**UNIVERSITY OF TRENTO**

DEPARTMENT OF MATHEMATICS

MASTER DEGREE IN MATHEMATICS

# AN ATTACK ON A RECENT CLASS OF POST-QUANTUM CFS-LIKE DIGITAL SIGNATURE SCHEMES

*Graduand:*
Paolo Piasenti

*Supervisor:*
Dr. Alessio Meneghetti

*Co-Supervisor:*
Giuseppe D'Alconzo

Academic Year 2021/2022

*To all those*
*who will never read the following pages.*
*Don't worry, I gotcha.*

*Why do we fall, Bruce?*

# Contents

# Introduction

Today's public-key cryptography heavily relies on some difficult-to-solve mathematical problems like factoring a huge integer or finding the order of a certain element within a group (this last known as DLP, short for Discrete Logarithm Problem). Right now, no polynomial-time algorithm running on a *classical* computer and able to overcome these challenges is known. This means that cryptographic schemes depending on these problems can still be considered safe and their security unaffected, in the sense that an eventual malicious intruder would require an unfeasible amount of time to perform their attack. But what if more powerful computers were to be invented?

This is exactly what is happening nowadays with the ever more concrete advent of *quantum computers*. More in detail, in 1994 Peter Shor conceived an algorithm running on a quantum machine able to solve both the integer factorization problem and DLP in any group in polynomial time (and space). Clearly, this represents a serious threat to the whole cryptographic asset, or at least to the current one. The development of quantum computers is real and relatively fast and cryptographic research, as its counterpart, is trying to keep up in an attempt to establish new post-quantum standards by continuously constructing up-to-date quantum-resistant cryptosystems. It is exactly in light of this that in December 2016 the National Institute of Standards and Technology (NIST) launched a competition, which goes by the name of NIST PQC, to determine one or more new state-of-the-art protocols that the entire community will agree upon. The competition virtually ended in July 2022 with a standardization of four cryptosystems: one key-establishment protocol and three digital signature schemes. NIST has also planned to issue a new call for proposals for post-quantum digital signature algorithms by the end of summer 2022, primarily in the perspective of diversifying its signature portfolio.

In this work, we will focus specifically on *digital signature* algorithms (DS), which are mathematical mechanisms used to validate the authenticity and the integrity of a message (and can hence be defined as the modern cyber replacement to signing documents with paper and pen), and especially on code-based ones. Indeed, one of the few effective (and therefore most studied and understood) options to build post-quantum cryptographic schemes is to employ coding theory and error-correcting codes in order to obtain unorthodox trapdoors essentially based on decoding problems. More in detail, this thesis will be divided as specified below.

In the first chapter, a comprehensive preamble to the framework we are going to explore is drawn, inclusive of several mathematical and cryptographic backgrounds and extensive for what concerns the classic literature about code-based cryptography (that is, McEliece and Niederreiter schemes, to be concise).

The second chapter is then devoted to the presentation of the history of code-based post-quantum cryptography with a brief roundup of the latest panorama and the most prominent and renowned results. A special emphasis is laid on two historical proposals, namely CFS and KKS, which laid the foundation for the adaptation of the *hash-and-sign* heuristic to code-based protocols (to be specific, trying to solve the problem of the decodability of the digest, caused by the irreversibility of the code-based encrypting function, as we are going to discuss in depth).

Finally, a specific CFS-like digital signature scheme is reviewed and analyzed in the third chapter. Moreover, we point out some security issues affecting this signing algorithm and show a simple forgery attack that can be mounted even against a given type of generalization of such a scheme. In conclusion, some general arguments and theoretical ideas regarding the improvement of the strategy to obtain a decodable hash are discussed.

# Chapter 1

# Preliminaries

In this chapter, we will set up the foundations for the development of our argument in terms of contextualization and basic notions.

## 1.1 The Setting

Contemporary public-key cryptography bases its security on some tricky and troublesome mathematical challenges which can ultimately be led back to a single crucial problem known as *Discrete Logarithm Problem*, or simply DLP, whose formulation is as follows.

Let $G$ be a finite abelian group of order $n$. For generic $g \in G$ and $x \in \mathbb{Z}$ we define

$$g^x := \begin{cases} \underbrace{g \cdot \ldots \cdot g}_{x \text{ times}} & \text{if } x > 0 \\ \mathbb{1}_G & \text{if } x = 0 \\ \underbrace{g^{-1} \cdot \ldots \cdot g^{-1}}_{x \text{ times}} & \text{if } x < 0 \end{cases}$$

where, using the multiplicative notation, $\cdot$ is the operation in $G$, the term $g^{-1}$ denotes the inverse of $g$ and $\mathbb{1}_G$ is the neutral element of $G$. Also, we denote with $\langle g \rangle \subseteq G$ the (cyclic) subgroup generated by $g$.

**Problem 1** (DLP)**.** Let $G$ be a finite abelian group.
GIVEN: $g, h \in G$ such that $h \in \langle g \rangle$.
FIND: $x$ such that $h = g^x$.

All the most popular and widespread key-exchange protocols are based on DLP, i.e. Diffie-Hellman (both the $\mathbb{Z}_p^*$ variant and the Elliptic-Curve one) and RSA. Indeed, even if the latter's security is designed to rely on the issue of factorizing a huge integer number, the problem can actually be traced back to the computation of a discrete logarithm, to some extent[1]. In particular, if you want to factor $N$, with high probability it will suffice to compute discrete logarithms modulo $N$, as accurately explained in [7].

Up to now, no polynomial-time algorithm accomplishing these tasks has been discovered, making us suspect that such a quick resolvent technique may possibly *not exist*. Recalling the definitions of the different computational complexity classes, it can be said that there are many suspicions that DLP does not belong to P. However, it has not (yet) been proven that such an efficient algorithm cannot exist. Furthermore, one can easily see that DLP belongs to NP, even though there are reasons to think that it is not NP-complete. All these arguments make sense in a *classical* computational framework, with a regular Turing machine as a reference model. Yet, how does the situation change if we take into account also the *quantum* computational framework?

### 1.1.1 The Quantum Threat

In 1994, the American mathematician Peter Shor working at Bell Labs in Murray Hill (New Jersey) formulated an algorithm able to solve DLP and the prime factorization problem in polynomial time. In actual fact, the problem addressed is a more general problem having these lasts as particular subinstances, known as *Hidden Subgroup Problem* (HSP); see [15] for further details. The downside of his brilliant idea, which is contained in his masterpiece [56], is that his algorithm is conceived to run on a quantum computer instead of on a classical one. In this regard, a natural question arises: what is a quantum computer in concrete terms?

In a nutshell, a quantum computer is a computational device whose basic unit of information is a so-called *qubit*, instead of the traditional bit. In practice, the concrete realization of a qubit is obtained by the (indirect) observation of a particle (like a photon or an electron) examined by taking into account the principles of quantum mechanics. Without going too far into the matter, it is worth to say that instead of assuming just the two status values "0" and "1" (as a normal bit does), a qubit can occupy any state which is in a coherent *superposition* of both the previous ones. Thereby, this innova-

[1]please note that we are **not** saying that there exists a reduction between the two problems; to avoid misunderstandings, please consult [7] for a proper comprehension.

tive approach that models the amount of information carried by a basic unit marks the transition from the traditional discrete design to a continuous one. As one might guess, it is clear that a computer able to store information in this way has a massive advantage in terms of performance over a classical computer. Indeed, besides solving HSP, a quantum computer can drastically reduce the time needed to solve any generic computational problem, exploiting Grover's quantum search algorithm (first described in [38]) to achieve a quadratic speedup over the normal brute-force alternative. This is the reason why also symmetric ciphers like AES [23], although still inherently secure, need to be revised and updated by increasing the key lengths.

Up to now, no real quantum computer has been built. Rather, what exists are only prototypes, i.e. machines that are not yet able to launch quantum algorithms on cryptographically relevant inputs. However, large companies are steadily investing in these new technologies. Their development proceeds at very high speed and although quantum processors can only complete simple tasks at the moment, the day quantum computers will gain the upper hand is not that far away. According to professor Michele Mosca, director of the Institute for Quantum Computing at the University of Waterloo:

> *"There is a 1 in 7 chance that some fundamental public-key crypto will be broken by quantum by 2026, and a 1 in 2 chance of the same by 2031."*

This notion of complete evolution of quantum computers to the point of solving hitherto intractable problems takes the name of *quantum supremacy*.

### 1.1.2 Post-Quantum Cryptography

Post-Quantum Cryptography is the science of designing new cryptographic algorithms which are able to resist against cryptanalytic attacks by a quantum computer. In general, this goal can be achieved by considering problems which are tricky for both standard and quantum computers. Nevertheless, this kind of approach aims to produce ciphers that can be efficiently implemented still on classical computers, rather than on quantum ones: this last is a task Quantum Cryptography takes care of, starting right from the dynamics of the quantum framework.

This section is intended to be a general overview of the different typologies of post-quantum approaches. Indeed, depending on the problem around which their security revolves, post-quantum ciphers can be categorized into several families:

◇ **Code-Based Cryptography:** it is perhaps the oldest and most understood type of post-quantum public-key cryptography. Its security mainly relies on the overall difficulty of decoding. In particular, given an error-correcting code $C$ and a message (row-vector) $m$ of proper length, it is difficult to recover the error pattern (vector) $e$ from

$$y = mG + e$$

looking only at $y$, where $G$ is a generator matrix of the code $C$. In other words, it is hard to find the closest codeword to a generic altered vector according to a fixed metric. This is known as the *General Decoding Problem* (GDP) but there are many other relevant instances of problems like this, e.g. the *Syndrome Decoding Problem* (SDP). The main idea is to construct a trapdoor based on the fact that only the party who generated the encrypting error-correcting code knows its secret structure and, consequently, also an efficient way to decode. Although this is not always true for all codes, there are classes of codes that inherently own a fast nontrivial decoding procedure. Binary Goppa codes are one of the best choices in this sense. The scheme requires the receiver to generate a private code representing it by a generator matrix $G$ and to publish a shuffled version of it (usually $G_{pub} = SGP$ for some invertible secret matrix $S$ and permutation matrix $P$). The sender transforms their message $m$ into a valid codeword $c$ according to the confused code $G_{pub}$, computing $c = mG_{pub}$. They then add a secret error vector $e$ to $c$ and finally sends $y = c + e$. Since the receiver is the only one who knows $G$ (and also $S$ and $P$), they are the only one capable to decode $y$, find the error $e$ and remove it from $y$ to recover $c$. The fact that the public code is indistinguishable from a random code is the key point: the moment an attacker finds out information about the structure of the secret code $G$, the security of the scheme begins to waver, as the hidden skeleton of the code is strictly related to the efficient decoding algorithm. What we have just described is a summarized version of the McEliece cryptosystem, presented in 1978. Since this is exactly the family of ciphers we are going to analyze, all these concepts will be discussed more in detail in the next pages.

◇ **Lattice-Based Cryptography:** it is an approach that exploits the difficulty of solving the so-called *Shortest Vector Problem* (SVP), which can be enunciated as follows. Given a lattice $\Lambda \subseteq \mathbb{Z}^n$, the issue is to find the shortest non-trivial vector $v$, i.e.

$$v \in \Lambda : \|v\| = \min_{w \in \Lambda^*} \|w\|$$

where $\Lambda^* := \Lambda \setminus \{\mathbf{0}\}$. The best-known lattice-based encryption algorithm is NTRU. It applies the logic just described to the vector space $\mathbb{Z}[x]/(x^N - 1)$ which is isomorphic to $\mathbb{Z}^N$, where $N$ is a parameter to be chosen. Ultimately, everything revolves around the construction of a specific lattice starting from some polynomials, themselves depending on some starting parameters. For a complete overview, see [39]. It is worth mentioning that three of the four NIST PQC winners fall into this category: CRYSTALS-Kyber [17], which is classified as a key-establishment mechanism, and the two digital signature algorithms CRYSTALS-Dilithium [29] and Falcon [33].

◇ **Hash-Based Cryptography:** is the generic term used to refer to the practice of constructing cryptographic primitives starting from secure hash functions. Up to now, hash-based cryptography exclusively applies to the domain of digital signatures, mainly thanks to the Merkle signature scheme (see [12] for further details). A good property of this category of schemes is that the security of the encrypting algorithm is no less than the security of the used hash function: this means that the employment of a hash function which is first-preimage-resistant and collision-resistant (hence also second-preimage-resistant) can ensure the overall security of the scheme. Yet again, it is meaningful to point out that the fourth winner of the NIST PQC, whose name is SPHINCS$^+$ [16], belongs to this class of cryptosystems.

◇ **Multivariate Cryptography:** schemes belonging to this category base their security on the difficulty of solving systems of sparse multivariate polynomial equations defined over a finite field $\mathbb{F}_q$ and with degree at least 2. Solving systems like these is proven to be NP-hard. Many cryptosystems of this type have been designed over the past years and many of these have been cryptanalyzed. One of the most interesting features of this kind of approach is that multivariate schemes provide the shortest signatures among all post-quantum algorithms. This could be one of the reasons for preferring multivariate cryptography for signing purposes rather than for encrypting ones.

◇ **Isogeny-Based Cryptography:** it is strictly related to the advanced theory of elliptic curves. It is a direct evolution of the conventional Elliptic-Curve Cryptography, in the sense that the role of points is played by curves and the relations among points are substituted by the relations among curves (in concrete terms, by special functions mapping one elliptic curve into another, known as *isogenies*). The so-called Isogeny Path Problem is the cornerstone of this setup. Given two elliptic curves $E$ and $E'$ defined over a finite field $\mathbb{F}_q$ and such that $|E| = |E'|$, it consists in finding an isogeny

$$\varphi : E(\mathbb{F}_q) \longrightarrow E'(\mathbb{F}_q)$$

such that its degree is $B$-smooth, for a certain $B \in \mathbb{N}$. This stream of thought can be used to create the legitimate post-quantum heir of the classical Diffie-Helman key exchange (DH and ECDH): the three main isogeny-based algorithms are: SIDH (Supersingular Isogeny Key Exchange), CRS (Couveignes, Rostovtsev, Stolbunov) and CSIDH (Commutative Supersingular Isogeny Key Exchange). For additional details about either the underlying isogeny theory or the ciphers, see [27].

### 1.1.3 NIST Standardization Process

In December 2016 NIST launched a competition aimed at standardizing new encryption and digital signature algorithms resistant against the upcoming threat. According to a direct quotation from NIST's official website[2], the contest should

> *"*[...] *solicit, evaluate, and standardize one or more quantum-resistant public-key cryptographic algorithms* [...] *the new public-key cryptography standards will specify one or more additional unclassified, publicly disclosed digital signature, public-key encryption, and key-establishment algorithms that are available worldwide, and are capable of protecting sensitive government information well into the foreseeable future, including after the advent of quantum computers."*

After careful consideration by NIST, in July 2022 the first group of winners was announced, resulting in the four first standardised post-quantum ciphers. Only one of these is a public-key encryption algorithm and the other three are digital signature algorithms. The following table, where the names of the winners are listed, provides a recap and also shows the classification of the ciphers:

| CATEGORY | PKE/KEM | DS |
|---|---|---|
| *lattice* | CRYSTALS-Kyber | CRYSTALS-Dilithium<br>Falcon |
| *hash* | | SPHINCS$^+$ |

[2]https://csrc.nist.gov/Projects/post-quantum-cryptography/Post-Quantum-Cryptography-Standardization

It is recalled that PKE stands for Public Key Encryption, KEM for Key Encapsulation Mechanism and DS for Digital Signature. Quoting the official NIST website again, it is stated that

> *"* [...] NIST *has identified four candidate algorithms for standardization.* NIST *will recommend two primary algorithms to be implemented for most use cases:* CRYSTALS-Kyber *(key-establishment) and* CRYSTALS-Dilithium *(digital signatures). In addition, the signature schemes* FALCON *and* SPHINCS$^+$ *will also be standardized."*

This suggests that the staple ciphers are the first two, and that the second two are to be intended as substitutes in certain circumstances (i.e. when the signatures provided by CRYSTALS-Dilithium are too long and, in the case of SPHINCS$^+$, to avoid relying solely and exclusively on lattice-based cryptography).

Although the competition is nearing an end, it has not officially ended yet and NIST expects some of the alternate candidates to be considered in a fourth round. NIST also hints it may re-open the signature category for new DS proposals towards the end of summer 2022, in an effort to deviate towards non-lattice-based alternatives as opposed to the just standardised digital signatures. For this reason, signature schemes that are not based on structured lattices are of greatest interest and NIST suggests, directly on its website, (trivial) guidelines for the future submissions: short signatures and fast verification processes.

The contest was designed to be divided into more turns: at each round, all submitted proposals were analyzed and either discarded or admitted to the next round or even temporarily rejected with the possibility by the designers to correct and refine their project. The selection process started with a total of 72 candidates, but only 69 of them were deemed complete and proper to be admitted to the first round. The following table provides a general round-up, showing all the 69 candidates divided by category:

| CATEGORY | PKE/KEM | DS |
|---|---|---|
| *code* | 23 | 5 |
| *lattice* | 17 | 3 |
| *hash* | 3 | 7 |
| *multivariate* | 0 | 2 |
| *isogeny* | 1 | 0 |
| *other* | 5 | 3 |

In the third of the planned four rounds of the competition, the algorithms which made it that far (and which could be considered the finalists of the competition) were only 7, with other 8 algorithms as alternate substitutes. It is worth emphasizing that code-based signatures have not appeared in that list. On the other hand, the Classic McEliece (which we are going to describe in the following pages) has withstood up to that point of the contest despite being born more than forty years ago. Even though it was not ultimately standardised, it will be re-analysed in the fourth round of the competition. This suggests that McEliece's idea is intrinsically sturdy but tainted by some practical inconveniences and hard to turn into a signature scheme without loss of security. Indeed, all attempts to build a digital signature starting from the coding theory context have been in vain, for either security or efficiency troubles. Our work is oriented in this precise direction: we are going to go deeply investigate the whys and wherefores of this issue.

