfloor \log_q(r-1) \rfloor$.

As for the case of random Goppa codes we also obtain formulas different from those of alternant codes. Note however that the Goppa codes are generated by means of a random irreducible $g(z)$ of degree r and hence $g(z)$ has no multiple roots. In particular, we can apply Theorem 2.7 in the binary case.

Experimental Fact 2 (Goppa Case)**.** *As long as $N - m\Delta_{Goppa} < k$, with very high probability the normalized dimension Δ_{Goppa} has the following value T_{Goppa}:*

$$T_{Goppa} = \begin{cases} \frac{1}{2}(r-1)(r-2) = T_{alternant} \text{ for } r < q-1 \\ \\ \frac{1}{2}r\Big((2e+1)r - 2q^e + 2q^{e-1} - 1\Big) \text{ for } r \geqslant q-1 \end{cases} \tag{6.1.4}$$

where e is the unique integer such that:

$$q^e - 2q^{e-1} + q^{e-2} < r \leqslant q^{e+1} - 2q^e + q^{e-1}.$$

Based upon these experimental observations, we are now able to define a *distinguisher* between random codes, alternant codes and Goppa codes. This distinguisher will be in particular useful to distinguish between McEliece public keys and random matrices.

Definition 6.2 (Random Code Distinguisher)**.** *Let m and r be integers such that $m \geqslant 1$ and $r \geqslant 1$. Let G be a $k \times n$ matrix whose entries are in $\mathbb{F}_q$ with $n \leqslant q^m$ and $k \stackrel{def}{=} n - rm$. Without loss of generality, we assume that G is systematic i.e., $G = (I_k \mid P)$. Let $\mathcal{L}_{\mathbf{P}}$ be the linear system associated to G as defined in* (6.1.2)*, and Δ the normalized dimension of $\mathcal{L}_{\mathbf{P}}$. We define the* Random Code Distinguisher $\mathcal{D}$ *as the mapping which takes in input G and outputs b in $\{-1, 0, 1\}$ such that:*

$$\mathcal{D}(G) = \begin{cases} -1 & \text{if } \Delta = T_{alternant} \\ 0 & \text{if } \Delta = T_{Goppa} \\ 1 & \text{otherwise.} \end{cases} \tag{6.1.5}$$

6.2 The random case

The purpose of this section is to study the behavior of D_{random}, namely the dimension of the solution space of $\mathcal{L}_{\mathbf{P}}$ when the entries of the matrix P are drawn independently from the uniform distribution over $\mathbb{F}_q$. In this case, we can show that:

Theorem 6.1. *Assume that $N \leqslant k$ and that the entries of P are drawn independently from the uniform distribution over $\mathbb{F}_q$. Then for any function $\omega(x)$ tending to infinity as x goes to infinity, we have*

$$\mathbf{prob}\left(D_{random} \geqslant mr\omega(mr)\right) = o(1),$$

as mr goes to infinity.

Notice that if we choose $\omega(x) = \log(x)$ for instance, then asymptotically the dimension D_{random} of the solution space is with very large probability smaller than $mr\log(mr)$. When m and r are of the same order (which is generally chosen in practice) this quantity is smaller than $D_{\text{alternant}}$ or D_{Goppa} which are of the form $\Omega(mr^2)$. The main ingredient for proving Theorem 6.1 consists in analyzing a certain (partial) Gaussian elimination process on the matrix

$$M \stackrel{\text{def}}{=} (p_{ij}p_{ij'})_{\substack{1 \leqslant i \leqslant k \\ k+1 \leqslant j < j' \leqslant n}}.$$

We can see the matrix M in block form, each block consists of the matrix $B_j = (p_{i,k+j}p_{i,k+j'})_{\substack{1 \leqslant i \leqslant k \\ 1 \leqslant j < j' \leqslant n-k}}$. Each block B_j is of size $k \times (rm - j)$.

$$M = \underbrace{\left(\begin{array}{ccc|ccc} P_{1,k+1}P_{1,k+2} & \dots & P_{1,k+1}P_{1,k+n} & P_{1,k+2}P_{1,k+3} & \dots & P_{1,n-1}P_{1,n} \\ P_{2,k+1}P_{2,k+2} & \dots & P_{2,k+1}P_{2,k+n} & P_{2,k+2}P_{2,k+3} & \dots & P_{2,n-1}P_{2,n} \\ \vdots & & \vdots & \vdots & & \vdots \\ P_{k,k+1}P_{k,k+2} & \dots & P_{k,k+1}P_{k,k+n} & P_{k,k+2}P_{k,k+3} & \dots & P_{k,n-1}P_{k,n} \end{array}\right)}_{B_1} \underbrace{\qquad\qquad}_{B_{rm-1}} \tag{6.2.1}$$

Notice that in B_j, the rows for which $p_{i,k+j} = 0$ consist only of zeros. To start the Gaussian elimination process with B_1, we will therefore choose $rm - 1$ rows for

which $p_{i,k+1} \neq 0$. This gives a square matrix M_1. We perform Gaussian elimination on M by adding rows involved in M_1 to put the first block B_1 in standard form. We continue this process with B_2 by picking now $rm-2$ rows which have not been chosen before and which correspond to $p_{i,k+2} \neq 0$. This yields a square submatrix M_2 of size $rm-2$ and we continue this process until we reach the last block. The key observation is that:

$$\operatorname{rank}(M) \geqslant \operatorname{rank}(M_1) + \operatorname{rank}(M_2) + \cdots + \operatorname{rank}(M_{rm-1}).$$

A rough analysis of this process yields the Theorem 6.1. The important point is that what happens for different blocks are independent processes and it corresponds to looking at different rows of the matrix P. We give all the previous results that we need in order to prove Theorem 6.1.

It will be convenient to assume that the columns of M are ordered lexicographically. The index of the first column is $(j, j') = (k+1, k+2)$, the second one is $(j, j') = (k+1, k+3)$, while the last one is $(j, j') = (n-1, n)$. The matrices M_i's which are involved in the Gaussian elimination process mentioned above are defined inductively as follows. Let E_1 be the subset of $\{1, \ldots, k\}$ of indices s such that $p_{s,k+1} \neq 0$. Let F_1 be the subset of E_1 formed by its first $rm-1$ elements (if these elements exist). Now , we set

$$M_1 \stackrel{\text{def}}{=} (p_{s,k+1} p_{s,j})_{\substack{s \in F_1 \\ k+1 < j \leq n}}. \tag{6.2.2}$$

Let r_1 be the rank of M_1. To simplify the discussion, we assume that:

1. $F_1 = \{1, 2, \ldots, rm-1\}$,
2. the submatrix N_1 of M_1 formed by its first r_1 rows and columns is of full rank.

Note that we can always assume this by performing suitable row and column permutations. In other words M has the following block structure:

$$M = \begin{pmatrix} N_1 & B_1 \\ A_1 & C_1 \end{pmatrix}.$$

We denote:

$$M^{(1)} \stackrel{\text{def}}{=} \begin{pmatrix} N_1^{-1} & O \\ -A_1 N_1^{-1} & I \end{pmatrix} M,$$

where O is a matrix of size $r_1 \times (k - r_1)$ with only zero entries and I is the identity matrix of size $k - r_1$. Notice that $M^{(1)}$ takes the block form:

$$M^{(1)} = \begin{pmatrix} I & B_1' \\ O & C_1' \end{pmatrix}.$$

This is basically performing Gaussian elimination on M in order to have the first r_1 columns in standard form. We then define inductively the $E_i, F_i, M_i, M^{(i)}$ and N_i as

follows:

$$E_i \stackrel{\text{def}}{=} \{s | 1 \leq s \leq k, p_{s,k+i} \neq 0\} \setminus \bigcup_{u=1}^{i-1} F_{i-u},$$
$$F_i \stackrel{\text{def}}{=} \text{the first } rm-i \text{ elements of } E_i.$$

M_i is the submatrix of $M^{(i-1)}$ obtained from the rows in F_i and the columns associated to the indices of the form $(k+i, j')$ where j' ranges from $k+i+1$ to n. $M^{(i)}$ is obtained from $M^{(i-1)}$ by first choosing a square submatrix N_i of M_i of full rank and with the same rank as M_i and then by performing Gaussian elimination on the rows in order to put the columns of $M^{(i-1)}$ involved in N_i in standard form (i.e., the submatrix of $M^{(i-1)}$ corresponding to N_i becomes the identity matrix while the other entries in the columns involved in N_i become zero). It is clear that the whole process leading to $M^{(rm-1)}$ amounts to perform (partial) Gaussian elimination to M. Hence:

Lemma 6.1. *When $|E_i| \geq rm - i$, for all $i \in \{1, \ldots, rm-1\}$, we have:*

$$\text{rank}(M) \geq \sum_{i=1}^{rm-1} \text{rank}(M_i).$$

Another observation is that M_i is equal to the sum of the submatrix $(p_{s,k+i}p_{s,j})_{\substack{s \in F_i \\ k+i<j\leq n}}$ of M and a certain matrix which is some function on the entries $p_{t,k+i}p_{t,j}$ where t belongs to $F_1 \cup \ldots F_{i-1}$ and j ranges over $\{k+i+1, n\}$. Since by definition of F_i, $p_{s,k+i}$ is different from 0 for s in F_i. In addition, the rank of M_i does not change by multiplying each row of index s by $p_{s,k+i}^{-1}$. Then, it turns out that the rank of M_i is equal to the rank of a matrix which is the sum of the matrix $(p_{s,j})_{\substack{s \in F_i \\ k+i<j\leq n}}$, another matrix depending on the $p_{t,k+i}p_{t,j}$'s (where t ranges over $F_1 \cup \ldots F_{i-1}$) and the $p_{s,k+1}$'s with $s \in F_i$. This proves that:

Lemma 6.2. *Assume that $|E_i| \geq rm - i$ for all $i \in \{1, \ldots, rm-1\}$. Then, the random variables* $\text{rank}(M_i)$ *are independent and* $\text{rank}(M_i)$ *is distributed as the rank of a square matrix of size $rm - i$ with entries drawn independently from the uniform distribution on $\mathbb{F}_q$.*

Another essential ingredient for proving Theorem 6.1 is the following well known lemma (see for instance [31][Theorem 1])

Lemma 6.3. *There exist two positive constants A and B depending on q such that the probability $p(s, \ell)$ that a random $\ell \times \ell$ matrix over $\mathbb{F}_q$ is of rank $\ell - s$ (where the coefficients are drawn independently from each other from the uniform distribution on $\mathbb{F}_q$) satisfies*

$$\frac{A}{q^{s^2}} \leq p(s, \ell) \leq \frac{B}{q^{s^2}}.$$

This enables to control the exponential moments of the defect of a random matrix. For a square matrix M of size $\ell \times \ell$, we define the defect $d(M)$ by $d(M) \stackrel{\text{def}}{=} \ell - \text{rank}(M)$.

Lemma 6.4. *If M is random square matrix whose entries are drawn independently from the uniform distribution over $\mathbb{F}_q$, then there exists some constant K such that for every $\lambda > 0$,*

$$\mathbb{E}\left(q^{\lambda d(\boldsymbol{M})}\right) \leq K q^{\frac{\lambda^2}{4}},$$

$\mathbb{E}(.)$ *denoting the expectation.*

Proof. By using Lemma 6.3, we obtain:

$$\mathbb{E}\left(q^{\lambda d(M)}\right) \leq \sum_{d=0}^{\infty} q^{\lambda d} \frac{B}{q^{d^2}} \leq B \sum_{d=0}^{\infty} q^{\lambda d - d^2}.$$

Observe that the maximum of the function $d \mapsto q^{\lambda d - d^2}$ is reached for $d_0 = \frac{\lambda}{2}$ and is equal to $q^{\frac{\lambda^2}{4}}$. Then, we can write the sum above as:

$$\sum_{d=0}^{\infty} q^{\lambda d - d^2} = \sum_{d \leq d_0} q^{\lambda d - d^2} + \sum_{d > d_0} q^{\lambda d - d^2}$$

Finally, we notice that:

$$\begin{aligned} \frac{q^{\lambda(d+1)-(d+1)^2}}{q^{\lambda d - d^2}} &\leq \frac{q^{\lambda(d_0+1)-(d_0+1)^2}}{q^{\lambda d_0 - d_0^2}} = \frac{1}{q} \text{ for } d > d_0, \\ \frac{q^{\lambda(d-1)-(d-1)^2}}{q^{\lambda d - d^2}} &\leq \frac{q^{\lambda(d_0-1)-(d_0-1)^2}}{q^{\lambda d_0 - d_0^2}} = \frac{1}{q} \text{ for } d \leq d_0. \end{aligned}$$

This leads to:

$$\begin{aligned} \sum_{d=0}^{\infty} q^{\lambda d - d^2} &\leq \sum_{d \leq d_0} q^{d - \lfloor d_0 \rfloor} q^{\frac{\lambda^2}{4}} + \sum_{d > d_0} q^{\lceil d_0 \rceil - d} q^{\frac{\lambda^2}{4}} \\ &= O\left(q^{\frac{\lambda^2}{4}}\right). \end{aligned}$$

□

We can use now the previous lemma together with Lemma 6.1 and Lemma 6.2 to derive

Lemma 6.5. *Assuming that $|E_i| \geq rm - i$ for all $i \in \{1, \ldots, t\}$, we get:*

$$\mathbf{prob}\left(\sum_{i=1}^{t} d(M_i) \geq u\right) \leq K^t q^{-\frac{u^2}{t}}$$

where K is the constant appearing in the previous lemma.

Proof. Let $D \stackrel{\text{def}}{=} \sum_{i=1}^{t} d(\boldsymbol{M}_i)$. Using Markov's inequality:

$$\mathbf{prob}(D \geq u) \leq \frac{\mathbb{E}(q^{\lambda D})}{q^{\lambda u}} \tag{6.2.3}$$

for some well chosen $\lambda > 0$. The exponential moment appearing at the numerator is upper-bounded with the help of the previous lemma and by using the independence of the random variables $q^{\lambda d(\boldsymbol{M}_i)}$, i.e.,:

$$\begin{aligned}
\mathbb{E}(q^{\lambda D}) &= \mathbb{E}\left(q^{\lambda \sum_{i=1}^{t} d(M_i)}\right) \\
&= \prod_{i=1}^{t} \mathbb{E}\left(q^{\lambda d(\boldsymbol{M}_i)}\right) \\
&\leq K^t q^{\frac{t\lambda^2}{4}}. \qquad (6.2.4)
\end{aligned}$$

Using now (6.2.4) in (6.2.3), we obtain $\mathbf{prob}(D \geq \alpha t) \leq K^t \frac{q^{\frac{t\lambda^2}{4}}}{q^{\lambda u}} = K^t q^{\frac{t\lambda^2}{4} - \lambda u}$. We choose $\lambda = \frac{2u}{t}$ to minimize this upper-bound, leading to:

$$\mathbf{prob}(D \geq u) \leq K^t q^{-\frac{u^2}{t}}.$$

□

The last ingredient for proving Theorem 6.1 is a bound on the probability that E_i is too small to construct F_i.

Lemma 6.6. *Let* $u_i \stackrel{def}{=} \binom{mr}{2} - \frac{(2rm-i)(i-1)}{2}$*, then*

$$\mathbf{prob}\left(|E_i| < rm - i \mid \ |F_1| = rm-1, \ldots, |F_{i-1}| = rm - i + 1\right) \leq e^{-2\frac{\left(\frac{q-1}{q}u_i - rm - i + 1\right)^2}{u_i}}$$

Proof. When all the sets F_j are of size $rm - j$ for j in $\{1, \ldots, i-1\}$, it remains $N - \sum_{j=1}^{i-1}(rm - j) = N - \frac{(2rm-i)(i-1)}{2} = u_i$ rows which can be picked up for E_i. Let S_t be the sum of t Bernoulli variables of parameter $\frac{q-1}{q}$. We obviously have

$$\mathbf{prob}\left(|E_i| < rm - i || F_1| = rm-1, \ldots, |F_{i-1}| = rm - i + 1\right) = \mathbf{prob}(S_{u_i} < rm - i)\,.$$

It remains to use the Hoeffding inequality on the binomial tails to finish the proof. □

We are ready now to prove Theorem 6.1:

Proof of Theorem 6.1. Let $u = \lceil \sqrt{mr\omega(mr)} \rceil$. We observe now that if all E_j's are of

size at least $rm - j$ for $j \in \{1, \ldots, u\}$, we can write

$$\begin{aligned} D &= N - \text{rank}(M) \\ &\leq N - \sum_{i=1}^{rm-u} \text{rank}(M_i) \text{ (by Lemma 6.1)} \\ &= \sum_{i=1}^{rm-1} (rm - i) - \sum_{i=1}^{rm-u} \text{rank}(M_i) \\ &= \sum_{i=1}^{rm-u} d(M_i) + \sum_{i=rm-u+1}^{rm-1} (rm - i) \\ &= \sum_{i=1}^{rm-u} d(M_i) + \frac{u(u-1)}{2} \\ &< \sum_{i=1}^{rm-u} d(M_i) + \frac{mr\omega(mr)}{2}. \end{aligned}$$

From this we deduce that

$$\mathbf{prob}(D_{\text{random}} \geq mr\omega(mr)) \leq \mathbf{prob}(A \cup B) \leq \mathbf{prob}(A) + \mathbf{prob}(B)$$

where A is the event "$\sum_{i=1}^{rm-u} d(M_i) \geq \frac{mr\omega(mr)}{2}$" and B is the event "for at least one E_j with $j \in \{1, \ldots, rm-u\}$ we have $|E_j| < rm - j$". We use now Lemma 6.5 to prove that $\mathbf{prob}(A) = o(1)$ as rm goes to infinity. We finish the proof by noticing that the probability of the complementary set of B satisfies

$$\begin{aligned} \mathbf{prob}(\bar{B}) &= \mathbf{prob}\left(\bigcap_{i=1}^{rm-u} |E_i| \geq rm - i \right) \\ &= \prod_{i=1}^{rm-u} \mathbf{prob}\left(|E_i| \geq rm - i \,||F_1| = rm - 1, \ldots, |F_{i-1}| = rm - i + 1 \right) \\ &= 1 - o(1) \text{ (by Lemma 6.6).} \end{aligned}$$

□

6.3 The alternant case

We first consider the case of alternant codes over $\mathbb{F}_q$ of degree r. The goal of this section is to identify a set of vectors which, after decomposition according to a basis of $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$, provides a basis of the solution space of $\mathcal{L}_{\mathbf{P}}$. First observe that to set up the linear system $\mathcal{L}_{\mathbf{P}}$ as defined in (6.1.2), we have used the trivial identity $Y_iY_iX_i^2 = (Y_iX_i)^2$. Actually, we can use any identity $Y_iX_i^aY_iX_i^b = Y_iX_i^cY_iX_i^d$ with $a, b, c, d \in \{0, 1, \ldots, r-1\}$ such that $a + b = c + d$. It is straightforward to check that we obtain the same algebraic system $\mathcal{L}_{\mathbf{P}}$ with:

$$\sum_{j=k+1}^{n} \sum_{j'>j} p_{i,j}p_{i,j'} \left(Y_jX_j^aY_{j'}X_{j'}^b + Y_{j'}X_{j'}^aY_jX_j^b + Y_jX_j^cY_{j'}X_{j'}^d + Y_{j'}X_{j'}^cY_jX_j^d \right) = 0. \tag{6.3.1}$$

So, the fact that *there are many different ways of combining the equations of the algebraic system together yielding the same linearized system* $\mathcal{L}_{\mathbf{P}}$ explains why the dimension of the vector space solution V_{q^m} is large.

For larger values of r, the automorphisms of $\mathbb{F}_{q^m}$ of the kind $x \mapsto x^{q^\ell}$ for some $\ell \in \{0, \ldots, m-1\}$ can be used to obtain the identity but the decomposition over $\mathbb{F}_q$ of the entries of vectors obtained from such equations give vectors that are dependent of those coming from the identity $Y_iX_i^aY_i^{q^{\ell-\ell'}}X_i^{bq^{\ell-\ell'}} = Y_iX_i^cY_i^{q^{\ell-\ell'}}X_i^{dq^{\ell-\ell'}}$ if we assume $\ell' \leqslant \ell$. Therefore, we are only interested in vectors that satisfy equations obtained with $0 \leqslant a, b, c, d < r$, $0 \leqslant \ell < m$ and $a + q^\ell b = c + q^\ell d$.

Definition 6.3. *Let a, b, c and d be integers in $\{0, \ldots, r-1\}$ and an integer ℓ in $\{0, \ldots, \lfloor \log_q(r-1) \rfloor\}$ such that $a + q^\ell b = c + q^\ell d$. We define*

$$\boldsymbol{Z}_{a,b,c,d,\ell} \stackrel{def}{=} \left(\boldsymbol{Z}_{a,b,c,d,\ell}[j, j'] \right)_{k+1 \leqslant j < j' \leqslant n}$$

where

$$\boldsymbol{Z}_{a,b,c,d,\ell}[j, j'] \stackrel{def}{=} Y_jX_j^aY_{j'}^{q^\ell}X_{j'}^{q^\ell b} + Y_{j'}X_{j'}^aY_j^{q^\ell}X_j^{q^\ell b} + Y_jX_j^cY_{j'}^{q^\ell}X_{j'}^{q^\ell d} + Y_{j'}X_{j'}^cY_j^{q^\ell}X_j^{q^\ell d},$$

for any j and j' satisfying $k + 1 \leqslant j < j' \leqslant n$.

Without loss of generality, we can assume that $d > b$ and set $\delta = d - b$. Moreover, as we have $a + q^\ell b = c + q^\ell d$, it implies that $a = c + q^\ell \delta$. Note that any vector $\boldsymbol{Z}_{a,b,c,d,\ell}$ is uniquely described by the tuple (b, c, δ, ℓ) by setting $d = b + \delta$ and $a = c + q^\ell\delta$ provided that $1 \leqslant \delta \leqslant r - 1 - b$ and $0 \leqslant c + q^\ell\delta \leqslant r - 1$.

The next proposition shows that some vectors $\boldsymbol{Z}_{c+q^\ell\delta,b,c,b+\delta,\ell}$ can be expressed as a linear combination of vectors defined with $\delta = 1$.

Proposition 6.1. *Let ℓ, δ, b and c be integers such that $\ell \geqslant 0$, $\delta \geqslant 1$, $1 \leqslant b+\delta \leqslant r-1$ and $1 \leqslant c + q^\ell\delta \leqslant r - 1$. Let us assume that $\delta \geqslant 2$ and let $b_i \stackrel{def}{=} b + i - 1$ and $c_i \stackrel{def}{=} c + q^\ell(\delta - i)$. We have*

$$\boldsymbol{Z}_{c+q^\ell\delta,b,c,b+\delta,\ell} = \sum_{i=1}^{\delta} \boldsymbol{Z}_{c_i+q^\ell,b_i,c_i,b_i+1,\ell}. \tag{6.3.2}$$

From Proposition 6.1, we deduce that the set of vectors $\boldsymbol{Z}_{c+q^\ell\delta,b,c,b+1,\ell}$ i.e., $\delta = 1$ form a spanning set for the vector space generated by all the vectors $\boldsymbol{Z}_{c+q^\ell\delta,b,c,b+\delta,\ell}$. To prove Proposition 6.1, we require the following lemma.

