Post-quantum cryptography: a study of the decoding of QC-MDPC codes

Valentin Vasseur

Université de Paris

Inria

Université de Paris

École doctorale Informatique, Télécommunications et Électronique de Paris

Inria équipe-projet COSMIQ

Post-quantum cryptography: a study of the decoding of QC-MDPC codes

Cryptographie post-quantique : étude du décodage des codes QC-MDPC

Par **Valentin Vasseur**

Thèse de doctorat d'informatique

Dirigée par Nicolas Sendrier

Présentée et soutenue publiquement le 29 mars 2021

Devant un jury composé de :

Nicolas Sendrier	Inria	Directeur
Jérôme Lacan	ISAE-Supaero	Rapporteur
Pierre Loidreau	Université de Rennes	Rapporteur
Alain Couvreur	Inria	Examinateur
Caroline Fontaine	ENS Paris-Saclay	Examinatrice
Philippe Gaborit	Université de Limoges	Examinateur
Sophie Laplante	Université de Paris	Examinatrice
Jean-Pierre Tillich	Inria	Examinateur

Équipe-projet COSMIQ
Inria,
2 rue Simone Iff,
75 012 Paris

Remerciements

Je dois d'abord remercier mon directeur de thèse, Nicolas, de m'avoir proposé de travailler sur cet algorithme si simple à décrire, mais qui a conduit à tant de questionnements et qui m'a permis de découvrir un domaine si riche. Merci pour ta disponibilité, tes explications, tes conseils et ton humour.

Je remercie Jérôme Lacan et Pierre Loidreau d'avoir accepté d'être rapporteurs. Je remercie également Alain Couvreur, Caroline Fontaine, Philippe Gaborit, Sophie Laplante et Jean-Pierre Tillich d'avoir accepté de faire partie du jury.

Il règne toujours une très bonne ambiance au sein de l'équipe SECRET/COSMIQ, merci donc à Anne, André, Pascale, Gaëtan, Anthony, María, Léo, Nicolas, Jean-Pierre et Christelle. Bien évidemment, je remercie aussi ceux qui étaient de passage dans l'équipe et ceux qui sont de passage aujourd'hui (en espérant n'oublier personne, la liste est longue) : Magali, Augustin, Yann, Xavier, Christina, Clémence, Pierre, Rémi, Étienne, Rodolfo, Kevin, Kaushik, Julia, Daniel, Nicolas, Thomas, Loïc, Aurélie, Sébastien, Simona, Antonio, Paul, Antoine, Lucien, Vivien, Johana, Rocco, Andrea, Clara, Yann, André et Christophe. Je pourrais même ajouter Étienne, Thierry et Bernard qui font presque partie de l'équipe. Et bien sûr, je n'oublie pas les membres du bureau Tapdance canal historique Mathilde, Matthieu et Ferdinand qui par leur bonne humeur ont contribué à ce que ce soit toujours un plaisir de venir au bureau.

Je remercie toutes les personnes que j'ai rencontrées au cours de ces dernières années et avec lesquelles je partage de bons souvenirs. Merci donc tout d'abord à Moran, Sonia, Lolo, Cyril, Céline, Benjamin, Clément, Marion et Vincent puis Caro, Cindy, Nico et Quentin et enfin Pascal, Pauline et Louis.

Je tiens à remercier Daniel Agier qui a réussi à me passionner pour les mathématiques dès le lycée.

Pour finir, merci à ma famille proche qui me tolère depuis toutes ces années. Merci à Mumu sans qui je serais probablement à la rue en ce moment ou pire, en banlieue. Merci à mes parents qui m'ont permis de faire ces études si longues mais passionnantes. Les contraintes du calendrier font que je soutiens ma thèse le jour d'un bien triste anniversaire, mais je sais que tu aurais été fière que ton fils soit docteur.