## 1.2 Digital Signatures

Digital Signatures (DS) are a very large and important branch of cryptography. Informally, digital signatures are mechanisms that bind a person/entity to a certain piece of data. In other terms, they provide *authenticity*, *integrity*, and *non-repudiation* of data, and are hence widely used in identification and authentication protocols, and therefore in the management of the whole Internet.

**Remark 1.1.** To be more precise, let us explain in a more technical way these three just mentioned concepts:

◇ **Authenticity:** is the property of a certain amount of data of being genuine and being able to be verified and trusted. In other words, it is an assessment of confidence in the validity of a message and its purported source of origin.

◇ **Integrity:** is the assurance that data is protected from unauthorized changes and guarantees that it is reliable and correct and that has not been corrupted or tampered with respect to its original form.

◇ **Non-repudiation:** is the capability to ensure that someone cannot deny or contest the validity of something and can be hence defined as the protection against an individual falsely denying having performed a particular action (e.g. creating information, sending a message, approving a decision).

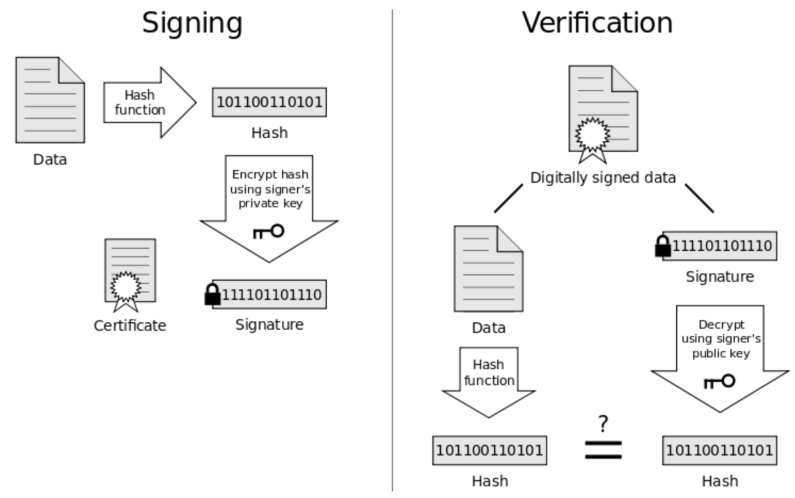
Concretely, a digital signature can be thought of as a string of bits associated to a message forwarded by a sender, in the same way they would append her signature at the bottom of the sheet of paper with a pen. A formal definition is instead the following one:

**Definition 1.1.** A digital signature scheme is determined by a triple of algorithms $(\mathsf{KGen}, \mathsf{Sign}, \mathsf{Verify})$, where:

1. $\mathsf{KGen}(1^\lambda)$ is the *key generation algorithm*: it selects a private (secret) key $\mathsf{sk}$ and the corresponding public key $\mathsf{pk}$ to be paired with $\mathsf{sk}$ starting from a security parameter $\lambda$.

2. $\mathsf{Sign}(m, \mathsf{sk})$ is the *signing algorithm*: given a message $m$ and a private key $\mathsf{sk}$, this algorithm allows the sender to produce a valid signature to attach to her message. First of all, $m$ is hashed into the digest $z := h(m)$, where $h$ is a suitable hash function. Starting from $z$, the signature $\sigma$ is finally produced using the private key $\mathsf{sk}$. Usually, a random quantity $r$ (known as *salt*) is integrated in this design and the string $z := h(h(m) \| r)$ is computed instead of simply $h(m)$. This practice is used to avoid known-plaintext attacks.

3. $\mathsf{Verify}(\sigma, m, \mathsf{pk})$ is the *verification algorithm*: given the message $m$, the public key $\mathsf{pk}$ and the signature $\sigma$ (and eventually the random seed $r$), this algorithm outputs $\top$ or $\bot$ depending on whether the signature is valid or not. Usually, the verification phase is started by triggering the backward process on input $\sigma$ and public key $\mathsf{pk}$ to obtain $z$. The random part is then removed from $z$ and the remaining part is finally compared to $h(m)$: if the two sides are equal, then the signature is valid, otherwise some tampering must have occurred.



This kind of approach is also known as *hash-and-sign* technique. There are some examples of algorithms achieving this kind of signature protocol, like DSA and ECDSA, both relying on DLP (the former on the group $\mathbb{Z}_p$ and the latter on the group associated to an elliptic curve).

**Remark 1.2.** A further property, inherent in the DS sphere and which would be desirable to ensure a "good" signature, is *unforgeability*. Roughly speaking, unforgeability alone generally means that adversaries cannot create valid signatures for messages on their own. More specifically, there exist two types of unforgeability: *existential* and *strong*. The former, which is often followed by the locution "under chosen message attack"and hence referred to by the acronym EUF-CMA, is the property whereby no adversary, after picking some arbitrary message/signature valid pairs of their choice

$$(m_1, \sigma_1), (m_2, \sigma_2), \dots, (m_k, \sigma_k)$$

can end up with a valid signature for whatever message $m \notin \{m_1, \dots, m_k\}$. The latter, which is usually referred to as SUF-CMA, ensures not only that the adversary cannot output a signature for a new message, but even that they cannot produce a new signature for a previously signed message. In other words, suppose an attacker is given a random valid message/signature pair $(m, \sigma)$ along with other valid message/signature pairs of their choice, like in the previous case. Then the signature algorithm is strongly unforgeable if the attacker cannot generate a new signature $\sigma' \neq \sigma$ for $m$. Obviously, the tacit assumption that underlies this set-up is that the malicious entity, in their attempt to overcome this kind of challenges, is not in possession of the private key.

Among the previously mentioned examples, there is an algorithm which, to some extent, can essentially be regarded as the inverse of RSA and which is used for signing instead of for encrypting. For instructive purposes only, it is worth explaining its mechanics, so let us describe how it works. This will help the comprehension of code-based digital signatures and the main trouble linked to them.

**Example 1.1** (RSA)**.** In the case of RSA-based digital signatures, the key generation algorithm is the same as in the case of RSA encryption, and consists in choosing $N = p \cdot q$ as an RSA modulus, with $p$ and $q$ primes (having certain "good" properties), $e \in \mathbb{Z}_N$ such that it is invertible in $\mathbb{Z}_{\varphi(N)}$ and $f$ its inverse, i.e. such that $ef \equiv_{\varphi(N)} 1$. The public key is specified by the couple $(N, e)$ along with a chosen hash function $h$. The triple $(p, q, f)$ represents instead the private key. The signing phase consists in taking the document $d$, hashing it with $h$ to obtain the digest $x = h(d)$, interpreting $x$ as an element of $\mathbb{Z}_N$ and computing the signature $s = x^f \mod N$. Someone who wants to verify the signature has to carry out the verification process by taking the signature $s$ and computing $y = s^e \mod N$ and finally comparing $y$ with the hash of the document $x = h(d)$. In the end, the signature is valid if and only if $x = y$ and the reason why is explained in the following lines:

$$\begin{aligned} y &= s^e \bmod N \\ &= (x^f)^e \bmod N \\ &= x^{fe} \bmod N \\ &= x^1 \bmod N \\ &= x. \end{aligned}$$

Notice that an eventual verifier is expected to perform only operations which they can actually accomplish, i.e. raise $s$ to the public exponent $e$, reduce it modulo the public modulo $N$, compute the hash of the public document $d$ using the public hash function $h$. Taking a close look at the situation, one notices that we are using exactly the reverse of the encryption function of RSA as signing function. In formal terms, the encrypting function is the automorphism of $\mathbb{Z}_N$ given by

$$\begin{aligned} \mathscr{E} : \mathbb{Z}_N &\longrightarrow \mathbb{Z}_N \\ a &\longmapsto a^e \end{aligned}$$

where $e$ is the public exponent. Its inverse is the decryption function given by the automorphism

$$\begin{aligned} \mathscr{D} : \mathbb{Z}_N &\longrightarrow \mathbb{Z}_N \\ a &\longmapsto a^f \end{aligned}$$

where $f$ is now the secret exponent. In this context, we are ultimately saying that we are using $\mathscr{D}$ to sign and $\mathscr{E}$ to verify. At a more abstract level, what we have just outlined is the conversion of a public key encryption algorithm into a digital signature one. Indeed, instead of waiting for a ciphertext which was encrypted using the public algorithm, the owner of the private key hashes the document to be signed and applies the decrypting algorithm on the digest. This action by the signatory can be thought of as they pretending to have received the cipher $h(d)$ and wanting to decrypt it. This is tantamount to saying that they have to find a certain plaintext that, if encrypted with the public key, gives precisely $h(d)$ as a result. This way, since in an asymmetric context the computationally complex part is the decryption one (in the sense that only the receiver is able to perform it), the capability of the signer to get the cleartext whose image under the encrypting function is $h(d)$ serves as proof of ownership of the private key. In fact, on the other side, once the signature $s$ is disclosed to the public, everyone who wants to check its validity only needs to encrypt it (and this is the accessible-to-all and easy operation) and control if $s$ is effectively mapped to $h(d)$ by the encrypting function. Again, observe that the owner of the private key associated to a given public key is the only one who can make this check test available to all. In conclusion, the concept to pinpoint is the following one: given an asymmetric cryptography framework and denoting with $\mathscr{E}$ and $\mathscr{D}$ the generic encrypting and decrypting functions respectively, what makes this paradigm work is the fact that the digest $h(d)$ can be regarded as an image of some plaintext under $\mathscr{E}$, or equivalently, it admits a preimage under $\mathscr{E}$. We can state this condition using the set notation of preimage as

$$\mathscr{E}^{-1}(h(d)) \neq \emptyset$$

or again, equivalently, by saying that the computation of $\mathscr{D}(h(d))$ yields a successful result. This only occurs when $h(d)$ happens to be in the set of decryptable ciphers. In the just analyzed example of the RSA-based digital signature, this requirement is fulfilled since $\mathscr{E}$ and $\mathscr{D}$ are one the inverse function of the other. However, notice that this is not always obvious, in the sense that the encrypting function is not always a bijection (unlike the example above, where $\mathscr{E}$ was even an isomorphism). Indeed, there might be cases where it is not surjective and hence this model cannot be replicated, as there exist elements which cannot be reached via the encryption function. The McEliece encryption, which we are going to present a little further down in this chapter, is precisely affected by this issue.

## 1.3 Error-Correcting Codes

Suppose you want to send a message to someone through a noisy channel. Then your message is likely to be altered along the way and the receiver might not be able to understand the received message. Error-Correcting Codes, usually abbreviated to ECC, are powerful mathematical tools used to protect communications from error occurrence; a code acts by adding extra bits of information to your message so that, even if some errors occur during the transmission, the receiver can discern whether something went wrong and, in the best case, even reconstruct the original message. Among all types of codes, we are interested in *linear* ones. In practice, a linear code is a particular category of codes obtained by means of linear algebra, as we are going to see. After giving a general overview of ECC, Binary Goppa Codes are analyzed. This class of codes is the best-suited one to be used as the core of the encrypting function in the McEliece scheme. For any further insights on ECC the reference textbook is the cornerstone piece [48].

### 1.3.1 Generalities

When it comes to ECC, the first thing to agree on is the set of symbols that can be sent through the channel, which is referred to as the *alphabet*. Since we are only interested in *binary* codes, our reference alphabet is the set containing only the two bits 0 and 1, i.e. the finite field $\mathbb{F}_2$, written in algebraic terms. In the following, unless otherwise specified, the adjective "linear" will be implied when referring to a code, in order to avoid repetitions since we will basically only deal with this kind of codes.

**Definition 1.2.** An $[n, k]$ *binary linear code* $C$ is a vectorial subspace of $\mathbb{F}_2^n$ of dimension $k$.

The elements of $C$ are called *codewords*. An $[n, k]$ linear code can be represented as a matrix whose rows form a basis for that linear subspace. Such a matrix is usually denoted with $G$ and is called *generator matrix* of the code. Since there exist several bases for the same subspace, this matrix is not unique, reason why the definite article has not been used. The codewords belonging to the code are all and only the linear combinations of the rows of this matrix. Alternatively, a code can be seen as the kernel of an $(n-k) \times n$ matrix, which is usually denoted with $H$ and is called *parity-check matrix* of the code. Throughout the whole thesis, all the vectors will be interpreted as column vectors.

**Definition 1.3.** Given an $[n, k]$ linear code and a vector $x \in \mathbb{F}_2^n$, the *syndrome* of $x$ is the vector $s = Hx \in \mathbb{F}_2^{n-k}$, where $H$ is the parity-check matrix of the code.

For every vector $x \in \mathbb{F}_2^n$, we can define its *Hamming weight*, that is the number of its non-zero entries:

$$w_H(x) := |\{i : x_i \neq 0 \wedge 1 \leq i \leq n\}|.$$

Through the definition of Hamming weight, the distance between two vectors can be defined. Given $x, y \in \mathbb{F}_2^n$, the *Hamming distance* between $x$ and $y$ is defined as the number of positions on which the two vectors differ; more formally, it is given by

$$d_H(x, y) := w_H(x - y).$$

Notice that this is not the only way to define a metric on $\mathbb{F}_2^n$. As we are going to see in the next chapter, there are a few other options to quantify the distance between vectors, among which the most widely used alternative named *rank metric*. Nevertheless, being more widespread, we will use the notation $d(*,*)$ instead of $d_H(*,*)$ to indicate the Hamming distance. In case of ambiguity, the notation will be specified on each occasion.

The principle of coding is the following: once a code $C$ has been chosen, the sender sends only codewords $c \in C$ to communicate through the channel. If the receiver gets a word $x \in \mathbb{F}_2^n$, they can then check whether $x$ belongs to the code $C$ or not. If $x$ is not in $C$, since the agreement was to send only codewords in the code, then they will know that something went wrong during the transmission. At this point, they can try to *decode*, that is to attempt to recover the original message from the received one. There are reasonable grounds (see [45] for a complete insight) to think that the sent codeword coincides with the *closest* (according to the adopted metric) codeword to the received vector (which, beware, may or may not be a codeword of $C$, and indeed it is nearly always a corrupted vector). In this context, the following definition is crucial.

**Definition 1.4.** The *distance* of a linear code is the minimum weight among its nonzero codewords, or equivalently, the minimum distance between distinct codewords:

$$d := \min_{\substack{c,c' \in C \\ c \neq c'}} d(c, c') = \min_{\substack{c,c' \in C \\ c \neq c'}} d(c - c', \mathbf{0}) = \min_{\substack{c \in C \\ c \neq \mathbf{0}}} d(c, \mathbf{0}).$$

It is classically denoted with $d$ and a linear code of length $n$, dimension $k$ and distance $d$ is commonly regarded as an $[n, k, d]$ code.

Still in this context, there are two other fundamental notions to bear in mind: the *detection capability* and the *correction capability* of a code. The first one is a natural number which represents the maximum number of transmission errors that the structure of the code allows one to recognize. The second one is, on the other hand, a natural number which specifies the maximum number of transmission errors that the structure of the code allows one (to recognize but additionally) to correct. Taking into account the geometrical interpretation of this setup, i.e. analyzing the behaviour of the correction spheres centered in the codewords as the radius varies (or, in a more formal manner, exploiting the triangular inequality), the following two theorems can be easily proved.

**Theorem 1.3.1.** *An $[n, k, d]$ code $C$ can detect any pattern of at most $d-1$ errors.*

**Theorem 1.3.2.** *An $[n, k, d]$ code $C$ can correct any pattern of at most $\lfloor \frac{d-1}{2} \rfloor$ errors.*

In general, computing the distance of a linear code is not an easy task since, intuitively, one would have to compute the distance between all possible (unordered) pairs of codewords and then take the minimum among those values. As can be guessed, this applies not only to the calculation of the distance of a code but also to the decoding of a message. Notice that these two tasks are strictly interlinked indeed.

### 1.3.2 Decoding

As already pointed out, if we consider a generic code $C$, the most natural way to decode a word $x$ received after a transmission (i.e. to retrieve the codeword $c$ that was sent) is that of taking the closest codeword to $x$. This principle is known as *maximum likelihood decoding.* One can either compare the received vector with all the codewords in $C$ and then take the closest one (if unique) or carry out a more clever (albeit equivalent in terms of computational cost) procedure. Starting from $x$, one constructs the spheres centered in $x$ and of increasing radius until a sphere that contains at least one codeword of $C$ is obtained. At this point, if there is only one codeword in the sphere, it is chosen as the (presumed) sent codeword $c$. On the other hand, if in the first case where the sphere and $C$ have a non-null intersection happens that unfortunately there are two (or more) codewords in the sphere, then the decoding is ambiguous and cannot be successfully completed. This algorithm is, of course, not very practical but one cannot expect much better if no information about the code and the alphabet is given.

In the case of linear codes, there is though another natural way to perform the decoding operation and it exploits the mathematical subgroup structure of the code $C$ with respect to the total space $\mathbb{F}_2^n$, which induces a canonical partition on it. This partition is obtained via the cosets of $C$. The idea is to choose a privileged element in each coset of $C$ that will be assumed to be the error occurred during transmission. In applying this method, the criterion used for the choice of this element is therefore of fundamental importance: in according to the principle of maximum likelihood decoding, the designated element is defined as the word with minimum weight within the coset.

**Definition 1.5.** A *coset leader* of a given coset of $C$ is an element belonging to the coset with the lowest Hamming weight within the coset.

Notice the use of the indefinite article in the definition, meaning that the coset leader is not necessarily uniquely identified. Anyway, in the case of multiple coset leaders, a choice can be made by selecting a specific one of them.