Lemma 6.7. *For any integers a, b, c, d, e, f in $\{0,\ldots,r-1\}$, and an integer ℓ in $\left\{0,\ldots,\lfloor\log_q(r-1)\rfloor\right\}$ such that $a+q^\ell b = c+q^\ell d$ we have:*

$$\boldsymbol{Z}_{a,b,c,d,\ell} + \boldsymbol{Z}_{c,d,e,f,\ell} = \boldsymbol{Z}_{a,b,e,f,\ell} \tag{6.3.3}$$

Proof of Proposition 6.1. Let $b^* \stackrel{\text{def}}{=} b+1$, $\delta^* \stackrel{\text{def}}{=} \delta - 1$ and $c^* \stackrel{\text{def}}{=} c + q^\ell\delta^*$. Then c^* is the integer such that $c^* + q^\ell = c + q^\ell\delta$, one can see that $c + q^\ell\delta^* = c + q^\ell(\delta-1) = c^*$ and by Lemma 6.7 we have:

$$\boldsymbol{Z}_{c^*+q^\ell,b,c^*,b+1,\ell} + \boldsymbol{Z}_{c+q^\ell\delta^*,b^*,c,b^*+\delta^*,\ell} = \boldsymbol{Z}_{c^*+q^\ell,b,c,b^*+\delta^*,\ell} = \boldsymbol{Z}_{c+q^\ell\delta,b,c,b+\delta,\ell}$$

which means that

$$\boldsymbol{Z}_{c+q^\ell\delta,b,c,b+\delta,\ell} = \boldsymbol{Z}_{c^*+q^\ell,b,c^*,b+1,\ell} + \boldsymbol{Z}_{c+q^\ell\delta^*,b^*,c,b^*+\delta^*,\ell} \tag{6.3.4}$$

The proof follows by induction. □

We can characterize more precisely the set of vectors $\boldsymbol{Z}_{c+q^\ell\delta,b,c,b+1,\ell}$ i.e., $\delta = 1$:

Definition 6.4. *Let $\mathcal{B}_r$ be the set of* nonzero *vectors $\boldsymbol{Z}_{c+q^\ell\delta,b,c,b+\delta,\ell}$ obtained with tuples (δ,b,c,ℓ) such that $\delta = 1$ and satisfying the following conditions:*

$$\begin{cases} 0 \leqslant b \leqslant r-2 \text{ and } 0 \leqslant c \leqslant r-1-q^\ell & \text{if} \quad 1 \leqslant \ell \leqslant \lfloor\log_q(r-1)\rfloor \\ 0 \leqslant b < c \leqslant r-2 & \text{if} \quad \ell = 0. \end{cases}$$

Proposition 6.2. *Let r be an integer such that $r \geqslant 3$. The cardinality of $\mathcal{B}_r$ is equal to $T_{alternant}$.*

Proof. Let us set $e \stackrel{\text{def}}{=} \lfloor\log_q(r-1)\rfloor$. Then the number of elements in $\mathcal{B}_r$ is given by the number of tuples (b,c,ℓ). Therefore we get:

$$\begin{aligned} |\mathcal{B}_r| &= \frac{1}{2}(r-1)(r-2) + \sum_{\ell=1}^{e}\sum_{b=0}^{r-2}(r-q^\ell) = \frac{1}{2}(r-1)\left(r-2+2er-2\sum_{\ell=1}^{e}q^\ell\right) \\ &= \frac{1}{2}(r-1)\left((2e+1)r-2\sum_{\ell=0}^{e}q^\ell\right) = T_{\text{alternant}} \end{aligned}$$

□

Proposition 6.2 gives an explanation of the value of $D_{\text{alternant}}$. To see this, let us introduce the following definition:

Definition 6.5. *Consider a certain decomposition of the elements of $\mathbb{F}_{q^m}$ in a $\mathbb{F}_q$ basis. Let $\pi_i : \mathbb{F}_{q^m} \mapsto \mathbb{F}_q$ be the function giving the i-th coordinate in this decomposition. By extension we denote for a vector $\boldsymbol{z} = (z_j)_{1\leq j\leq n} \in \mathbb{F}_{q^m}^n$ by $\pi_i(\boldsymbol{z})$ the vector $(\pi_i(z_j))_{1\leq j\leq n} \in \mathbb{F}_q^n$.*

We have the following heuristic.

Heuristic 1. *For random choices of x_i's and y_i's with $1 \leqslant i \leqslant n$ the set $\{\pi_i(\boldsymbol{Z}) | 1 \leq i \leq m, \boldsymbol{Z} \in \mathcal{B}_r\}$ forms a basis of the vector space of solution of $\mathcal{L}_{\mathbf{P}}$.*

6.4 The binary Goppa case

In this section we will explain Experimental Fact 2 in the case of a binary Goppa code. We denote by r the degree of the Goppa polynomial. In this case, it is readily seen that the theoretical expression T_{Goppa} has a simpler expression given by

Proposition 6.3. *Let us define* $e \stackrel{def}{=} \lceil \log_2 r \rceil + 1$ *and* $N \stackrel{def}{=} \binom{mr}{2}$. *When* $q = 2$, *the formula in Equation* (6.1.4) *can be simplified to* $T_{Goppa} = \frac{1}{2} r \Big((2e+1)r - 2^e - 1 \Big)$.

Theorem 2.7 shows that a binary Goppa code of degree r can be regarded as a binary alternant code of degree $2r$. This seems to indicate that we should have

$$D_{\text{Goppa}}(r) = m T_{\text{alternant}}(2r).$$

This is not the case however. It turns out that $D_{\text{Goppa}}(r)$ is significantly smaller than this. In our experiments, we have found out that the vectors of $\mathcal{B}_{2r}$ still form a generating set for $\mathcal{L}_{\mathbf{P}}$, but that they are not independent anymore.

We are really interested in the dependencies over the binary field $\mathbb{F}_2$, but we are first going to find linear relations over the extension field $\mathbb{F}_{2^m}$. There are many of them, as shown by the following proposition which exploits that the Y_i's are derived from the Goppa polynomial $g(z)$ by $Y_i = g(X_i)^{-1}$.

Proposition 6.4. *Let* t, ℓ *and* c *be integers such that* $0 \leqslant t \leqslant r-2$, $1 \leqslant \ell \leqslant \lfloor \log_2(2r-1) \rfloor$ *and* $0 \leqslant c \leqslant 2r - 2^\ell - 1$. *We set* $c^* \stackrel{def}{=} c + 2^{\ell-1}$. *It holds that:*

$$\sum_{b=0}^{r} \gamma_b^{2^\ell} \boldsymbol{Z}_{c+2^\ell, t+b, c, t+b+1, \ell} = \boldsymbol{Z}_{c^*+2^{\ell-1}, 2t, c^*, 2t+1, \ell-1} + \boldsymbol{Z}_{c+2^{\ell-1}, 2t+1, c, 2t+2, \ell-1}. \quad (6.4.1)$$

Propositon 6.4 which needs Lemma 6.8 is actually a particular case of Proposition 6.5.

Lemma 6.8. *Let* ℓ, δ, b *and* c *be integers such that* $\ell \geqslant 0$, $\delta \geqslant 1$, $1 \leqslant b + \delta \leqslant r-1$, $1 \leqslant c + q^\ell \delta \leqslant r-1$. *We have for any* j *and* j' *such that* $k+1 \leqslant j < j' \leqslant n$:

$$\boldsymbol{Z}_{c+q^\ell\delta, b, c, b+\delta, \ell}[j, j'] = \left(X_j^\delta + X_{j'}^\delta \right)^{q^\ell} \left(Y_j X_j^c \left(Y_{j'} X_{j'}^b \right)^{q^\ell} + Y_{j'} X_{j'}^c \left(Y_j X_j^b \right)^{q^\ell} \right) \quad (6.4.2)$$

Proof. Let $d = b + \delta$ and $a = c + q^\ell \delta$. We can write that:

$$\begin{aligned} \boldsymbol{Z}_{c+q^\ell\delta, b, c, b+\delta, \ell}[j, j'] &= \boldsymbol{Z}_{a, b, c, d, \ell}[j, j'] \\ &= Y_j Y_{j'}^{q^\ell} \left(X_j^a X_{j'}^{q^\ell b} + X_j^c X_{j'}^{q^\ell d} \right) + Y_{j'} Y_j^{q^\ell} \left(X_{j'}^a X_j^{q^\ell b} + X_{j'}^c X_j^{q^\ell d} \right) \\ &= Y_j Y_{j'}^{q^\ell} X_{j'}^{q^\ell b} \left(X_j^a + X_j^c X_{j'}^{q^\ell \delta} \right) + Y_{j'} Y_j^{q^\ell} X_j^{q^\ell b} \left(X_{j'}^a + X_{j'}^c X_j^{q^\ell \delta} \right) \end{aligned}$$

Using the identity $a = c + q^\ell \delta$, we also have:

$$\begin{aligned} \boldsymbol{Z}_{c+q^\ell\delta, b, c, b+\delta, \ell}[j, j'] &= Y_j Y_{j'}^{q^\ell} X_{j'}^{q^\ell b} X_j^c \left(X_j^{q^\ell \delta} + X_{j'}^{q^\ell \delta} \right) + Y_{j'} Y_j^{q^\ell} X_j^{q^\ell b} X_{j'}^c \left(X_{j'}^{q^\ell \delta} + X_j^{q^\ell \delta} \right) \\ &= \left(X_j^{q^\ell \delta} + X_{j'}^{q^\ell \delta} \right) \left(Y_j Y_{j'}^{q^\ell} X_{j'}^{q^\ell b} X_j^c + Y_{j'} Y_j^{q^\ell} X_j^{q^\ell b} X_{j'}^c \right) \end{aligned}$$

□

Proposition 6.5. *Let t, ℓ, δ and c be integers such that $t \geqslant 0$, $\ell \geqslant 1$, $\delta \geqslant 1$, $t+\delta \leqslant r-1$, $c \geqslant 0$ and $c+2^\ell\delta \leqslant 2r-1$. We have:*

$$\sum_{b=0}^{r} \gamma_b^{2^\ell} \boldsymbol{Z}_{c+2^\ell\delta,t+b,c,t+b+\delta,\ell} = \boldsymbol{Z}_{c'+2^{\ell'}\delta',b',c',b'+\delta',\ell'} \tag{6.4.3}$$

where $\ell' = \ell-1$, $\delta' = 2\delta$, $b' = 2t$, $c' = c$.

Proof. By Lemma 6.8, we have that:

$$\begin{aligned}\boldsymbol{Z}_{c+2^\ell\delta,t+b,c,t+b+\delta,\ell}[j,j'] &= \left(X_j^\delta + X_{j'}^\delta\right)^{2^\ell}\left(Y_j X_j^c Y_{j'}^{2^{\ell-1}} X_{j'}^{2^\ell t}\left(Y_{j'}X_{j'}^{2b}\right)^{2^{\ell-1}}\right) + \\ &\quad \left(X_j^\delta + X_{j'}^\delta\right)^{2^\ell}\left(Y_{j'} X_{j'}^c Y_j^{2^{\ell-1}} X_j^{2^\ell t}\left(Y_j X_j^{2b}\right)^{2^{\ell-1}}\right)\end{aligned}$$

Using the fact that $Y_j \sum_{b=0}^r \gamma_b^2 X_j^{2b} = 1$ and $Y_{j'} \sum_{b=0}^r \gamma_b^2 X_{j'}^{2b} = 1$ we also have:

$$\begin{aligned}\sum_{b=0}^{r} \gamma_b^{2^\ell} \boldsymbol{Z}_{c+2^\ell\delta,t+b,c,t+b+\delta,\ell}[j,j'] &= \left(X_j^\delta + X_{j'}^\delta\right)^{2^\ell}\left(Y_j X_j^c Y_{j'}^{2^{\ell-1}} X_{j'}^{2^\ell t}\left(Y_{j'}\sum_{b=0}^{r}\gamma_b^2 X_{j'}^{2b}\right)^{2^{\ell-1}}\right) \\ &+ \left(X_j^\delta + X_{j'}^\delta\right)^{2^\ell}\left(Y_{j'} X_{j'}^c Y_j^{2^{\ell-1}} X_j^{2^\ell t}\left(Y_j\sum_{b=0}^{r}\gamma_b^2 X_j^{2b}\right)^{2^{\ell-1}}\right) \\ &= \left(X_j^{2\delta} + X_{j'}^{2\delta}\right)^{2^{\ell-1}}\left(Y_j X_j^c\left(Y_{j'}X_{j'}^{2t}\right)^{2^{\ell-1}} + Y_{j'}X_{j'}^c\left(Y_j X_j^{2t}\right)^{2^{\ell-1}}\right) \\ &= \boldsymbol{Z}_{c'+2^{\ell'}\delta',b',c',b'+\delta',\ell'}[j,j']\end{aligned}$$

with $\ell' = \ell-1$, $\delta' = 2\delta$, $b' = 2t$, $c' = c$. Since $c'+2^{\ell'}\delta' = c+2^\ell\delta$ and $c+2^\ell\delta \leqslant 2r-1$ we have $c'+2^{\ell'}\delta' \leqslant 2r-1$. Moreover, we require $b'+\delta' \leqslant 2r-1$ which means $2(t+\delta) \leqslant 2r-1$. This last inequality implies $t+\delta \leqslant r-1$. □

Proof of Proposition 6.4. By Proposition 6.5 when $\delta = 1$ we have the following equality:

$$\sum_{b=0}^{r} \gamma_b^{2^\ell} \boldsymbol{Z}_{c+2^\ell,t+b,c,t+b+1,\ell} = \boldsymbol{Z}_{c+2^\ell,2t,c,2(t+1),\ell-1}$$

Moreover by Proposition 6.1, we also have:

$$\boldsymbol{Z}_{c+2^\ell,2t,c,2(t+1),\ell-1} = \boldsymbol{Z}_{c^*+2^{\ell-1},2t,c^*,2t+1,\ell-1} + \boldsymbol{Z}_{c+2^{\ell-1},2t+1,c,2t+2,\ell-1}$$

where by definition c^* is equal to $c+2^{\ell-1}$. □

As a consequence of Proposition 6.4, $\mathcal{B}_{2r}$ can not be a basis of the linearized system in the Goppa case. We count the number of such equations in the following proposition.

Proposition 6.6. *The number N_L of equations of the form* (6.4.1) *is equal to* $2(r-1)\left(ru+1-2^u\right)$ *where* $u \stackrel{def}{=} \lfloor \log_2(2r-1) \rfloor$.

Proof of Proposition 6.6. Each equation is defined by a triple (t, c, ℓ). As $0 \leqslant t \leqslant r-2$, $1 \leqslant \ell \leqslant u$ and $0 \leqslant c \leqslant 2r - 2^\ell - 1$, we therefore have:

$$N_L = \sum_{t=0}^{r-2} \sum_{\ell=1}^{u} (2r - 2^\ell).$$

One can easily check that this expression is exactly the same as given in the proposition. □

Notice that each equation of the form (6.4.1) involves one vector of $\mathcal{B}_{2r}$ that does not satisfy the other equations. These equations are therefore independent and by denoting by $< \mathcal{B}_{2r} >_{\mathbb{F}_{2^m}}$ the vector space over $\mathbb{F}_{2^m}$ generated by the vectors of $\mathcal{B}_{2r}$ we should have

$$\dim < \mathcal{B}_{2r} >_{\mathbb{F}_{2^m}} \leq |\mathcal{B}_{2r}| - N_L.$$

The experiments we have made indicate that actually equality holds here. However, this does not mean that the dimension of the vector space over $\mathbb{F}_2$ generated by the set $\{\pi_i(\boldsymbol{Z}), \boldsymbol{Z} \in \mathcal{B}_{2r}, 1 \leq i \leq m, \boldsymbol{Z} \in \mathcal{B}_{2r}\}$ is equal to $m \dim < \mathcal{B}_{2r} >_{\mathbb{F}_{2^m}}$. It turns out that there are still other dependencies among the $\pi_i(\boldsymbol{Z})$'s. The following proposition gives an explanation of how such dependencies occur.

Proposition 6.7. *Let* $\boldsymbol{Q}_{a,b,c,d,\ell} \stackrel{def}{=} \left(\boldsymbol{Q}_{a,b,c,d,\ell}[j,j']\right)_{k+1 \leqslant j < j' \leqslant n}$, *with* $\boldsymbol{Q}_{a,b,c,d,\ell}[j,j'] = \left(\boldsymbol{Z}_{a,b,c,d,\ell}[j,j']\right)^2$. *For any integers* $b \geqslant 0$, $t \geqslant 0$, $\delta \geqslant 1$ *and* ℓ *such that* $0 \leqslant \ell \leqslant \lfloor \log_2(2r-1) \rfloor - 1$, $b + \delta \leq 2r - 1$ *and* $t + 2^\ell \delta \leqslant r - 1$, *we have*

$$\boldsymbol{Z}_{2t+2^{\ell+1}\delta,b,2t,b+\delta,\ell+1} = \sum_{c=0}^{r} \gamma_c^2 \boldsymbol{Q}_{c+2^\ell\delta,b,t+c,b+\delta,\ell}. \tag{6.4.4}$$

Proof of Proposition 6.7. For any j and j' such that $k+1 \leqslant j < j' \leqslant n$, we have:

$$\begin{aligned}
&\sum_{c=0}^{r} \gamma_c^2 \left(\boldsymbol{Z}_{c+2^\ell\delta,b,t+c,b+\delta,\ell}\right)^2 [j,j'] = \\
&\left(X_j^\delta + X_{j'}^\delta\right)^{2^{\ell+1}} \left(\left(Y_{j'} X_{j'}^b\right)^{2^{\ell+1}} X_j^{2t} Y_j^2 \sum_{c=0}^{r} \gamma_c^2 X_j^{2c} + \left(Y_j X_j^b\right)^{2^{\ell+1}} X_{j'}^{2t} Y_{j'}^2 \sum_{c=0}^{r} \gamma_c^2 X_{j'}^{2c} \right) \\
= \quad &\left(X_j^\delta + X_{j'}^\delta\right)^{2^{\ell+1}} \left(\left(Y_{j'} X_{j'}^b\right)^{2^{\ell+1}} Y_j X_j^{2t} + \left(Y_j X_j^b\right)^{2^{\ell+1}} Y_{j'} X_{j'}^{2t} \right) = \boldsymbol{Z}_{c'+2^{\ell'}\delta,b',c',b'+\delta',\ell'}[j,j']
\end{aligned}$$

with $\ell' = \ell + 1$, $\delta' = \delta$, $b' = b$, $c' = 2t$ and $c' + 2^{\ell'}\delta' = 2t + 2^{\ell+1}\delta$. In particular, one can easily check that the necessary conditions are $b + \delta \leq 2r - 1$ and $t + 2^\ell \delta \leqslant r - 1$ in order for this equation to hold. □

Proposition 6.8. *The number N_Q of vectors of $\mathcal{B}_{2r}$ satisfying Equation* (6.4.4) *is equal to* $(2r-1)(ru-2^u+1)$ *where* $u \stackrel{def}{=} \lfloor \log_2(2r-1) \rfloor$.

Proof of Proposition 6.8. By Proposition 6.7 we know that N_Q is the number of vectors $\boldsymbol{Z}_{2t+2^{\ell+1}\delta,b,2t,b+\delta,\ell+1}$ obtained with $\delta = 1$, $b \geqslant 0$, $t \geqslant 0$ and satisfying $0 \leqslant \ell \leqslant u-1$, $b+\delta \leq 2r-1$ and $t+2^\ell\delta \leqslant r-1$. Therefore we have:

$$N_Q = \sum_{l=0}^{u-1} \sum_{t=0}^{r-1-2^\ell} (2r-1) \tag{6.4.5}$$

which is equal to the desired expression. □

Each of such equations gives rise to m linear equations over $\mathbb{F}_2$ involving the $\pi_i(\boldsymbol{Z})$ for $\boldsymbol{Z}$ in $\mathcal{B}_{2r}$. Therefore, it could be expected that $\Delta_{\text{Goppa}} = |\mathcal{B}_{2r}| - N_L - N_Q$. But, some vectors in $\mathcal{B}_{2r}$ appear both in linear relations of the form (6.4.1) and "quadratic" equations of the form (6.4.4). More precisely, let $\mathcal{B}_{2r}^{\text{quad}}$ be the subset of vectors of $\mathcal{B}_{2r}$ which are involved in an Equation of type (6.4.4). There are equations of type (6.4.1) which involve only vectors of $\mathcal{B}_{2r}^{\text{quad}}$. Let N_1 be their numbers. Moreover, it is possible by adding two equations of type (6.4.1) involving at least one vector which is not in $\mathcal{B}_{2r}^{\text{quad}}$ to obtain an equation which involves only vectors of $\mathcal{B}_{2r}^{\text{quad}}$. Let N_0 be the number of such sums. Finally, let $N_{L\cap Q} \stackrel{\text{def}}{=} N_1 + N_0$. It is possible to count such equations to obtain

Proposition 6.9. $N_{L\cap Q} = (r-1)\Big((u-\frac{1}{2})r - 2^u + 2\Big)$ *where* $u \stackrel{def}{=} \lfloor \log_2(2r-1) \rfloor$.