Contents

Introduction

Most cryptosystems in use today rely on the hardness of number theory problems such as the factorization or the discrete logarithm problems, either in finite fields or in elliptic curves. These problems with the current parameters would not resist an adversary with a sufficiently powerful quantum computer [Sho99; Joz01]. In fact the security of these problems would scale poorly against such an adversary, see for example [BHLV17].

In anticipation of the development of a powerful quantum computer, alternatives to cryptosystems based on number theory are gaining momentum in research. The National Institute of Standards and Technology (NIST) launched in 2014 a process[1] to standardize the first post-quantum public key encryption (PKE), key encapsulation mechanisms (KEM) and signatures. Initially, 82 proposals were submitted. At the time of writing, we are in the 3rd round and there are 7 finalists in the running, most likely candidates for standardization in the short term, and 8 alternate candidates, who may need another round of evaluation. The breakdown by domain of the remaining proposals is presented in Table 1.

Table 1: NIST Post-Quantum Cryptography Standardization Process — Round 3

	PKE/KEM	Signature
Finalists		
Code	1	0
Lattice	3	2
Multivariate	0	1
Alternate candidates		
Code	2	0
Lattice	2	0
Hash	0	1
Isogeny	1	0
Multivariate	0	1
Zero-knoweldge	0	1

We can see that the proposed cryptosystems are based on problems that cover a wide range of mathematical fields. Let us focus on code-based cryptography. The only code-based finalist in the NIST standardization process is based on [McE78], published in 1978 in which McEliece introduced the idea of using error-correcting codes in cryptography.

Error correcting codes are extensively used in telecommunications. To transmit a message, it is first transformed into a codeword, a process that adds redundancy. If the codeword is transmitted over a noisy channel, it will contain errors when it reaches its recipient. Redundancy implies that we are still able, as long as the number of errors is limited, to recover the original message from this noisy codeword. A good error-correcting

[1] https://csrc.nist.gov/projects/post-quantum-cryptography

code has a certain structure that makes it possible to have an efficient decoder, that is to say, an algorithm that removes many errors in a timely manner. On the other hand, a random code is hard to decode.

McEliece had the idea of hiding a message by taking the corresponding codeword and adding as many errors as it is possible to remove. He used Goppa codes. On the one hand, they have good decoders, which allows decryption. And on the other hand, once scrambled, these codes are hard to distinguish from random codes. So anyone who does not know how the code was scrambled will not have a good decoder. In other words, we have a public-key encryption system: if the scrambled code is made public, anyone can use it to encrypt, but only the person who knows how it was scrambled can decrypt.

It is remarkable that, so far, this system has not been affected by any major attack, classical or quantum, and that it is now being considered for standardization. However, the public key size is in the order of a few megabits. This can be a hindrance for some applications. In order to reduce the key size, one can therefore consider using codes other than Goppa codes. Many attempts have been made in this direction and most of them have resulted in broken cryptosystems.

The Quasi-Cyclic Moderate Density Parity Check (QC-MDPC) codes, proposed in [MTSB13], appear to be good candidate for replacing Goppa codes in a McEliece cryptosystem. They have small key sizes and have not suffered major attacks. However, unlike Goppa codes, the principle of their decoder does not depend on algebraic properties but on probabilistic properties. Consequently, decoding may fail.

It has been shown in [GJS16] that the decoder leaks information about the secret key in case of failures. In fact, when the same key is used, an adversary who encrypts a large number of messages and has the ability to determine which ones failed to decrypt, can retrieve the secret key from the set of these failure-triggering messages. Thus, to use keys that are not ephemeral but static, it is necessary to ensure that the decoding failure rate (DFR) is very low. This is also a necessary condition to be able to verify strong security constraints such as indistinguishability under chosen ciphertext attack (IND-CCA).

QC-MDPC decoders and their DFR are the primary focus of the works presented in this document. Our motivation is threefold: to improve decoding algorithms, to better understand their workings and to ensure that they fail with negligible probability. We will focus on [BIKE] a Key Encapsulation Mechanism that uses QC-MDPC codes. It is an alternate candidate to the NIST Post-Quantum Cryptography Standardization Process.