The method involving coset leaders, known as *syndrome decoding*, works as follows. Let $H$ be a parity-check matrix for the $[n, k]$ linear code $C$. The vector $x$ is in $C$ if and only if $Hx = \mathbf{0}$. For any received vector $x$, the syndrome $s = Hx$ of length $n - k$ can be read as a measure of whether the $n$-tuple $x$ belongs to the code or not. More than just that, the syndrome voices information about the error vector. Instead of directly retrieving the message, this method is geared towards calculating the error (which is essentially the same thing). Since the syndrome of a codeword is $\mathbf{0}$, two vectors $x$ and $e$ that differ by a codeword $c$ will have the same syndrome:

$$Hx = H(c + e) = Hc + He = \mathbf{0} + He = He.$$

That is, syndromes are constant on the cosets of $C$ in $\mathbb{F}_2^n$. Similarly, distinct cosets have different syndromes, since the difference of vectors from distinct cosets cannot be a codeword and thus has a nonzero syndrome. We interpret the above equation by saying that the received vector $x$ and the corresponding error vector $e$ introduced by the channel will have the same syndrome, namely that of the coset to which they both belong. It is now clear that, given a coset and a word $x$ in it, the closest codeword to $x$ is given by the difference between $x$ and the coset leader, thanks to how the latter has been defined. At this point, we only need to store a so called *syndrome dictionary* containing all possible syndromes $\{s_1, \dots, s_{2^{n-k}}\}$ together with the corresponding coset leaders $\{e_1, \dots, e_{2^{n-k}}\}$ such that $He_i = s_i$. While decoding, when $x$ is received, first calculate the syndrome $s = Hx$. Next, look up $s$ in the syndrome dictionary as $s = s_i$. Finally, decode $x$ to $c = x - e_i$.

**Remark 1.3.** Like in the case of brute force decoding, also in this case the decoding procedure may fail, depending on the uniqueness of the coset leader. As previously mentioned, it all has to do with the geometry of the code and its distance. Notably, the issue of determining the distance of a code is no trivial matter: as shown in [60], it is a NP-hard problem.

Actually, the decision problem associated to syndrome decoding is also NP-complete, as can be read in [14]. In particular, the formal definition of this major problem, which represents the building block of the upcoming discussion on the security of certain ciphers and consequently of the whole thesis, follows. It is known as *Syndrome Decoding Problem* and is commonly abbreviated as SDP.

**Problem 2** (SDP)**.** Let $C$ be a binary linear code.
GIVEN: an $r \times n$ parity-check matrix $H$ for $C$, a syndrome $s \in \mathbb{F}_2^r$ and an integer $w > 0$.
FIND: a word $e \in \mathbb{F}_2^n$ such that $w_H(e) \leq w$ and $He = s$.

If one can solve this problem, one can essentially decode any kind of linear code. Although the difficulty of SDP may seem a drawback (and indeed it is) because it places limits on our ability to decode, it is also a benefit because it provides us with a brand-new trapdoor for constructing ciphers based on a problem other than DLP or factorization.

In fact, if it were possible to figure out one or more classes of codes that inherently possess an efficient decoding algorithm and if it were possible to set up a scheme in which the one who knows the decoding algorithm keeps it hidden as a secret key so that they are the only one knowing how to quickly decrypt, then we would have achieved a post-quantum asymmetric cipher. The public information, derived by confusion and obfuscation from the private one, would in fact be such that the decryption algorithm cannot be traced, in the same way that, in the case of RSA, the two primes cannot be traced from the public modulus. The good news is that the class of codes we were talking about actually exists and it is precisely with this in mind that we are now going to introduce Binary Goppa Codes.

### 1.3.3 Binary Goppa Codes

In the context of algebraic coding theory, Binary Goppa codes are one of the best answers to the issue of efficient decoding since, as we shall see, a very fast decoding algorithm naturally follows from their definition.

**Definition 1.6** (Binary Goppa Code)**.** Let $\mathbb{F}_{2^m}$ be a finite field of characteristic 2 and let $g(x) \in \mathbb{F}_{2^m}[x]$. Let $L = \{L_1, \ldots, L_n\} \subseteq \mathbb{F}_{2^m}$ such that $g(L_i) \neq 0 \ \ \forall i$. Then, a *binary Goppa code* $\Gamma(L, g)$ is defined as follows:

$$\Gamma(L,g) := \left\{ c \in \mathbb{F}_2^n \ : \ \sum_{i=1}^{n} \frac{c_i}{x - L_i} \equiv 0 \mod g(x) \right\}.$$

**Remark 1.4.** The condition requiring that no root of $g$ appears among the elements of $L$ is necessary to ensure that the polynomial $x - L_i$ is invertible in $\mathbb{F}_{2^m}[x]/(g)$.

Recalling that Goppa codes are linear codes (easy to prove), the first question that arises is about the $[n, k, d]$ parameters. Trivially, $n = |L|$. For what concerns the dimension $k$, a theorem guarantees that $k \geq n - mt$, where $t := \deg(g)$. Again a theorem ensures that the distance $d \geq t + 1$ (look at [40] for both the arguments). A particular class of binary Goppa codes is obtained by choosing the polynomial $g$ so that it is irreducible. In this case we speak of an *irreducible binary Goppa code.* A peculiarity of this type of codes is that we have a stronger condition on the lower bound for the distance, which reads that $d \geq 2t + 1$. Therefore, the following proposition holds.

**Proposition 1.3.3.** *The parameters for an irreducible binary Goppa code are* $[|L|, \geq |L| - mt, \geq 2t + 1]$.

For the proofs of this theorem, consult again [40].

Since the main goal of this brief subsection is to show that binary Goppa codes inherently feature a fast decoding algorithm, we now have to deal with their parity-check matrix, being closely related to the decoding procedure.

**Proposition 1.3.4.** *The matrix $H$ obtained as*

$$H := \begin{bmatrix} g_t & & & & \\ g_{t-1} & g_t & & & \\ g_{t-2} & g_{t-1} & \ddots & & \\ \vdots & \vdots & \ddots & \ddots & \\ g_1 & g_2 & \cdots & g_{t-1} & g_t \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ L_1^1 & L_2^1 & L_3^1 & \cdots & L_n^1 \\ L_1^2 & L_2^2 & L_3^2 & \cdots & L_n^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ L_1^{t-1} & L_2^{t-1} & L_3^{t-1} & \cdots & L_n^{t-1} \end{bmatrix} \begin{bmatrix} \frac{1}{g(L_1)} & & & & \\ & \frac{1}{g(L_2)} & & & \\ & & \frac{1}{g(L_3)} & & \\ & & & \ddots & \\ & & & & \frac{1}{g(L_n)} \end{bmatrix}$$

*is a parity check matrix for the binary Goppa $\Gamma(L, g)$, where $g_i$ is the coefficient of $x^i$ in $g(x)$.*

The proof is quite simple and is based on the following two easily provable claims:

$$\frac{1}{x - L_i} = \frac{g(L_i) - g(x)}{g(L_i)(x - L_i)}$$

$$c \in \Gamma(L, g) \iff \sum_i c_i \frac{g(L_i) - g(x)}{g(L_i)(x - L_i)} \equiv 0 \mod g(x).$$

To secondly construct the generator matrix $G$, just remember that $GH^\top = \mathbf{0}$.

For what concerns encoding, the first thing to do is to find the generator matrix $G$ for the chosen $\Gamma(L, g)$ Goppa code. Then, as always, the encoding of a (row-vector) message $m$ is simply $G^\top m$, taking care to choose $m$ of length $k$, where $k$ is the number of (linearly independent) rows of $G$, i.e. the dimension of the code. Things get more interesting when it comes to decoding. In this respect, the following definition is fundamental.

**Definition 1.7** (Error Locator Polynomial)**.** Let $y = (y_1, y_2, \ldots, y_n)$ be the received vector and assume that at most $t$ errors occurred, where $2t+1 \leq d$, so that $t$ corresponds to the maximum number of errors that can always be corrected. Hence, we have that

$$y = (y_1, y_2, \ldots, y_n) = \overbrace{(c_1, c_2, \ldots, c_n)}^{c} + \overbrace{(e_1, e_2, \ldots, e_n)}^{e}$$

where $e_i \neq 0$ at exactly $t$ positions and $c$ is a codeword. Let $E$ be the set of error positions, defined as $E := \{i : 1 \leq i \leq n \wedge e_i \neq 0\} =: \mathrm{supp}(e)$ and let $L = \{L_1, \ldots, L_n\}$. The polynomial $\sigma(x) \in \mathbb{F}_{2^m}[x]$ defined as

$$\sigma(x) := \prod_{i \in E}(x - L_i)$$

is called the *error locator polynomial* related to the binary Goppa code $\Gamma(L, g)$.

The goal of this polynomial is to locate the positions where the errors occurred and to correct them with the right symbol of the alphabet. Since we are only dealing with binary Goppa codes, it is sufficient to simply locate the errors, since there is only one other possible alternative. If we were to use generic Goppa codes, we would need to calculate a separate polynomial as well, known as *error evaluator polynomial*. Clearly, if one had the error locator polynomial at hand, one could quickly find all the error positions either by factoring $\sigma(x)$ or by evaluating it on each element of $L$ in order to discover its roots. So, the key point here is to find a feasible way to compute $\sigma(x)$ starting from the received vector $y$. The algorithm presented on the next page, known as *Patterson algorithm*, gathers all the operations to be done step by step in order to calculate $\sigma(x)$.

The theoretical arguments explaining the correct functioning of this algorithm can be found either in [51] (which is the original paper by Patterson) or in [49]. Basically, the claim heavily exploits the requirement that the code is binary and that $g$ is irreducible. In a nutshell, from the key equation

$$\sigma(x)s(x) \equiv \sigma'(x) \mod g(x)$$

which is only valid in the binary case (and which can be easily proved), one appeals to the fact that a generic polynomial in $\mathbb{F}_{2^m}[x]$ can always be written in terms of the squares of two others smaller polynomials. That is because, in a field of a characteristic 2, a polynomial containing only even degree terms can be seen as the square of the polynomial obtained by halving the exponent to each term of the starting one (e.g. $x^6 + x^2 + 1 = (x^3 + x + 1)^2$). Hence, the error locator polynomial can be thought of as $a(x)^2 + xb(x)^2$ and from this equation necessary conditions on $a(x)$ and $b(x)$ are derived.

**Algorithm 1** (Patterson)

```
Input: the received vector $y = (y_1, \dots, y_n)$
Output: the error locator polynomial $\sigma(x)$
1  compute the syndrome polynomial $s(x) := \sum_i \frac{y_i}{x-L_i} \mod g(x)$
2  if $s(x) = 0$ then
3      $y$ is a codeword
4      return
5  end
6  compute $h(x)$ such that $s(x)h(x) \equiv 1 \mod g(x)$
7  if $h(x) = x$ then
8      return $\sigma(x) = x$
9  end
10 compute $d(x)$ such that $d(x)^2 \equiv h(x) + x \mod g(x)$
11 compute $a(x)$ and $b(x)$ such that $a(x) \equiv h(x)b(x) \mod g(x)$, where $b(x)$ is
   the polynomial with least degree among the two
12 return $\sigma(x) = a(x)^2 + xb(x)^2$
```

Summing up, the moral of this short subsection on Goppa codes is that it is possible, when choosing a binary Goppa code with generator polynomial $g$ of degree $t = \deg(g)$, to efficiently correct up to $t$ errors by triggering the Patterson machinery, which is a polynomial-time algorithm with respect to the length of the received word. This opportunity, if made somehow exclusive to a single party, can be used to set up one of the most relevant and valuable ciphers ever invented: the McEliece cryptosystem.

## 1.4 McEliece's Idea

The first ever cryptosystem based on coding theory is a public key encryption scheme, presented in 1978 by Robert McEliece. This cryptosystem uses a linear error-correcting code in order to create a public and a private key. In the original paper, the proposal for the error-correcting code was a binary Goppa code. In 1986, Harald Niederreiter proposed a similar PKC based on error-correcting codes. This proposal was later shown to be security equivalent to McEliece's proposal. Among others, Niederreiter estimated Generalized Reed-Solomon codes as suitable codes for his cryptosystem, which were assumed to allow smaller key sizes than Goppa codes. Unfortunately, in 1992 it was shown that Niederreiter's proposal to use GRS codes was insecure. More in general, this repartee of proposal and objection was seen repeated in the following years, during which several attempts were made to modify McEliece's original scheme in order to reduce the public key size. However, most of them turned out to be insecure or inefficient compared to McEliece's original proposal. The variety of possible cryptographic applications provides sufficient motivation to have a closer look at these two crucial cryptosystems based on coding theory. The aim is to establish a serious alternative to the classic public-key cryptosystems like the ones based on number theory.

### 1.4.1 The McEliece Cryptosystem

Let us begin this subsection by immediately presenting the schematization of the key generation step and of the encryption and decryption steps of the McEliece cryptosystem.

**Algorithm 2** (McEliece PKC)

```
Parameters: $n, t \in \mathbb{N}$
1  Key Generation: from $n$ and $t$ generate the following matrices
2  $G \leftarrow$ a $k \times n$ generator matrix of a binary Goppa code $\Gamma(L, g)$ of dimension
   $k$ and minimum distance $d \geq 2t + 1$
3  $S \leftarrow$ a $k \times k$ random binary non-singular matrix
4  $P \leftarrow$ an $n \times n$ random permutation matrix
5  $G_{pub} = SGP$
6  Public Key: $(G_{pub}, t)$
7  Private Key: $(S, G, P, \delta_\Gamma)$ where $\delta_\Gamma$ is an efficient decoding algorithm for
   the chosen code $\Gamma$, i.e. the Patterson algorithm
8  Encryption: given a plaintext $m \in \mathbb{F}_2^k$
9  randomly choose a vector $z \in \mathbb{F}_2^n$ of weight $t$
10 compute the ciphertext as $c = G_{pub}^\top m + z$
11 Decryption: given a ciphertext $c \in \mathbb{F}_2^n$
12 let $x = Pc = P(G_{pub}^\top m + z) = P(P^\top G^\top S^\top)m + Pz = G^\top(S^\top m) + Pz$
13 run the decoding algorithm $\delta_\Gamma$ on $x$; since by its construction it has Hamming
   distance from $\Gamma$ equal to $t$, from $\delta_\Gamma(x)$ we get the codeword $G^\top(S^\top m)$
14 from $G^\top(S^\top m)$ easily retrieve the vector $S^\top m \in \mathbb{F}_2^k$ and finally multiply on
   the left by $(S^\top)^{-1}$ to obtain the original message $m$
```

The just presented algorithm is Robert McEliece's original version from 1978, which has remained unbroken to this day, even if more than 40 years have passed since its proposal. The trapdoor for this cryptosystem consists of the knowledge of an efficient error-correcting algorithm for the chosen class of codes (which is always available, as highlighted in the last subsection, if one chooses the family of Goppa codes) together with a permutation. Concretely, the private and the public key are two matrices: the public key is based on the private key but in such a way that makes it unfeasible to retrieve the private key.

The security of this cryptosystem relies on a decoding problem. An eventual attacker will have a hard time trying to separate $G_{pub}$ from $G$ because, in order to do so, she would need to know the inverse of the matrices $S$ and $P$, which are part of the private key and hence not published. Also, even if she knew $P$, she would still have to decode the code $G$ in order to retrieve $G^\top(S^\top m)$. As claimed in the literature, 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. Randomizing the order of the columns serves to make the public matrix look random and to ensure the difficulty of obtaining the decoding method from the encoding matrix. This way, the public key is indistinguishable from the private key. More formally, the problem whose complexity establishes the foundations for the security of the McEliece cryptosystem is slightly different from the SDP and can be sketched as follows:

**Problem 3** (McEliece Problem)**.** Let $\Gamma$ be a binary Goppa code.
GIVEN: a McEliece public key $(G_{pub}, t)$, where $G_{pub}$ is a $k \times n$ matrix, and a ciphertext $c \in \mathbb{F}_2^n$.
FIND: the (unique) message $m \in \mathbb{F}_2^k$ such that $w_H(G_{pub}^\top m - c) = t$.

It is simple to understand that someone who is able to solve SDP is also able to solve the McEliece Problem. The reverse is allegedly not true, as the code generated by $G_{pub}$ is not a random one but permutationally equivalent to a code of a known class (a Goppa code, in the definition). Therefore, we can not assume that the McEliece Problem is NP-hard. Solving the McEliece-Problem would only solve the General Decoding Problem for a certain class of codes and not for all codes.

On the other hand, what is really of interest is that no one is able to trace back to the generating matrix of the original code from the shuffled matrix. Here we are not referring to the fact of figuring out what the two matrices $P$ and $S$ are (since that is quite improbable) but rather to the fact that there might be attacks able to derive information on some property or on the structure itself of the code (e.g. the generating polynomial in the case of a Goppa code). That is because, unlike random codes, a code belonging to a certain family is a "structured" code and it always conceals within itself, albeit latently (e.g. even after a column swap), information about its algebraic definition.

**Remark 1.5.** This property has a name and is known as *indistinguishability*, which can be informally summarized as the computational infeasibility to tell apart a generator matrix of a random code from a generator matrix belonging to a particular family, like the binary Goppa codes. When the latter are involved, the challenge to draw such a distincion is known as the *Goppa Code Distinguisher Problem*, usually abbreviated to GD.