Proof of Proposition 6.9. We will consider vectors $\boldsymbol{Z}_{c+2^\ell,b,c,b+1,\ell}$ of $\mathcal{B}_{2r}$ that satisfy Equation (6.4.4) and such that there exists a linear relation that link them. In other words, we consider all the linear relations of the form $\sum_i \alpha_i \boldsymbol{Z}_{c_i+2^{\ell_i},b_i,c_i,b_i+1,\ell_i} = 0$ with α_i in $\mathbb{F}_{2^m}$ and where each $\boldsymbol{Z}_{c_i+2^{\ell_i},b_i,c_i,b_i+1,\ell_i}$ is equal to a linear relation of the form (6.4.4). We will see that the number of *independent* equations is equal to $N_{L\cap Q}$. First, one can observe that for any such vectors we necessary have c_i even and $1 \leqslant \ell_i \leqslant u$. We also know by Proposition 6.4 that for any integers t, ℓ and c such that $0 \leqslant t \leqslant r-2$, $1 \leqslant \ell \leqslant u$ and $0 \leqslant c \leqslant 2r-2^\ell-1$, we have the following linear relation:

$$\sum_{b=0}^{r} \gamma_b^{2^\ell} \boldsymbol{Z}_{c+2^\ell,t+b,c,t+b+1,\ell} = \boldsymbol{Z}_{c^*+2^{\ell-1},2t,c^*,2t+1,\ell-1} + \boldsymbol{Z}_{c+2^{\ell-1},2t+1,c,2t+2,\ell-1}$$

where by definition $c^* = c + 2^{\ell-1}$. Note in particular that whenever c is even then c^* is also even and if $\ell \geqslant 2$ then we obtain a linear relation between some vectors that also satisfy quadratic equations of the form (6.4.4). Each equation enables to remove one quadratic equation. So if we denote by N_1 the number of equations of the form (6.4.1) with c even and $\ell \geqslant 2$, we have then:

$$N_1 = \sum_{t=0}^{r-2} \sum_{\ell=2}^{u} \left(\frac{1}{2}(2r-2^\ell)\right) = (r-1)\sum_{\ell=1}^{u-1}(r-2^\ell) = (r-1)\Big((u-1)r - 2^u + 2\Big). \tag{6.4.6}$$

Moreover in the case $\ell = 1$ Equation (6.4.3) becomes

$$\sum_{b=0}^{r} \gamma_b^2 \boldsymbol{Z}_{c+2,t+b,c,t+b+1,1} = \boldsymbol{Z}_{c+2,2t,c,2t+2,0}.$$

In particular when c is even, say for instance $c = 2t'$ for some integer, then this last equation can be rewritten as:

$$\sum_{b=0}^{r} \gamma_b^2 \boldsymbol{Z}_{2t'+2,t+b,2t',t+b+1,1} = \boldsymbol{Z}_{2t'+2,2t,2t',2t+2,0}. \tag{6.4.7}$$

We know that when $t' = t$ then $\boldsymbol{Z}_{2t'+2,2t,2t',2t+2,0}$ is zero. In that case we obtain new relations between vectors satisfying quadratic equations that are independent even from those obtained with $\ell \geqslant 2$. As for the case when $t \neq t'$ we also have $\boldsymbol{Z}_{2t'+2,2t,2t',2t+2,0} = \boldsymbol{Z}_{2t+2,2t',2t,2t'+2,0}$. From this identity and from Equation (6.4.7) we then obtain new relations of the following form:

$$\sum_{b=0}^{r} \gamma_b^2 \boldsymbol{Z}_{2t'+2,t+b,2t',t+b+1,1} = \sum_{b=0}^{r} \gamma_b^2 \boldsymbol{Z}_{2t+2,t'+b,2t,t'+b+1,1} \tag{6.4.8}$$

This last equation involves only vectors that satisfy also quadratic equations. So the number N_0 of equations of the form (6.4.8) is given by the number of sets $\{t, t'\}$. But by assumption t and t' should satisfy $0 \leqslant t \leqslant r-2$ and $c = 2t'$ with $0 \leqslant c \leqslant 2r-3$, which implies that $0 \leqslant t' \leqslant r-2$. Therefore, N_0 is equal to the number (t, t') such that $t \leqslant t'$ and thus we get:

$$N_0 = \sum_{t=0}^{r-2} \sum_{t'=t}^{r-2} = \frac{1}{2}(r-1)r. \tag{6.4.9}$$

Finally, by gathering all the cases we therefore obtain that:

$$N_{L\cap Q} = N_1 + N_0 = (r-1)\Big((u-1)r - 2^u + 2\Big) + \frac{1}{2}(r-1)r.$$

□

Proposition 6.10. *For any integer $r \geqslant 2$, we have $T_{Goppa}(r) = |\mathcal{B}_{2r}| - N_L - N_Q + N_{L\cap Q}$.*

Proof. Set $u \stackrel{\text{def}}{=} \lfloor \log_2(2r-1) \rfloor$. From Equation (6.1.3), we have

$$|\mathcal{B}_{2r}| = (2r-1)\left((2u+1)r - 2^{u+1} + 1\right)$$

which implies from Proposition 6.8

$$\begin{aligned} |\mathcal{B}_{2r}| - N_Q &= (2r-1)\left((2u+1)r - 2^{u+1} + 1 - (ru - 2^u + 1)\right) \\ &= (2r-1)((u+1)r - 2^u). \end{aligned}$$

Moreover, from Proposition 6.6 and Proposition 6.9, we can write:

$$\begin{aligned} N_L - N_{L\cap Q} &= (r-1)\Big(2ur + 2 - 2^{u+1} - (ur - \frac{r}{2} - 2^u + 2)\Big) \\ &= (r-1)\Big((u+\frac{1}{2})r - 2^u\Big) \end{aligned}$$

Therefore by gathering all these equalities we obtain:

$$|\mathcal{B}_{2r}| - (N_L + N_Q - N_{L\cap Q}) = r\left((u+\frac{3}{2})r - 2^u - \frac{1}{2}\right)$$

On the other hand from Proposition 6.3, we have $T_{\text{Goppa}}(r) = \frac{1}{2}r\left((2e+1)r - 2^e - 1\right)$ where $e = \lceil \log_2 r \rceil + 1$. Using the basic inequality $2r - 1 < 2r < 2(2r-1)$, we have therefore $\log_2(2r-1) < \log_2(r) + 1 < \log_2(2r-1) + 1$ which finally implies $\lceil \log_2 r \rceil = u$. Thus, $T_{\text{Goppa}}(r) = \frac{1}{2}r\left((2u+3)r - 2^{u+1} - 1\right)$ and the proposition is proved. □

6.5 Conclusion and cryptographic implications

The existence of a distinguisher for the specific case of binary Goppa codes has consequences for code-based cryptographic primitives because it represents, and by far, the favorite choice in such primitives. We focus in this part on secure parameters that are within the range of validity of our distinguisher. The simple expression given in Proposition 6.3 is not valid for any value of r and m but tends to be true for codes that have a code rate $\frac{n-mr}{n}$ that is close to one. This kind of codes are mainly encountered with the public keys of the CFS signature scheme [34].

If we assume that the length n is equal to 2^m and we denote by $r_{\min}$ the smallest integer r such that $N - mT_{\text{Goppa}} \geqslant 2^m - mr$ then any binary Goppa code defined of degree $r \geqslant r_{\min}$ cannot be distinguished from a random linear code by our technique. This value is gathered in Table 6.1.

Table 6.1: Smallest order r of a binary Goppa code of length $n = 2^m$ for which our distinguisher does not work.

m	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
$r_{\min}$	5	8	8	11	16	20	26	34	47	62	85	114	157	213	290	400

One can notice for instance that the binary Goppa code obtained with $m = 13$ and $r = 19$ corresponding to a McEliece public key of 90 bits of security, fits in the range of validity of our distinguisher. The values of $r_{\min}$ in Table 6.1 are checked by experimentations for $m \leqslant 16$ whereas those for $m \geqslant 17$ are obtained by solving the equation $\frac{mr}{2}\left((2e+1)r - 2^e - 1\right) = \frac{1}{2}mr(mr-1) - 2^m + mr$. Eventually, all the keys proposed in [47, table 4] for the CFS scheme can be distinguished.

Part II

On Hash Based Signature Schemes

Chapter 7

Signature schemes

7.1 Introduction

A handwritten signature is a small message that is added to a document to proove the identity of the author. It is used in everyday situations and is why we need an equivalent to be able to sign electronic documents. A handwritten signature is physically part of the document that is being signed, it can be authenticated (by comparing it with the signature of an ID document for example) and is very hard to copy. In the electronic case we solve these three situations by forcing the signature to depend on: the message that we want to sign and a public key that allows anybody to verify the signature. We also add some additional information in the message like for example, the date and time such that if any other person copy the signature it does not have any validity.

We saw that in 1976 Diffie and Hellman proposed to use *trapdoor one-way functions* in order to create a public-key cryptosystem. Let $\mathcal{X}$ be the set of plain texts, $\mathcal{Y}$ the set of ciphertexts, $e_{pk} : \mathcal{X} \to \mathcal{Y}$ the encryption function that depends on the public key (pk) and $d_{sk} : \mathcal{Y} \to \mathcal{X}$ the decryption function that depends of the secret key (sk), such that $d_{sk} \circ e_{pk} = id$. In the introduction we saw that with these cryptosystems Alice and Bob can communicate in a safe way. Now, Bob wants to be sure that it was Alice the one who sent the message. Note that we are not interested anymore in the secrecy of the message. For this:

- Alice signs the message $m \in \mathcal{X}$ by using her secret key (sk) to compute the signature of m: $\sigma \stackrel{\text{def}}{=} d_{sk}(m)$.
- For the verification, we just have to apply the encryption function to σ using Alice's public key (pk) and check if $e_{pk}(\sigma) = m$.

A signature scheme needs an algorithm to compute a signature for any message such that the desired person (Alice in this case) is the only person that is able to sign. It also needs a public verification algorithm which output is "yes" or "no": the answer to the question: *using the given public key is this a valid signature for the message?* The signature scheme is illustrated in Figure 7.1. In this chapter we will give a short introduction on signatures schemes and their security, it is mainly based in [54, 111].

7.2 Security of signature schemes

In this section we assume that Alice is the person who is signing the message. There are two kinds of attacks:

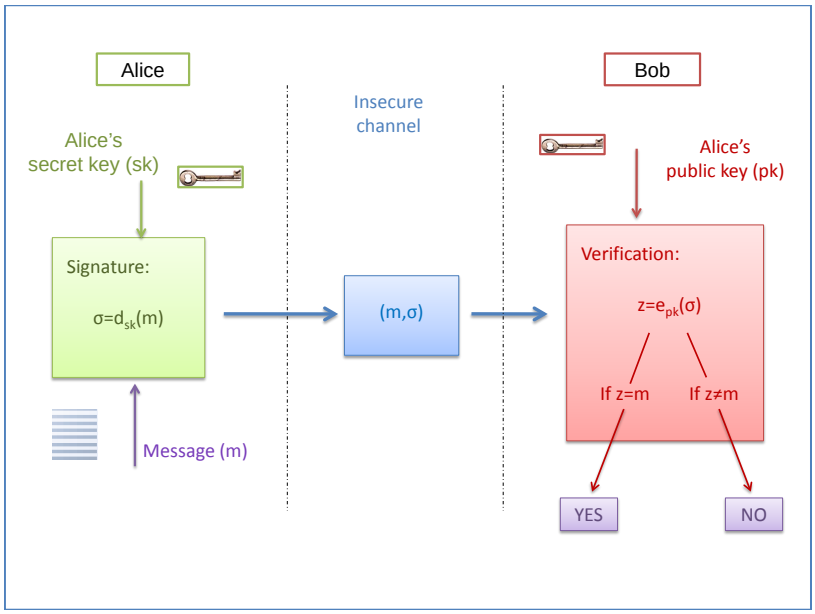

Figure 7.1: Signature scheme.

- *Key-only attacks*: the adversary only knows Alice's public key.
- *Message attacks*: in this case the adversary has access to some message/signature pairs before he tries to break the scheme. There are four kinds of message attacks:
  - *Known-message attack*: The adversary has access to the signatures of a set of messages $m_1, \ldots, m_t$ that are NOT chosen by her/him.
  - *Generic chosen-message attack*: In this case the adversary can obtain the signatures of a set of messages $m_1, \ldots, m_t$ that she/he has chosen. These messages do not depend on Alice's public key and are chosen before the attacker has saw any signature.
  - *Directed chosen-message attack*: As in the previous case the adversary can obtain the signatures of a set of messages $m_1, \ldots, m_t$ that she/he has chosen. The messages are chosen before any signatures are seen, but may depend on Alice's public key.
  - *Adaptive chosen-message attack*: The attacker is allowed to use Alice as an oracle. He can request signatures of messages that depend on her public key and which depend additionally on previously obtained signatures.

We say that the signature scheme is "broken" if the attacker can do any of the following with a non-negligible probability:

- *A total break*: the attacker is able to recover Alice's secret key.

- *Universal forgery*: the attacker finds an efficient signing algorithm functionally equivalent to Alice's signature algorithm (based on an equivalent, possible different, trapdoor information).
- *Selective forgery*: the attacker can sign a single message of her/his choice.
- *Existential forgery*: the attacker is able to create a valid signature for at least one message which is chosen by another person.

We say that a scheme is respectively *totally break, universally forgeable, selectively forgeable or existentially forgeable* if it is breakable in one of the above senses. We assume that the signature scheme provides *non-repudiation*, i.e., is such that if Alice sings a message, then she cannot deny that she has signed it.

There is another attack that is trivial: the attacker chooses an arbitrary value for the signature σ' and using Alice's public key she/he computes $m' \stackrel{\text{def}}{=} e_{pk}(\sigma')$. Then the pair (m', σ') is a valid pair, even if we do not have any control over m'. We will see in the next section that we can avoid this trivial attack by using hash functions.

7.3 Signatures and hash functions

A *cryptographic hash function* maps strings of arbitrary length to strings of fix length, say n, that is typically between 128 and 512 bits. We denote a hash function by $h : \{0,1\}^* \to \{0,1\}^n$. The output of the cryptographic hash function has being called with different names like for example: hash message, message digest, fingerprint.

A hash function has to satisfy certain properties in order to be used in cryptography. They are three main properties that are commonly required: and is that the best attack for the following three problems is the brute force attack:

- **Preimage:** Given $y \stackrel{\text{def}}{=} h(x)$ find a string x' such that $h(x') = y$.
- **Second preimage:** Given x find a string $x' \neq x$ such that $h(x') = h(x)$.
- **Collision:** Find two strings x and x' such that $x' \neq x$ and $h(x) = h(x')$.

By doing a brute force attack we can solve the preimage and second preimage problem after 2^n applications of h. And the collision problem after $2^{n/2}$ applications of h.

By using hash functions in the signature scheme introduced in the previous section we can avoid some problems. In fact if we sign the hash message $h(m)$, instead of the message m, we can avoid the trivial attack. In this case

- To sign the message $m \in \mathcal{X}$: Alice computes $h(m)$ and by using her secret key (sk) computes the signature of m: $\sigma \stackrel{\text{def}}{=} d_{sk}(h(m))$. Then Alice sends to Bob the valid pair (m, σ).

- For the verification, we have to compute $h(m)$ and then using Alice's public key (pk) encrypt σ and check if $e_{pk}(\sigma) = h(m)$.

In the trivial attack the attacker chooses σ' and computes $y \stackrel{\text{def}}{=} e_{pk}(\sigma')$. But now he has to find a message m' such that $y = h(m')$ i.e., he has to solve the preimage problem. Another advantage that we get by hashing the message is that we can avoid the cases where the message has a very big size.

Chapter 8

One-time signature schemes

A one-time signature scheme is a digital signature where the key is allowed to be used only once. In the signature process, parts of the signature key are revealed, and therefore, if it is used several times, an attacker can use the revealed parts to generate a valid signature. The basic idea to sign a one-bit message m, is to choose two n-bit strings x_0 and x_1 and let $f : \{0,1\}^n \to \{0,1\}^n$ be a one-way function. We compute $y_0 = f(x_0)$ and $y_1 = f(x_1)$, then we authenticate them and make them public. To sign the message m, we send x_m. The verifier can check if $f(x_m) = y_m$.

The drawback of the schemes that we will describe in this chapter is that each key pair can only be used once. If we want to do r signatures, we can of course call the function r independent times, but in that case the total signature size will always grow with the number of messages signed. And the public key size will be also r times bigger.

In all the schemes we want to sign a k-bit message $m = (m_0, \ldots, m_{k-1})$. We choose an integer n that is a security parameter (e.g., $n = 128$) and a one-way function $f : \{0,1\}^n \to \{0,1\}^n$.

Note that messages of greater length can be signed by first hashing them with a hash function h. In those cases we first compute the k-bits message $m^* \stackrel{\text{def}}{=} h(m)$ and then sign m^*. We should consider the case when the signer is not honest. She/he may find two different messages m and m' such that $h(m) = h(m')$. In this case both messages will have the same signature, so the signer will be able to sign m and later claim that she/he had sign m' and not m. To avoid this case we assume that h is collision resistant.

8.1 Lamport's signature scheme

The idea of using a hash function to produce a digital signature apparently originated from Lamport, who proposed a solution in a personal communication with Diffie [68]. The idea seems to have been first made public in the classic paper by Diffie and Hellman [36], and was later described in a technical report by Lamport [69]. Lamport's signature scheme is described in Algorithm 6.

Each key pair can only be used to sign a single message. If two messages are signed, then it might be possible to forge a signature on a third message. As an example, consider (for $k = 3$) the two messages (written in binary) $m = \texttt{100}$ and $m' = \texttt{111}$. The signature on m consists of the values $(x_{0,1}, x_{1,0}, x_{2,0})$, and the signature on m' consists of the values $(x_{0,1}, x_{1,1}, x_{2,1})$. An adversary will now be able to produce a valid signature for the messages `110` and `101`. As an example, the signature for the message

Algorithm 6 Lamport's signature scheme

- **Key generation:**
  - Choose $2k$ random n-bit strings $x_{i,j}$ for $0 \leq i < k$ and $j \in \{0,1\}$.
  - Compute $y_{i,j} \stackrel{\text{def}}{=} f(x_{i,j})$ for $0 \leq i < k$ and $j \in \{0,1\}$.
  - Authenticate and make public the $y_{i,j}$ for $0 \leq i < k$ and $j \in \{0,1\}$.

  The secret key is $(x_{0,0}, x_{0,1} \ldots, x_{k-1,1})$ and the public key is $(y_{0,0}, y_{0,1} \ldots, y_{k-1,1})$.
- **Signature generation:** Sign m by revealing x_{i,m_i} for all i, $0 \leq i < k$.
- **Signature verification:** The signature is verified by computing $z_i \stackrel{\text{def}}{=} f(x_{i,m_i})$ and checking that $z_i = y_{i,m_i}$ for all i, $0 \leq i < k$.

`110` is $(x_{0,1}, x_{1,1}, x_{2,0})$; all three values are known from the two valid signatures made by the signer.

The secrets $x_{i,j}$ may be pseudorandomly generated from a, say, ℓ-bit seed, in which case the length of the secret key is ℓ bits. The length of the public key is $2kn$ bits and the signature length is kn bits.

Remark 8.1. *Since kn of the $2kn$ public key bits can be computed by the verifier, and the remaining kn bits could be included in the signature, the public key does not have to consist of all the values $y_{i,j}$, but may simply be the single (say, n-bit) value $Y = h(y_{0,0} \| y_{0,1} \| \ldots \| y_{k-1,0} \| y_{k-1,1})$, where h is a cryptographic hash function.*

Using this remark, the key length is n bits and the signature length is $2kn$ bits.

8.2 Improvements of Lamport's signature scheme

In [80] Merkle proposes an improvement of Lamport's scheme. The main idea is to have a secret key of the form $X = (x_0, \ldots, x_{k-1})$, where the x_i are random n-bit numbers and sign m by revealing x_i for all i, $0 \leq i < k$, such that $m_i = 1$. In this case an adversary will be able to produce a valid signature for the messages which does not have `1`-bits in positions where m does not have `1`-bits. For example if we sign the message $m =$ `0110`, an adversary will also be able to sign the messages `0100` and `0010`. To avoid this we add to the end of m the number of `0`-bits it has.

We define $k' \stackrel{\text{def}}{=} k + \lfloor \log_2(k) \rfloor + 1$. The signature scheme is explained in Algorithm 7.

Example: Let $k = 8$ and $m =$ `01101101`. In this case $k' = k + 4$, $a = 3$ and $a_b =$ `0011`. We define $m' =$ `011011010011`. The signature will be $\sigma = (x_1, x_2, x_4, x_5, x_7, x_{10}, x_{11})$. Any message with fewer `1`-bits will have a `1`-bit instead of a `0`-bit in the binary representation of a. Therefore, an adversary will not be able to sign it.

Algorithm 7 Merkle's improvement of Lamport's signature scheme

- **Key generation:**
  - Choose k' random n-bit strings x_i for $0 \leq i < k'$.
  - Compute $y_i \stackrel{\text{def}}{=} f(x_i)$ for $0 \leq i < k'$.
  - Authenticate and make public the y_i for $0 \leq i < k'$.

  $X = (x_0, \ldots, x_{k'-1})$ is the secret key and $Y = (y_0, \ldots, y_{k'-1})$ is the public key.
- **Signature generation:**
  - Count the number of `0`-bits in m, call this number a. Let a_b be the binary representation of a (with k'-bits) and $m' \stackrel{\text{def}}{=} (m \| a_b)$.
  - Sign m by revealing x_i for all i, $0 \leq i < k'$ such that $m'_i = 1$.
- **Signature verification:**
  - Find a, a_b and generate m' as above.
  - Compute $z_i \stackrel{\text{def}}{=} f(x_i)$ and check that $z_i = y_i$ for all i such that $m'_i = 1$.

The key length is now roughly $n(k + \log_2(k))$ bits; the signature length depends on the message, but is about $n(k + \log_2(k))/2$ bits on average. As in Remark 8.1, the public key elements that cannot be computed from the signature may be included in the signature, which makes it possible to achieve a key length of n bits and a signature length of $n(k + \log_2(k))$ bits.

Bleichenbacher and Maurer described [21] another, more complicated but also theoretically more efficient variant; Dods et al. [37] analyzed the proposal and found that in practice, it does not perform as well as Winternitz' scheme (see Section 8.3). In 1992, Bos and Chaum described [22] another variant of the Lamport scheme. And in 2002 Reyzin and Reyzin introduced a very similar variant [98] that we will present in Section 9.1. It is essentially the same as Bos and Chaum, the main difference is that Bos and Chaum wanted to minimize the public key size, while Reyzin and Reyzin wanted to minimize the signature size.

8.3 Winternitz's signature scheme

This scheme appears first in Merkle's thesis [79], he wrote that Winternitz suggested him this method in 1979. The idea is to apply several times the one way function f as explained in the following example.

Example: We wish to sign the message m that can be `0`, `1`, `2`, or `3`. We authenticate and make public $y_0 = f^3(x_0)$ and $y_1 = f^3(x_1)$. Then we reveal $\sigma_1 = f^m(x_0)$ and $\sigma_2 = f^{3-m}(x_1)$. The verifier can easily find m by counting how many applications of f does he need to apply to σ_1 to reach y_0. Is very important to send both σ_1 and σ_2, if not the verifier could claim that she/he received a bigger power than the real one.

We present a more detailed description of this scheme, given in [24], in Algorithm 8.

Note that in this version instead of sending σ_1 and σ_2, we add an additional information at the end of the message, to apply the idea of the Merkle's improvement presented in the previous section. In fact, t_1 bits are used for the signature part and t_2 bits for the additional information that we will add. The public key and the signature size are nt, where $t = t_1 + t_2$. The main advantage here is that the $y_i's$ can be hashed together (as in Remark 8.1) without increasing the signature length.

In Table 8.1 we can see the public key and the signature size of the different signature schemes introduced in this chapter.

Table 8.1: Signature and key size for different hash-based signature schemes. We include the variant (indicated by an asterisk) proposed in Remark 8.1.

Scheme	Public key size	Signature size
Lamport	$2kn$	kn
Lamport*	n	$2kn$
Merkle's improvement of Lamport	$(k + \log_2(k))n$	$(\frac{k+\log_2(k)}{2})n$
Merkle's improvement of Lamport*	n	$(k + \log_2(k))n$
Winternitz	n	tn

There are variants of the one-time signature schemes described above which allow multiple messages to be signed with the same key pair. We will introduce some of them in the following chapter.

Algorithm 8 Winternitz's signature scheme

- **Key generation:**
  - Choose a Winternitz's parameter $w \geq 2$, to be the number of bits to be signed simultaneously.
  - Let
$$t_1 = \left\lceil \frac{k}{w} \right\rceil, \qquad t_2 = \left\lceil \frac{\lfloor \log_2(t_1) \rfloor + 1 + w}{w} \right\rceil \qquad \text{and} \qquad t = t_1 + t_2.$$
  - Choose t random n-bit strings x_i for $0 \leq i < t$.
  - Compute, authenticate and make public $y_i \stackrel{\text{def}}{=} f^{2^w-1}(x_i)$ for $0 \leq i < t$.
  - $X = (x_0, x_1, \ldots, x_{t-1})$ is the secret key and $Y = (y_0, y_1 \ldots, y_{t-1})$ the public key.
- **Signature generation:**
  1. Add a minimum number of zeros at the beginning of m such that the length of m is divisible by w. The extended m is split into t_1 w-bit strings $b_{t-1}, \ldots, b_{t-t_1}$ such that $m = b_{t-1} \| \ldots \| b_{t-t_1}$.
  2. Identify each b_i with an integer in $\{0, 1, \ldots, 2^w - 1\}$ and define the checksum
$$c = \sum_{i=t-t_1}^{t-1} (2^w - b_i).$$
     Since $c \leq t_1 2^w$, the binary representation of c has a length at most $\lfloor \log_2 t_1 2^w \rfloor + 1 = \lfloor \log_2 t_1 \rfloor + w + 1$. We add a minimum number of zeros at the beginning of c such that the length of c is divisible by w, and we can define the t_2 w-bit blocks $b_{t_2-1}, \ldots, b_0$ such that $c = b_{t_2-1} \| \ldots \| b_0$.
  - The signature is $\sigma = (f^{b_0}(x_0), f^{b_1}(x_1), \ldots, f^{b_{t-1}}(x_{t-1}))$.
- **Signature verification:** The signature $\sigma \stackrel{\text{def}}{=} (\sigma_0, \ldots, \sigma_{t-1})$ is verified by computing the bit string $b_0, \ldots, b_{t-1}$ as above and checking if
$$(f^{2^w-1-b_0}(\sigma_0), \ldots, f^{2^w-1-b_{t-1}}(\sigma_{t-1})) = (y_0, \ldots, y_{t-1}).$$

Chapter 9

Multiple-time signature schemes

9.1 Reyzin-Reyzin signature scheme

In 2002, Reyzin and Reyzin proposed a signature scheme [98] . Let b, t and k be integers such that $\binom{t}{k} \geq 2^b$, $\mathcal{T} = \{1, 2, \ldots, t\}$ and $\mathcal{T}_k$ is the family of k-subsets of $\mathcal{T}$. Let

$$S : \{0, 1, \ldots, 2^b - 1\} \to \mathcal{T}_k$$

be an injective function such that $S(m)$ is the m-th k-element subset of $\mathcal{T}_k$. Reyzin and Reyzin propose two ways to implement S, the computational cost of the first one is $\mathcal{O}(kt(log_2 t)^2)$ and of the second one is $\mathcal{O}(k^2(log_2 t)(log_2 k))$, both need $\mathcal{O}(k^2(log_2 t)^2)$ bits of memory. We want to sign a b-bit message $m = (m_0, \ldots, m_{b-1})$. We choose an integer n that is a security parameter (e.g., $n = 128$) and a one-way function $f : \{0,1\}^n \to \{0,1\}^n$. The signature scheme is presented in Algorithm 9.