NIST has expressed concerns about its IND-CCA security and DFR analysis in [Ala+20], but nonetheless considers it one of the most promising candidates. This document is a step toward addressing these concerns.

This document is divided in four parts.

The first part recalls the necessary background on coding theory, public key cryptography, security reductions, code-based cryptography and the specificities of the QC-MDPC based scheme BIKE.

The second part presents new decoding algorithms and reviews some aspects of their implementation. The performance and tuning of these algorithms will be discussed from an essentially empirical point of view.

The third part describes two theoretical probabilistic models for some MDPC decoders, the ultimate goal being to predict their DFR.

The fourth part introduces a new decoding assumption and the statistical framework it implies for extrapolating the DFR from simulation measurements. We then study specific parity check matrices or error patterns that, due to the structural properties of QC-MDPC codes, are great candidates to challenge this new assumption.

Publications

[BIKE] Carlos Aguilar Melchor, Nicolas Aragon, Paulo S L M Barreto, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Ghosh Santosh, Shay Gueron, Tim Güneysu, Rafael Misoczki, Edoardo Persichetti, Nicolas Sendrier, Jean-Pierre Tillich, Valentin Vasseur, and Gilles Zémor. *BIKE*. NIST Round 3 submission for Post-Quantum Cryptography. Aug. 2020. URL: https://bikesuite.org.

[SV19] Nicolas Sendrier and Valentin Vasseur. "On the Decoding Failure Rate of QC-MDPC Bit-Flipping Decoders". In: *Post-Quantum Cryptography (PQCrypto)*. Ed. by Jintai Ding and Rainer Steinwandt. Vol. 11505. LNCS. Chongqing, China: Springer, May 2019, pp. 404–416. DOI: 10.1007/978-3-030-25510-7_22.

[SV20a] Nicolas Sendrier and Valentin Vasseur. "About Low DFR for QC-MDPC Decoding". In: *Post-Quantum Cryptography (PQCrypto)*. Ed. by Jintai Ding and Jean-Pierre Tillich. Vol. 12100. LNCS. Paris, France: Springer, Apr. 2020, pp. 20–34. DOI: 10.1007/978-3-030-44223-1_2.

[SV20b] Nicolas Sendrier and Valentin Vasseur. *On the existence of weak keys for QCMDPC decoding*. Cryptology ePrint Archive, Report 2020/1232. 2020. URL: https://eprint.iacr.org/2020/1232.

Notations

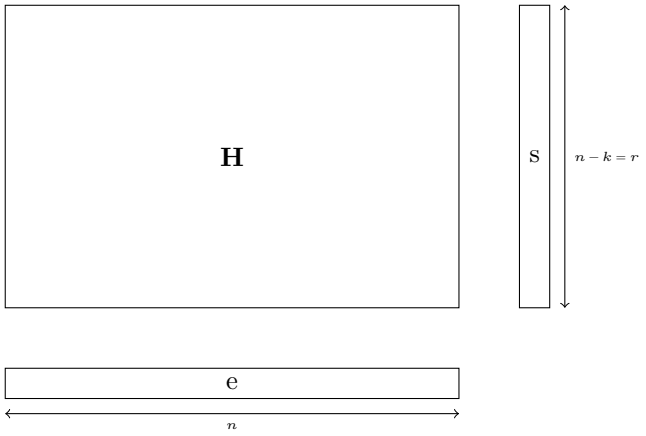
General.

- Vectors and polynomials are denoted in roman type (e.g. h) and matrices in bold (e.g. $\mathbf{H}$).
- The support $\mathrm{Supp}(\mathrm{v})$ of a vector v is the set of indices of its nonzero entries. [Def. 1.15 p. 11]
- The weight $|\mathrm{v}|$ of a vector v is always the Hamming weight. [Def. 1.16 p. 11]
- The vector operator $\star$ is the Schur product, both for vectors and polynomials. [Def. 1.24 p. 12]
- The notation $x \xleftarrow{\$} S$ means drawing uniformly at random an element from S and assign it to x.