**Problem 4** (GD)**.** Let $k < n$ be two integers and let $\mathbb{F}_2^{k\times n}$ denote the set of all possible binary $k \times n$ matrices. Similarly, let $\Gamma^{k\times n}$ denote the set of all binary $k \times n$ generator matrices spanning a binary Goppa code. Clearly, $\Gamma^{k\times n} \subsetneq \mathbb{F}_2^{k\times n}$.
GIVEN: a matrix $G$, which either belongs to $\Gamma^{k\times n}$ or to $\mathbb{F}_2^{k\times n} \setminus \Gamma^{k\times n}$ depending on the answer of a random oracle.
FIND: the provenance of $G$, i.e. output $\top$ if $G \in \Gamma^{k\times n}$, otherwise output $\perp$ if $G \in \mathbb{F}_2^{k\times n} \setminus \Gamma^{k\times n}$.
An algorithm which can always win this game is called a *distinguisher*.

Up to now, no distinguisher attack has been found for a general Goppa code, nor even for the general case of a generic code, fact which widely suggests that the Goppa Code Distinguishing Problem is a hard decision problem. A distinguisher has been instead discovered for high-rate Goppa codes in [30]. Indistinguishability is the strongest security assumption related with structural attacks. When it holds, the security of the corresponding public-key system can be truly reduced to the hardness of decoding, for which very strong arguments exist.

## 1.4.2 The Niederreiter Variant

The Niederreiter cryptosystem was developed in 1986 by Harald Niederreiter. It is the dual variant of the McEliece cryptosystem in the sense that, instead of using the generator matrix to encrypt the message, the parity-check matrix is used. In this setting, the message is represented as a binary error vector of length $n$ and of Hamming weight (at most) $t$ and the ciphertext is computed as the syndrome associated to this error pattern. The receiver decrypts the ciphertext using a fast decoding algorithm and straight obtains the message; anyone else who wants to eavesdrop on the communication has to deal with SDP. The security of the Niederreiter PKC and the McEliece PKC are equivalent indeed, in the sense that an attacker who can break the first is able to break the second and vice versa. Clearly, the security always depends on the chosen family of codes and in fact Niederreiter's original proposal using Generalised Reed-Solomon codes (GRS) was broken by Sidelnikov and Shestakov in [57].

One small difference in comparison to the McEliece cryptosystem that can appear quite annoying is the need to convert the message in the form of a binary vector of weight $t$. Thankfully, there is a quite simple algorithm that can perform this conversion; moreover, since PKC are often used to communicate random data to be later used as key of a symmetric cipher, this mapping is really not even an issue. Below, a sketch summarising all the steps of the scheme is provided:

**Algorithm 3** (Niederreiter PKC)

```
   Parameters: $n, t \in \mathbb{N}$
1  Key Generation: from $n$ and $t$ generate the following matrices
2  $H \leftarrow$ an $(n-k) \times n$ parity-check matrix of a code $C$ which can correct up
   to $t$ errors
3  $P \leftarrow$ an $n \times n$ random permutation matrix
4  $H_{pub} = MHP$, whose columns span the column space of $HP$, that is, $M$ is
   the matrix that transforms $HP$ in systematic form, i.e. $H_{pub}^{\{1,\ldots,n-k\}} = I_{n-k}$
5  Public Key: $(H_{pub}, t)$
6  Private Key: $(M, H, P, \delta_C)$ where $\delta_C$ is an efficient decoding algorithm for
   the chosen code $C$
7  Encryption: given a plaintext $m \in \mathbb{F}_2^k$
8  represent it as an error vector $e \in \mathbb{F}_2^n$ with $w_H(e) = t$ using whichever
   reasonable algorithm of embedding
9  compute the ciphertext as the syndrome $s = H_{pub}e$
10 Decryption: given a ciphertext $c \in \mathbb{F}_2^n$
11 calculate $x = M^{-1}s = M^{-1}H_{pub}e = M^{-1}(MHP)e = HPe = H(Pe)$
12 apply the decoding algorithm $\delta_C$ on $x$, which can be seen as a syndrome with
   respect to the matrix $H$; observe that the coset leader corresponding to that
   syndrome will be precisely $Pe$ as it has Hamming weight $t$ by construction
13 from $Pe$ easily retrieve the error pattern as $e = P^{-1}(Pe)$
```

The advantages of this dual variant are essentially two. The first is a smaller ciphertext size: vectors of length $n-k$ versus vectors of length $n$.

The second is a smaller public key size, since it is sufficient to store and publish just the redundant part of the matrix $H_{pub}$ (as it has been reduced in systematic form). It is clear that this expedient cannot be used in the case of the classical version of the McEliece cryptosystem, as there would be a decrease in the semantic security. Indeed, remembering that the McEliece encryption occurs as $G_{pub}^{\top}m + z$, the assumption of a systematic $G_{pub}$ would result in the initial bits of the ciphertext being identical to the bits of $m$, unless situated in positions affected by the error $z$. Since $w_H(z)$ is negligible compared to the length of a codeword, a dramatically high percentage of bits of $m$ would be revealed.

**Remark 1.6.** It is worth emphasizing that, in the form in which we have presented them, neither McEliece nor Niederreiter cryptosystems are feasible for practical applications due to the huge size of their keys. One can see that, to achieve the same security level as the currently used version of RSA, the public key of the McEliece scheme needs to be more than 100 times greater than that of RSA. Therefore, reducing the key sizes of the two code-based cryptosystems introduced in this chapter is one of the starting points of great interest for a continuous research in this field.

# Chapter 2

# Code-Based Digital Signatures

Among all the cryptographic applications of coding theory, in this chapter we will deal precisely with digital signatures. McEliece's and Niederreiter's advancements have been moved in the right direction but they only provided paradigms for asymmetric ciphers. Nevertheless, as has always been the case in the cryptographic literature, these lasts are a very good starting point for the construction of signature schemes. Digital signatures need their own quantum-resistant replacement too and we will analyze all the most significant and prominent code-based proposals to date, including the most recent and present-day ones and unraveling merits and shortcomings of each of them.

## 2.1 Two "Historical" Proposals

The two schemes presented below are arguably the two most emblematic and exemplary ideas of code-based digital signatures, substantiating their potentialities but also their issues, and definitely among the first to have been proposed, being temporally placed around the turn of the millennium. CFS (2001) and KKS (1997) fall under what is called the *hash-and-sign* heuristic, which is a general approach for signing messages of arbitrary length by hashing them first and then signing the digest only. However, it will become clear how, in the case of the code-based realm, it is not always straightforward to apply this paradigm.

### 2.1.1 CFS

This signature scheme is named after its inventors: Courtois, Finiasz and Sendrier, who published their work [22] in 2001. Based on the Niederreiter cryptosystem (thus relying on the hardness of SDP and of GD), the CFS scheme has been conceived as an attempt to convert the latter PKC into a DS. The practicality of this signature scheme is questionable because of its large public key (which is the same problem as for the Niederreiter cipher) and a long signing time, as will soon become clear. Although the large key size might be a problem for some applications, a storage requirement of a few megabytes on the verifier's side is not always an issue, since it is usually the case that the receiver is the one holding the lion's share of the computational power. Unfortunately, the drawback concerning the long signing time is instead inescapable. In fact, as will be explained in detail later on, each signature requires a large number (several hundreds of thousands) of algebraic decoding attempts. To give a concrete idea of the duration, the first reported signing time in software was about one minute, in comparison to the sub-second computation of an ECDSA signature.

The logic behind CFS is very simple and is based on reversing the Niederreiter cryptosystem. Recall that the latter works by taking an error vector $e$ as a message and turning it into a ciphertext by computing the syndrome $s = H_{pub}e$. So far, one would be led to believe that CFS works by hashing the document, interpreting it as a syndrome and decoding it to obtain the signature as the minimum-weighted vector related to that syndrome. Now, herein lies the crux of the matter.

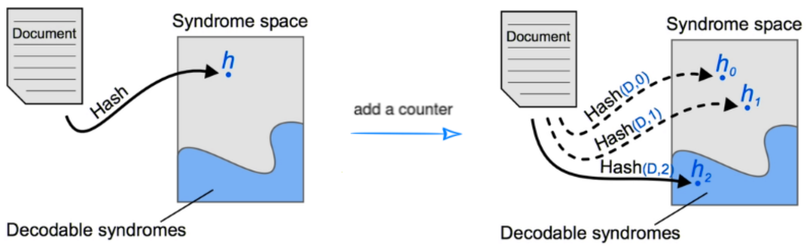
**Remark 2.1.** Not all syndromes are decodable.

It is therefore clear that, unlike in Example 1.1 (RSA) where the hash of the document (whatever it was) could always be interpreted as an element of $\mathbb{Z}_N$, in the case of code-based cryptography we cannot proceed in the same way, as $h(d)$ may not belong to the space of decodable syndromes. Recall that trapdoor functions allow digital signatures to take advantage of the unique capability of the public key owner to invert those functions and it is thus evident that only messages whose hashes fall within the ciphertext space can be signed in this way. It is also evident that, given the premises, it would be desirable for the ciphertext space to be as large as possible. In our framework, it coincides with the space of all decodable syndromes. In formal terms, this last wish translates into having a meaningful portion of vectors $s \in \mathbb{F}_2^{n-k}$ for which there exists a corresponding error pattern $e \in \mathbb{F}_2^n$ of Hamming weight less than or equal to the correcting capability $t$ of the code, such that $s$ is the syndrome of $e$.

**Remark 2.2.** There exist codes for which it is possible to cover the whole space $\mathbb{F}_2^n$ by taking the union of the balls centered on codewords and of

Hamming radius equal to the correction capability $t$. These codes are known as *perfect codes*. It is straightforward that the existence of such a sphere packing allows any received vector (as well as any syndrome) to be decoded, since any vector of $\mathbb{F}_2^n$ will always be enclosed within one and only one ball.

A naive cryptographer would therefore be tempted to use perfect codes in place of Goppa codes as keystones of the scheme, so as to obviate the problem of undecodable syndromes. This would of course be a serious mistake, due to the excessive leak of information they would disclose about their very structure. The option of using perfect codes is therefore to be ruled out for security reasons and the smartest play to make remains to stick to Goppa codes and repeatedly hash the message until a decodable syndrome is found.



This procedure is, in the end, nothing more than a *hash-and-sign* routine which inevitably requires several tries. Concretely, given a suitable hash function $h$, one produces a sequence $d_0, \ldots, d_{i^*}$ of elements in $\mathbb{F}_2^{n-k}$ such that

$$d_0 = h(m\|0),\ d_1 = h(m\|1),\ \ldots,\ d_i = h(m\|i),\ \ldots,\ d_{i^*} = h(m\|i^*)$$

where $i^*$ is the smallest integer such that $d_{i^*}$ is a decodable syndrome and the symbol $\|$ denotes the concatenation operator. The signature is then composed by the corresponding error pattern $e_{i^*}$ (which only the signer can compute) and by the counter $i^*$.

The first straightforward question that arises is about the number of attempts that are needed in order to obtain a useful syndrome. The answer can be easily provided by comparing the total number of syndromes against the number of (efficiently) correctable syndromes:

$$\frac{\#\ \text{DECODABLE SYNDROMES}}{\#\ \text{TOTAL SYNDROMES}} = \frac{\sum_{i=0}^{t}\binom{n}{i}}{2^{n-k}} \simeq \frac{\binom{n}{t}}{2^{n-k}} \simeq \frac{\frac{n^t}{t!}}{n^t} = \frac{1}{t!}$$

which represents the probability of a syndrome to be decodable. In the equation above, the relations among the parameters of a generic binary Goppa code have been used, i.e. $k = n - mt$ and $n = 2^m$.

**Remark 2.3.** In order to raise the probability of encountering a decodable syndrome and thus speed up the (successful) execution of the scheme, an apparently good guess would be to reduce $t$. Unfortunately, this ploy is unfeasible since a small $t$ would imply a code rate, defined as

$$r := \frac{k}{n} = \frac{n - mt}{n} = 1 - \frac{m}{2^m} t$$

tending towards 1 for the parameter values used in practice, i.e. for high $n$ (and thus also high $m$, being $n = 2^m$). As a result, a high value of $r$ jeopardizes the security of the whole scheme, as it exposes the code to a distinguisher attack, namely the one shown in [31]. As already claimed, this is an extremely specific, and for this very reason solvable, instance of GD.

**Algorithm 4** (CFS)

**Parameters:** $n, t \in \mathbb{N}$

```
1  $\boxed{\mathsf{KGen}_{\mathrm{CFS}}(1^\lambda)}$: select appropriate $n$ and $t$ consistent with the $\lambda$ parameter[3], then generate a Niederreiter PKC key pair with a code $C$ randomly drawn from the class of binary irreducible Goppa codes of length $n$ and correction capability $t$; recall that $H_{pub} = MHP$ where $M$ is an invertible matrix, $P$ is a permutation matrix and $H$ is a parity-check matrix of $C$
2  $\mathsf{pk} = (H_{pub}, t, h)$ where $h$ is an agreed suitable hash function
3  $\mathsf{sk} = (M, H, P, \delta_C)$ where $\delta_C$ is an efficient decoding algorithm for $C$
4  $\boxed{\mathsf{Sign}_{\mathrm{CFS}}(m, \mathsf{sk})}$: given the message $m$, starting from $i = 0$
5  $d_i = h(m \| i)$
6  while $d_i$ is not a decodable syndrome do
7      $i \leftarrow i + 1$
8      $d_i = h(m \| i)$
9  end
10 $e = \delta_C(M^{-1} d_i)$
11 $\sigma = (P^\top e, i)$
12 $\boxed{\mathsf{Verify}_{\mathrm{CFS}}(\sigma, m, \mathsf{pk})}$: given $\sigma = (z, i^*)$ verify that $w_H(z) \leq t$ then compute
13 $a = h(m \| i^*)$
14 $b = H_{pub} z$
15 $\sigma$ is valid $\iff a = b$
```

[3]the notation $1^\lambda$ indicates, by convention, the unary representation of $\lambda$, i.e a string $11\ldots1$ of lambda ones. This is a standard notation that comes from computer science and is used to emphasize that the complexity of functions is described as a function of

Above, the step-by-step instructions to set up the CFS scheme are shown. Notice that, as in the case of the Niederreiter cryptosystem, its security relies on the hardness of both SDP and GD. The proof of the correctness of the signature is evidenced by the following chain of equalities (remember that $P$ is a permutation matrix):

$$b = H_{pub}z = H_{pub}(P^\top e) = (MHP)(P^\top e) = MHe =$$
$$= M(M^{-1}d_{i^*}) = d_{i^*} = h(m \| i^*) = a.$$

In [25], Dallot proposes a modified version of the CFS signature, called mCFS. In this modified version, the author proposes to replace the counter $i$ with a random nonce uniformly distributed over $\{1, \ldots, 2^{n-k}\}$.

### 2.1.2 KKS

KKS is the acronym for the creators of this digital signature scheme: Kabatianskii, Krouk, and Smeets. The idea behind this proposal (dated 1997) is to get rid of the concept of fast decoding algorithm and to bypass the annoying problem concerning non-correctable syndromes. In their paper [41] the three authors define a scheme that does not rely on Goppa codes and that can be instantiated with random codes. Moreover, unlike in the CFS signature scheme, the signature is not computed by using a decoding algorithm for the chosen code and is therefore completely free from the restrictive constraint of using families of structured codes with a hidden intrinsic trapdoor.

The most curious aspect that emerges from this introduction (and which may sound rather strange) is the fact that the signing phase of KKS is done without decoding. Let us denote the set of all binary vectors of length $n$ and of Hamming weight $t$ by $\mathbb{F}_2^{n,t} := \{x \in \mathbb{F}_2^n : w_H(x) = t\}$, the set of decodable syndromes by $H\mathbb{F}_2^{n,t} := \{Hx : x \in \mathbb{F}_2^{n,t}\}$ and the set of all documents by $\mathcal{D}$ (which may be more formally defined as $\mathcal{D} := \mathbb{F}_2^*$ i.e. the set of all binary strings having an arbitrary finite length). The design remains essentially unchanged, with a public code specified through its parity-check matrix $H$, but the idea is to *directly* define a secret application

$$\phi : \mathcal{D} \longrightarrow \mathbb{F}_2^{n,t}$$

in order to automatically generate a signature $\phi(d)$ for a generic document $d \in \mathcal{D}$. The signer then sends $(d, \phi(d))$. To complete the picture, one more

the length of the input, rather than the input itself. Since it is always desirable for a cryptographic function to have time complexity that is polynomial in the length of the input, writing $\mathsf{KGen}(\lambda)$ would appear inconsistent given the premises.

function is evidently needed, which is the <u>public</u> application

$$\chi : \mathcal{D} \longrightarrow H\mathbb{F}_2^{n,t}$$

to be used in the verification step. Given a pair $(d, \sigma)$, the signature $\sigma$ is valid if and only if $\chi(d) = H\sigma$ and $w_H(\sigma) = t$. Against this backdrop, the function $\chi$ is conceived *a posteriori*, to make the calculations match when an attempt to validate the signature is made.

**Remark 2.4.** When outlining the two functions $\phi$ and $\chi$, in order to keep the dissertation at a more general and practice-oriented level, the domain of such functions has been defined as the rather informal set of all documents $\mathcal{D}$. It is clear that, from a theoretical point of view, what happens is that instead of starting from $\mathcal{D}$, depending on the mathematical characteristics of the cipher, a *finite* domain (e.g. $\mathbb{F}_2^k$ or $\mathbb{F}_2^{n-k}$, as we shall see in some of the analyzed cases) is chosen. Such a set is commonly interpreted as the message space, which can be easily reached from $\mathcal{D}$ by means of an appropriate hash function. The interposition of a third set between domain and codomain constitutes that necessary and naturally forced step to take, given that there are an infinite number of documents but a finite number of signatures in this setting. For this reason, in the following $\phi$ and $\chi$ will be defined as

$$\phi : X \longrightarrow \mathbb{F}_2^{n,t}$$
$$\chi : X \longrightarrow H\mathbb{F}_2^{n,t}$$

where $X$ is a finite set (the message space indeed), being this situation equivalent to the foregoing one if a public hash function $h$ is taken into account. These arguments are slightly different when it comes to CFS, since in that case $\phi$ is not even defined (remember that CFS does not compute the signature directly by means of a dedicated function but rather by decoding) and also $\chi$ is not defined a priori, being interpreted by a public hash function $h$ that proceeds by trial and error to end up into $H\mathbb{F}_2^{n,t}$ directly from $\mathcal{D}$. It is precisely due to this tentative approach that it makes sense to start directly from $\mathcal{D}$ in the case of CFS.