Algorithm 9 Reyzin-Reyzin's signature scheme

- **Key generation:**
  - Choose t random n-bit strings x_i for $0 \leq i < t$.
  - Compute $y_i \stackrel{\text{def}}{=} f(x_i)$ for $0 \leq i < t$.
  - Authenticate and make public the y_i for $0 \leq i < t$.
  - $X = (x_0, \ldots, x_t)$ is the secret key and $Y = (y_0, \ldots, y_t)$ is the public key.
- **Signature generation:** Given the message m, interpret m as an integer between 0 and $2^b - 1$.
  - Compute $S(m) \stackrel{\text{def}}{=} \{i_0, i_1, \ldots, i_{k-1}\} \in \mathcal{T}_k$.

  The signature is $\sigma = (x_{i_0}, x_{i_1}, \ldots, x_{i_{k-1}})$.
- **Signature verification:** Given the message m, and the signature $\sigma \stackrel{\text{def}}{=} (\sigma_0, \ldots, \sigma_{k-1})$
  - Interpret m as an integer between 0 and $2^b - 1$ and compute $S(m) = \{i_0, i_1, \ldots, i_{k-1}\}$.
  - Check if $f(\sigma_j) = y_{i_j}$ for all $j \in \{0, \ldots, k-1\}$.

The public key size is nt, the key size is nk and the most expensive part is to compute S.

They are many choices for the parameters t and k such that $\binom{t}{k} \geq 2^b$. Notice that the public key size is linear in t and the signature size is linear in k. We must do a trade-off between these two variables to find a small signature scheme with a public key of reasonable size.

The important property of S is that it is *impossible* to find any two distinct m and m' such that $S(m) \subseteq S(m')$. In the same paper, Reyzin and Reyzin proposed another scheme called HORS (for "Hash to Obtain Random Subset"), that follows the same idea but instead of using S they use a function S' such that it has the following weaker property: that it is *infeasible* to find any two distinct m and m' such that $S'(m) \subseteq S'(m')$. A function S' with such a property is called a *subset-resilient* function.

9.2 HORS signature scheme

We want to sign the message M. We choose an integer n that is a security parameter (e.g., $n = 128$), a one-way function $f : \{0,1\}^n \to \{0,1\}^n$. We also choose an integer k and a k-bit hash function $h : \{0,1\}^* \to \{0,1\}^k$. Let d and t be chosen such that $d < t$ and $d \cdot \lceil \log_2(t) \rceil \leq k$. The HORS signature scheme is presented in Algorithm 10.

The public key size is nt and the key size is nd. The security requirements on the hash function h are somewhat non-standard: h must be subset-resilient.

When r messages have been signed using the same key pair, the probability that the signature of a random message can be forged (in a non-adaptive attack) is $(rd/t)^d$; in other words, the complexity of forging is around $2^k/(rd)^d$.

Applying Remark 8.1 in this case, the signature would grow from dn bits to tn bits.

9.3 HORS++ signature scheme

In 2003, Pieprzyk et al. [96] proposed a method to construct a multiple-time signature scheme following the idea of the HORS scheme. As stated in Section 9.1, the main condition of S is that for any two distinct messages M_1 and M_2, we have that $S(M_2) \nsubseteq S(M_1)$. Now if we want to sign r messages, we will be able to do it if we find a function S such that for any $r+1$ distinct messages $M_1, \ldots, M_r, M_{r+1}$, we have that $S(M_{r+1}) \nsubseteq \bigcup_{i=1}^{r} S(M_i)$. The proposal uses *cover-free families* introduced by Erdös et al. [40]. An (ℓ, t, r)-cover-free family is a set $\mathcal{X}$ of t elements, together with $\mathcal{B} = \{B_i \subseteq \mathcal{X} | i = 1, \ldots, \ell\}$, such that for all j and all sets I of r distinct integers between 1 and ℓ, not including j, we have

$$B_j \nsubseteq \bigcup_{i \in I} B_i.$$

Algorithm 10 HORS signature scheme

- **Key generation:**
  - Choose t random n-bit strings x_i for $0 \leq i < t$.
  - Compute $y_i \stackrel{\text{def}}{=} f(x_i)$ for $0 \leq i < t$.
  - Authenticate and make public the y_i for $0 \leq i < t$.
  - $X = (x_0, \ldots, x_t)$ is the secret key and $Y = (y_0, \ldots, y_t)$ is the public key.
- **Signature generation:**
  - Compute $m = h(M)$ and split m into d chunks $m_0, \ldots, m_{d-1}$ of $\lceil \log_2(t) \rceil$ bits each. If $d \cdot \lceil \log_2(t) \rceil < k$, ignore some bits of m.
  - Interpret each m_i as an integer $\langle m_i \rangle$ between 0 and $t - 1$.

  The signature is $\sigma = (x_{\langle m_0 \rangle}, x_{\langle m_1 \rangle}, \ldots, x_{\langle m_{d-1} \rangle})$.
- **Signature verification:** Given the signature $\sigma \stackrel{\text{def}}{=} (\sigma_0, \ldots, \sigma_{k-1})$
  - Compute $m = h(M)$ and split m into d chunks $m_0, \ldots, m_{d-1}$ of $\lceil \log_2(t) \rceil$ bits each. If $d \cdot \lceil \log_2(t) \rceil < k$, ignore some bits of m.
  - Interpret each m_i as an integer $\langle m_i \rangle$ between 0 and $t - 1$.
  - Check that $f(\sigma_i) = y_{\langle m_i \rangle}$ for all i, $0 \leq i < d$.

We want to sign the message M. We choose an integer n that is a security parameter (e.g., $n = 128$), a one-way function $f : \{0,1\}^n \to \{0,1\}^n$, an (ℓ, t, r)-cover-free family $(\mathcal{X}, \mathcal{B})$, an integer k such that $2^k \leq \ell$, a one-to-one mapping $S : \{0,1\}^k \to \mathcal{B}$ and h a k-bit hash function. The signature scheme is explained in Algorithm 11.

Algorithm 11 HORS++ signature scheme

- **Key generation:**
  - Choose t random n-bit strings x_i for $0 \leq i < t$.
  - Compute $y_i \stackrel{\text{def}}{=} f(x_i)$ for $0 \leq i < t$.
  - Authenticate and make public the y_i for $0 \leq i < t$.
  - $X = (x_0, \ldots, x_t)$ is the secret key and $Y = (y_0, \ldots, y_t)$ is the public key.
- **Signature generation:**
  - Compute $m = h(M) = (m_0, \ldots, m_{k-1})$ and interpret m as an integer number between 0 and $2^k - 1$.
  - Compute $S(m) \stackrel{\text{def}}{=} \{i_0, i_1, \ldots, i_{k-1}\} \in \mathcal{B}$.

  The signature is $\sigma = (x_{i_0}, x_{i_1}, \ldots, x_{i_{k-1}})$.
- **Signature verification:** Given the signature $\sigma \stackrel{\text{def}}{=} (\sigma_0, \ldots, \sigma_{k-1})$
  - Compute $m = h(M) = (m_0, \ldots, m_{k-1})$ and interpret m as an integer number between 0 and $2^k - 1$.
  - Compute $S(m) \stackrel{\text{def}}{=} \{i_0, i_1, \ldots, i_{k-1}\} \in \mathcal{B}$.
  - Check that $f(\sigma_j) = y_{i_j}$ for all j in $\{0, \ldots, k-1\}$.

The public key size is nt and the signature size is nk. Pieprzyk et al. give three different ways to build the function S. We will explain the constructions based on polynomials and on error-correcting codes.

***Constructing S based on Polynomials*:** This construction was first proposed by Erdös et al. [40]. Let d and c be integers and $\mathcal{X} = \mathbb{F}_{2^c} \times \mathbb{F}_{2^c}$ ($\mathbb{F}_{2^c}$ being the finite field of 2^c elements). Consider the polynomials of degree less than d over $\mathbb{F}_{2^c}$ and associate to each of those polynomials g the set $B_g = \{(x, g(x)) | x \in \mathbb{F}_{2^c}\} \subseteq \mathcal{X}$. We define

$$\mathcal{B} = \{B_g | g \text{ is a polynomial of degree at most } d-1\}.$$

Let g_1 and g_2 be two different such polynomials, then $|B_{g_1} \cap B_{g_2}| \leq d-1$, since $g(x) \stackrel{\text{def}}{=} g_1(x) - g_2(x)$ is a polynomial of degree less than d with at most $d-1$ different roots. Let $g, g_1, \ldots, g_r$ be polynomials of degree less than d over $\mathbb{F}_{2^c}$. Using the fact that $|B_g| = 2^c$, if $2^c \geq r(d-1) + 1$ we have that

$$\begin{aligned} |B_g \backslash (B_{g_1} \cup \cdots \cup B_{g_r})| &\geq |B_g| - (|B_g \cap B_{g_1}| + \cdots + |B_g \cap B_{g_r}|) \\ &\geq 2^c - r(d-1) \\ &\geq 1. \end{aligned}$$

Since $|\mathcal{B}| = 2^{dc}$ and $|\mathcal{X}| = 2^{2c}$, if $2^c \geq r(d-1)+1$ then $(\mathcal{X}, \mathcal{B})$ is a $(2^k, 2^{2c}, r)$-cover-free family.

We can now build the function $S(m)$, where m is a message of length $k \leq cd$. See m as the concatenation of d c-bit substrings ($m \stackrel{\text{def}}{=} a_0||a_1||\ldots||a_{d-1}$) and interpret each one of these substrings as an element in $\mathbb{F}_{2^c}$. Define $g_m(x) = \sum_{i=0}^{d-1} a_i x^i$ and the mapping $S : \{0,1\}^k \to \mathcal{B}$ by

$$S(m) = \{(\alpha, g_m(\alpha)) | \alpha \in \mathbb{F}_{2^c}\}.$$

Since $\mathcal{X}$ does not consist of random secrets in this case, we need another set Z of the same size as $\mathcal{X}$ consisting of secret values z_i, $0 \leq i < 2^{2c}$, and a mapping from pairs in $\mathbb{F}_{2^c} \times \mathbb{F}_{2^c}$ to integers between 0 and $2^{2c}-1$.

S is an efficient function since implementing it involves only polynomial evaluations in $\mathbb{F}_{2^c}$. The public key size is $n2^{2c}$ and the signature size is $n2^c$. Applying Remark 8.1 in this case, the signature size is $n2^{2c}$.

***Constructing S based on error-correcting codes*:** Let Y be an alphabet of q elements and let $\mathscr{C}$ be a linear (N, K, D) code over Y (i.e., subspace of Y^N of dimension K such that the Hamming distance between two distinct vectors in $\mathscr{C}$ is at least D). Each codeword is denoted by $c_{ij} = (c_{i1}, \ldots, c_{iN})$ with $c_{ij} \in Y$ for all $1 \leq i \leq q^K$ and $1 \leq j \leq N$.

A cover-free family can be defined by letting

$$\mathcal{X} = \{1, \ldots, N\} \times Y \text{ and } B_i = \{(j, c_{ij}) : 1 \leq j \leq N\}$$

i.e., $t = |\mathcal{X}| = Nq$, $|B_i| = N$ and there are $\ell = q^K$ subsets B_i.

For each pair $i \neq k$, we have that $|B_i \cap B_k| = |\{j : c_{ij} = c_{kj}\}| \leq N - D$. Now, as long as $r \leq \frac{N-1}{N-D}$ holds, we have a (q^K, Nq, r)-cover-free family. In fact, taking $B_s, B_{i_1}, \ldots, B_{i_r}$ $(r+1)$ different sets and assuming that $r \leq N - 1N - D$ we have that

$$\begin{aligned} |B_s \backslash (B_{i_1} \cup \cdots \cup B_{i_r})| &\geq |B_s| - (|B_s \cap B_{i_1}| + \cdots + |B_s \cap B_{i_r}|) \\ &\geq N - r(N-D) \\ &\geq 1. \end{aligned}$$

The public key size is Nq elements, and the signature size is N elements.

As above, since X does not consist of random secrets, we need another set Z of size Nq consisting of secret values z_i, $0 \leq i < Nq$, and a mapping from pairs in $\{1, \ldots, N\} \times Y$ to integers between 0 and $Nq - 1$.

Assume q is a power of two, say $q = 2^c$. We then identify Y with the finite field $\mathbb{F}_{2^c}$. In order to be able to sign a k-bit message m, we must have $q^K \geq 2^k$, so $K \geq k/c$. We assume c divides k and choose $K = k/c$.

We split m into K substrings of length c bits each, interpret each c-bit substring as an element in $\mathbb{F}_{2^c}$ and identify m with the K-vector $(m_1, m_2, \ldots, m_K) \in Y^K$. Let G be a $K \times N$ generating matrix for $\mathscr{C}$; then we define

$$S(m) = \{(1, u_1), (2, u_2), \ldots, (N, u_N)\},$$

where $(u_1, u_2, \ldots, u_N) = (m_1, m_2, \ldots, m_K)G$.

Note that the polynomial construction can be seen as a special case of this construction using Reed-Solomon codes.

The advantages comparing to HORS are that: HORS++ is secure against an adaptive chosen-message attack and that the security requirements on the hash function h are weaker than in HORS.

Pieprzyk et al. also propose to use hash chaining in order to increase the number of messages that can be signed with their signature scheme. This method is almost identical to the extension described in the following section.

9.4 HORSE signature scheme

HORSE [86] is HORS Extended. The extension consists in the idea of forming a hash chain for each secret x_i, e.g., $x_i^0 = x_i$, $x_i^1 = f(x_i)$, $x_i^2 = f(x_i^1)$, $\ldots$, $x_i^s = f(x_i^{s-1})$. The public key consists of the values x_i^s, and the secret key consists of the hash chains x_i^j, $j = 0, \ldots, s-1$. There is a technique by Coppersmith and Jakobsson [32] that allows values in the hash chain to be computed efficiently with limited storage.

A message is signed as described for the HORS scheme above, but the public key is refreshed for each signature by replacing all revealed values x_i^j by x_i^{j-1}. This allows at least a factor s more signatures to be produced with the same key pair; in practice, often a lot more signatures may be produced before all secrets x_i^j have been revealed for some i. Moreover, security is not reduced from one signature to the next, since whenever a secret value is revealed, it is replaced by another.

There is a problem with synchronization of this scheme, however, since signer and verifier must agree on the state of the public key. The signer can include some information in the signature about the state, but while a verifier is unsynchronized with the signer, it becomes easier to forge a signature for that verifier. If, for instance, the verifier has missed j updates to the public key, then a signature for this verifier can be forged with complexity around $2^k/(jd)^d$, since the situation is identical to HORS where j messages have been signed.

9.5 Are HORS, HORS++ and HORSE better than Winternitz's scheme?

We argue that HORS and its extension HORS++ and HORSE are not really better than Winternitz's method applied multiple times.

Winternitz's scheme

In this scheme to sign one k-bit message we fix two parameters n and w. We define $t_1 = \left\lceil \frac{k}{w} \right\rceil$, $t_2 = \left\lceil \frac{\lfloor \log_2(t_1) \rfloor + 1 + w}{w} \right\rceil$ and $t = t_1 + t_2$. We saw that the public key size is n and the signature size is nt. If we want to sign r messages, we will then have a public key of size rn and a signature key of size rtn.

HORS

Recall that one needs to choose two parameters, d and t, such that $d\lceil \log_2(t) \rceil \leq k$ and $d < t$, where k is the size of a subset-resilient hash function h. In the following we will assume that t is a power of two, then $\log_2(t)$ is an integer. Take as an example $k = 256$, we have five options for the pair (d, t): $(2, 2^{128})$, $(4, 2^{64})$, $(8, 2^{32})$, $(16, 2^{16})$ and $(32, 2^8)$.

Since the public key size is t, we are only interested in the pair $(32, 2^8)$.

The complexity of forging a signature after r messages have been signed is roughly $2^k/(rd)^d$. This is for an adversary that is not able to adaptively choose the messages to be signed. If $d = 32$, $t = 2^8$, the complexity of forging a signature after seeing a single valid signature is about 2^{96}, while the complexity drops to 2^{64} after two valid signatures and 2^{32} after three valid signatures. A better security is obtained by choosing $d = 16$ and $t = 2^{16}$, but this requires $2^{16} = 65,536$ elements in the secret and public key. In Winternitz' scheme, the public key consists of a single hash value. By concatenating $65,536$ hash values together, one can sign $65,536$ messages.

HORS++ with the cover-free family based on polynomials

Recall that in order to be able to sign k-bit messages, we need to choose two parameters c and d such that $cd \geq k$. As an example, with $k = 256$, assuming that c and d are powers of two and trying to minimize the size of c and d, we only deal with the cases where $cd = k$. We then have only nine possibilities for the pair (c, d): $(1, 256)$, $(2, 128)$, $(4, 64)$, $(8, 32)$, $(16, 16)$, $(32, 8)$, $(64, 4)$, $(128, 2)$, $(256, 1)$. However, since the public key consists of as many elements as there are elements in the set $\mathcal{X}$, namely 2^{2c}, we are only interested in the pairs $(1, 256)$, $(2, 128)$, $(4, 64)$ and $(8, 32)$. With these, we can securely sign $r = \lfloor \frac{2^c - 1}{d-1} \rfloor$ messages. The first three pairs do not constitute proper cover-free families (one has $r = 0$). By choosing $c = 8$ and $d = 32$, $r = 8$ messages can be securely signed, but still 2^{16} elements are needed in the public key. We might choose $c = d = 16$ and obtain a $(2^{256}, 2^{32}, 4369)$-cover-free family that can be used to sign 4369 messages. However, the public key will consist of 2^{32} elements.

In both cases, the public key grows by a factor greater than r, meaning that Winternitz' scheme applied r times yields shorter public keys and shorter signatures.

HORS++ with the cover-free family based on error-correcting codes

We assume that the linear code is MDS [76], since MDS codes are optimal with respect to this construction. Hence, $D = N-K+1$, so one can sign $r \leq \frac{N-1}{K-1}$ messages securely. We note that the linear code $(N, K, N-K+1)$ may not exist over a given alphabet Y.

We again assume that $k = 256$, and we need to choose c and K such that $cK = k$. The public key consists of $Nq = N2^c$ elements, and the signature size is N elements. We have $N \geq K$, and assuming we want to be able to sign eight messages (as in one of the examples above), we obtain the condition $N \geq 8(K-1)+1$.

We have the same possibilities for c and K as we had for c and d above. In order to minimize the length of the public key, one would choose one of the first few options in the list. For instance, one might choose $c = 1$ and $K = 256$ and obtain a binary $(2041, 256, 1786)$ code, which does not exist. Even if it did, 2041 elements would be needed in the public key in order to sign 8 messages, compared to which Winternitz' scheme applied multiple times is superior. The signature size is 256 elements, which is also worse than in Winternitz' scheme.

In general, for variable r, the size of the public key is at least $Nq = (r(K-1)+1)2^c$ elements, which is always greater than r elements as in Winternitz' scheme (except for $K = 1$, in which case the signer must choose 2^k secrets, i.e., one for each message that he wants to sign. He must then hash all secrets, and in order to sign the message m, reveal the corresponding secret).

HORSE

HORSE in principle has the same properties as HORS, but the number of messages that one can sign increases by a factor about $s/(1-e^{-d/t})$. On the other hand, increasing s amplifies synchronizing issues, and also increases key generation time. Additionally, refreshing the key after each signature requires about $\log_2(s)$ hash function evaluations.

Ignoring these issues, HORSE has very good properties compared to Winternitz's scheme. As an example, with $k = 256$, $d = 32$, $t = 2^8$, and $s = 2^{10}$, one can sign about $8,715$ messages using a public key containing 256 elements. The security level is about 2^{96} in a non-adaptive attack.

We can see that HORS and its extension HORS++ (using cover-free families based on polynomials and error-correcting codes) and HORSE are not really better than Winternitz's method applied multiple times.

9.6 Cover-free families based on orthogonal arrays

We may try to use a different cover-free family than the one proposed by Pieprzyk et al. in order to decrease the key and the signature size. In [73] we can see the

constructions of different kinds of cover-free families. An orthogonal array $OA(t, \ell, s)$ is an $\ell \times s^t$ array with entries from a set of $s \geq 2$ symbols such that in any t rows, every $t \times 1$ column vector appears exactly once. Let q be a prime power and $t < q$, we know from [30] that there exists an $OA(t, q+1, q)$, and from [73] that if this orthogonal array exists, then we are able to construct a $(q^t, q^2+q, \lfloor \frac{q-1}{t-1} \rfloor)$-cover-free family. Let $\mathcal{X}$ be the set $\mathbb{F}_q \times \mathbb{F}_{q+1}$, the subsets B_i are defined by $\{(s_1, 1), (s_2, 2), \ldots (s_{q+1}, q+1)\}$ where $(s_1, s_2, \ldots, s_{q+1})$ is a column vector of $OA(t, q+1, q)$. In order to sign a k-bit message m, we choose t such that $k \leq q^t$, then m can be mapped to its corresponding column vector of $OA(t, q+1, q)$. Using the same idea for the signature scheme as in HORS++, the size of the private key will be $q(q+1)$ elements and the signature size will be $q+1$ elements.

With $k = 256$ we need to choose q and t such that $q^t \geq 2^{256}$. For implementation reasons, it is a good idea to choose q to be a power of 2, so we choose some α and define $q = 2^\alpha$. In order to compare with one of the HORS++ instances mentioned above, we want to be able to sign 8 messages. So we also need that $\lfloor \frac{q-1}{t-1} \rfloor \geq 8$. The smallest α which makes it possible to satisfy both inequalities is $\alpha = 8$, so we have $q = 2^8$ and we choose $t = 32$. So $q^t = 2^{256}$ and we can sign 8 messages. With this choice of q and t, the public key size is $q(q+1) \approx 2^{16}$ elements, and the signature size is 257 elements. This is almost exactly the same as in HORS++.

Table 9.1 shows a comparison between different signature schemes in terms of public key size and signature size. In all schemes except Winternitz's, we include the variant proposed in Remark 8.1.

Table 9.1: Signature and key size for different 8-time hash-based signature schemes assuming a 256-bit message. We include the variant (indicated by an asterisk) proposed in Remark 8.1. HORS++ uses cover-free family based on polynomials.

Scheme	Public key size	Signature size
Lamport	4,096n	256n
Lamport*	8n	512n
Winternitz ($w = 4$)	8n	67n
HORS++ $(2^{256}, 2^{16}, 8)$	65,536n	256n
HORS++ $(2^{256}, 2^{16}, 8)$*	n	65,536 n
Scheme based on orthogonal array	65,792n	257n
Scheme based on orthogonal array*	n	65,792 n

We can see that HORS++ using cover-free families based on orthogonal arrays is neither better than Winternitz's method applied multiple times.