Coding theory.

- The following variables are restricted to the following usage unless stated otherwise:
  - n: the length of a code;
  - k: the dimension of a code;
  - $r = n - k$: the dimension of the dual; the block size of a quasi-cyclic parity check matrix;
  - d: the column weight of a parity check matrix;
  - w: the row weight of a parity check matrix;
  - t: the weight of an error pattern.
- For all $j \in \{0, \dots, n-1\}$, $\sigma_j := \left|\mathrm{h}_j \star \mathrm{s}\right|$. [Def. 5.1 p. 31]
- For a matrix $\mathbf{H}$, we denote its i-th row by $\mathbf{h}_i^\top$, and its j-th column by $\mathbf{h}_j$.
- We will represent information vectors m, error patterns e and codewords c as row vectors; and the syndrome s as a column vector:

$$\mathbf{G} \in \mathbb{F}_q^{k \times n}\,, \qquad \mathbf{H} \in \mathbb{F}_q^{(n-k) \times n}\,, \qquad \mathrm{c} = \mathrm{m}\mathbf{G}\,, \qquad \mathrm{s} = \mathbf{H}\mathrm{e}^\top\,.$$


H
S
$n - k = r$
e
n

Part I

Preliminaries

Chapter 1

Coding theory

In this chapter we will recall the fundamental notions from coding theory that we will need to construct QC-MDPC codes.

1.1 Linear codes

Definition 1.1. Let k and n be two positive integers with $k \leq n$. An $\mathbb{F}_q$-*linear code* $\mathcal{C}$ of *length* n and *dimension* k is a linear subspace of dimension k of the vector space $\mathbb{F}_q^n$.

Such a code is designated as an $[n, k]$-code.

Definition 1.2. The *rate* of a code $\mathcal{C}$ of dimension k and length n is the ratio

$$R = \frac{k}{n}\,.$$

In the context of telecommunications, when using an $[n, k]$-linear code to send information, for every k symbols of useful information, $(n - k)$ redundant symbols are also sent. The code rate is therefore the proportion of useful information that is sent through a channel. The reduntant information is used for error-detection and correction.

Definition 1.3. A matrix $\mathbf{G} \in \mathbb{F}_q^{k\times n}$ is said to be a *generator matrix* of the linear code $\mathcal{C}$ if its rows form a basis of $\mathcal{C}$.

Definition 1.4. The dual code $\mathcal{C}^\perp \subset \mathbb{F}_q^n$ of a code $\mathcal{C}$ is the orthogonal space

$$\mathcal{C}^\perp := \{\mathrm{v} \in \mathbb{F}_q^n \mid \forall \mathrm{w} \in \mathcal{C}, \mathrm{v} \cdot \mathrm{w} = 0\}$$

where $\mathrm{v} \cdot \mathrm{w}$ is the scalar product $\sum_{i=0}^{n-1} v_i w_i$.

A generator matrix for the dual code $\mathcal{C}^\perp$ is called a *parity check matrix* for $\mathcal{C}$.

We will call *equations* the rows of parity check matrix.

Definition 1.5. The generator matrix $\mathbf{G}$ (resp. the parity check matrix $\mathbf{H}$) of a linear code $\mathcal{C}$ (of dimension k and length n) is said to be in *systematic form* if is written as

$$\mathbf{G} = [\mathbf{I}_k \quad P] \qquad \text{resp.} \qquad \mathbf{H} = [\mathbf{I}_{n-k} \quad P]\,.$$