Designing the application $\phi$ is clearly the most crucial step in a *signing-without-decoding* technique. Next, the definition $\chi$ follows accordingly, having to satisfy the equation $\chi(d) = H\phi(d)$ for the chosen function $\phi$ on every input $d$. Notice that, in any case, there is the need for both $\phi$ and thereafter $\chi$ not to be a mere enumeration of their images, for efficiency reasons which will be better explained in one of the forthcoming examples. Furthermore, for security reasons, the public $\chi$ must not reveal anything about the secret $\phi$.

The function $\chi$, on the other hand, deserves special attention, as it turns out to be the centerpiece of the whole theory about constructing any digital signature algorithm that aims to rely on deterministic decoding. In fact, although born within the *signing-without-decoding* context as a feedback function, it can still subsist without $\phi$ and, more generally, outside of the latter context, as it can be independently intended as a smart and efficient way to map $\mathcal{D}$ to $H\mathbb{F}_2^{n,t}$ in order to consequently apply the logic of CFS. In a parallelism with a classic *hash-and-sign* scenario, this map plays the role of the hash function whose output has to be compared with the outcome of a certain sequence of operations on the signature to check its validity. The function $\chi$ is the available-to-all verification function, and precisely for this reason it should not disclose indications on how to easily decode from $H$. If this property is guaranteed, then the security of the resulting scheme is equivalent to that of the Niederreiter cryptosystem: to forge a signature from the two public data $\chi$ and $H$, an opponent would indeed have to solve for $\sigma \in \mathbb{F}_2^{n,t}$ the fundamental equation

$$\chi(m) = H\sigma$$

with $m \in X$ the message whose signature is to be forged, and this is actually an instance of SDP. In short, $\chi$ is the keystone of every CFS-like digital signature algorithm. It is hence very important to be familiar with these notions because most of the scientific investigation in this field revolves around this function and how best to design it.

**Example 2.1.** Precisely in relation to security, an example of a very foolish and ingenuous choice for $\chi$ is

$$\begin{aligned} \chi : X &\longrightarrow H\mathbb{F}_2^{n,t} \\ m &\longmapsto H\varepsilon(m) \end{aligned}$$

where $\varepsilon$ is the canonical embedding (enumeration) function towards constant weight vectors (which has already been mentioned in the Niederreiter encryption algorithm). It is clear that such a candidate would lead the signature to an immediate forgery since an opponent can create a valid message/signature just by taking $(m, \varepsilon(m))$.

**Example 2.2.** Observe that there is no real need for $\chi$ to be bound to $H$ in an algebraic sense. In fact, an opposite choice to the previous example could paradoxically be a function that, given $m \in X$, sends it to a *random* $e \in H\mathbb{F}_2^{n,t}$. This is equivalent to saying that $\chi$ is be randomly picked among all possible $|H\mathbb{F}_2^{n,t}|^{|X|}$ maps from $X$ to $H\mathbb{F}_2^{n,t}$. Forging such a scheme seems to be as hard as breaking the Niederreiter cryptosystem, since the only thing one is left with to forge a signature is the decoding of the syndrome $\chi(m)$, which has no direct mathematical connection with $m$ as it has been derived at random. However, the issue here is that this scheme is not efficient because it requires a huge public key, considering that $\chi$ should become a common part of the public key and for its storing $|X|(n-k)$ bits are needed. Practically speaking, the definition of $\chi$ should be better than an enumeration of its images, for which too much memory is required. Ultimately, a random application $\chi$ would be a good choice in terms of security but a bad one for concrete use.

Let us now proceed to deal with the technical part of KKS. For its design, the authors exploit the fact that for every linear code the set of its correctable syndromes contains a linear subspace of relatively large dimension $l$. Thanks to this linear structure, one can efficiently generate correctable syndromes covering a great portion of the set $H\mathbb{F}_2^{n,t}$. Unfortunately, due to this same property, the proposed scheme can be broken after approximately $l$ of its uses. Let $V$ be a binary $[n, n-r]$ code with minimal Hamming distance $d_V > 2t$ and let $C$ be an *equidistant* $[n', k']$ code with minimal Hamming distance $d_C = t$ and $n > n'$ (recall that an equidistant code is a code in which the distance between every two codewords is constant). Let $H = [H_1 \ldots H_n]$ be a $r \times n$ parity-check matrix of the code $V$ (where $H_i$ denotes the $i$-th column of $H$) and let $G$ be a $k \times n$ generator matrix of the code $C$. Define the $r \times k'$ matrix

$$F := H(J)G^{\top}$$

where $J \subset \{1, \ldots, n\}$ with $|J| = n'$, and $H(J)$ denotes the $r \times n'$ submatrix of $H$ consisting of the columns $H_j$ for $j \in J$. It is easy to prove the following.

**Proposition 2.1.1.** *$\forall x \in \mathbb{F}_2^{k'} \setminus \{\mathbf{0}\}$ it happens that $Fx \in H\mathbb{F}_2^{n,t}$.*

Starting from these premises, a signature scheme can be designed as follows: the signer publishes the two matrices $H$ and $F$, and keeps instead hidden the set $J$ and the matrix $G = [G_1 \ldots G_{n'}]$ as private keys. The signer then computes the $k' \times n$ matrix $G^* := [G_1^* \ldots G_n^*]$ whose columns are defined as follows:

$$G_i^* := \begin{cases} \mathbf{0} & \text{if } \ i \notin J \\ G_{\eta(i)} & \text{if } \ i \in J \end{cases}$$

where $\mathbf{0}$ is the zero column of length $k'$ and $\eta : J \longrightarrow \{1, \ldots, n'\}$ is the bijective function that enumerates the integers in $J$ in increasing order. The signer finally signs a message $m \in \mathbb{F}_2^{k'} \setminus \{0\}$ by a signature $e \in \mathbb{F}_2^n$ that they compute as $e := G^{*\top}m$. One can easily see that $e$ is a unique solution of the following signature equation:

$$He = Fm : w_H(e) = t.$$

Using the notation introduced at the beginning of this subsection, we can say that the signing function is

$$\begin{aligned} \phi_{\mathrm{KKS}} : \mathbb{F}_2^{k'} \setminus \{\mathbf{0}\} &\longrightarrow \mathbb{F}_2^{n,t} \\ m &\longmapsto G^{*\top}m \end{aligned}$$

while the verification function is

$$\begin{aligned} \chi_{\mathrm{KKS}} : \mathbb{F}_2^{k'} \setminus \{\mathbf{0}\} &\longrightarrow H\mathbb{F}_2^{n,t} \\ m &\longmapsto Fm. \end{aligned}$$

The proof of the correctness of the signature comes from the simply demonstrable fact that $HG^{*\top} = H(J)G^{\top}$ and the proof of its uniqueness comes from the fact that $d_V > 2t = 2d_C$.

Such a signature scheme is not resistant against homomorphism attacks, i.e after observing two signed messages $(m_1, e_1)$ and $(m_2, e_2)$ an opponent can create a false but valid message/signature pair $(m_1 + m_2, e_1 + e_2)$. This is not really an issue, since to avoid this kind of attack the usual practice is to consider any "enough nonlinear" function, such as any good hash function $h$, and to preliminarily apply it to $m$ so that $e := G^{*\top}h(m)$. This yields to the modified signature equation

$$He = Fh(m) : w_H(e) = t.$$

By the way, homomorphism attacks are not the biggest problem with this scheme. A positive aspect is that discovering the trapdoors $J$ and $G$ seems to be as difficult as decoding the code $V$, since every column of the public matrix $F$ is a linear combination of exactly $t$ columns of the matrix $H$ taken from the positions $J$ and to find this linear combination is the same as decoding $V$ for some particular value of the syndrome. Nevertheless, although every opponent's attempt to solve the previous equation should fail because of the hardness of decoding an arbitrary linear code, the straight application of this basic version of KKS unfortunately demands a too large code length. In fact, to have a sufficiently large work factor, since we are using equidistant codes, the first useful equidistant code has a length which is too large for any practical application. In order to make an attacker face a sufficiently challenging work factor, the length $n$ should be at least of the order of $10^{15}$. This is because the use of equidistant codes imposes important conditions on the parameters $n'$ and $k'$ and consequently also on $n$.

In order to reduce the parameters so that the scheme becomes practically usable, the idea is to replace the equidistant code with any random $[n', k']$ code $C$ whose nonzero codewords $c$ have all weight $t_1 \leq w_H(c) \leq t_2$. This leads to the further modification of the signature equation

$$He = Fh(m) : t_1 \leq w_H(e) \leq t_2.$$

The only thing to pay attention to is the condition $d_V > 2t_2 = 2d_C$ which guarantees the uniqueness of a signature, i.e. the uniqueness of the solution of the above equation. Either way, this requirement is not that important for the proposed scheme since one message can have multiple signatures. The only requirement is that solving the signature equation should be challenging if the number of errors falls within the range $[t_1, t_2]$.

At this point, the question of how to obtain codes with such properties arises. In other words, the focus shifts to how frequent it is for the distribution of the weights of a random code to be confined between two (nontrivial) integers $t_1$ and $t_2$. The answer is provided by the following pair of propositions:

**Proposition 2.1.2** (Carlitz-Uchiyama Bound)**.** *Let $U$ be the dual of a binary BCH code of length $n' = 2^m - 1$ and designed distance $d = 2s + 1$. Then $\forall u \in U$ it holds that*

$$\left| w_H(u) - \frac{n' + 1}{2} \right| \leq (s - 1)\sqrt{n' + 1}.$$

**Proposition 2.1.3.** *Let $U$ be a code defined by a random $k' \times n$ systematic generator matrix $G$ and let $\varepsilon \in (0, 1)$. Then the probability that $G$ generates a code where the weight of every nonzero codeword lies in $[\frac{n'}{2}(\varepsilon - 1), \frac{n'}{2}(\varepsilon + 1)]$ is at least $1 - 2^{-(n'-k')+n'h(\varepsilon)+1}$ where $h(x) := -x\log_2(x) - (1 - x)\log_2(1 - x)$ is the binary entropy function.*

For a complete insight on BCH codes and on this kind of bounds, consult [48] and the original KKS paper [41]. Observe that the last theorem does not certainly guarantee the bound, but works at a probabilistic level. The odds of the property not occurring can be lowered at will, and this concept is somewhat the same that recurs with the RSA pseudoprimes. The usefulness of this brief theoretical digression is that one can actually recreate the setting required to apply the ultimate version of KKS, using either BHC codes or random ones.

Without going into too much detail, let it be known that there are no less than four versions of KKS, trivially called KKS-1, KKS-2, KKS-3 and KKS-4. Each version attempts to apply the basic model of KKS in such a way as to optimize all the computational aspects. Below, a diagram summarising the general KKS paradigm is shown:

**Algorithm 5** (KKS)

**Parameters:** $k', n', k, n \in \mathbb{N}$

```
1  $\boxed{\mathsf{KGen}_{\mathrm{KKS}}(1^\lambda)}$: according to the security parameter $\lambda$, a random $k' \times (n'-k')$
   matrix $D$ is chosen and the systematic $k' \times n'$ matrix $G = [I_{k'}|D]$ is built;
   with a certain high probability, the codewords belonging to this code will
   have weight in the range $[t_1, t_2]$
2  choose a random systematic $k \times n$ matrix $H$ for suitable $n \geq n'$ and $k$
3  $J \leftarrow$ a random subset of $\{1, \dots, n\}$ of cardinality $n'$
4  $F = H(J)G^\top$
5  $\mathsf{pk} = (F, H, t_1, t_2)$
6  $\mathsf{sk} = (J, G)$
7  $\boxed{\mathsf{Sign}_{\mathrm{KKS}}(m, \mathsf{sk})}$: given a message $m \in \mathbb{F}_2^{k'} \setminus \{0\}$
8  as accurately explained on page 37, preliminary compute the matrix $G^*$
9  $\sigma = G^{*\top} m$
10 $\boxed{\mathsf{Verify}_{\mathrm{KKS}}(\sigma, m, \mathsf{pk})}$: given $(m, \sigma)$, check that $t_1 \leq w_H(\sigma) \leq t_2$, then
11 $\sigma$ is valid $\iff$ $Fm = H\sigma$
```

In conclusion, it is worth pointing out that all four versions of KKS have been broken. The main concern is that each use of this digital signature algorithm, generating a signature $e$, inevitably reveals $w_H(e)$ positions of $J$. Once $J$ has been discovered, one can easily derive $G$ by solving linear systems given by the equation $H(J)G = F$. In both [18, 50], efficient attacks on all concrete KKS proposals can be found. Just to give an idea, it emerges how, on average, it takes only about 20 signatures to break the first three variants of KKS. For these reasons, KKS has sadly to be considered at best a one-time digital signature scheme.

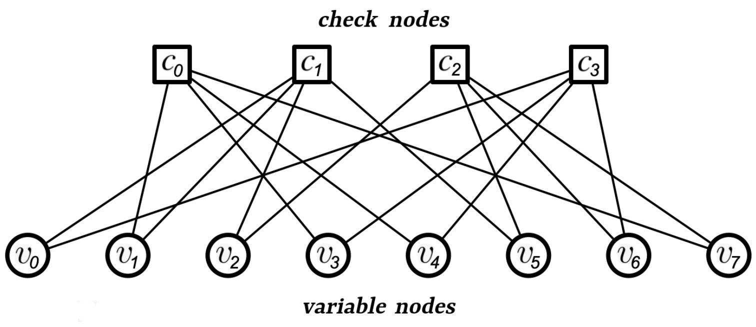
## 2.2 Overview On The Latest Panorama

Although since the two earliest and historical proposals CFS [22] and KKS [41] plenty of ideas and projects have followed, the issue of finding a viable candidate is still an open and tough problem. As usual, the heart of the matter is always the atavistic duel between security and efficiency, which do not seem to be able to coexist within the same code-based digital signature scheme. The main drawback of CFS is the signing time, since the procedure of hashing a message with a counter until a decodable digest is found is an extremely wasteful approach, but at least its security is still intact. On the other hand, KKS suffers from several security issues, which blows up its good aspect of being quite efficient.

The recent history of code-based digital signatures has witnessed the emergence of several schemes which, with the exception of some sporadic examples that have attempted quite innovative and extravagant approaches, can all ultimately be placed under a quite simple classification. The first and foremost outlier is Wave [28], which relies on the brilliant idea of exploiting the hardness of decoding in a new way. In fact, solving SDP for a certain matrix $H$ is well known to be hard when we seek a word of weight below a certain threshold. Beyond this point the problem becomes easy to solve with linear algebra, until a second threshold is reached and beyond which the problem becomes hard again. Wave relies on the difficulty of decoding with very high weight, and what is even more peculiar is that this approach works to its maximum effect in the ternary field $\mathbb{F}_3$ (instead of the classic binary one), where the family of the generalized $(U, U+V)$ codes takes full advantage of its potential.

Besides the aforementioned exceptions and the schemes that have tried to follow the paradigm provided by CFS in a relatively canonical way, like the two submissions to NIST PQC named RaCoSS and pqsigRM (both broken), three main branches can be identified. It should be noted that a given scheme may obviously belong to more than one category, as the groupings proposed below are not mutually exclusive. These three strands of study are heavily founded on the advanced notions of:

◇ **LDPC codes:** schemes belonging to this category aim to substitute the classical (deterministic) error-correcting codes with a typology of efficiency-oriented codes that, however, operate with a certain degree of uncertainty. This alternative line of development represented by probabilistic decoding has been directly inspired by Shannon's probabilistic approach to coding theory. In his doctoral thesis [36] dated 1962, Gallager dealt for the first time with Low-Density Parity-Check codes, commonly abbreviated as LDPC. These codes, as their name suggests, are defined by a parity check matrix which is a sparse matrix (i.e. with a low number of ones). For over 30 years they have not received much attention from the scientific community and have only recently been rediscovered. The missing piece came in 1981 with the game-changer idea by Tanner in [59], where a particular class of bipartite graphs related to ECC is illustrated. Tanner graphs fit LDPC codes like a glove, as this representation allows the theoretical notion of parity bit to be effectively and efficiently revisited in a computer-oriented guise.