9.7 Using (near-)collisions to sign twice

In this section, we describe a novel method that allows two signatures for each public key, i.e., a two-time signature scheme. This is achieved without an increase in the

public key size, nor in the signature size, but it requires a non-negligible amount of offline work.

We want to sign two 1-bit messages m_0 and m_1. We choose an integer n that is a security parameter (e.g., $n = 128$) and a one-way function $f : \{0,1\}^n \to \{0,1\}^n$ which is not collision intractable. The scheme is described in Algorithm 12.

Algorithm 12 Signature scheme using collisions to sign twice

- **Key generation:**
  - Find four n-bit strings x_0, x'_0, x_1 and x'_1 such that:

$$\begin{cases} f(x_0) = f(x'_0) = y_0 \text{ and} \\ f(x_1) = f(x'_1) = y_1, \end{cases}$$

    and such that x_0 and x_1 have a `0`-bit in the most significant position, and x'_0 and x'_1 have a `1`-bit in the most significant position.
  - Authenticate and make public y_0 and y_1.
  - (x_0, x'_0, x_1, x'_1) is the secret key and (y_0, y_1) is the public key.
- **Signature generation:**
  - In order to sign m_0: reveal the string $0\|x_{m_0}$.
  - In order to sign m_1: reveal $1\|x'_{m_1}$.

  The signatures are $\sigma_1 = 0\|x_{m_0}$ and $\sigma_2 = 1\|x'_{m_1}$.
- **Signature verification:**
  1. The signature on m_0 is verified by
     - Checking that $f(x_{m_0}) = y_{m_0}$.
     - Checking that x_{m_0} has a leading `0`-bit.
  2. The signature on m_1 is verified by
     - Checking that $f(x'_{m_1}) = y_{m_1}$.
     - Checking that x'_{m_1} has a leading `1`-bit.

This two-time signature scheme can be combined with, e.g, Merkle's method (described in Section 8.2) to allow the signing of messages of several bits. This would allow two k-bit messages to be signed using a single public key of length about $(k+\log_2(k))n$ bits.

It can also be combined with Winternitz' one-time signature scheme, but here it seems that two different one-way functions f_1 and f_2 are required. One finds collisions of the type $f_1^{2^w-1}(x_0) = f_2^{2^w-1}(x'_0) = y_0$, etc.

The drawback of this method is the need to find collisions. With, say, $n = 80$,

finding a collision requires about 2^{40} hash function evaluations (finding k collisions requires only roughly a factor $\sqrt{k}$ more hash function evaluations). The work required to forge a signature, however, is still about 2^n, which currently seems infeasible with $n = 80$. We could also try to find a one-way second preimage resistant function that is not collision resistant.

In order to expand the gap between the amount of work required by the signer, and the amount of work required to forge a signature, one might make use of near-collisions. Hence, it is no longer required that (e.g.) x_0 and x'_0 collide in all bits, but only in, say, $n - t$ bits. The expected number of hash function evaluations required to find a collision in $n - t$ out of n bits, when the t bit positions that do not collide are not fixed beforehand, is about

$$\sqrt{2^n / \binom{n}{t}}.$$

However, it is necessary that the signer includes in the public key an indication of the t bit positions to ignore when the signature is checked. This increases the public key size (using Merkle's one-time signature scheme) from about $(k + \log_2(k))n$ bits to about $(k + \log_2(k))(n + t\lceil \log_2(n) \rceil)$ bits. Hence, there is a trade-off here; larger values of t will expand the gap between the signer's and the forger's work, but will also increase the public key size.

As an example, with $n = 96$ and $t = 5$, one may find a near-collision after about $2^{35.1}$ hash function evaluations. The work required to forge is about $2^{96-5} = 2^{91}$. With $k = 256$, this means an increase in the public key size from $25,440$ bits to $34,715$ bits. With $t = 10$, the work required by the signer is about $2^{26.3}$, and the work required by a forger is about 2^{86}. The public key size increases to $43,990$ bits in this case.

Chapter 10

Merkle tree signature schemes

In order to sign several messages using one-time signature schemes, we need to store a huge amount of verification keys, especially if the verifier needs to ask signatures from many people. An idea will be to send at the same time the signature and the verification key, this will solve the problem of the memory but will open the problem of the authentication process. Merkle proposes two different ways to authenticate the verification key with a modest memory requirement. The first method [82], called *Merkle tree authentication*, allows us to do a finite number of signatures by building a tree of finite length (a static tree). We will explain it in Section 10.1.1.

The second method [81] is a similar scheme that allows to sign an infinite number of times. Nevertheless this has a cost, in fact, the signature size grows after each signature. This is a problem for the efficiency and also because it reveals an additional information about the number of signatures that have been made. This proposal uses a dynamically expanding tree method, we will explain it in Section 10.1.2. We denoted this method *dynamic tree* in opposition of the first method denoted *static tree*.

The first method was published two years later than the second one, but was written eight years before, and is the most cited one [1]. The fact that it can only do a finite number of signatures may be an important difference compared with the signatures schemes based on a one-way function with a trapdoor (like RSA). But in practice these schemes are also limited to a finite number of signatures, this limitation can be due to the devices where the scheme is implemented or to the policies of the signature schemes.

10.1 Merkle tree authentication

10.1.1 Static tree

In this method we fix the number of messages that we want to sign, say, 2^D (for simplicity we fix it to be power of two). We choose a one-way function $f : \{0,1\}^* \to \{0,1\}^n$. In order to sign 2^D (k-bits)messages $(M_1, M_2, \ldots, M_{2^D})$, the signer chooses a one-time signature scheme and 2^D secret/public key pairs (s_i, p_i) for $0 \leq i < 2^D$. In Algorithm 13 we can see the key generation. We say that D is the *depth* or *height* of the tree.

[1] [82] " was submitted to Ron Rivest, then editor of the Communications of the ACM, in 1979. It was accepted subject to revisions and was revised and resubmitted in November 1979. Unfortunately, Ron Rivest passed over the editorship to someone else, the author became involved in a startup, and the referees reportedly never responded to the revised draft".

Algorithm 13 Key generation

- Generate 2^D one-time key pairs (s_i, p_i) for $0 \leq i < 2^D$.
- Build a binary tree from the bottom to the top as follows:
  - The leaves of the tree are $y_0^i \stackrel{\text{def}}{=} f(p_i)$ for $0 \leq i \leq n-1$.
  - The nodes at height j are $y_j^i \stackrel{\text{def}}{=} f(y_{j-1}^{2i} \| y_{j-1}^{2i+1})$ for $0 \leq j \leq D$ and $0 \leq i < 2^{D-j}$.
- Authenticate and make public the root of the three $Y = y_D^0$.

The public key is the root of the three $Y = y_D^0$ and the secret key are the one-time key pairs (s_i, p_i) for $0 \leq i < 2^D$.

We assume that the message M_i is signed using the one-time key pairs (s_i, p_i). To be able to verify the signature one have to be able to authenticate p_i using the public key Y. This can be done if the signer adds to the signature an additional information called *authentication path* that allows the verifier to authenticate p_i (from Y). In the following example we fix $D = 3$ and we show how we can authenticate p_4 using Y.

Example: We fix $D = 3$, the binary tree is represented in Figure 10.1.

$Y = y_3^0$ is an authenticated public value. In order to authenticate p_4 using Y:

1. The signer sends y_2^0 and y_2^1 so the verifier can compute $y_3^0 = f(y_2^0 \| y_2^1)$ and check if it is the right value. This will authenticate y_2^1.
2. The signer sends y_1^2 and y_1^3 so the verifier can compute $y_2^1 = f(y_1^2 \| y_1^3)$ and check if it is the right value. This will authenticate y_1^2.
3. The signer sends y_0^5 and from p_4, the verifier can compute $y_0^4 \stackrel{\text{def}}{=} f(p_4)$ and then check if $y_1^2 = f(y_0^4 \| y_0^5)$ is equal to the authenticated value. This will authenticate p_4.

We can see that half of the transmissions are redundant, if we send the elements in the opposite order (i.e., from the leaves to the root), from $y_0^4 = f(p_4)$ and y_0^5 the verifier can compute y_1^2, then the signer can send y_1^3 and the verifier can compute y_2^1. Finally the signer will send y_2^0 and the verifier can compute the root of the tree y_3^0 and compare it with the public value Y. This process is illustrated in Figure 10.1, the red nodes represent the public values and the purple nodes the computed values.

If the signature is made in this order, instead of transmitting $2\log_2(n)$ elements we will just send $\log_2(n)$ elements. In the previous example, the signer will send: y_0^5, y_1^3 and y_2^0. We call these elements the *authentication path* for p_4.

This new method is very good to reduce the verifier's memory, she/he will only need to save the root of the tree instead of $p_0, p_1, \ldots, p_{2^D-1}$.

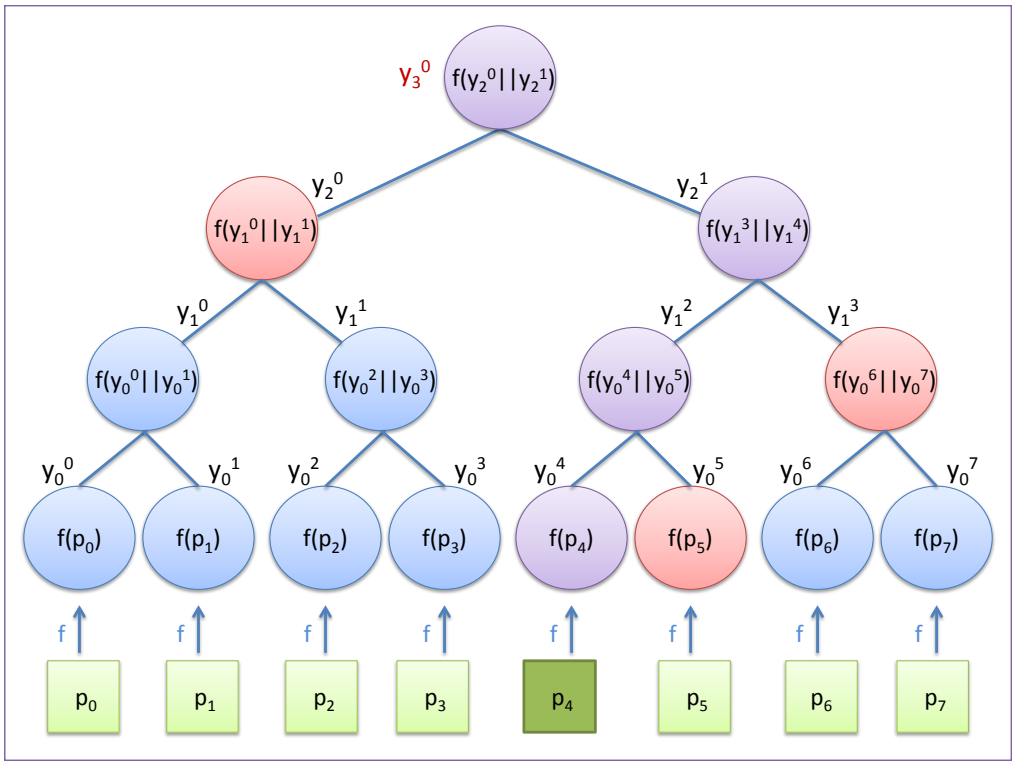

Figure 10.1: Merkle static tree, $D = 3$.

But we still need a good method to save the information in the tree, or the authentication paths, so that the signer does not have to build it for each message. In Table 10.1 we can see all the authentication paths for each leaf in our previous example.

Table 10.1: Authentication path for $D = 3$.

p_0	y_2^1	y_1^1	y_0^1
p_1	y_2^1	y_1^1	y_0^0
p_2	y_2^1	y_1^0	y_0^3
p_3	y_2^1	y_1^0	y_0^2
p_4	y_2^0	y_1^3	y_0^5
p_5	y_2^0	y_1^3	y_0^4
p_6	y_2^0	y_1^2	y_0^7
p_7	y_2^0	y_1^2	y_0^6

An idea is to use the different p_i in order, i.e., first authenticate p_1, then p_2, etc. Since the authentication path from p_{i+1} uses a big part of the path of p_i, we will be able to delete from Table 10.1 all the redundant elements, we will then save only the information written in red.

Now that we had define the authentication path, we can describe the signature and the verification scheme in Algorithm 14. The computation of the authentication path can be shared between signatures, so that the number of hash function calls per signature is constant. There are many ways to do this, e.g. [14, 60, 82, 113]. See also Section 10.2.

Algorithm 14 Signature and verification

- **Signature generation:** Sign M_i by revealing (s_i, p_i) and the authentication path of p_i.
- **Signature verification:** The signature is verified by
  - Checking if m_i is signed with the key pair (s_i, p_i) using the selected one-time signature scheme.
  - Authenticating p_i by computing y_D^0 (from p_i and the authentication path) and comparing with the authenticated and public value Y.

With this method, we can sign a fixed number of times, since once the tree is built, no other leaf can be added such that the root value is respected. This "inflexibility" makes it impossible to add new Y_i's but also makes it practically impossible to add invalid leaves.

10.1.2 Dynamic tree

The second proposal of Merkle [81] presents an infinite tree of one-time signatures. This time we start to build the tree from the root. Each node has associated with it

a label (which is an integer):

$$\begin{cases} \text{- The label of the root is 0.} \\ \text{- The label of the left children-node of the node } i \text{ is } (2i+1). \\ \text{- The label of the right children-node of the node } i \text{ is } (2i+2). \end{cases}$$

Note that in this way we can have an infinite tree. It is very easy to compute the sub-nodes from the parental node, and vice-versa. We denote $\pi(i)$ to the parent of the node i (i.e., the node $\lfloor (i-1)/2 \rfloor$). Each node is going to have three functions:

1. Authenticate the left sub-node.

2. Authenticate the right sub-node.

3. Sign a k-bits message M.

Hence, each node i needs three one-time secret/public key pairs denoted (s_i^ℓ, p_i^ℓ), (s_i^r, p_i^r) and (s_i^m, p_i^m), in order to authenticate the left and right sub-nodes and sign the message respectively. Given a cryptographic hash function h, we define p_i^* by

$$p_i^* \stackrel{\text{def}}{=} h(p_i^\ell) \| h(p_i^r) \| h(p_i^m).$$

A complete node can also be hashed; we define $H(i) = h(p_i^*)$ to be the hash of node i. We define $\sigma_j^m(A)$, $\sigma_j^\ell(A)$ and $\sigma_j^r(A)$ to be the one-time signature on a message A using the key pair (s_j^m, p_j^m), (s_j^ℓ, p_j^ℓ) and (s_j^r, p_j^r) respectively. We can see the key and signature generation algorithm using the dynamic tree to sign a message M in Algorithm 15, and the verification algorithm in Algorithm 16.

In this scheme, the signature size grows logarithmically with the number of nodes that we have used, and so do the signature generation and verification times. The verifier has to repeat the second step of the verification process $\log_2(i+1)$ times. We fixed the tree to be binary but it can have k branches (instead of two), in this case the number of time that the verifier has to repeat the second step of the verification process is reduced to $\log_k(i+1)$. But k cannot be too large since the computation of p_i^* will take longer:

$$p_i^* \stackrel{\text{def}}{=} p_i^1 \| p_i^2 \| \ldots \| p_i^k \| p_i^m.$$

10.1.3 Remarks:

1. These two *meta-systems* can use any one-time signature scheme and any hash function h. Of course the security, sizes and computational cost depend on them. Coronado [33] proved that if h is collision resistant, then the *Merkle tree* is existentially unforgeable under an adaptive chosen message attack.

2. We have presented the original *Merkle tree* introduced by Merkle, but there are some improved versions that give an efficient and practical method, they can be found in [24] and [35].

Algorithm 15 Merkle's dynamic tree signature scheme

- **Key generation:**
  - Generate three one-time key pairs denoted: (s_0^ℓ, p_0^ℓ), (s_0^r, p_0^r) and (s_0^m, p_0^m).
  - Compute $p_0^* = h(p_0^\ell)\|h(p_0^r)\|h(p_0^m)$ and $H(0) = h(p_0^*)$
  - Authenticate and make public $Y \stackrel{\text{def}}{=} H(0)$.
- **Signature generation:** Sign the message M given an existing tree. The signature S is constructed iteratively; initially S is the empty string.
  1. The signer choose some leaf node, say, i.
  2. Generate three one-time key pairs denoted: (s_i^ℓ, p_i^ℓ), (s_i^r, p_i^r) and (s_i^m, p_i^m) (this add two new leaves to the tree).
  3. Compute $\sigma_i^m(M)$, the one-time signature on a message M using the key pair (s_i^m, p_i^m).
  4. $S \leftarrow S\|M\|\langle i\rangle\|\sigma_i^m(M)\|p_i^*$, where $\langle i\rangle$ is a binary representation of the integer i.
  5. – **IF** $i = 0$, the signature has been created.
     – **ELSE**
       * If i is odd (i.e., is a left sub-node) compute $\sigma_{\pi(i)}^\ell(H(i))$ and
       $$S \leftarrow S\|\sigma_{\pi(i)}^\ell(H(i))\|p_{\pi(i)}^*.$$
       * If i is even (i.e., is a right sub-node) compute $\sigma_{\pi(i)}^r(H(i))$ and
       $$S \leftarrow S\|\sigma_{\pi(i)}^r(H(i))\|p_{\pi(i)}^*.$$

     Replace i by $\pi(i)$; go back to steps 5.

Algorithm 16 Signature verification algorithm in the Merkle's dynamic tree signature scheme

The signature is verified by parts:

1. S can be seen as $S \leftarrow S\|M\|\langle i\rangle\|\sigma_i^m(M)\|p_i^*\|S'$.

   - Verify that $\sigma_i^m(M)$ is a valid signature of M.
   - We need to authenticate p_i^m, for this we check that $h(p_i^m)$ corresponds to the correct suffix of p_i^*.
   - Compute $H(i)$ from p_i^*.

   If $H(i)$ is valid, then the signature is also valid. If $i = 0$ compare $H(i)$ with Y else got o step 2.

   - **IF** $i = 0$, we compare $H(i)$ with the public value $Y = H(0)$ and **STOP**.
   - **ELSE** go to step 2.

2. S' can be seen as $S' \leftarrow \sigma^{\ell,r}_{\pi(i)}(H(i))\|p^*_{\pi(i)}\|S''$. By $\sigma^{\ell,r}_{\pi(i)}$ we mean $\sigma^{\ell}_{\pi(i)}$ if i is odd and $\sigma^{r}_{\pi(i)}$ if i is even.

   - Verify that $\sigma^{\ell,r}_{\pi(i)}(H(i))$ is a valid signature of $H(i)$.
   - We need to authenticate $p^{\ell,r}_{\pi(i)}$, for this we check that $h(p^{\ell,r}_{\pi(i)})$ corresponds to the correct suffix of $p^*_{\pi(i)}$.
   - Compute $H(\pi(i))$
   - – **IF** $\pi(i) = 0$, we compare $H(\pi(i))$ with the public value $Y = H(0)$.
     – **ELSE** $S' \leftarrow S''$ and replace i by $\pi(i)$. Go back to steps 2.

3. In the static method, a large binary tree is generated in the key generation phase. This fixes the number of messages that can be signed to the number of leaves in the tree. One may for instance generate a tree of depth 20 capable of signing about 1 million messages. The time required for key generation is not negligible, but an advantage of this scheme is that the signature size and signature generation and verification times are constant (in contrast with the dynamic tree method).

4. Doing a brute-force attack we are able to find a preimage or a second preimage on a hash function after roughly 2^n evaluations of the hash function, and a collision after roughly $2^{n/2}$. In a quantum computer, using the Grover algorithm [56] we only require $2^{n/3}$ applications of the hash function to find a collision with probability at most $1/2$. Grover's algorithm will change the parameters of security but the complexity remains exponential, so we can say that these schemes are *quantum resistant*.

10.2 Simple and efficient hash tree traversal

In this section we describe a relatively simple and efficient hash tree traversal algorithm. The algorithm resembles one described by Szydlo [113].

Hash tree traversal is used to compute authentication paths in the static tree scheme described in Section 10.1.1. We note that there is a simple procedure, called TREEHASH (see e.g., [24, Alg. 2.1]), which given the leaves of a hash tree of depth D computes the root using $2^D - 1$ hash function evaluations and at most D units of memory. TREEHASH can, of course, also be used to compute authentication nodes.

10.2.1 Preliminaries

We start with some observations on authentication paths. We use the term *round* to denote the time interval in between two signatures; round i means the interval between signature no. $i-1$ and signature no. i, where counting starts from 0.

Future authentication nodes are computed in some round before the node is needed. The hash tree traversal algorithms tries to schedule these computations. We keep the same notation used in Section 10.1.1 and we try to find the authentication paths for the leaves in order, i.e., first find the authentication path of y_0^1, then y_0^2, etc. We assume that we know the authentication path of y_0^0.

An authentication node at level d must be replaced in round i whenever i is a multiple of 2^d. In round i, find $d, r \in \mathbb{Z}$ such that i can be written $r \times 2^d$, where r is odd.

- The authentication node at level d must change from a right to a left node: $y_d^r \rightarrow y_d^{r-1}$.
- The authentication nodes at levels below d must change from left to right nodes (its cousin): $y_j^{r2^{d-j}-2} \rightarrow y_j^{r2^{d-j}+1}$ for $0 \leq j < d$.

Note that no other authentication nodes need to change, since if r is odd, then i is not a multiple of 2^{d+1}. One may conclude that there is at most one new left authentication node per signature. If a left node y_d^{r-1} of height d is computed in the round $i = r2^d$ (for r odd), i.e., the round where it is needed, then it can be computed using $d+1$ hash function evaluations (we call these *units* in the following). The reason is that the authentication path at that point contains exactly the nodes needed to quickly compute y_d^{r-1}. In other words, y_d^{r-1} is on the path from the leaf y_0^{i-1} to the root.

Example ($D = 4$): In Table 10.2 we can compare the authentication path of the leaves y_0^j for $7 \leq j \leq 10$ in the Merkle's static tree with $D = 4$. We can see that an authentication node at level d must be replaced in round i whenever i is a multiple of 2^d. And writing $i = r2^d$, if r is odd we notice that the change is from a right node to its sibling. And if r is even the change is from a left node to a right node (its cousin). This is illustrated in Figure 10.2 for round $i = 8$.

Table 10.2: Authentication path (AP) of the leaves y_0^j for $7 \leq j \leq 10$ in the Merkle's static tree with $D = 4$. We write $i = r2^d$ and if r is odd we notice that the change is from a right node to its sibling (in red). And that if r is even the change is from a left node to a right node (in blue).

	AP of y_0^7	$i = 8$	AP of y_0^8	$i = 9$	AP of y_0^9	$i = 10$	AP of y_0^{10}
$d = 3$	y_3^1	1×2^3	y_3^0	–	y_3^0	–	y_3^0
$d = 2$	y_2^0	2×2^2	y_2^3	–	y_2^3	–	y_2^3
$d = 1$	y_1^2	4×2^1	y_1^5	–	y_1^5	5×2^1	y_1^4
$d = 0$	y_0^6	8×2^0	y_0^9	9×2^0	y_0^8	10×2^0	y_0^{11}

In this example, we can also illustrate the fact that a left node y_d^{r-1} of height d is computed in the round $i = r2^d$ (for r odd) by using $d+1$ hash function evaluations. Taking for example $i = 8$, $y_3^1 \to y_3^0$. As y_0^6, y_1^2 and y_2^0 form the authentication path of y_0^7, then we can compute y_3^0 by applying 3 times the hash function h: the first time to compute $y_1^3 = h(y_0^6 \| y_0^7)$, the seconde time to compute $y_2^1 = h(y_1^2 \| y_1^3)$ and the last time to compute $y_3^0 = h(y_2^0 \| y_2^1)$.