Remark 1.6. A code is fully defined by its dual, therefore if $\mathbf{H} \in \mathbb{F}_q^{(n-k)\times n}$ is a parity check matrix for an $[n, k]$-code $\mathcal{C}$ then

$$\mathcal{C} = \{\mathrm{c} \in \mathbb{F}_q^n \mid \mathbf{H}\mathrm{c}^\top = 0\}\,.$$

Remark 1.7. If $\mathbf{G} \in \mathbb{F}_q^{k\times n}$ and $\mathbf{H} \in \mathbb{F}_q^{(n-k)\times n}$ are respectively a generator and a parity check matrices of a code $\mathcal{C}$ then we have

$$\mathbf{H}\mathbf{G}^\top = \mathbf{0}\,.$$

Definition 1.8. Let $\mathcal{C}$ be a code of length n and dimension k with parity check matrix $\mathbf{H} \in \mathbb{F}_q^{(n-k)\times n}$. The *syndrome* of a vector $\mathrm{x} \in \mathbb{F}_q^n$ is the vector

$$\mathbf{H}\mathrm{x}^\top \in \mathbb{F}_q^{n-k}\,.$$

Definition 1.9. Let $\mathcal{C}$ be a binary code of length n and dimension k with parity check matrix $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}$. We say that the i-th equation is *satisfied* if the corresponding syndrome bit s_i is zero, and we say that the equation is *unsatisfied* otherwise.

1.2 Decoding

Definition 1.10. Let $\mathcal{C}$ be a code of length n and dimension k with generator matrix $\mathbf{G} \in \mathbb{F}_q^{k\times n}$ and parity check matrix $\mathbf{H} \in \mathbb{F}_q^{(n-k)\times n}$.

Let $\mathrm{x} \in \mathbb{F}_q^n$, *decoding* is the process of finding a vector $\mathrm{e} \in \mathbb{F}_q^n$ such that

$$\mathrm{x} - \mathrm{e} \in \mathcal{C} \qquad \textit{i.e.} \qquad \exists \mathrm{m} \in \mathbb{F}_q^k, \mathrm{x} = \mathrm{m}\mathbf{G} + \mathrm{e}\,.$$

Let $\mathrm{s} \in \mathbb{F}_q^{n-k}$, *syndrome-decoding* is the process of finding a vector $\mathrm{e} \in \mathbb{F}_q^n$ such that

$$\mathbf{H}\mathrm{e}^\top = \mathrm{s}\,.$$

Remark 1.11. There exists different flavors of decoders, for example:

- a minimum distance decoder minimizes the Hamming weight of the vector e,
- a maximum likelihood decoder finds m that maximizes $\Pr[\mathrm{x} \text{ received} \,|\, \mathrm{m} \text{ sent}]$ for a specific channel.

Remark 1.12. In code-based cryptography we rely on the *(syndrome) decoding problem* in which the challenge is to decode a given instance (see §3.3).

Definition 1.13. The *Tanner graph* of a binary parity check matrix $\mathbf{H}$ is the bipartite graph defined by the biadjacency matrix $\mathbf{H}$.

If $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}$, in the Tanner graph, the n columns correspond to n nodes called the *variable nodes*, and the $(n-k)$ rows correspond to $(n-k)$ nodes called the *check nodes*.

While the use of the graph structure was already somehow practiced by Gallager in [Gal63], the notion was named after Tanner because of his work on the construction of long codes from smaller codes in [Tan81].

Related to the probalities computation in the Tanner graph, the following important classic result on the parity of a sum of binary random variables will be needed in this document (see [Gal63, Lemma 4.1] for a proof).