This computational model amazingly simplifies the implementation of the so-called message-passing algorithms, like the belief propagation algorithm or the bit-flipping algorithm. Informally, these routines work by iteratively exchanging information between the two sets of nodes of the Tanner graph (known as variable nodes, upon which the bits of the codeword to be decoded are arranged, and check nodes) and by repeatedly updating the values of certain bits of the variable nodes in such a way as to make every check node agree with the values of the corresponding linked variable nodes. More precisely, the requirement to be satisfied is that each check node must have an even number of edges hitting a 1 arising from its connected variable nodes. This is because a check node of the Tanner graph plays the role of a row-condition of the parity-check matrix of the code, and this condition imposes on the codeword that wants to belong to the code to have an even number of ones in the specified positions. This process continues until total compatibility is obtained, which is the graph-wise counterpart of the fact that the product between the considered vector and the parity-check matrix returns the **0** vector as the result. This way, super fast decoding able to correct both erasures and errors in polynomial time can be achieved. Since a Tanner graph is used as the graphical/computational representation of a parity-check matrix, the property of an LDPC matrix to be sparse is the key point of the efficiency of this approach, because it implies a streamlined form of the graph which therefore has few arcs between the check nodes and the variable nodes. Nevertheless, the price to pay is the possibility of failure on the part of these algorithms, since there might be cases in which the execution does not terminate. It is precisely by virtue of the existence of such outputs on which the aforementioned decoding algorithms are faced with an endless loop that this method is usually referred to as probabilistic (as opposed to deterministic): the Decoding Failure Rate (DFR) is generally low, but non-null. Although failures may seem irrelevant because of their low occurrence, they should not be underestimated. The example of the asymmetric cipher LEDAcrypt [8] is emblematic in this respect, as the observation of the events of decoding failure leaking information about the secret key [54] allowed the construction of an attack on the cipher [3]. Besides LDPC, there are also other similar families of codes, like the Moderate-Density Parity-Check codes (MDPC) and the Low-Density Generator-Matrix codes (LDGM), whose definition is embedded in their name and whose logic is basically the same as the LDPC codes. A further advancement is the use of Quasi-Cyclic LDPC codes (QC-LDPC), which resort to circulant matrices exploiting their algebraic definition to spread the ones around the parity-check matrix. This trick results in great savings in terms of storage size. As for their application, the best known signature schemes exploiting LDPC codes are [9] in the Niederreiter framework and [44, 52] in the Schnorr-Lyubashevsky context.

◇ **Rank Metric:** let us consider a finite field $\mathbb{F}_{q^m}$, where $q$ is a prime, and let's consider the vectorial space $V := \mathbb{F}_{q^m}^n$ of dimension $n$ over $\mathbb{F}_{q^m}$. In turn, also $\mathbb{F}_{q^m}$ can be seen as a vectorial space of dimension $m$ over $\mathbb{F}_q$ and it has therefore a basis $x_1, \ldots, x_m$ of cardinality $m$, with $x_i \in \mathbb{F}_{q^m}$. Let $v := (v_1, \ldots, v_n) \in V$. Every component of $v$, being an element of $\mathbb{F}_{q^m}$, can be represented as $v_i = a_{1i}x_1 + a_{2i}x_2 + \ldots + a_{mi}x_m$. Hence, exploiting this representation, every vector $v = (v_1, \ldots, v_n) \in V$ can in turn be represented as a matrix

$$A_v = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \ldots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \ldots & a_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \ldots & a_{mn} \end{bmatrix}.$$

The rank metric is defined precisely via this matrix. For $x, y \in V$:

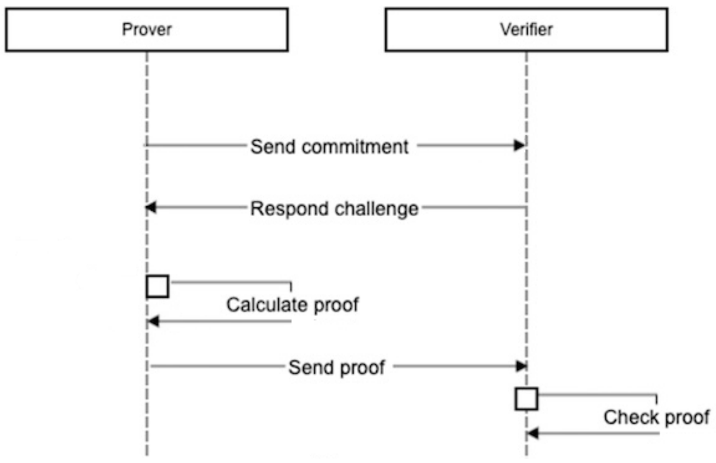
$$d_r(x, y) := \text{rank}(A_{x-y}).$$

The definition of rank weight is instead the following: given a vector $v \in V$, we define its rank weight as

$$w_r(v) := \text{rank}(A_v).$$

Clearly, in order for the definition to be consistent, one can prove that this definition does not depend on the choice of the basis. The first cryptosystem based on rank metric codes, called GPT, was proposed in 1991 by Gabidulin, Paramonov and Tretjakov in [34]. Since then, several proposals have tried to exploit some old settings revisiting them with the rank metric instead of the Hamming one. One of the most recent and prominent digital signature schemes is called Durandal [4] and consists in the adaptation of the Lyubashevsky scheme (which is a lattice-based digital signature scheme) to rank metric codes. Other two interesting and very recent proposals belonging to this category are MURAVE [43] and the NIST PQC proposal RankSign (broken).

◇ **Fiat-Shamir Transformation:** it is well known that a valid alternative for building digital signature schemes is provided by the conversion of zero-knowledge identification schemes using the Fiat-Shamir transform, as extensively explained in [32]. Briefly speaking, a zero-knowledge identification scheme (ZKID) is a three-move protocol that allows one party (the prover) to convince another party (the verifier) that he knows a certain secret while avoiding communicating any further information on the secret itself other than the fact that the secret is actually known. The idea behind zero-knowledge proofs is to solve the issue of demonstrating a (secret) knowledge without adopting the most straightforward and banal solution of publicly exposing it, but rather trying not to disclose any extra detail at all. Practically speaking, the first of the three main steps sees the prover committing to a certain secret value and sending this commitment to the verifier. In the second step, the verifier binds the commitment and the message to a challenge, and sends the challenge to the prover. In the third step, the prover uses their secret value to produce a valid answer to the challenge and sends the response to the verifier. Finally, the verifier checks if the response is consistent with the commitment and the challenge. To ensure that the prover is not blindly guessing and thus getting the correct answers by chance, the verifier can challenge them multiple times. By repeating this interaction, the possibility of the prover faking to know the secret knowledge can be reduced at will until the verifier is satisfied. Indeed, if the prover is guessing, they will eventually be proven wrong by the verifier's tests. Instead, if the secret is actually known, the prover will always pass the verifier's test without any problems.



What the Fiat-Shamir transformation does within this context is, in a nutshell, to remove interactivity from this kind of protocols. Concretely, it replaces the second step with a cryptographic hash function: the challenge is bound to the commitment via the verdict of an available-to-all function (which is the same to be used to verify the digital signature), so that the prover is able to autonomously and locally perform each step of the protocol without the need for external involvement. The pros of such an approach are that no trapdoors are involved, since the protocol merely uses a random instance of a hard-to-solve problem; the cons are instead the large signature sizes, since many repetitions are needed to ensure security. The most prominent ZKID schemes on which the Fiat-Shamir heuristic has been applied (and which, by the way, are usually all instantiated using random codes on the classical decoding problem) are the works by Stern [58] and Veron [61], the two improvements of their idea known as CVE [19] and AGS [2], the reinterpretation of the Veron scheme from the rank metric perspective known as RVDC [13] and finally LESS [11], which is based on the problem of determining whether two codes are permutationally equivalent or not, and the brand-new MEDS [21], which can be characterized as the rank-metric version of LESS.

One last scheme must be named, as far as we are concerned, to complete the picture of the recent history of code-based DS. Among the newest proposals of code-based digital signature schemes, a paper called "*An Efficient Code Based Digital Signature Algorithm*" [53] deserves to be named. The reason is that this paper appears quite revolutionary since, according to its title, it would seem to present an efficient code-based algorithm. Given all the complications of creating a code-based digital signature that is both secure and efficient encountered over the course of the entire thesis, the existence of such a scheme would be of great impact to the crypto-community. In reality, a more careful investigation has ascertained the presence of *a flaw* in the signing algorithm described in the just quoted paper. A complete description of this flawed scheme is given in the next and final chapter, as is our cryptanalysis.

# Chapter 3

# A Flawed Proposal

As already mentioned in the last section of the previous chapter, in this third and last one we will deal with a precise digital signature scheme whose name is mCFS$_c$ and which has first been described in [53] by Ren, Zheng and Wang. The proposed scheme follows the benchmark of CFS with the idea of revisiting the way in which messages are sent towards the space of decodable syndromes. In other words, using the concepts and the notation introduced in Chapter 2, the three authors' attempt is to design a smart and streamlined $\chi$, instead of proceeding by attempts. The outlined hash function is built starting from coding theory, in particular exploiting the Merkle-Damgård construction on the compression function consisting in the simple multiplication by a parity-check matrix. Unfortunately, the so-defined code-based hash function has a single but major problem: leaks information on how to sign. This breach opens the door for our attack, which also holds for a certain type of generalization of the mCFS$_c$ approach.

## 3.1 mCFS$_c$

One of the main reasons why most of the cryptographic studies regarding code-based digital signatures keep orbiting around CFS is its connatural security. In fact, most CFS-like schemes persist to be unbroken, despite their slow speed in the signing process and their cumbersome keys. Of these times, the trend that can be spotted in this research field is to fuse the KKS idea of the pursuit of decodable syndromes together with the CFS skeleton. An iconic example is the mCFS$_c$ signature scheme [53], which hashes the message into a decodable syndrome using a code-based hash function. Thus, let us now start by describing how a hash function can be achieved from coding theory and then proceed to describe the mCFS$_c$ algorithm.

### 3.1.1 Code-Based Hash Functions

Hash functions are one of the cornerstone notions of the entire cryptographic world. They are deterministic functions that map bit arrays of arbitrary size (messages) to a bit array of a fixed size (digest). Their main feature is that they are one-way functions, i.e. easy to compute but practically infeasible to invert. Ideally, it would be desirable that the only method to find a message that leads to a particular hash was to perform a brute-force search on several inputs to determine whether any of them can produce a match. This property is known as *first preimage resistance*. A further property a good hash function should have is the so-called *collision resistance*, which translates into the computational impossibility of finding two different messages with the same hash value. Notice that this last property implies the so-called *second preimage resistance*. In general, what we want to occur when applying a hash function is the well-known *avalanche effect*: a small change on the message should entail a so drastic change on the digest that the new hash value appears totally uncorrelated with the prior one.

As far as we are concerned for the continuation of our argument, it is relevant to point out that a clever way to achieve hash functions is, among others, that of exploiting the Merkle-Damgård construction. This design (whose original and complete description can be found at [26]) is based on the iterative application of a certain compression function until the whole message is read. A *compression function* is a map

$$c : \mathbb{F}_2^s \longrightarrow \mathbb{F}_2^r$$

that sends $s$-bit vectors into $r$-bit vectors, for certain fixed values of $s, r \in \mathbb{N}$. Roughly speaking, a compression function can be thought of as the early stage of a hash function, in the sense that it can only process fixed-length input strings. Apart from that, it is desirable for a compression function to have the same properties mentioned above for a hash function. For this reason, a good way to draw compression functions is to derive them from mathematical challenges, considering the trapdoor scenarios the latter offer. Multiplication by a parity check matrix is a striking example of the above: throughout this entire thesis it has become glaring how difficult it is to decode a syndrome.

The crucial point of the work by Merkle and Damgård is the proof that the security of the compression function implies the security of the resulting hash function. The original paper [26] is all about the proof of this property indeed. A hash function built starting from the Merkle-Damgård construction is as resistant to collisions as it is its originating compression function; any collision for the full hash function can be traced back to a collision in the corresponding compression function. The same holds for the one-way property. Since a hash function must be able to digest arbitrary-length messages to produce a fixed-length digest, this can be achieved by breaking the input up into a series of equal-sized blocks and operating on them in sequence using a one-way compression function. The compression function can either be specially designed for hashing or be built from a block cipher. Usually, the last processed block should also be length padded, since it is not always the case that the starting input is an exact multiple of the chosen sub-length.

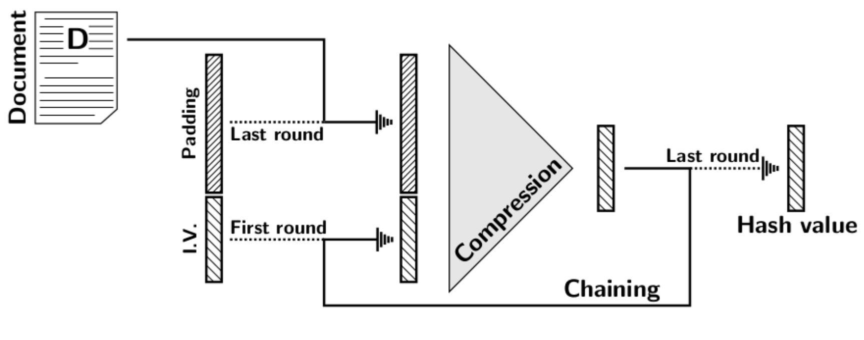
Concretely, let $c : \mathbb{F}_2^s \longrightarrow \mathbb{F}_2^r$ be a generic compression function. The length of the digest will be $r$. Given a random string $d \in \mathbb{F}_2^*$ of arbitrary length, the Merkle-Damgård construction leads to the definition of a hash function $h_c : \mathbb{F}_2^* \longrightarrow \mathbb{F}_2^r$ following this procedure:

$\boxed{1}$ consider the string $d_{pad}$, obtained by padding $d$ in such a way that its final length $|d_{pad}|$ is a multiple of $s$ and split $d_{pad}$ in $|d_{pad}|/s$ blocks of length $s$;

$\boxed{2}$ in the first round, combine the first block of $d$ with a fixed initial vector (IV) to obtain the first "state" $S_1$ of length $s$ and compute $c(S_1)$;

$\boxed{3}$ in the $i$-th round, combine the $i$-th block of $d$ with the output of $c(S_{i-1})$ with the $i$-th block of $m$ obtaining the $i$-th state $S_i$ and compute $c(S_i)$;

$\boxed{4}$ the output of the hash function is given by $h_c(d) := c(S_{|m|/s})$.



In 2003 Augot together with two of the inventors of CFS came out with a brilliant idea (fully availabe in [5]) of a code-based hash function based on

Merkle-Damgård. The underlying compression function is nothing more than a purpose-designed multiplication by a parity check matrix $H$, which will be exhaustively defined in the upcoming example. What makes this new approach really valuable is that such code-based compression functions, as opposed to traditional compression functions, are *provably secure*. In fact, classical compression functions, although very fast, cannot in general be proven secure, since they do not depend on hard-to-solve mathematical problems whose difficulty can be studied and classified. Conversely, the compression function proposed relies on a well-known problem called *Regular Syndrome Decoding Problem* (RSD) which is a particular instance of the more general SDP and which has regardless been proven to be equally secure. Indeed, its NP-completeness has been assessed by a reduction (available in the appendix of both [5,6]) from the well-understood Three Dimensional Matching. Before laying down its formal statement, let us recall what is meant by regular word.

**Definition 3.1.** Let $n, w \in \mathbb{N}^*$ such that $w \mid n$. A $(n, w)$-*regular word* is a vector $v \in \mathbb{F}_2^n$ of Hamming weight $w$ that in each interval of positions of the form $\left(\frac{(j-1)n}{w}, \frac{jn}{w}\right]$ contains exactly one nonzero entry $\forall\, 1 \leq j \leq w$.

On an intuitive level, this means that if we chop up the vector in $w$ equal parts and imagine it as the concatenation of these blocks, then each part contains exactly one nonzero entry.

**Problem 5** (RSD)**.** Let $C$ be a binary linear code.
GIVEN: an $r \times n$ parity-check matrix $H$ for $C$, a syndrome $s \in \mathbb{F}_2^r$ and an integer $w \mid n$.
FIND: a $(n, w)$-regular word $e \in \mathbb{F}_2^n$ such that $w_H(e) = w$ and $He = s$.

Code-based hash functions belonging to the family proposed by Augot et al. have been named *Fast Syndrome-Based hash functions* (FSB). As already said, their security is certified by the NP-completeness of RSD, which consequently implies the solidity of the compression function which, in turn, thanks to Merkle-Damgård gimmick, finally guarantees the full resistance of the deriving hash function. Furthermore, observe that, unlike asymmetric ciphers where, in addition to the arduous challenge, there is always trapdoor acting like a smokescreen to hide it, in this case the latter is missing and security can be readily related to the difficulty of decoding. As almost always, the sturdiness of a scheme comes yet at the cost of being less efficient. FSB hash functions are slower than traditional ones and use quite a lot of memory.

In both their two main papers [5,6], but in particular in the second and latest one dated 2005, the authors work precisely on the establishment of the optimal parameters to guarantee practical security against the best-known decoding techniques (like Information Set Decoding (ISD) and Wagner's Generalized Birthday Paradox (GBP)). As a matter of fact, much of their work is devoted to this investigation of the behaviour of the cryptanalitic attacks as the parameters change, in order to find a good compromise between security and feasibility of implementation. The description of FSB hash functions is contained in the example below and is of fundamental importance for the culmination of our dissertation since such a code-based hash function is used as the decodable-syndromes-targeting function $\chi$ for the mCFS$_c$ digital signature scheme. So, let us now proceed to explain their construction.