We are now ready for a high-level description of the hash tree traversal algorithm.

10.2.2 Algorithm description

The authentication path needed for the first signature is computed and stored during key generation. Hence, round 0 consists in key generation and nothing else.

We explain how we can compute authentication nodes by spending (at most) D hash function evaluations (units) per round. The main ideas are:

- Left nodes are computed as late as possible, i.e., the node y_d^{r-1} is computed in round $i = r2^d$ for r odd. We saw in the previous section that this be computed using $d+1$ hash function evaluations. Then, once we are in round $i = r2^d$ (with

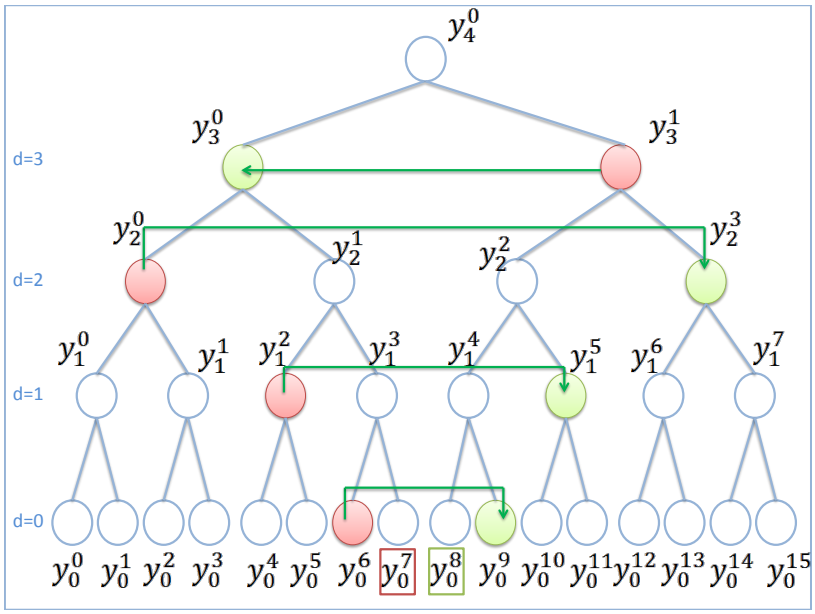

Figure 10.2: In red the authentication path for y_0^7 and in green the one for y_0^8. The green arrows show the changes that must be done in round $i = 8$.

r odd) the computation of left nodes has highest priority and is the first node to be computed.

- Any remaining time is spent on right nodes. Highest priority is given to nodes of smallest height. Hence, if the next right node at level j is not computed yet, then we work on that node before moving to level $j + 1$, etc.
- We never work on more than one node at a given height at a time.

Since there will often not be enough units to finish the computation of a right node, we may have to stop and continue in a later round. Hence, we imagine having access to a function `update(i,j,t)` that given the current round number i, a height j, and a number t of units that it is allowed to spend, is able to start or continue working on the next right authentication node of height j. The function returns the number of units it spent. It is able to figure out which node it needs to work on, and it remembers how far it got the last time it was working on that node. It also knows if there is no work to be done at height j.

As mentioned above, in round $i = r2^d$, where r is odd, the right authentication nodes $y_j^{r2^{d-j}+1}$, for $0 \leq j \leq d-1$, must be ready. Computation of the node $y_j^{r2^{d-j}+1}$ cannot begin sooner than round $i - 2^{j+1}$ if we are working on only one authentication node per level at a time. We imagine that `update(i,j,t)` is aware of this.

We have a maximum of $D-1$ authentication node computations at a time (one left node, and at most $D-2$ right nodes, since the right node at level $D-1$ is computed during key generation).

It has been experimentally verified that the memory requirements are never more than $3D - 4$ hash values, and that nodes will always be available on time. We give a proof sketch of the last fact below. Algorithm 17 describes the algorithm in pseudocode for a tree of height D; the description refers to the authentication path computations needed in round i. The first time the algorithm is called, we have $i = 1$.

Algorithm 17 Hash tree traversal algorithm

Input: i; the round number. `update(i,j,t)` is an external function maintaining its own local state.

1. $T \leftarrow D$ //time budget.
2. Find integers d and odd r such that $i = r2^d$.
3. Compute the left node y_d^{r-1} needed in the next authentication path, using nodes of height less than d in the current authentication path (if $d > 0$).
4. $T \leftarrow T - (d + 1)$ //$d + 1$ units spent
5. $j \leftarrow 0$
6. **while** $T > 0$ and $j \leq D - 2$
   - $t \leftarrow \texttt{update}(\texttt{i}, \texttt{j}, \texttt{T})$
   - $T \leftarrow T - t$ //t units spent.
   - $j \leftarrow j + 1$

   end while
7. Replace the node at level d in the authentication path by y_d^{r-1}. Replace the node at level j by $y_j^{r2^{d-j}+1}$, $0 \leq j \leq d - 1$.

10.2.3 Algorithm justification

We now argue why all authentication nodes are always ready when they are needed. Consider a height-2 binary tree, where the authentication path for the first leaf is already known. See Figure 10.3.

- In round 1, i.e., before signing the second message, we need to compute the leaf y_0^0. This requires one unit. Hence, we have one unit left to compute the next right leaf y_0^3, which requires exactly one unit (and is needed in round 2).
- In round 2, we spend both units computing the left node y_1^0 needed in this round.
- In round 3, we need to compute the left leaf y_0^2, which requires one unit. There is nothing more to compute.

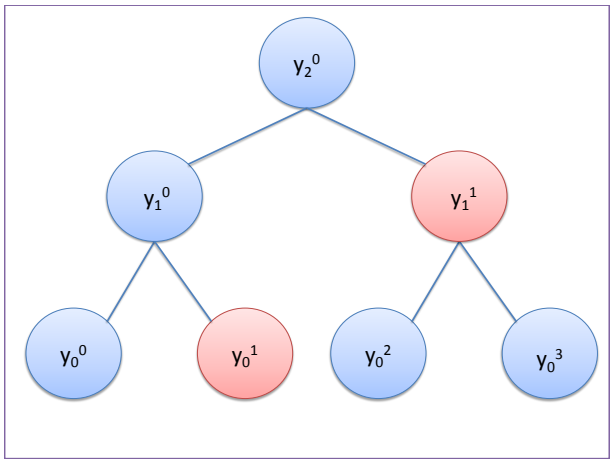

Figure 10.3: Merkle static tree, $D = 2$. In red we represent the authentication path for y_0^0.

Hence, we might say that we have one unit in spare, and we have only done computations in three rounds, so in fact we have a full round of two additional units in spare as well. In total, we have three units in spare.

Now let us double the size of the tree by increasing its height by one. The computations needed in the first half of the rounds are the same as before. There, we had three units in spare, but since we have increased the height by one, we also increase the "budget" of computations by one for each leaf. So there are an additional four units in spare from the first four rounds, in total seven units. But we also have to do more work, since there are three authentication nodes that have to be computed, which did not have to be computed before (see Figure 10.4). These are the right nodes y_0^5 and y_1^3, and the left node y_2^0. The work required to compute these is (respectively) one, three, and three units, in total seven. Hence, given the seven spare units from the first half of the rounds, we can compute these nodes in time, and we'll end up with seven units in spare again after the last round.

In general, when we double the size of the tree, and we look at the situation after the first 2^{D-1} rounds (where D is the new height of the tree), we need to compute D new nodes in order to be in the same situation as before round 0. In fact in round $i = 2^{D-1}$ we have to compute:

- One left node: y_{D-1}^0, that requires D units (as discussed in Section 10.2.1, y_{D-1}^0 is on the path from the leaf $y_0^{2^{D-1}-1}$ to the root).
- And $(D-2)$ right nodes, one at each level j for $0 \leq j \leq D-2$, requiring a total of $\sum_{j=0}^{D-2}(2^{j+1}-1)$ units.

Hence, we need $D + \sum_{j=0}^{D-2}(2^{j+1}-1) = (\sum_{j=1}^{D-1} 2^j) + 1 = 2^D - 1$ spare units at the end of the first half to be ready to start the second half.

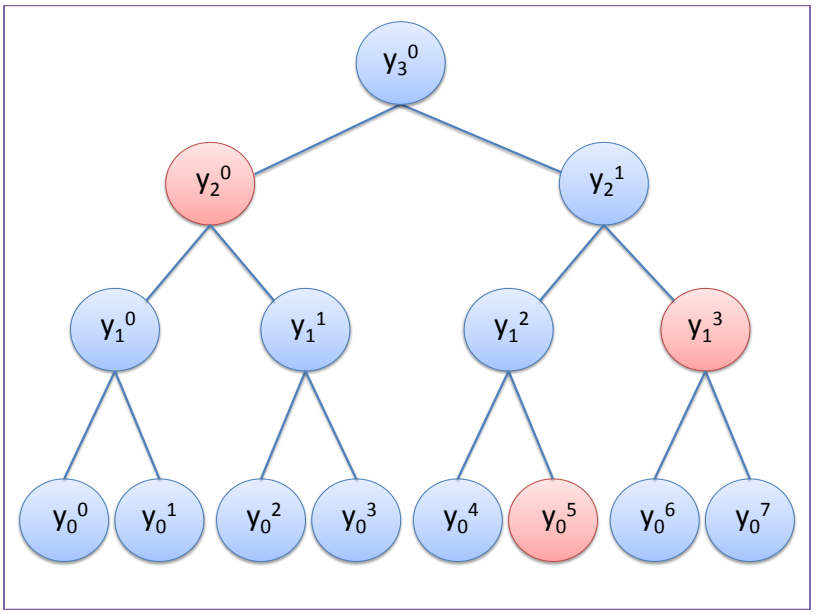

Figure 10.4: A binary tree of height three. The red nodes are the authentication nodes that must be computed before round 4.

We define $g(D)$ to be the total number of hash function evaluations needed to compute authentication nodes throughout the rounds in a tree of height D. We can see that $g(D) = (2^D - 1) + 2g(D-1)$, since until round 2^{D-1} we needed $g(D-1)$ by definition, we saw above that in round $i = 2^{D-1}$ we need $(2^D - 1)$ and after round $i = 2^D$ we are in the same case as in a tree of height $D-1$. This means that $g(D) = \sum_{j=0}^{D-1}(2^D - 2^j) = D2^D - (2^D - 1) = (D-1)2^D + 1$.

As we want to use D units per round, until round $i = 2^{D-1}$ we can use $D2^{D-1}$ units. We have used $g(D-1) = (D-2)2^{D-1}+1$, then we have $D2^{D-1} - ((D-2)2^{D-1}+1) = 2^D - 1$ spare units at the end of the first half. Then, there will just be enough units to prepare for the second half of the rounds.

To conclude, authentication nodes will always be available on time, and the total number of hash function evaluations needed to compute authentication nodes throughout the rounds is $(D-1)2^D + 1$.

10.2.4 Comparisons

Table 10.3 compares our hash tree traversal algorithm with others in terms of the number of hash function evaluations per round, and the maximum memory requirements. The complexities of the descriptions of the algorithms varies substantially.

Table 10.3: Comparison of our hash tree traversal algorithm with others in terms of the number of hash function evaluations per round, and the maximum memory requirements.

Algorithm	Time (HF eval.) per round	Memory requirements
Jakobsson et al. [60]	$2D/\log(D)$	$1.5D^2/\log(D)$
Szydlo [113]	$2D$	$3D-2$
Berman et al. [14]	$2D/\log^2(D)$	$(D/\log^2(D)+1)\log(D)+2D$
Our results	D	$3D-4$

Chapter 11

Conclusion

The security of most public-key cryptosystems used in practice depends only on the hardness of solving the factoring and discrete logarithm problem. This fact is enough motivation to study cryptosystems based on other trapdoor one-way functions. After Peter Shore solved in 1994 the two problems mentioned above, we are even more motivated to find an alterative solution that may be used in classical and quantum computers. Maybe a large quantum computer will never be constructed and maybe no one will ever solve the factoring and discrete logarithm problem for classical computers. But if these ever occurs, we should be able to have an alternative solution that is secure, efficient and that inspires confidence.

The main purpose of these thesis was to study two of the four families that we believe are quantum resistant and hopefully motivate the research in these areas.

Code-based cryptography

The first family we were interested in was the *code-based cryptography*. McEliece cryptosystem was published almost at the same time that the RSA and is very efficient in encryption and decryption, the main drawback is that it has a public key with a huge size. McEliece PKC uses Goppa codes but many variants have been proposed to decrease the public-key size by using a different family of codes.

We know that it is hard to decode random codes; if we also have the statement that the codes in the McEliece system cannot be distinguished from a random code, then decoding the codes appearing in the McEliece system is hard. The distinguisher problem (with binary Goppa codes) was introduced in 2001 by Courtois et al. [34].

Code based public-key cryptosystems are an interesting alternative to classical cryptography since we can build cryptosystems, signatures schemes, hash functions, etc. Even if it may not be interesting to redefine all the cryptographic primitives that we have in classical cryptography, it is still interesting to keep working in this area since we may find even more interesting alternatives. Remember for example that the efficiency in encryption and decryption makes McEliece's PKC suitable for constrained devices [38, 59].

Contributions and future work: In Chapter 5 we present a structural attack on two promising variants: one that uses quasi-cyclic alternant codes by Berger et al. [11] and the other that uses quasi-dyadic matrices by Barreto and Misoczki [85]. This chapter is based on the paper [52], joint work with Gregor Leander. It is very hard to generalize the attacks of the McEliece variants based on different kinds of

codes, since the attacks exploit the structure of the codes and therefore depend on them. In 2011 Persichetti [95] proposed a very similar variant to the one in [85], which uses Srivastava codes. The attack presented in [52] cannot be applicable in this case. It will be interesting as a future work to modify the attack presented in Chapter 5 to be able to attack it. And also to attack the signature scheme based on quasi-dyadic codes proposed by Barreto's et al. [10].

In Chapter 6 we present a deterministic polynomial-time distinguisher (between Goppa, alternant and random codes) for high rate codes, i.e. the range of parameters used in the CFS signature scheme. This is not an attack on the system, but it invalidates the hypothesis of the security proof. This chapter is based on the paper [43], that is a joint work with Jean-Charles Faugère, Ayoub Otmani, Ludovic Perret and Jean-Pierre Tillich.

Hash-based signatures schemes

Digital signatures are very important in electronic commerce, internet security and many other daily applications. In the second part of the thesis we studied *hashed-based signatures schemes*, that a are a good quantum-resistant alternative to the schemes based on trapdoor one-way function. The first proposals are also from the late 70's, when Lamport and Winternitz proposed some one-way signature schemes based on one-way functions and Merkle proposed the chaining methods to be able to sing more than one times. We did on overview on one-time and multiple time signature schemes and we present Merkle's chaining methods. These proposals are not enough efficient to be able to substitute the actual signature schemes like RSA for example.

Contributions and future work: This part is based on joint work with Lars R. Knudsen and Søren S. Thomsen that appears in [64]. We analyzed some of the multiple-time signature schemes (HORS, HORS++, HORSE) and showed that it is better to use multiple times Winternitz's scheme. We proposed a new signature scheme that allows to sign two messages without increasing the public key and signature size, but that requires a non-negligible amount of offline work. We also give a new, simple and efficient algorithm for traversing a tree in tree-based signature schemes.

We still would like to decrease the signature size and to make the chaining methods more efficient in order to have a scheme that could substitute the used schemes, like for example RSA. It is also interesting to ask what is a good security notion for post-quantum signatures and how to make more efficient the schemes in a "quantum-hard instances".

Bibliography

[1] ECRYPT II Yearly Report on Algorithms and Keysizes. http://www.ecrypt.eu.org/documents/D.SPA.17.pdf (accessed 2011/10/19), 2010-2011.

[2] Carlisle M. Adams and Henk Meijer. Security-Related Comments Regarding McEliece's Public-Key Cryptosystem. In Pomerance [97], pages 224–228.

[3] Mohssen Alabbadi and Stephen B. Wicker. Security of Xinmei digital signature scheme. *Electronic Letters*, 28:890–891, 1992.

[4] Mohssen Alabbadi and Stephen B. Wicker. Digital signature scheme based on error-correcting codes. In *IEEE International Symposium on Information – ISIT'93*, volume 26, page 199, 1993.

[5] Mohssen Alabbadi and Stephen B. Wicker. Susceptibility of Digital Signature Schemes Based on Error-Correcting Codes to Universal Forgery. In Andrew Chmora and Stephen B. Wicker, editors, *Error Control, Cryptology, and Speech Compression*, volume 829 of *Lecture Notes in Computer Science*, pages 6–12. Springer, 1993.

[6] Mohssen Alabbadi and Stephen B. Wicker. A Digital Signature Scheme Based on Linear Error-correcting Block Codes. In Josef Pieprzyk and Reihaneh Safavi-Naini, editors, *ASIACRYPT*, volume 917 of *Lecture Notes in Computer Science*, pages 238–248. Springer, 1994.

[7] Roberto M. Avanzi, Simon Hoerder, Dan Page, and Michael Tunstall. Side-Channel Attacks on the McEliece and Niederreiter Public-Key Cryptosystems. In *2nd international Workshop on constructive Side-Channel Analysis and Secure Sedign COSADE*, 2011.

[8] Marco Baldi, Marco Bodrato, and Franco Chiaraluce. A New Analysis of the McEliece Cryptosystem Based on QC-LDPC Codes. In Rafail Ostrovsky, Roberto De Prisco, and Ivan Visconti, editors, *SCN*, volume 5229 of *Lecture Notes in Computer Science*, pages 246–262. Springer, 2008.

[9] Marco Baldi and Franco Chiaraluce. Cryptanalysis of a new instance of McEliece cryptosystem based on QC-LDPC Codes. *IEEE International Symposium on Information Theory*, pages 2591–2595, 2007.

[10] Paulo S. L. M. Barreto, Pierre-Louis Cayrel, Rafael Misoczki, and Robert Niebuhr. Quasi-Dyadic CFS Signatures. In Xuejia Lai, Moti Yung, and Dongdai Lin, editors, *Inscrypt*, volume 6584 of *Lecture Notes in Computer Science*, pages 336–349. Springer, 2010.

[11] Thierry P. Berger, Pierre-Louis Cayrel, Philippe Gaborit, and Ayoub Otmani. Reducing Key Length of the McEliece Cryptosystem. In Bart Preneel, editor, *AFRICACRYPT*, volume 5580 of *Lecture Notes in Computer Science*, pages 77–97. Springer, 2009.

[12] Thierry P. Berger and Pierre Loidreau. How to Mask the Structure of Codes for a Cryptographic Use. *Des. Codes Cryptography*, 35(1):63–79, 2005.

[13] Elwyn R. Berlekamp, Robert J. McEliece, and Henk C.A. van Tilborg. On the inherent intractability of certain coding problems. *IEEE Transactions on Information Theory*, 24(3):384–386, 1978.

[14] Piotr Berman, Marek Karpinski, and Yakov Nekrich. Optimal trade-off for Merkle tree traversal. *Theoretical Computer Science*, 372(1):26–36, 2007.

[15] Daniel J. Bernstein, Johannes Buchmann, and Erik Dahmen, editors. *Post-Quantum Cryptography*. Springer, 2009.

[16] Daniel J. Bernstein, Tanja Lange, and Christiane Peters. Attacking and Defending the McEliece Cryptosystem. In Buchmann and Ding [25], pages 31–46.

[17] Daniel J. Bernstein, Tanja Lange, and Christiane Peters. Wild McEliece. In Biryukov et al. [20], pages 143–158.

[18] Daniel J. Bernstein, Tanja Lange, and Christiane Peters. Smaller Decoding Exponents: Ball-Collision Decoding. In Phillip Rogaway, editor, *CRYPTO*, volume 6841 of *Lecture Notes in Computer Science*, pages 743–760. Springer, 2011.

[19] Thomas A. Berson. Failure of the McEliece Public-Key Cryptosystem Under Message-Resend and Related-Message Attack. In Burton S. Kaliski Jr., editor, *CRYPTO*, volume 1294 of *Lecture Notes in Computer Science*, pages 213–220. Springer, 1997.

[20] Alex Biryukov, Guang Gong, and Douglas R. Stinson, editors. *Selected Areas in Cryptography - 17th International Workshop, SAC 2010, Waterloo, Ontario, Canada, August 12-13, 2010, Revised Selected Papers*, volume 6544 of *Lecture Notes in Computer Science*. Springer, 2011.

[21] Daniel Bleichenbacher and Ueli M. Maurer. Directed Acyclic Graphs, One-way Functions and Digital Signatures. In Yvo Desmedt, editor, *CRYPTO*, volume 839 of *Lecture Notes in Computer Science*, pages 75–82. Springer, 1994.

[22] Jurjen N. Bos and David Chaum. Provably Unforgeable Signatures. In Ernest F. Brickell, editor, *CRYPTO*, volume 740 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 1992.

[23] Wieb Bosma, John J. Cannon, and Catherine Playoust. The Magma Algebra System I: The User Language. *J. Symb. Comput.*, 24(3/4):235–265, 1997.

[24] Johannes Buchmann, Erik Dahmen, and Michael Zydlo. *Post-Quantum Cryptography*, chapter Hash-based Digital Signature Schemes, pages 35–94. In Bernstein et al. [15], 2009.

[25] Johannes Buchmann and Jintai Ding, editors. *Post-Quantum Cryptography, Second International Workshop, PQCrypto 2008, Cincinnati, OH, USA, October 17-19, 2008, Proceedings*, volume 5299 of *Lecture Notes in Computer Science*. Springer, 2008.

[26] Anne Canteaut and Florent Chabaud. Improvements of the Attacks on Cryptosystems Based on Error-Correcting Codes. *Rapport interne du Departement Mathematiques et Informatique*, LIENS:95–21, 1995.

[27] Anne Canteaut and Florent Chabaud. A New Algorithm for Finding Minimum-Weight Words in a Linear Code: Application to McEliece's Cryptosystem and to Narrow-Sense BCH Codes of Length 511. *IEEE Transactions on Information Theory*, 44(1):367–378, 1998.

[28] Anne Canteaut and Nicolas Sendrier. Cryptoanalysis of the Original McEliece Cryptosystem. In Kazuo Ohta and Dingyi Pei, editors, *ASIACRYPT*, volume 1514 of *Lecture Notes in Computer Science*, pages 187–199. Springer, 1998.

[29] Pierre-Louis Cayrel, Ayoub Otmani, and Damien Vergnaud. On Kabatianskii-Krouk-Smeets Signatures. In Claude Carlet and Berk Sunar, editors, *WAIFI*, volume 4547 of *Lecture Notes in Computer Science*, pages 237–251. Springer, 2007.

[30] Charles J. Colbourn and Jeffrey H. Dinitz. *The CRC Handbook of Combinatorial Designs*. CRC Press, 1st edition, 1996.

[31] Colin Cooper. On the distribution of rank of a random matrix over a finite field. *Random Struct. Algorithms*, 17(3-4):197–212, 2000.

[32] Don Coppersmith and Markus Jakobsson. Almost Optimal Hash Sequence Traversal. In Matt Blaze, editor, *Financial Cryptography*, volume 2357 of *Lecture Notes in Computer Science*, pages 102–119. Springer, 2002.

[33] Carlos Coronado. On the security and the efficiency of the merkle signature scheme. Cryptology ePrint Archive, Report 2005/192, 2005. `http://eprint.iacr.org/`.

[34] Nicolas Courtois, Matthieu Finiasz, and Nicolas Sendrier. How to Achieve a McEliece-Based Digital Signature Scheme. In Colin Boyd, editor, *ASIACRYPT*, volume 2248 of *Lecture Notes in Computer Science*, pages 157–174. Springer, 2001.

[35] Erik Dahmen. *Post-quantum signatures for today*. PhD thesis, Technische Universität Darmstadt, 2009.