Proposition 1.14. *Let k be a positive integer, $p_1, \ldots, p_k \in [0,1]$ and $X_1, \ldots, X_k$ be k independent binary random variables following Bernoulli distributions with probabilities respectively $p_1, \ldots, p_k$. Then*

$$\forall b \in \{0,1\}, \Pr[X_1 + \cdots + X_k \bmod 2 = b] = \frac{1 + (-1)^b \prod_{\ell=1}^{k}(1-2p_\ell)}{2}\,.$$

1.3 Minimum distance & Gilbert-Varshamov distance

Definition 1.15. The *support* of a vector $\mathrm{v} \in \mathbb{F}_q^n$ is the set of indices of its nonzero entries:

$$\mathrm{Supp}(\mathrm{v}) := \left\{ i \in \{0, \dots, n-1\} \,\middle|\, v_i \neq 0 \right\} .$$

We use the same notation for polynomials

$$\mathrm{Supp}\left(\sum_{i=0}^{n-1} v_i x^i \right) := \left\{ i \in \{0, \dots, n-1\} \,\middle|\, v_i \neq 0 \right\} .$$

Definition 1.16. The *Hamming weight* $|\mathrm{v}|$ of a vector (or a polynomial) v is the number of its nonzero entries

$$|\mathrm{v}| := \left|\mathrm{Supp}(\mathrm{v})\right| .$$

The *Hamming distance* $d(\mathrm{a}, \mathrm{b})$ between two vectors a and b is the Hamming weight of the difference between a and b:

$$d(\mathrm{a}, \mathrm{b}) := |\mathrm{a} - \mathrm{b}| .$$

Definition 1.17. The *minimum distance* of a code $\mathcal{C}$ is the minimum distance between two distinct codewords:

$$\min_{\substack{\mathrm{c}_0, \mathrm{c}_1 \in \mathcal{C} \\ \mathrm{c}_0 \neq \mathrm{c}_1}} d(\mathrm{c}_0, \mathrm{c}_1) .$$

Remark 1.18. For a linear code, we have

$$\min_{\substack{\mathrm{c}_0, \mathrm{c}_1 \in \mathcal{C} \\ \mathrm{c}_0 \neq \mathrm{c}_1}} d(\mathrm{c}_0, \mathrm{c}_1) = \min_{\substack{\mathrm{c} \in \mathcal{C} \\ \mathrm{c} \neq 0}} |\mathrm{c}| .$$

Definition 1.19. Let q, n and k be integers. The *Gilbert-Varshamov distance* $d_{GV}(q, n, k)$ is the smallest integer d such that

$$\sum_{i=0}^{d-1} \binom{n}{i} (q-1)^i \geq q^{n-k} .$$

Remark 1.20. Let q, n and k be integers. For an $[n, k]$-code drawn uniformly at random, the expected number of possible solutions of weight below or equal to w to a decoding problem is

$$\frac{\sum_{i=0}^{w} \binom{n}{i} (q-1)^i}{q^{n-k}} .$$

So, on average, there is one solution to a decoding problem with weight $d_{GV}(q, n, k)$.

Definition 1.21. The *binary entropy* function is:

$$\mathrm{H}(X) := \mathrm{H}(p) = -p \log_2 p - (1-p) \log_2 (1-p) .$$

1.4 Regularity

Definition 1.22. Let l and r be two positive integers. An (l, r)-*regular* code is a code such that, in its Tanner graph, every variable node has degree l and every check node has degree r. The integers l and r are respectively the *left* and *right* degrees.

Equivalently if $\mathbf{H}$ is a parity check matrix, the code it defines is (l, r)-regular if all its column have the same weight l and all its rows have the same weight r.

1.5 Channel

In telecommunications, when it comes to error correction, the model of the channel used is often specified. In cryptography, error patterns are often drawn uniformly at random among all fixed weight patterns. If a pattern has a weight t and a length n, it can be seen, as a first approximation, as coming from a binary symmetric channel with a crossover probability t/n.