**Example 3.1** (FSB)**.** Recalling that it is still challenging to recover vectors of weight $w$ in the set of vectors with a certain syndrome, then, computing syndromes can serve as a one-way compression function: let us see how. First of all, the length $r$ of the digest is chosen. Given a generic $r \times n$ parity-check matrix $H$ of an $[n, n-r]$ binary code $C$, let $w$ be an integer dividing $n$ and set $l := n/w$ and $s := w\lfloor \log_2(l) \rfloor$. The trick is to embed the $s$ input bits in a $(n, w)$-regular word, so that the result of the multiplication of the latter by the chosen parity-check matrix $H$ will be a syndrome whose provenience can be traced. The compression function $c_H : (\mathbb{F}_2)^s \longrightarrow (\mathbb{F}_2)^r$ based on $H$ is computed by carrying out the following steps:

$\boxed{1}$ let $H_1, \dots, H_w$ be $r \times l$ sub-matrices such that $H = [H_1 \dots H_w]$ using block notation;

$\boxed{2}$ given $x \in \mathbb{F}_2^s$, split it in $w$ blocks of length $\lfloor \log_2(l) \rfloor$; according to the way each block is made, it can be interpreted as an integer number between 0 and $l-1$, hence we can represent $x$ as $x = (z_1, \dots, z_w)$ with $0 \le z_i \le l-1$;

$\boxed{3}$ set $c_H(x)$ as the vectorial sum of the $(z_i + 1)$-th column of the matrix $H_i$ for $i = 1, \dots, w$. In formulas, if $(H_i)_j$ is the $j$-th column of $H_i$, we have that

$$c_H(x) := \sum_{i=1}^{w} (H_i)_{z_i+1} \, .$$

Unsurprisingly, the compression function has been labelled with $H$, since it strongly depends on the chosen parity-check matrix. A first remark must be made on the definition of $s := w\lfloor \log_2(l) \rfloor$ and the consequent interpretation of $x$ which may sound rather strange, considering that not all the columns of $H_i$ can be accessed when $l$ is not of the form $l = 2^m$ for a certain $m$. In practice, $l$ is always a power of 2, so that its binary logarithm is always a round (integer, without decimals) number. A second observation concerns the code-based hash function $h_{c_H} : \mathbb{F}_2^* \longrightarrow \mathbb{F}_2^r$ that can be now constructed using the Merkle-Damgård construction on the just illustrated $c_H$. First of all, in the case of this kind of matrix-reliant compression functions, we are going to use the notation $h_H : \mathbb{F}_2^* \longrightarrow \mathbb{F}_2^r$ for the corresponding hash function instead of the previous one, so as to streamline the writing and avoid a captious and redundant sequence of labelling subscripts. Furthermore, the construction proposed in Augot's original paper presents a slight (albeit irrelevant) difference with the general construction we have outlined a few pages back. Namely, in the creation of a new state, instead of combining the $s$ bits coming from the document to be hashed with the $r$ bits coming from the compression, only $s - r$ bits are taken from the document. These are then juxtaposed (using the $\|$ operation) with the $r$ bits of the syndrome so as to recreate a legit state of $s = (s - r) + r$ bits. With this set-up, to digest the entire message will require $|d_{pad}|/(s - r)$ rounds instead of the standard $|d_{pad}|/s$. While adopting this method, it is important to choose $w$ in such a way that $s > r$. Lastly, observe that the computation of $h_H$ presupposes the knowledge of $H$.

**Proposition 3.1.1.** *For every state $S_i$ of the hash function $h_H$, the value of $c_H(S_i)$ is a syndrome of an $(n, w)$-regular vector.*

*Proof.* According to the scheme, the state $x = S_i$ is split in $w$ sub-vectors of length $\lfloor \log_2(l) \rfloor$ which, as already pointed out, can be intended as integers $z_1, \ldots, z_w$ between 0 and $l - 1$. Let $e_i \in \mathbb{F}_2^n$ be the vector having support (i.e. the set of the non-null positions)

$$\begin{aligned}\operatorname{supp}(e_i) = \{\, & (z_1 + 1) + 0 \cdot l, \\ & (z_2 + 1) + 1 \cdot l, \\ & \quad \cdots \\ & (z_w + 1) + (w - 1) \cdot l \,\}.\end{aligned}$$

By construction, such a $e_i$ has Hamming weight $w$ and, additionally, its ones are distributed so that every sub-vector obtained by breaking down $e_i$ in $w$ equal parts contains one and only one 1. Hence $e_i$ is a $(n, w)$-regular word. Finally, $c(S_i) = He_i$ i.e. $c_H(S_i)$ is the syndrome of $e_i$. $\square$

We can summarize the compression function as follows: let $n, w, l, s$ be defined as before. Consider the function (bijection when $l$ is a power of 2)

$$\begin{aligned}\text{split} : \mathbb{F}_2^s &\longrightarrow (\mathbb{F}_2^{\lfloor \log_2(l) \rfloor})^w \\ x &\longmapsto (z_1, \ldots, z_w)\end{aligned}$$

that splits a binary vector $x = (x_1, \dots, x_s)$ of length $s$ into $w$ vectors of length $\lfloor \log_2(l) \rfloor$. Every vector in $\mathbb{F}_2^{\lfloor \log_2(l) \rfloor}$ can be seen as an integer between 0 and $l-1$. Define the map

$$\begin{aligned} \mathrm{reg}_{(n,w)} : \mathbb{F}_2^s &\longrightarrow \mathbb{F}_2^n \\ x &\longmapsto (e_1, \dots, e_n) \end{aligned}$$

where $(e_1, \dots, e_n)$ is the vector of Hamming weight $w$ whose components are defined by cases as

$$e_i := \begin{cases} 1 & \text{if } \; i \in \{(\mathrm{split}(x)_j + 1) + (j-1)l : 1 \le j \le w\} \\ 0 & \text{otherwise.} \end{cases}$$

The compression function $c_H$ can be hence written as

$$\begin{aligned} c_H : \mathbb{F}_2^s &\longrightarrow \mathbb{F}_2^r \\ x &\longmapsto H \mathrm{reg}_{(n,w)}(x). \end{aligned}$$

### 3.1.2 The Signature Scheme

The mCFS$_c$ digital signature scheme is built starting from the by now well-known protocol mCFS [22, 25] using a particular code based hash function. Recall that mCFS differs from CFS only for the fact that instead of using a counter to keep track of the attempts, a random seed is used in its place. The hash function used in mCFS$_c$ is exactly a FSB one, like those described in the example at the end of the previous subsection. Although it will seem trivial, it is worth remarking the following property in relation to the code based hash function $h_H$:

**Proposition 3.1.2.** *With the above definitions and notation, the output of the code based hash function $h_H$ is a syndrome of a $(n, w)$-regular word.*

*Proof.* According to the loop construction by Merkle and Damgård of the hash function, the output of $h_H$ is nothing less than the output of the final round $c_H(S_{|d_{pad}|/(s-r)})$. According to Proposition 3.1.1, for any state $S_i$, $c_H(S_i)$ is a syndrome of a $(n, w)$-regular word and so must be the ultimate output of the has function $h_H$. □

In [53], the authors of mCFS$_c$ pick $H$ as the $r \times n$ parity-check matrix of a $[n, n-r]$ binary Goppa code. The parameter $w$ is then chosen to be an integer less than or equal to the correcting capacity $t$ of the code and dividing $n$. This yields to the hash function $h_H : \mathbb{F}_2^* \longrightarrow \mathbb{F}_2^r$ based on $H$. The paper is unclear about the construction of this hash function, in the sense that it is not specified if the hash function is based on the secret matrix $H$ or the public matrix $H_{pub}$. In this connection, we first observe that since the hash function must be part of the public key, the option involving $H$ has to be rejected, since disclosing the secret matrix $H$ (which, remember, is part of the secret key) would reveal information on the efficient decoding algorithm $\delta_C$. Hence, only the other option is left valid and assuming that the hash is based on the public matrix $H_{pub}$, the signature scheme is described by the three algorithms in the following table.

**Algorithm 6** (mCFS$_c$)

```
Parameters: $n, k, t \in \mathbb{N}$
1  $\boxed{\mathsf{KGen}_{\mathrm{mCFS}_c}(1^\lambda)}$: select $n, k, t$ according to the security parameter $\lambda$, then
   choose a random $[n, k]$ binary Goppa code $C$ with correcting capacity $t$ and
   parity-check matrix $H$ and let $\delta_C$ be an efficient syndrome decoding algorithm
   for $C$
2  $P \leftarrow$ a $n \times n$ random permutation matrix
3  $H_{pub} = HP$
4  choose an integer $w \leq t$ and such that $w \mid t$
5  construct the hash function $h_{H_{pub}} : \mathbb{F}_2^* \longrightarrow \mathbb{F}_2^{n-k}$ based on $H_{pub}$
6  $\mathsf{pk} = (H_{pub}, t, h_{H_{pub}})$
7  $\mathsf{sk} = (H, P, \delta_C)$
8  $\boxed{\mathsf{Sign}_{\mathrm{mCFS}_c}(m, \mathsf{sk})}$: given a message $m$
9  pick a random $R$ in $\{1, \ldots, 2^{n-k}\}$ and compute $d = h_{H_{pub}}(h_{H_{pub}}(m) \,\|\, R)$
10 $e = \delta_C(d)$
11 $\sigma = (P^\top e, R)$
12 $\boxed{\mathsf{Verify}_{\mathrm{mCFS}_c}(\sigma, m, \mathsf{pk})}$: let $\sigma = (u, R)$; verify that $w_H(u) \leq t$, then compute
13 $a = h_{H_{pub}}(h_{H_{pub}}(m) \,\|\, R)$
14 $b = H_{pub}u$
15 $\sigma$ is valid $\iff a = b$
```

To check the correctness of this signature scheme, the same argument used for CFS is valid. The idea behind this approach is very simple: since the newly-built FSB hash function can hash the message into a syndrome deriving from a $(n, w)$-regular word, choosing $w$ in such a way that it is smaller than the correction capability of the chosen Goppa code will allow one to always get correctable syndromes. Notice that it is not really important that the syndrome is linked to a regular word in itself, but rather that it is linked to a word of weight $w \leq t$, since this implies that the syndrome is correctable. The regularity of the underlying error vector is a mere side effect of the construction of the compression function used to define the FSB hash function and is not actually relevant in this scenario.

Readdressing the situation using the concepts introduced in the second chapter, it is clear that the impulse to use the hash function $h_{H_{pub}}$ as the function $\chi_{\mathrm{mCFS}_c}$ is very strong, as it offers a very simple way to access the space of decodable syndromes without having to proceed by trial and error. Nevertheless, the simplest way is not always the right one, and indeed it is often the most fallacious one: from a security point of view, this approach *cannot work*.

## 3.2 The Attack

In this subsection we are going to explicit an attack on mCFS$_c$. The attack is based on the fact that the code-based hash function $h_{H_{pub}}$ leaks information on how to sign, due to its cyclic structure. This issue leaves room for an attacker to forge a valid signature on any message of their choice without necessarily knowing the secret key. The vulnerability and the consequent exploit are described in the following proposition.

**Proposition 3.2.1.** *Let* $(\mathsf{sk}, \mathsf{pk})$ *be the output of* $\mathsf{KGen}_{\mathrm{mCFS}_c}(1^\lambda)$. *An attacker knowing the public key* $\mathsf{pk}$ *can forge a valid signature compatible with the private key* $\mathsf{sk}$ *for any message* $m$.

*Proof.* Given a mCFS$_c$ public key $\mathsf{pk} = (H_{pub}, t, h_{H_{pub}})$ and a message $m$, the attacker picks a random $R \in \{1, \dots, 2^{n-k}\}$ and preliminary computes $f = h_{H_{pub}}(m) \,\|\, R$. Then they proceed to compute $d = h_{H_{pub}}(f)$, but it stops before the last round of hash function $h_{H_{pub}}$ thus obtaining the round state $\dot{S} := S_{|m_{pad}|/(s-r)} \in \mathbb{F}_2^s$ instead of the digest $d = c_{H_{pub}}(\dot{S})$. Finally, they compute $u = \mathrm{reg}_{(n,w)}(\dot{S})$ and outputs the couple $\sigma = (u, R)$ as a signature for $m$. Anyone can verify the validity of this signature of $m$ compatible with the secret key $\mathsf{sk} = (H, P, \delta_C)$. In fact, we can compute $a = h_{H_{pub}}(h_{H_{pub}}(m) \,\|\, R)$ and $b = H_{pub}u$ and observe that $a$ is equal to $b$ due to the fact that the multiplication of $u = \mathrm{reg}_{(n,w)}(\dot{S})$ by $H_{pub}$ is exactly the last step of the hash function $h_{H_{pub}}$. Therefore $\sigma$ is a valid signature. $\square$

The key point is that it is sufficient to truncate the execution of the hash function $h_{H_{pub}}$ on input $m$ at the last round to obtain a straightforward and practical way to sign $m$ itself. Once again, using the terminology introduced in the second chapter, the ingenuity is that the function $\phi_{\mathrm{mCFS}_c}$ that tells how to sign has been embedded in $\chi_{\mathrm{mCFS}_c}$ and to retrieve it, one only needs to prematurely stop the computation of the latter. In fact, we can state that

$$\phi_{\mathrm{mCFS}_c}(m) = \delta_w(S_{|m_{pad}|/(s-r)}).$$

Wanting to keep track of where messages land gives too much information, because the signature end up to be precisely the regular vector used to select the columns of $H$ that, summed up, will produce the controlled-origin decodable syndrome. Summing up, the following table shows the schematic steps to perform the attack.

**Algorithm 7** (Attack on mCFS$_c$)

```
Given: the public key $\mathsf{pk}$ of mCFS$_c$
1 Forgery Attack: taken any document $d \in \mathbb{F}_2^*$ proceed to
2 pick a random $R \in \{1, \dots, 2^{n-k}\}$
3 compute $f = h_{H_{pub}}(d) \,\|\, R$
4 begin to compute $h_{H_{pub}}(f)$ but stop before the last compression
5 said $\dot{S} = S_{|d_{pad}|/(s-r)} \in \mathbb{F}_2^s$ the last state, compute $u = \mathrm{reg}_{(n,w)}(\dot{S})$
6 $\sigma = (u, R)$ is a valid signature for $d$
```

## 3.3 Generalization

In these conclusive lines of this chapter we want to generalize the strategy adopted in the mCFS$_c$ signature and show that such an approach still leads to an attack. Indeed, what we have just outlined actually applies also to a certain type of generalization of the mCFS$_c$ scheme. The goal is to consider a modification of $h_H$ and to prove that this new entire family of hash functions is vulnerable to the same attack we have just described for $h_H$ and that is not therefore suitable for secure applications.

Let $B_{n,t} := \{x \in \mathbb{F}_2^n : w_H(x) \leq t\}$ be the set of vectors in $\mathbb{F}_2^n$ of Hamming weight less than or equal to $t$. Let $\gamma_t : \mathbb{F}_2^s \longrightarrow \mathbb{F}_2^n$ be a generic function subject only to the condition that $\mathrm{Im}(\gamma_t) \subseteq B_{n,t}$, i.e. $\gamma_t$ is a function mapping bit strings of length $s$ into bit string of length $n$ with a Hamming weight bounded by $t$:

$$w_H(\gamma_t(v)) \leq t \quad \forall v \in \mathbb{F}_2^s.$$

We define the function $\bar{h}_H : \mathbb{F}_2^* \longrightarrow \mathbb{F}_2^{n-k}$ the function mapping messages into syndromes associated to the parity-check matrix $H$ defined by the formula

$$m \longmapsto \bar{h}_H(m) = H\gamma_t(h(m))$$

where $h(\cdot)$ is any efficient function $\mathbb{F}_2^* \longrightarrow \mathbb{F}_2^s$. For simplicity of notation, we refer to $h$ as a hash function, even though in this case no security property on $h$ is required (although it would be a good practice to choose a cryptographically-secure hash function). With this definition, we can consider the following version of CFS, that we call $\widetilde{\mathrm{CFS}}$ and that can be considered the most generic case of a CFS-like digital signature scheme relying on a code-based management of the hashes.

**Algorithm 8** ($\widetilde{\mathrm{CFS}}$)

```
Parameters: $n, k, t \in \mathbb{N}$
1  $\boxed{\mathsf{KGen}_{\widetilde{\mathrm{CFS}}}(1^\lambda)}$: select $n, k$ and $t$ according to the security parameter $\lambda$, then randomly choose an $[n, k]$ code $C$ with parity-check matrix $H$ to be used in the CFS algorithm, i.e. $C$ must be a code for which there exists an efficient decoder $\delta_C$ able to correct up to $t$ errors; besides, $C$ should be such that any code equivalent to $C$ is indistinguishable from a random code
2  $M \leftarrow$ an $(n-k) \times (n-k)$ invertible matrix
3  $P \leftarrow$ an $n \times n$ random permutation matrix
4  $H_{pub} = MHP$
5  choose an efficient map $\gamma_t$ and a hash $h$
6  $\mathsf{pk} = (H_{pub}, t, h, \gamma_t)$
7  $\mathsf{sk} = (M, H, P, \delta_C)$
8  $\boxed{\mathsf{Sign}_{\widetilde{\mathrm{CFS}}}(m, \mathsf{sk})}$: given a message $m$
9  compute $d = \bar{h}_{H_{pub}}(m)$ according to the just presented definition of $\bar{h}_{H_{pub}}$
10 compute $s = M^{-1}d$
11 decode $s$ obtaining $e = \delta_C(s)$
12 $\sigma = P^\top e$
13 $\boxed{\mathsf{Verify}_{\widetilde{\mathrm{CFS}}}(\sigma, m, \mathsf{pk})}$: let $\sigma = u$; verify that $w_H(u) \leq t$ and then compute:
14 $a = \bar{h}_{H_{pub}}(m)$
15 $b = H_{pub}u$
16 $\sigma$ is valid $\iff a = b$
```

Once again, the correctness of the above signature scheme is straightforward and follows directly from the correctness of CFS. We remark that $\mathrm{mCFS}_c$ can be essentially obtained as a particular sub-case of $\widetilde{\mathrm{CFS}}$ by adopting particular instances of the involved mathematical tools. In particular:

- ◇ $C$ is a binary irreducible Goppa code;
- ◇ $S = I_{n-k}$ is the identity matrix of order $n-k$;

- ◇ $\gamma_t = \mathrm{reg}_{(n,w)}$ is the function defined on page 53;
- ◇ $h$ is the code-based hash function $h_{H_{pub}}$ stopped before the last application of $\mathrm{reg}_{(n,w)}$ and multiplication by $H_{pub}$ which we momentarily denote with $h^{stop}_{H_{pub}}$.