[36] Whitfield Diffie and Martin E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[37] C. Dods, Nigel Paul Smart, and Martijn Stam. Hash Based Digital Signature Schemes. In Nigel P. Smart, editor, *Cryptography and Coding 2005, Proceedings*, volume 3796 of *Lecture Notes in Computer Science*, pages 96–115. Springer, 2005.

[38] Thomas Eisenbarth, Tim Güneysu, Stefan Heyse, and Christof Paar. MicroEliece: McEliece for Embedded Devices. In Christophe Clavier and Kris Gaj, editors, *CHES*, volume 5747 of *Lecture Notes in Computer Science*, pages 49–64. Springer, 2009.

[39] Daniela Engelbert, Raphael Overbeck, and Arthur Schmidt. A Summary of McEliece-Type Cryptosystems and their Security. Cryptology ePrint Archive, Report 2006/162, 2006. `http://eprint.iacr.org/`.

[40] Paul Erdös, Peter Frankl, and Zoltán Füredi. Families of finite sets in which no set is covered by the union of r others. *Israel Journal of mathematics*, 51:79–89, 1985.

[41] Jean-Charles Faugère. A new efficient algorithm for computing Gröbner bases (F4). *Journal of pure and Applied Algebra*, 139:61–88, 1999.

[42] Jean-Charles Faugère. A new efficient algorithm for computing Gröbner bases without reduction to zero (F5). In *Proceedings of the 2002 international symposium on Symbolic and algebraic computation*, ISSAC '02, pages 75–83, New York, NY, USA, 2002. ACM.

[43] Jean-Charles Faugère, Valérie Gauthier Umaña, Ayoub Otmani, Ludovic Perret, and Jean-Pierre Tillich. A Distinguisher for High Rate McEliece Cryptosystems. In *IEEE Information Theory Workshop (ITW 2011)*, pages 1–5, October 2011.

[44] Jean-Charles Faugère, Ayoub Otmani, Ludovic Perret, and Jean-Pierre Tillich. Algebraic Cryptanalysis of McEliece Variants with Compact Keys. In Henri Gilbert, editor, *EUROCRYPT*, volume 6110 of *Lecture Notes in Computer Science*, pages 279–298. Springer, 2010.

[45] Cédric Faure and Lorenz Minder. Cryptanalysis of the McEliece cryptosystem over hyperelliptic codes. In *Proceedings of the 11th international workshop on Algebraic and Combinatorial Coding Theory, ACCT 2008*, pages 99–107, 2008.

[46] Matthieu Finiasz. Parallel-CFS - Strengthening the CFS McEliece-Based Signature Scheme. In Biryukov et al. [20], pages 159–170.

[47] Matthieu Finiasz and Nicolas Sendrier. Security Bounds for the Design of Code-Based Cryptosystems. In Mitsuru Matsui, editor, *ASIACRYPT*, volume 5912 of *Lecture Notes in Computer Science*, pages 88–105. Springer, 2009.

[48] Ernst M. Gabidulin. Theory of codes with maximum rank distance. *Problems of Information Transmission*, 21(1):1–12, 1985.

[49] Ernst M. Gabidulin, Alexei V. Ourivski, Bahram Honary, and Bassem Ammar. Reducible rank codes and their applications to cryptography. *IEEE Transactions on Information Theory*, 49(12):3289–3293, 2003.

[50] Ernst M. Gabidulin, A. V. Paramonov, and O. V. Tretjakov. Ideals over a Non-Commutative Ring and thier Applications in Cryptology. In Donald Davies, editor, *EUROCRYPT*, volume 547 of *Lecture Notes in Computer Science*, pages 482–489. Springer, 1991.

[51] Philippe Gaborit. Shorter keys for code based cryptography. In Øyvind Ytrehus, editor, *WCC*, volume 3969 of *Lecture Notes in Computer Science*, pages 81–90. Springer, 2005.

[52] Valérie Gauthier Umaña and Gregor Leander. Practical Key Recovery Attacks On Two McEliece Variants. Cryptology ePrint Archive, Report 2009/509, 2009. `http://eprint.iacr.org/`.

[53] J.K Gibson. Equivalent Goppa Codes and Trapdoors to McEliece's Public Key Cryptosystem. In *EUROCRYPT*, volume 547 of *Lecture Notes in Computer Science*, pages 517–521. Springer, 1991.

[54] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. *SIAM J. Comput.*, 17(2):281–308, 1988.

[55] Valery Denisovich Goppa. A new class of linear error-correcting codes. *Problems of Information Transmission*, 6:207–212, 1970.

[56] Lov K Grover. A fast quantum mechanical algorithm for database search. In *STOC*, pages 212–219, 1996.

[57] Chris Hall, Ian Goldberg, and Bruce Schneier. Reaction Attacks against several Public-Key Cryptosystems. In Vijay Varadharajan and Yi Mu, editors, *ICICS*, volume 1726 of *Lecture Notes in Computer Science*, pages 2–12. Springer, 1999.

[58] L. Harn and D.C. Wang. Cryptoanalysis and modification of digital signature scheme based on error-correcting codes. *Electronic Letters*, 28:157–159, 1992.

[59] Stefan Heyse. Low-reiter: Niederreiter encryption scheme for embedded microcontrollers. In Nicolas Sendrier, editor, *PQCrypto*, volume 6061 of *Lecture Notes in Computer Science*, pages 165–181. Springer, 2010.

[60] Markus Jakobsson, Frank Thomson Leighton, Silvio Micali, and Michael Szydlo. Fractal Merkle Tree Representation and Traversal. In Marc Joye, editor, *CT-RSA*, volume 2612 of *Lecture Notes in Computer Science*, pages 314–326. Springer, 2003.

[61] Heeralal Janwa and Oscar Moreno. McEliece Public Key Cryptosystems Using Algebraic-Geometric Codes. *Des. Codes Cryptography*, 8(3):293–307, 1996.

[62] Jørn Justesen and Tom Høholdt. *A Course in Error-Correcting Codes*. European Mathematical Society, 2004.

[63] Gregory Kabatianskii, E. Krouk, and Ben J. M. Smeets. A Digital Signature Scheme Based on Random Error-Correcting Codes. In Michael Darnell, editor, *IMA Int. Conf.*, volume 1355 of *Lecture Notes in Computer Science*, pages 161–167. Springer, 1997.

[64] Lars R. Knudsen, Søren S. Thomsen, and Valérie Gauthier Umaña. On hash-based digital signatures. Submitted to Designs, Codes and Cryptography, 2011.

[65] Kazukuni Kobara. Code-Based Public-Key Cryptosystems and Their Applications. In Kaoru Kurosawa, editor, *ICITS*, volume 5973 of *Lecture Notes in Computer Science*, pages 45–55. Springer, 2009.

[66] Kazukuni Kobara and Hideki Imai. Countermeasure against reaction attacks. *The Symposium on Cryptography and Information Security*, 2000.

[67] Kazukuni Kobara and Hideki Imai. Semantically Secure McEliece Public-Key Cryptosystems-Conversions for McEliece PKC. In Kwangjo Kim, editor, *Public Key Cryptography*, volume 1992 of *Lecture Notes in Computer Science*, pages 19–35. Springer, 2001.

[68] Leslie Lamport. The Writings of Leslie Lamport (web page). `http://research.microsoft.com/en-us/um/people/lamport/pubs/pubs.html` (accessed 2011/05/12).

[69] Leslie Lamport. Constructing digital signatures from a one way function. Technical Report CSL-98, SRI International, October 1979.

[70] Niels Lauritzen. *Concrete Abstract Algebra, From Numbers to Gröbner basis*. Cambridge University Press, 2005.

[71] Pil Joong Lee and Ernest F. Brickell. An Observation on the Security of McEliece's Public-Key Cryptosystem. In *EUROCRYPT*, volume 330 of *Lecture Notes in Computer Science*, pages 275–280. Springer, 1988.

[72] Jeffrey S. Leon. A probabilistic algorithm for computing minimum weights of large error-correcting codes. *IEEE Transactions on Information Theory*, 34(5):1354–1359, 1988.

[73] P. C. Li, G. H. J. van Rees, and R. Wei. Constructions of 2-cover-free families and related separating hash families. *Journal of Combinatorial Designs*, 14:423–440, 2006.

[74] Yuan Xing Li, Robert H. Deng, and Xin mei Wang. On the equivalence of McEliece's and Niederreiter's public-key cryptosystems. *IEEE Transactions on Information Theory*, 40(1):271–273, 1994.

[75] Pierre Loidreau and Nicolas Sendrier. Some weak keys in McEliece public-key cryptosystem. *IEEE International symposium on Information Theory*, page 382, 1998.

[76] Florence Jessie MacWilliams and Neil James Alexander Sloane. *The theory of error-correcting codes*. North Holland, Amsterdam, 1977.

[77] Irene Marquez-Corbella, Edgar Martinez-Moro, and Ruud Pellikaan. Evaluation of public-key cryptosystems based on algebraic geometry codes. In Cardona Castle in, editor, *Proceedings of the Third International Castle Meeting on Coding Theory and Applications*, pages 199–204. Cardona, J. Borges and M. Villanueva Eds, September 11-15 2011.

[78] Robert J. McEliece. A public key cryptosystem based on alegbraic coding theory. *DSN progress report*, 42-44:114–116, 1978.

[79] Ralph Charles Merkle. *Secrecy, authentication, and public key systems*. PhD thesis, Stanford University Stanford, CA, USA, 1979.

[80] Ralph Charles Merkle. Secrecy, authentication, and public key systems. *(Computer science), UMI Research Press*, 1982.

[81] Ralph Charles Merkle. A Digital Signature Based on a Conventional Encryption Function. In Pomerance [97], pages 369–378.

[82] Ralph Charles Merkle. A Certified Digital Signature. In Gilles Brassard, editor, *CRYPTO*, volume 435 of *Lecture Notes in Computer Science*, pages 218–238. Springer, 1989.

[83] Lorenz Minder. *Cryptography based on error-correcting codes*. PhD thesis, EPFL, no. 3846, 2007.

[84] Lorenz Minder and Amin Shokrollahi. Cryptanalysis of the Sidelnikov Cryptosystem. In Moni Naor, editor, *EUROCRYPT*, volume 4515 of *Lecture Notes in Computer Science*, pages 347–360. Springer, 2007.

[85] Rafael Misoczki and Paulo S. L. M. Barreto. Compact McEliece Keys from Goppa Codes. In Michael J. Jacobson Jr., Vincent Rijmen, and Reihaneh Safavi-Naini, editors, *Selected Areas in Cryptography*, volume 5867 of *Lecture Notes in Computer Science*, pages 376–392. Springer, 2009.

[86] William D. Neumann. HORSE: An Extension of an r-Time Signature Scheme With Fast Signing and Verification. In *International Conference on Information Technology: Coding and Computing*, pages 129–134. IEEE Computer Society, 2004.

[87] Harald Niederreiter. Knapsack-type cryptosystems and alegraic coding theory. *Problems of Control and Information Theory*, 15:157–166, 1986.

[88] Ayoub Otmani and Jean-Pierre Tillich. An Efficient Attack on all Concrete KKS Proposals. To appear at PQC 2011.

[89] Ayoub Otmani, Jean-Pierre Tillich, and Léonard Dallot. Cryptanalysis of Two McEliece Cryptosystems Based on Quasi-Cyclic Codes. *CoRR*, abs/0804.0409, 2008.

[90] Alexei V. Ourivski and Ernst M. Gabidulin. Column Scrambler for the GPT Cryptosystem. *Discrete Applied Mathematics*, 128(1):207–221, 2003.

[91] Raphael Overbeck. *Public key cryptography based on coding theory.* PhD thesis, Technische Universität Darmstadt, 2007.

[92] Raphael Overbeck. Structural attacks for public key cryptosystems based on Gabidulin codes. *Journal of cryptology*, 21(2):280–301, 2008.

[93] Raphael Overbeck and Nicolas Sendrier. *Post-Quantum Cryptography*, chapter Code-based cryptography, pages 95–145. In Bernstein et al. [15], 2009.

[94] N. Patterson. The algebraic decoding of Goppa codes. *IEEE Transactions on Information Theory*, 21(2):203–207, 1975.

[95] Edoardo Persichetti. Compact McEliece keys based on Quasi-Dyadic Srivastava codes. Cryptology ePrint Archive, Report 2011/179, 2011. `http://eprint.iacr.org/`.

[96] Josef Pieprzyk, Huaxiong Wang, and Chaoping Xing. Multiple-Time Signature Schemes against Adaptive Chosen Message Attacks. In Mitsuru Matsui and Robert J. Zuccherato, editors, *Selected Areas in Cryptography*, volume 3006 of *Lecture Notes in Computer Science*, pages 88–100. Springer, 2003.

[97] Carl Pomerance, editor. *Advances in Cryptology - CRYPTO '87, A Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, California, USA, August 16-20, 1987, Proceedings*, volume 293 of *Lecture Notes in Computer Science*. Springer, 1988.

[98] Leonid Reyzin and Natan Reyzin. Better than BiBa: Short One-Time Signatures with Fast Signing and Verifying. In Lynn Margaret Batten and Jennifer Seberry, editors, *ACISP*, volume 2384 of *Lecture Notes in Computer Science*, pages 144–153. Springer, 2002.

[99] Ron M. Roth and Gadiel Seroussi. On generator matrices of MDS codes. *IEEE Transaction on Information thepry*, 31(6):826–830, 1985.

[100] Nicolas Sendrier. On the structure of a randomly permuted concatenated code. *EUROCODE 94*, pages 169–173, 1994.

[101] Nicolas Sendrier. An algorithm for finding the permutation betwen two equivalent binary codes. Technical report, Technical Report RR-2460, INRIA, 1996.

[102] Nicolas Sendrier. On the Concatenated Structure of a Linear Code. *Appl. Algebra Eng. Commun. Comput.*, 9(3):221–242, 1998.

[103] Claude Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423 and 623–656, 1948.

[104] Amin Shokrollahi, Chris Monico, and Joachim Rosenthal. Using low density parity check codes in the McEliece cryptosystem. In *IEEE International Symposium on Information Theory (ISIT 2000)*, page 215, 2000.

[105] Peter W. Shor. Algorithms for Quantum Computation: Discrete Logarithms and Factoring. In *IEEE Symposium on Foundations of Computer Science*, pages 124–134, 1994.

[106] Abdulhadi Shoufan, Falko Strenzke, H. Gregor Molter, and Marc Stöttinger. A Timing Attack Against Patterson Algorithm in the McEliece PKC. In *ICISC*, 2009.

[107] Vladimir M. Sidelnikov. A public-key cryptosystem based on a binary Reed-Muller codes. *Discrete Mathematics and aplications, 2,*, 4:439–44, 1992.

[108] Vladimir M. Sidelnikov and Sergey O. Shestakov. On insecurity of cryptosystems based on gerenalized Reed-Solomon codes. *Discrete Mathematics and applications*, 2:439–444, 1992.

[109] Jacques Stern. A method for finding codewords of small weight. In Gérard Cohen and Jacques Wolfmann, editors, *Coding Theory and Applications*, volume 388 of *Lecture Notes in Computer Science*, pages 106–113. Springer, 1989.

[110] Jacques Stern. A New Identification Scheme Based on Syndrome Decoding. In Douglas R. Stinson, editor, *CRYPTO*, volume 773 of *Lecture Notes in Computer Science*, pages 13–21. Springer, 1993.

[111] Douglas R. Stinson. *Cryptography Theory and Practice*. Chapman & Hall/CRC, third edition, 2006.

[112] Falko Strenzke, Erik Tews, H. Gregor Molter, Raphael Overbeck, and Abdulhadi Shoufan. Side channels in the mceliece pkc. In Buchmann and Ding [25], pages 216–229.

[113] Michael Szydlo. Merkle Tree Traversal in Log Space and Time. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT*, volume 3027 of *Lecture Notes in Computer Science*, pages 541–554. Springer, 2004.

[114] Jeremy Thorpe. Low-density Parity-check codes. `http://www.ldpc-codes.com/l` (accessed 2011/10/11).

[115] K. Tzeng and K. Zimmermann. On extending Goppa codes to cyclic codes. In *IEEE Transactions of Information Theory*, volume 21, pages 716–721, 1975.

[116] Johan van Tilburg. Cryptanalysis of Xinmei digital signature scheme. *Electronic Letters*, 28:1935–1936., 1992.

[117] Johan van Tilburg. Cryptanalysis of the Alabbadi-Wicker digital signature scheme. In *Symposium on Information Theory*, pages 114–119, 1993.

[118] Joachim von zur gathen and Jürgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, 2003.

[119] Pascal Véron. *Problème SD, opérateur trace, schémas d'identification et codes de Goppa*. PhD thesis, Université Toulon et du Var, Toulon, France, 1995.

[120] Christian Wieschebrink. Two NP-complete problems in coding theory with an application in code based cryptography. In *IEEE International Symposium on Information Theory*, pages 1733–1737, 2006.

[121] Wang Xinmei. Digital signature scheme based on error-correcting codes. *Electronics Letters*, 26:898–899, 1990.

[122] Irving S. Reed y Gustave Solomon. Polynomial codes over certain finite fields. *Journal of the SIAM*, 8:300–304, 1960.

Appendices

Appendix A

The class $\mathcal{NP}$ and Asymptotic notation

A.1 The class $\mathcal{NP}$

The class $\mathcal{P}$ [13]: This class is defined to be a set of computational problems, which can be solved by an algorithm that is guaranteed to terminate in a number of steps bounded by a polynomial in the length of the input. Thus $\mathcal{P}$ corresponds to the class of *polynomial-time* algorithms.

The class $\mathcal{NP}$ [13]: This class is defined to be a set of computational problems which can be solved by non-deterministic algorithm, whose running time is bounded by a polynomial in the length of the input. A non deterministic algorithm is such that when it is confronted with a choice between two alternatives, it can create two copies of itself and simultaneously follow the consequences of both courses. This repeated splitting may lead to an exponentially growing number of copies; the algorithm is said to solve the given problem if any one of these copies produces the correct answer. $\mathcal{NP}$ correspond to the class of *non-deterministic-polynomial-time* algorithms.

The class $\mathcal{NP}$-hard: This class is defined to be a set of computational problems that are at least as hard as the hardest problems in $\mathcal{NP}$. This means that there exist hard instances, but the average case could be easy.

The class $\mathcal{NP}$-complete: This class is defined to be a set of computational problems that are in the set of $\mathcal{NP}$ problems and also in the set of $\mathcal{NP}$-hard problems, i.e., these are the hardest problems in $\mathcal{NP}$.

A.2 Asymptotic notation

- **Big-Oh notation:** We say that $f(n) \in O(g(n))$ if f is bounded above by g (up to constant factor) asymptotically. i.e., as $n \to \infty$,

$$\exists k > 0, n_0 | \forall (n > n_0), f(n) \leq g(n) \times k$$

- **Little-Oh notation:** We say that $f(n) \in o(g(n))$ if f is dominated by g asymptotically. i.e., as $n \to \infty$,

$$|f(n)| \leq |g(n)| \times \epsilon, \forall \epsilon$$

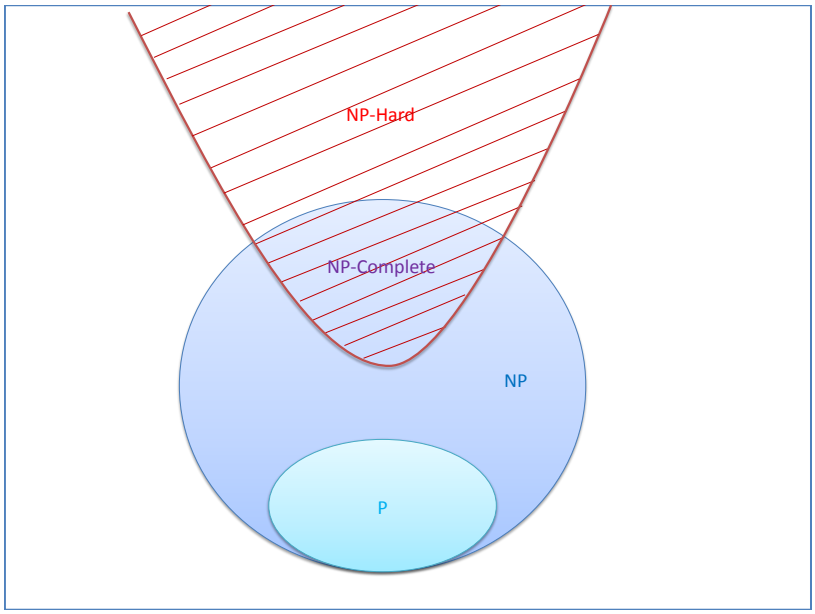

Figure A.1: Diagram for $\mathcal{P}$, $\mathcal{NP}$, $\mathcal{NP}$-hard and $\mathcal{NP}$-complete.

- **Big Omega notation:** We say that $f(n) \in \Omega(g(n))$ if f is bounded below by g (up to constant factor) asymptotically. i.e., as $n \to \infty$,

$$f(n) \geq g(n) \times k \text{ for some positive } k.$$

Appendix B

Definition of some codes

The purpose of this appendix is to give the definitions of the codes used in the first part that haven't been defined. For a more detailed definition and to see the properties of these codes, pleas refer to [62,76]. We are not going to define Low-density parity-check (LDPC) and Algebraic geometry codes, but we refer the reader to [114] and [62].

B.1 Combination of codes

One idea to obtain a long code is to do a combination of shorter codes, in this section we will introduce two examples. For a more complete description please refer to [62, Chapter 10] and [76, page 307].

Product codes: Is the simplest form of composite codes.

Definition B.1. *A product code is a vector space of n_1 by n_2 arrays such that each row is a codeword in a linear (n_1, k_1, d_1) code, and each column is a codeword in a linear (n_2, k_2, d_2) code.*

Theorem B.1. *The parameters of a product code are $(N, K, D) = (n_1 n_2, k_1 k_2, d_1 d_2)$.*

Concatenated codes: These codes are also called *serial encoding*, since the mane idea is to "place" codes next to each other and, the output of a code is the input of the next one. The term "*concatenated*" was originally used to indicate the combination of a Reed Solomon code and a binary code. In Figure B.1 we can see how the codes are "placed", the inner encoder and decoder use an (n, k, d) binary code (called *inner code*). Now, let see the combination of inner encoder, channel and inner decoder as a new channel, called *superchannel*, that transmit elements in $\mathbb{F}_2^k$. We can correct errors of the elements transmitted through the superchannel using an (N, K, D) code over $\mathbb{F}_2^k$, that we call *outer code*. The overall code (called sueprcode) is a binary code of length nN, dimension kK and rate $\frac{kn}{KN}$ [76, page 307].

Theorem B.2. *[62, page 113] The minimum distance of the concatenated code is at least $D \times d$.*

B.2 Other codes

Reed Muller (RM) codes: This section is based on [76, Chapter 13]. Reed Muller codes are one of the oldest and more studied codes and they are easy to decode.

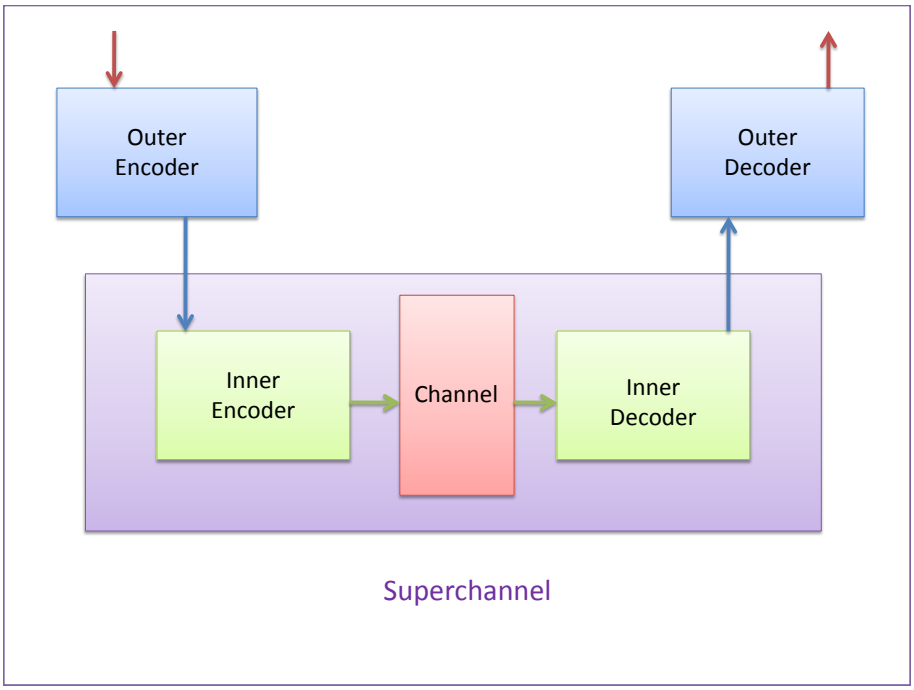

Figure B.1: Concatenated codes.

We fix $n = 2^m$ and $v = (v_1, \ldots, v_m) \in \mathbb{F}_2^m$. A Boolean function is any function $f(v) = f(v_1, \ldots, v_m)$ which output is 0 or 1.

Definition B.2. *The r^{th} order binary Reed-Muller (RM) code $\mathscr{R}(r,m)$ of length $n = 2^m$, for $0 \leq r \leq m$, is the set of all vectors f, where $f(v_1, \ldots, v_m)$ is a Boolean function which is a polynomial of degree at most r.*

Theorem B.3. *$\mathscr{R}(r, m)$ has minimum distance 2^{m-r}.*

Rank metric (Gabidulin) codes: The following definitions are from [12]. Let $\mathbb{F}_q$ be a finite field, and $a = (a_1, a_2, \ldots, a_n) \in \mathbb{F}_{q^m}^n$, the *rank weight* of a is by definition the rank of the $m \times n-$ matrix over $\mathbb{F}_q$ formed by extending every coordinate a_i on a basis of $\mathbb{F}_{q^m}/\mathbb{F}_q$. The rank weight define a metric.

Definition B.3. *Let $\mathscr{C}$ be a linear code over $\mathbb{F}_{q^m}$, the minimum rank distance of $\mathscr{C}$ is $d = min_{c \in \mathscr{C}^*}(rank(c))$.*

Definition B.4. *Let X be a $k \times n$ matrix with coefficients in $\mathbb{F}_{q^m}$. The column rank of X over $\mathbb{F}_q$ is equal to the maximum number of columns of X that are linearly independent over $\mathbb{F}_q$.*

In 1985 Gabidulin proposed a family of codes which are optimal for rank metric [48]. We fix $(g_1, g_2, \ldots, g_n)$, n elements in $\mathbb{F}_{q^m}$ which are linearly independent over q, the Gabidulin codes are defined by the following generator matrix

$$G = \begin{pmatrix} g_1^{[0]} & g_2^{[0]} & \cdots & g_n^{[0]} \\ g_1^{[1]} & g_2^{[1]} & \cdots & g_n^{[1]} \\ \vdots & \vdots & \ddots & \vdots \\ g_1^{[k-1]} & g_2^{[k-1]} & \cdots & g_n^{[k-1]} \end{pmatrix}$$

where $[i] = q^i$.

Appendix C

Gröbner basis

Gröbner basis have been defined in two simultaneous and independent works. In 1964 Hironaka introduced them in his work on resolution of singularities over $\mathbb{C}$ (for which he received the Field medal), and used the term *"standard basis"*. In 1965 Buchberger gave an independent description in his PhD thesis, he named them "Gröbner basis" in honor of his advisor W. Gröbner. In order to introduce them, we will first give some of the definitions and properties that we need, then we will give the Hilbert algorithm (that guarantees the existence of the basis), and finally we will explain how we can use these basis to solve a system of equations. This Appendix is based on [118, Chapter 21] and [70, Chapter 5].

C.1 Preliminaries

Let $\mathbb{F}$ be a field, $R = \mathbb{F}[x_1, \ldots, x_n]$ a polynomial ring in n variables over $\mathbb{F}$, and $f_1, \ldots, f_s \in R$. The polynomials $f_1, \ldots, f_s$ form a basis of the ideal

$$I = \langle f_1, \ldots, f_s \rangle = \Big\{ \sum_{1 \leq i \leq s} q_i f_i | q_i \in R \Big\}.$$

Definition C.1. *[118, Page 585]*

- *A **partial order** $<$ on a set S is an irreflexive and transitive relation, so that*

$$not(\alpha < \alpha) \text{ and } \alpha < \beta < \gamma \Rightarrow \alpha < \gamma \ \forall \alpha, \beta, \gamma \in S.$$

- *A partial order is a **total order** (or simply order) if either $\alpha = \beta$ or $\alpha < \beta$ or $\beta < \alpha$, for all $\alpha, \beta \in S$.*
- *A **well order** is a total order such that every nonempty subset of S has a least element.*

Definition C.2. *[118, Page 586] A **monomial order** in $R = \mathbb{F}[x_1, \ldots, x_n]$ is a relation $\prec$ on $\mathbb{N}^n$ such that*

1. *$\prec$ is a well order.*
2. *$\alpha \prec \beta \Rightarrow \alpha + \gamma \prec \beta + \gamma$ for all $\alpha, \beta, \gamma \in \mathbb{N}^n$*

Definition C.3. *[118, Page 587] Let $f = \sum_{\alpha \in \mathbb{N}^n} c_\alpha x^\alpha \in R$ be a nonzero polynomial with all $c_\alpha \in \mathbb{F}$ (not all zero), and $\prec$ a monomial order.*

1. *Each* $c_\alpha x^\alpha$ *with* $c_\alpha \neq 0$ *is a* ***term*** *of* f.

2. *The* ***multidegree*** *of* f *is* $mdeg(f) = max_\prec\{\alpha \in \mathbb{N}^n : c_\alpha \neq 0\}$, *where* $max_\prec$ *is the maximum with respect to* $\prec$.

3. *The* ***leading coefficient*** *of* f *is* $lc(f) = c_{mdeg(f)} \in \mathbb{F} \setminus \{0\}$.

4. *The* ***leading monomial*** *of* f *is* $lm(f) = x^{mdeg(f)} \in \mathbb{R}$.

5. *The* ***leading term*** *of* f *is* $lt(f) = lc(f) \times lm(f) \in \mathbb{R}$.

Algorithm 18 Multivariate division with remainder algorithm [118, Page 589]

- **Input:** Nonzero polynomials $f, f_1, \ldots, f_s \in R = \mathbb{F}[x_1, \ldots, x_n]$, where $\mathbb{F}$ is a field, and a monomial order $\prec$ on R.
- **Output:** $q_1, \ldots, q_s, r \in R$ such that $f = q_1 f_1 + \cdots + q_s f_s = r$ and no monomial in r is divisible by any of $lt(f_1), \ldots, lt(f_s)$.

1. $r \leftarrow 0$, $p \leftarrow f$ **for** $i = 1, \ldots s$ **do** $r \leftarrow 0$.
2. **while** $p \neq 0$ **do**
3. **if** $lt(f_i)$ divides $lt(p)$ for some $i \in \{1, \ldots, s\}$
   then choose some such i, $q_i \leftarrow q_i + \frac{lt(p)}{lt(f_i)}$, $p \leftarrow p - \frac{lt(p)}{lt(f_i)} f_i$
   else $r \leftarrow r + lt(p)$, $p \leftarrow p - lt(p)$
4. **Return** $q_1, \ldots, q_s, r$.

This algorithm may not be unique, the remainder r depend on the order of $f_1, \ldots, f_s$ (like in the following example). We will like to have a generating set such that the remainder is independent of the order of its elements. This set exist and is called a Gröbner basis.

Example (from [118, Page 588]): We define the *lexicographical order* ($\prec_{lex}$) by

$$\alpha \prec_{lex} \beta \iff \text{the leftmost nonzero entry in } \alpha - \beta \text{ is negative.}$$

Now let $\prec = \prec_{lex}$, $f = xy^2 + 1$, $f_1 = xy + 1$ and $f_2 = y + 1$. We want to apply the multivariate division with remainder algorithm, in order to find q_1, q_2 and r in R such that $f = q_1 f_1 + q_2 f_2 + r$ and no monomial in r is divisible by $lt(f_1)$ or $lt(f_2)$. In Table C.1 we can see the division, in the left side it starts with f_2 and in the right one with f_1. We can see that in fact the remainder output depend of the order of the set.

Table C.1: Example of multivariate division with remainder.

	$xy+1$	$y+1$		$xy+1$	$y+1$
xy^2+1		xy	xy^2+1	y	
$-(xy^2+xy)$			$-(xy^2+y)$		
$-xy+1$		$-x$	$-y+1$		-1
$-(-xy-x)$			$-(-y-1)$		
$x+1$			2		

C.2 Existence

Lemma C.1. *[118, Page 594] Let I be an ideal in $R = \mathbb{F}[x_1, \ldots, x_n]$. If $G \subseteq I$ is a finite subset such that $\langle lt(G)\rangle = \langle lt(I)\rangle$, then $\langle G\rangle = I$.*

Theorem C.1 (Hilbert basis theorem.)**.** *[118, Page 594] Every ideal I in $R = \mathbb{F}[x_1, \ldots, x_n]$ is finitely generated. More precisely, there exists a finite subset $G \subseteq I$ such that $\langle G\rangle = I$ and $\langle lt(G)\rangle = \langle lt(I)\rangle$.*

Definition C.4 (Gröbner basis)**.** *[118, Page 594] Let $\prec$ be a monomial order and $I \subseteq R$ an ideal. A finite set $G \subseteq I$ is a Gröbner basis for I with respect to $\prec$ if $\langle lt(G)\rangle = \langle lt(I)\rangle$.*

Corollary C.1. *[118, Page 595] Every ideal I in $R = \mathbb{F}[x_1, \ldots, x_n]$ has a Gröbner basis.*

A Gröbner basis is not unique, you can always add another polynomial $g_* \in I$ to the list of polynomials in the Gröbner basis $(g_1, \ldots, g_r)$. Then $(g_1, \ldots, g_r, g_*)$ will also be a Gröbner basis. This lead us to the following definition:

Definition C.5. *[70, Page 212] A reduced Gröbner basis $(g_1, \ldots, g_r)$ is a minimal Gröbner basis such that no term in g_i is divisible by $lt(g_i)$ for $i \neq j$.*

Buchberger proposed a method for computing Gröbner basis that is explained for example in [70,118]. There are other proposal to find these basis, like for example F4 and F5 due to Faugère [41, 42], that are optimizations of Buchberger's algorithm.

C.3 Solving equations using Gröbner basis

Gröbner bases can be applied for solving systems of non-linear (polynomial) equations. Let $I = \langle f_1, \ldots, f_s\rangle$ as before, we define the *variety of I* by

$$V(I) = \{u \in \mathbb{F}^n | f(u) = 0 \forall f \in I\} = \{u \in \mathbb{F}^n | f_1(u) = \cdots = f_s(u) = 0\}.$$

This variety is also denoted $V(f_1, \ldots, f_s)$ instead of $V(\langle f_1, \ldots, f_s\rangle)$ for short. Let $(g_1, \ldots, g_r)$ be a Gröbner basis of I, we get that

$$V(f_1, \ldots, f_s) = V(g_1, \ldots, g_r).$$

Often solving the system $g_1(x_1, \ldots, x_n) = 0, \ldots, g_r(x_1, \ldots, x_n) = 0$ is much easier than solving $f_1(x_1, \ldots, x_n) = 0, \ldots, f_s(x_1, \ldots, x_n) = 0$. The main idea is to eliminate variables by combining some equations. In the ideal case we will find equations that depend only on x_1, then only on x_1 and x_2, etc.

Theorem C.2. *[70, Page 215] Let $G = (g_1, \ldots, g_r)$ be a Gröbner basis for an ideal $I \subseteq R = \mathbb{F}[x_1, \ldots, x_n]$ with respect to the lexicographic ordering $\preceq_{lex}$ given by $x_1 \preceq_{lex} x_2 \preceq_{lex} \cdots \preceq_{lex} x_n$. Then $(G \cap R)$ is a Gröbner basis for the ideal $(I \cap R)$ in R.*

Example (from [70, Page 216]): find the solution of the system of equations in $\mathbb{R}^2$.

$$\begin{cases} y^2 - x^3 + x &= 0, \\ y^3 - x^2 &= 0 \end{cases} \tag{C.3.1}$$

By using Buchberger's algorithm we can find that the reduced Gröbner basis is $(y^3 - y^4 - 2y^6 + y^9, x + y^2 + y^4 - y^7)$. Then solve the system of equations C.3.1 is equivalent to find the solution of:

$$\begin{cases} y^3 - y^4 - 2y^6 + y^9 &= 0, \\ x + y^2 + y^4 - y^7 &= 0 \end{cases} \tag{C.3.2}$$

i.e., to solve

$$\begin{cases} y^3(1 - y - 2y^3 + y^6) &= 0, \\ x &= -y^2 - y^4 + y^7. \end{cases} \tag{C.3.3}$$

Appendix D

Experimental results for the distinguisher

We gathered samples of results we obtained through intensive computations with the Magma system [23] in order to confirm the formulas. We randomly generated alternant and Goppa codes over the field $\mathbb{F}_q$ with $q \in \{2, 4, 8, 16, 32\}$ for values of r in the range $\{3, \ldots, 50\}$ and several m. The Goppa codes are generated by means of an irreducible $g(z)$ of degree r and hence $g(z)$ has no multiple roots. In particular, we can apply Theorem 2.7 in the binary case. We compare the dimensions of the solution space against the dimension D_{random} of the system derived from a random linear code. Tables D.1-D.3 give figures for the binary case with $m = 14$. We define $T_{\mathsf{alternant}}$ and T_{Goppa} respectively as the expected normalized dimensions for an alternant and a Goppa code deduced from the formulas (6.1.3) and (6.1.4). We can check that D_{random} is equal to 0 for $r \in \{3, \ldots, 12\}$ and $D_{\text{random}} = N - k$ as expected. We remark that $D_{\text{alternant}}$ is different from D_{random} whenever $r \leq 15$, and D_{Goppa} is different from D_{random} as long as $r \leq 25$. Finally we observe that our formulas for $T_{\mathsf{alternant}}$ fit as long as $k \geq N - mT_{\mathsf{alternant}}$ which correspond to $r \leq 15$. This is also the case for binary Goppa codes since we have $mT_{\mathsf{Goppa}} = D_{\text{Goppa}}$ as long as $k \geq N - mT_{\mathsf{Goppa}}$ i.e., $r \leq 25$. We also give in Tables D.12-D.14 the examples we obtained for $q = 4$ and $m = 6$ to check that the arguments also apply. We also compare binary Goppa codes and random linear codes for $m = 15$ in Tables D.4-D.7 and $m = 16$ in Tables D.8-D.11. We see that D_{random} and D_{Goppa} are different for $r \leq 33$ when $m = 15$ and for $m = 16$ they are different even beyond our range of experiment $r \leq 50$.

Table D.1: $q = 2$ and $m = 14$

r	3	4	5	6	7	8	9	10	11	12
N	861	1540	2415	3486	4753	6216	7875	9730	11781	14028
k	16342	16328	16314	16300	16286	16272	16258	16244	16230	16216
D_{random}	0	0	0	0	0	0	0	0	0	0
$D_{\text{alternant}}$	42	126	308	560	882	1274	1848	2520	3290	4158
$mT_{\mathsf{alternant}}$	42	126	308	560	882	1274	1848	2520	3290	4158
D_{Goppa}	252	532	980	1554	2254	3080	4158	5390	6776	8316
mT_{Goppa}	252	532	980	1554	2254	3080	4158	5390	6776	8316

Table D.2: $q = 2$ and $m = 14$

r	13	14	15	16	17	18	19	20	21
N	16471	19110	21945	24976	28203	31626	35245	39060	43071
k	16202	16188	16174	16160	16146	16132	16118	16104	16090
D_{random}	269	2922	5771	8816	12057	15494	19127	22956	26981
$D_{\text{alternant}}$	5124	6188	7350	8816	12057	15494	19127	22956	26981
$mT_{\mathsf{alternant}}$	5124	6188	7350	8610	10192	11900	13734	15694	17780
D_{Goppa}	10010	11858	13860	16016	18564	21294	24206	27300	30576
mT_{Goppa}	10010	11858	13860	16016	18564	21294	24206	27300	30576

Table D.3: $q = 2$ and $m = 14$

r	22	23	24	25	26	27	28	29	30
N	47278	51681	56280	61075	66066	71253	76636	82215	87990
k	16076	16062	16048	16034	16020	16006	15992	15978	15964
D_{random}	31202	35619	40232	45041	50046	55247	60644	66237	72026
$D_{\text{alternant}}$	31202	35619	40232	45041	50046	55247	60644	66237	72026
$mT_{\mathsf{alternant}}$	19992	22330	24794	27384	30100	32942	35910	39004	42224
D_{Goppa}	34034	37674	41496	45500	50046	55247	60644	66237	72026
mT_{Goppa}	34034	37674	41496	45500	49686	54054	58604	63336	68250

Table D.4: $q = 2$ and $m = 15$

r	3	4	5	6	7	8	9	10	11	12	13
N	990	1770	2775	4005	5460	7140	9045	11175	13530	16110	18915
k	32723	32708	32693	32678	32663	32648	32633	32618	32603	32588	32573
D_{random}	0	0	0	0	0	0	0	0	0	0	0
D_{Goppa}	270	570	1050	1665	2415	3300	4455	5775	7260	8910	10725
mT_{Goppa}	270	570	1050	1665	2415	3300	4455	5775	7260	8910	10725

Table D.5: $q = 2$ and $m = 15$

r	14	15	16	17	18	19	20	21	22	23	24
N	21945	25200	28680	32385	36315	40470	44850	49455	54285	59340	64620
k	32558	32543	32528	32513	32498	32483	32468	32453	32438	32423	32408
D_{random}	0	0	0	0	3817	7987	12382	17002	21847	26917	32212
D_{Goppa}	12705	14850	17160	19890	22815	25935	29250	32760	36465	40365	44460
mT_{Goppa}	12705	14850	17160	19890	22815	25935	29250	32760	36465	40365	44460

Table D.6: $q = 2$ and $m = 15$

r	25	26	27	28	29	30	31	32	33	34
N	70125	75855	81810	87990	94395	101025	107880	114960	122265	129795
k	32393	32378	32363	32348	32333	32318	32303	32288	32273	32258
D_{random}	37732	43477	49447	55642	62062	68707	75577	82672	89992	97537
D_{Goppa}	48750	53235	57915	62790	67860	73125	78585	84240	90585	97537
mT_{Goppa}	48750	53235	57915	62790	67860	73125	78585	84240	90585	97155

Table D.7: $q = 2$ and $m = 15$

r	35	36	37	38	39	40	41	42	43	44
N	137550	145530	153735	162165	170820	179700	188805	198135	207690	217470
k	32243	32228	32213	32198	32183	32168	32153	32138	32123	32108
D_{random}	105307	113302	121522	129967	138637	147532	156652	165997	175567	185362
D_{Goppa}	105307	113302	121522	129967	138637	147532	156652	165997	175567	185362
mT_{Goppa}	103950	110970	118215	125685	133380	141300	149445	157815	166410	175230

Table D.8: $q = 2$ and $m = 16$

r	3	4	5	6	7	8	9	10	11	12	13
N	1128	2016	3160	4560	6216	8128	10296	12720	15400	18336	21528
k	65488	65472	65456	65440	65424	65408	65392	65376	65360	65344	65328
D_{random}	0	0	0	0	0	0	0	0	0	0	0
D_{Goppa}	288	608	1120	1776	2576	3520	4752	6160	7744	9504	11440
mT_{Goppa}	288	608	1120	1776	2576	3520	4752	6160	7744	9504	11440

Table D.9: $q = 2$ and $m = 16$

r	14	15	16	17	18	19	20	21	22	23	24
N	24976	28680	32640	36856	41328	46056	51040	56280	61776	67528	73536
k	65312	65296	65280	65264	65248	65232	65216	65200	65184	65168	65152
D_{random}	0	0	0	0	0	0	0	0	0	2360	8384
D_{Goppa}	13552	15840	18304	21216	24336	27664	31200	34944	38896	43056	47424
mT_{Goppa}	13552	15840	18304	21216	24336	27664	31200	34944	38896	43056	47424

Table D.10: $q = 2$ and $m = 16$

r	25	26	27	28	29	30	31	32	33	34
N	79800	86320	93096	100128	107416	114960	122760	130816	139128	147696
k	65136	65120	65104	65088	65072	65056	65040	65024	65008	64992
D_{random}	14664	21200	27992	35040	42344	49904	57720	65792	74120	82704
D_{Goppa}	52000	56784	61776	66976	72384	78000	83824	89856	96624	103632
mT_{Goppa}	52000	56784	61776	66976	72384	78000	83824	89856	96624	103632

Table D.11: $q = 2$ and $m = 16$

r	35	36	37	38	39	40	41	42	43
N	156520	165600	174936	184528	194376	204480	214840	225456	236328
k	64976	64960	64944	64928	64912	64896	64880	64864	64848
D_{random}	91544	100640	109992	119600	129464	139584	149960	160592	171480
D_{Goppa}	110880	118368	126096	134064	142272	150720	159408	168336	177504
mT_{Goppa}	110880	118368	126096	134064	142272	150720	159408	168336	177504

Table D.12: $q = 4$ and $m = 6$

r	3	4	5	6	7	8	9	10	11	12
N	153	276	435	630	861	1128	1431	1770	2145	2556
k	4078	4072	4066	4060	4054	4048	4042	4036	4030	4024
D_{random}	0	0	0	0	0	0	0	0	0	0
$D_{\text{alternant}}$	6	18	60	120	198	294	408	540	690	858
$mT_{\mathsf{alternant}}$	6	18	60	120	198	294	408	540	690	858
D_{Goppa}	18	60	120	198	294	408	540	750	990	1260
mT_{Goppa}	18	60	120	198	294	408	540	750	990	1260

Table D.13: $q = 4$ and $m = 6$

r	13	14	15	16	17	18	19	20	21
N	3003	3486	4005	4560	5151	5778	6441	7140	7875
k	4018	4012	4006	4000	3994	3988	3982	3976	3970
D_{random}	0	0	0	560	1157	1790	2459	3164	3905
$D_{\text{alternant}}$	1044	1248	1470	1710	2064	2448	2862	3306	3905
$mT_{\mathsf{alternant}}$	1044	1248	1470	1710	2064	2448	2862	3306	3780
D_{Goppa}	1560	1890	2250	2640	3060	3510	3990	4500	5040
mT_{Goppa}	1560	1890	2250	2640	3060	3510	3990	4500	5040

Table D.14: $q = 4$ and $m = 6$

r	22	23	24	25	26	27	28	29	30
N	8646	9453	10296	11175	12090	13041	14028	15051	16110
k	3964	3958	3952	3946	3940	3934	3928	3922	3916
D_{random}	4682	5495	6344	7229	8150	9107	10100	11129	12194
$D_{\text{alternant}}$	4682	5495	6344	7229	8150	9107	10100	11129	12194
$mT_{\mathsf{alternant}}$	4284	4818	5382	5976	6600	7254	7938	8652	9396
D_{Goppa}	5610	6210	6840	7500	8190	9107	10100	11129	12194
mT_{Goppa}	5610	6210	6840	7500	8190	8910	9660	10440	11250