Indeed, with these choices, we have that

$$\bar{h}_{H_{pub}}(m) = H_{pub}\,\mathrm{reg}_{(n,w)}(h^{stop}_{H_{pub}}(m)) = h_{H_{pub}}(m).$$

Also observe that in mCFS$_c$ there are other marginal differences with respect to our generalisation, which however do not impact the main substance of the scheme we sketched above (e.g. computing $h_{H_{pub}}(h_{H_{pub}}(m) \,\|\, R)$ instead of $h_{H_{pub}}(m)$).

As in the case of mCFS$_c$, an analogous attack can be carried out also on this more general class of CFS-like digital signatures schemes. The following proposition provides a constructive demonstration of the attack.

**Proposition 3.3.1.** *Let* $(\mathsf{sk}, \mathsf{pk})$ *be the output of* $\mathsf{KGen}_{\widetilde{CFS}}(1^\lambda)$*. An attacker knowing the public key* $\mathsf{pk}$ *can forge a valid signature compatible with the private key* $\mathsf{sk}$ *for any message* $m$*.*

*Proof.* An attacker knowing the public parameters $(H_{pub}, t, h, \gamma_t)$ is able to forge a signature for any message $m$. In fact, instead of performing the steps of the signature algorithm on $m$, they first compute $x = \gamma_t(h(m))$ and then outputs $x$ as the signature of $m$. Indeed, $x$ is a valid error vector in $\mathbb{F}_2^n$ of Hamming weight at most $t$ (by definition of $\gamma_t$) whose syndrome w.r.t. the parity-check matrix $H_{pub}$ is $s = \bar{h}_{H_{pub}}(m)$. Therefore, any verifier obtains

$$H_{pub}x = H_{pub}\gamma_t(h(m)) = \bar{h}_{H_{pub}}(m)$$

and the signature results valid. □

We highlight the fact that an attacker does not need to know the private key to forge any signature and the fact that the number of operations performed to successfully obtain a forgery are even less than the number of operations performed by a honest user to obtain a valid signature. In conclusion, the essence of the vulnerability of the scheme is that, to obtain a decodable syndrome, we force the application of a function $\gamma_t$ to the output of the hash function *before* computing the syndrome. Even though this step allows to obtain a correctable syndrome without having to rely on the expensive trial-and-error routine of the signature steps of the original CFS protocol, the usage of such a map forces to *explicitly* determine that particular decodable error, compatible with the output syndrome, during the signing phase.

# Conclusions

In this thesis we have dealt with post-quantum code-based digital signatures. First of all, a preliminary chapter on the theoretical background, indispensable for a coherent and linear development of the discourse, has been included. Then, after having placed the subject within the modern cryptographic framework, we retraced the history of code-based schemes and delved into the two fundamental cornerstones that triggered this thread and by which most code-based scheme is inspired: CFS and KKS. In particular, beyond the presentation of the protocols themselves, our attempt was to identify the key points of both these DS schemes that should be treasured as pivotal (and basically mandatory) imitation cues when trying to construct new code-based hash-and-sign digital signature schemes. In addition, code-based hash functions have been introduced and their construction explained. Finally, a particular CFS-like scheme based on a Merkle-Damgård-style code-based hash function has been analyzed. We proved this approach to be unsuccessful since the protocol allows an attacker who does not know the private key to produce a valid signature for any message of their choice. Furthermore, we showed that a generalization of this approach remains insecure: a hash function that sends arbitrarily long binary strings into the set of decodable syndromes can be constructed and yet an attack on this new variation of the signature continues to persist. Therefore, other solutions should be found in order to preserve the original security of the scheme and simultaneously to reduce the computational effort used in the signing process as well. The design of a suitable code-based signature should take into account both the provable security of CFS and the efficiency of KKS.

It is clear that the significance of this work is quite humble and our results will not certainly change the history of code-based cryptography. Rather than that, our work is to be intended as a reflection on what *cannot* be done when devising a signature scheme that aims to be based on hashing towards decodable syndromes. As already remarked, this thesis stresses the main issues of designing a code-based digital signature scheme that is both efficient and secure. Throughout the whole dissertation the importance of SDP as a quantum-resistant computational problem has become evident as well as the consequent importance of the design of what we have referred to as the function $\chi$. In fact, this work has been first of all formative to take the first steps into post-quantum code-based cryptography and helpful to comprehend some of its core dynamics: a certain level of familiarity with such notions is necessary to first master them and to be then able to make a contribution. The study of the literature is the only way to get to grips with new difficult concepts.

In this regard, to provide some final remarks on possible future advances, a new potential way that could be explored is that of using the theory of Boolean functions to formulate a suitable $\chi$. In fact, remember that the function $\chi$ is nothing more than an application between $\mathbb{F}_2^n$ and $\mathbb{F}_2^m$ and can be hence seen as a vectorial Boolean function. In particular, Example 2.2 from Chapter 2 is emblematic in this respect because it brings out that $\chi$ has not necessarily to be linked to the code in an algebraic manner and thus enables one to proceed in the following way. One option could be to take one of the random enumerations describing $\chi$ and try to use some of the features of Boolean functions, like their ANF representation, to reduce the space needed to store it. Alternatively, the other possibility could be to start directly from the description of a random $\chi$ by means of a vector of ANF Boolean formulas and try to deduce sufficient conditions on these so that the image of the message space under $\chi$ is completely contained in the space of decodable syndromes. Actually, even necessary conditions would be useful, so as to shrink the set of functions among which to look for a good $\chi$. The door that leads to facing the function $\chi$ from this perspective is hence left open, despite the fact that these arguments are, at this stage, nothing more than unrefined speculations.

For what concerns writing choices, the guideline we decided to maintain was that of drawing a logical flow of notions and implications from the beginning to the end. In this way, it was possible to assemble a coherent discourse without losing the train of thoughts and to rationally get to the final parts of the thesis, where the most interesting aspects are presented. In fact, rather than adding extra details and insights useless for the discourse but helpful just to enrich the sections and enlarge the work, only the most important and spontaneous questions that might have arisen during the reading were comprehensively answered. To give an example, the Patterson algorithm was presented solely and exclusively to demonstrate that there is indeed an efficient algorithm for decoding Goppa codes, since this property is crucial in all our arguments. For anyone interested in the reasons for its working, we have left some hints of demonstrations as well as bibliographical references. It is obvious that the chain of logical redirections that could be created is very long and ramified and therefore it was considered more appropriate to break it off at the initial stages with concise but illuminating answers that could then be further explored by a direct consultation of the sources.

# Acknowledgements

Particular thanks go to my supervisor, Dr. Alessio Meneghetti, for the suggestion of this topic of post-quantum code-based cryptography and for the collaboration and help offered especially during the early (and as always more complicated) stages of the birth of this work. In fact, PQC is a relatively pioneering topic and there is hardly ever time to deal with it in master's degree courses, nor even to introduce it. Other special thanks are due to my co-supervisor Giuseppe D'Alconzo, for always having been available for clarifications, explanations, discussions and corrections. Finally, heartfelt thanks to all those who have been close to me over the years, supporting me in every way: if this has been possible, it is also thanks to you.

# Bibliography

[1] Michel Abdalla, Jee Hea An, Mihir Bellare, and Chanathip Namprempre. From identification to signatures via the fiat-shamir transform: Minimizing assumptions for security and forward-security. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 418–433. Springer, 2002.

[2] Carlos Aguilar, Philippe Gaborit, and Julien Schrek. A new zero-knowledge code based identification scheme with reduced communication. In *2011 IEEE Information Theory Workshop*, pages 648–652. IEEE, 2011.

[3] Daniel Apon, Ray Perlner, Angela Robinson, and Paolo Santini. Cryptanalysis of ledacrypt. In *Annual International Cryptology Conference*, pages 389–418. Springer, 2020.

[4] Nicolas Aragon, Olivier Blazy, Philippe Gaborit, Adrien Hauteville, and Gilles Zémor. Durandal: a rank metric based signature scheme. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 728–758. Springer, 2019.

[5] Daniel Augot, Matthieu Finiasz, and Nicolas Sendrier. A fast provably secure cryptographic hash function. *Cryptology ePrint Archive*, 2003.

[6] Daniel Augot, Matthieu Finiasz, and Nicolas Sendrier. A family of fast syndrome based cryptographic hash functions. In *International Conference on Cryptology in Malaysia*, pages 64–83. Springer, 2005.

[7] Eric Bach. *Discrete logarithms and factoring.* University of California at Berkeley, 1984.

[8] Marco Baldi, Alessandro Barenghi, Franco Chiaraluce, Gerardo Pelosi, and Paolo Santini. Ledacrypt: Qc-ldpc code-based cryptosystems with bounded decryption failure rate. In *Code-Based Cryptography Workshop*, pages 11–43. Springer, 2019.

[9] Marco Baldi, Marco Bianchi, Franco Chiaraluce, Joachim Rosenthal, and Davide Schipani. Using ldgm codes and sparse syndromes to achieve digital signatures. In *International Workshop on Post-Quantum Cryptography*, pages 1–15. Springer, 2013.

[10] Marco Baldi, Marco Bodrato, and Franco Chiaraluce. A new analysis of the mceliece cryptosystem based on qc-ldpc codes. In *International Conference on Security and Cryptography for Networks*, pages 246–262. Springer, 2008.

[11] Alessandro Barenghi, Jean-François Biasse, Edoardo Persichetti, and Paolo Santini. Less-fm: fine-tuning signatures from the code equivalence problem. In *International Conference on Post-Quantum Cryptography*, pages 23–43. Springer, 2021.

[12] Georg Becker. Merkle signature schemes, merkle trees and their cryptanalysis. *Ruhr-University Bochum, Tech. Rep*, 2008.

[13] Emanuele Bellini, Florian Caullery, Philippe Gaborit, Marc Manzano, and Victor Mateu. Improved veron identification and signature schemes in the rank metric. In *2019 IEEE International Symposium on Information Theory (ISIT)*, pages 1872–1876. IEEE, 2019.

[14] E. Berlekamp, R. McEliece, and H. van Tilborg. On the inherent intractability of certain coding problems (corresp.). *IEEE Transactions on Information Theory*, 24(3):384–386, 1978.

[15] Daniel J Bernstein. Introduction to post-quantum cryptography. In *Post-quantum cryptography*, pages 23–32. Springer, 2009.

[16] Daniel J Bernstein, Andreas Hülsing, Stefan Kölbl, Ruben Niederhagen, Joost Rijneveld, and Peter Schwabe. The sphincs+ signature framework. In *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, pages 2129–2146, 2019.

[17] Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Crystals-kyber: a cca-secure module-lattice-based kem. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 353–367. IEEE, 2018.

[18] Pierre-Louis Cayrel, Ayoub Otmani, and Damien Vergnaud. On kabatianskii-krouk-smeets signatures. In *International Workshop on the Arithmetic of Finite Fields*, pages 237–251. Springer, 2007.

[19] Pierre-Louis Cayrel, Pascal Véron, and Sidi Mohamed El Yousfi Alaoui. A zero-knowledge identification scheme based on the q-ary syndrome decoding problem. In *International Workshop on Selected Areas in Cryptography*, pages 171–186. Springer, 2010.

[20] Lindsay N Childs. *A concrete introduction to higher algebra.* Springer, 2009.

[21] Tung Chou, Ruben Niederhagen, Edoardo Persichetti, Tovohery Hajatiana Randrianarisoa, Krijn Reijnders, Simona Samardjiska, and Monika Trimoska. Take your meds: Digital signatures from matrix code equivalence. *Cryptology ePrint Archive*, 2022.

[22] Nicolas T Courtois, Matthieu Finiasz, and Nicolas Sendrier. How to achieve a mceliece-based digital signature scheme. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 157–174. Springer, 2001.

[23] Joan Daemen and Vincent Rijmen. Aes proposal: Rijndael. 1999.

[24] Giuseppe D'Alconzo, Alessio Meneghetti, and Paolo Piasenti. Security issues of cfs-like digital signature algorithms. *arXiv preprint arXiv:2112.00429*, 2021.

[25] Léonard Dallot. Towards a concrete security proof of courtois, finiasz and sendrier signature scheme. In *Western European Workshop on Research in Cryptology*, pages 65–77. Springer, 2007.

[26] Ivan Bjerre Damgård. A design principle for hash functions. In *Conference on the Theory and Application of Cryptology*, pages 416–427. Springer, 1989.

[27] Luca De Feo. Mathematics of isogeny based cryptography. *arXiv preprint arXiv:1711.04062*, 12, 2017.

[28] Thomas Debris-Alazard, Nicolas Sendrier, and Jean-Pierre Tillich. Wave: A new code-based signature scheme. 2018.

[29] Léo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Crystals-dilithium: A lattice-based digital signature scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 238–268, 2018.

[30] Jean-Charles Faugere, Valérie Gauthier-Umana, Ayoub Otmani, Ludovic Perret, and Jean-Pierre Tillich. A distinguisher for high-rate mceliece cryptosystems. *IEEE Transactions on Information Theory*, 59(10):6830–6844, 2013.

[31] Jean-Charles Faugère, Valérie Gauthier-Umaña, Ayoub Otmani, Ludovic Perret, and Jean-Pierre Tillich. A distinguisher for high-rate mceliece cryptosystems. *IEEE Transactions on Information Theory*, 59(10):6830–6844, 2013.

[32] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the theory and application of cryptographic techniques*, pages 186–194. Springer, 1986.

[33] Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. Falcon: Fast-fourier lattice-based compact signatures over ntru. *Submission to the NIST's post-quantum cryptography standardization process*, 36(5), 2018.

[34] Ernst M Gabidulin, AV Paramonov, and OV Tretjakov. Ideals over a non-commutative ring and their application in cryptology. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 482–489. Springer, 1991.

[35] Philippe Gaborit, Gaétan Murat, Olivier Ruatta, and Gilles Zémor. Low rank parity check codes and their application to cryptography. In *Proceedings of the Workshop on Coding and Cryptography WCC*, volume 2013, 2013.

[36] Robert Gallager. Low-density parity-check codes. *IRE Transactions on information theory*, 8(1):21–28, 1962.

[37] Daniel Gottesman, H-K Lo, Norbert Lutkenhaus, and John Preskill. Security of quantum key distribution with imperfect devices. In *Information Theory, 2004. ISIT 2004. Proceedings. International Symposium on*, page 136. IEEE, 2004.

[38] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219, 1996.

[39] Jeffrey Hoffstein, Jill Pipher, and Joseph H Silverman. Ntru: A ring-based public key cryptosystem. In *International Algorithmic Number Theory Symposium*, pages 267–288. Springer, 1998.

[40] Ellen Jochemsz. Goppa codes & the mceliece cryptosystem. *Vrije Universiteit Amsterdam*, pages 3–4, 2002.

[41] Gregory Kabatianskii, Evgenii Krouk, and Ben Smeets. A digital signature scheme based on random error-correcting codes. In *IMA International Conference on Cryptography and Coding*, pages 161–167. Springer, 1997.

[42] Cameron F Kerry and Patrick D Gallagher. Digital signature standard (dss). *FIPS PUB*, pages 186–4, 2013.

[43] Terry Shue Chien Lau and Chik How Tan. Murave: A new rank code-based signature with multiple rank verification. In *Code-Based Cryptography Workshop*, pages 94–116. Springer, 2020.

[44] Zhe Li, Chaoping Xing, and Sze Ling Yeo. A new code based signature scheme without trapdoors. *Cryptology ePrint Archive*, 2020.

[45] San Ling and Chaoping Xing. *Coding theory: a first course*. Cambridge University Press, 2004.

[46] Vadim Lyubashevsky. Fiat-shamir with aborts: Applications to lattice and factoring-based signatures. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 598–616. Springer, 2009.

[47] Vadim Lyubashevsky. Lattice signatures without trapdoors. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 738–755. Springer, 2012.

[48] Florence Jessie MacWilliams and Neil James Alexander Sloane. *The theory of error correcting codes*, volume 16. Elsevier, 1977.

[49] Michiel Marcus, T Lange, and P Schwabe. White paper on mceliece with binary goppa codes, 2019.

[50] Ayoub Otmani and Jean-Pierre Tillich. An efficient attack on all concrete kks proposals. In *International Workshop on Post-Quantum Cryptography*, pages 98–116. Springer, 2011.

[51] N. Patterson. The algebraic decoding of goppa codes. *IEEE Transactions on Information Theory*, 21(2):203–207, 1975.

[52] Edoardo Persichetti. Efficient one-time signatures from quasi-cyclic codes: A full treatment. *Cryptography*, 2(4):30, 2018.

[53] Fang Ren, Dong Zheng, WeiJing Wang, et al. An efficient code based digital signature algorithm. *Int. J. Netw. Secur.*, 19(6):1072–1079, 2017.

[54] Paolo Santini, Massimo Battaglioni, Franco Chiaraluce, and Marco Baldi. Analysis of reaction and timing attacks against cryptosystems based on sparse parity-check codes. In *Code-Based Cryptography Workshop*, pages 115–136. Springer, 2019.

[55] Claude Elwood Shannon. A mathematical theory of communication. *Bell system technical journal*, 27(3):379–423, 1948.

[56] Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999.

[57] Vladimir Michilovich Sidelnikov and Sergey O Shestakov. On insecurity of cryptosystems based on generalized reed-solomon codes. 1992.

[58] Jacques Stern. A new identification scheme based on syndrome decoding. In *Annual International Cryptology Conference*, pages 13–21. Springer, 1993.

[59] R. Tanner. A recursive approach to low complexity codes. *IEEE Transactions on Information Theory*, 27(5):533–547, 1981.

[60] A. Vardy. The intractability of computing the minimum distance of a code. *IEEE Transactions on Information Theory*, 43(6):1757–1766, 1997.

[61] Pascal Véron. Improved identification schemes based on error-correcting codes. *Applicable Algebra in Engineering, Communication and Computing*, 8(1):57–69, 1997.

*Fire Walk With Me*