Definition 1.23. Let x be the binary random variable transmitted over a *binary symmetric channel* of *crossover probability* ϵ, let us write the received binary random variable y. Then we have

$$\Pr\left[x = 0 \middle| y = 0\right] = \Pr\left[x = 1 \middle| y = 1\right] = 1 - \epsilon\,,$$
$$\Pr\left[x = 0 \middle| y = 1\right] = \Pr\left[x = 1 \middle| y = 0\right] = \epsilon\,.$$

1.6 Schur product

Definition 1.24. The *Schur product* of two vectors $\mathrm{v}, \mathrm{w} \in \mathbb{F}_q^n$ is the componentwise product

$$\mathrm{v} \star \mathrm{w} := (v_0 \cdot w_0, v_1 \cdot w_1, \ldots, v_{n-1} \cdot w_{n-1})\,.$$

We use the same notation for polynomials

$$\left(\sum_{i=0}^{n-1} v_i x^i\right) \star \left(\sum_{i=0}^{n-1} w_i x^i\right) := \sum_{i=0}^{n-1} (v_i \cdot w_i) x^i\,.$$

Proposition 1.25. *The Schur product is distributive over the addition:*

$$\mathrm{u} \star (\mathrm{v} + \mathrm{w}) = \mathrm{u} \star \mathrm{v} + \mathrm{u} \star \mathrm{w}\,.$$

Proposition 1.26. *For any integer $n > 0$, let $v, w \in \mathbb{F}_2^n$, then*

$$|\mathrm{v} + \mathrm{w}| = |\mathrm{v}| + |\mathrm{w}| - 2\,|\mathrm{v} \star \mathrm{w}|\ .$$

Proof. $\mathrm{v} + \mathrm{v} \star \mathrm{w}$ and $\mathrm{w} + \mathrm{v} \star \mathrm{w}$ have disjoint support and weight respectively $|\mathrm{v}| - |\mathrm{v} \star \mathrm{w}|$ and $|\mathrm{w}| - |\mathrm{v} \star \mathrm{w}|$,

$$|\mathrm{v} + \mathrm{w}| = \big|(\mathrm{v} + \mathrm{v} \star \mathrm{w}) + (\mathrm{w} + \mathrm{v} \star \mathrm{w})\big|\ .$$

□

Corollary 1.27. *Let k be a positive integer and let $\mathrm{v}_1, \ldots, \mathrm{v}_k$ be k vectors of equal length, then*

$$\left|\sum_{i=1}^{k} \mathrm{v}_i\right| = \sum_{d=1}^{k} (-2)^{d-1} \sum_{1 \le i_1 < \cdots < i_d \le k} \left|\mathrm{v}_{i_1} \star \cdots \star \mathrm{v}_{i_d}\right|\ .$$

Proof. Proposition 1.26 shows the case where $k = 2$. Now, let k be any integer greater than 2,

$$\left|\sum_{i=1}^{k} \mathrm{v}_i\right| = \left|\left(\sum_{i=1}^{k-1} \mathrm{v}_i\right) + v_k\right| = \left|\sum_{i=1}^{k-1} \mathrm{v}_i\right| + |v_k| - 2\left|\sum_{i=1}^{k-1} (\mathrm{v}_i \star \mathrm{v}_k)\right|\ .$$

Then, by induction hypothesis, we have

$$\left|\sum_{i=1}^{k} \mathrm{v}_i\right| = \sum_{d=1}^{k-1}(-2)^{d-1} \sum_{1\le i_1<\cdots<i_{d-1}\le k-1} \left|\mathrm{v}_{i_1} \star \cdots \star \mathrm{v}_{i_{d-1}}\right| + \left|v_k\right| - 2\sum_{d=1}^{k-1}(-2)^{d-1} \sum_{1\le i_1<\cdots<i_{d-1}\le k-1} \left|\mathrm{v}_{i_1} \star \cdots \star \mathrm{v}_{i_{d-1}} \star \mathrm{v}_k\right| .$$

□

1.7 (QC-)MDPC codes and basic properties

Starting from this point in this document, we will only focus on binary codes (i.e. $\mathbb{F}_2$-linear code).

Definition 1.28 (Low Density Parity Check)**.** An $[n, k]$ *LDPC code* is a linear code for which there exists a sparse parity check matrix, by which we mean that the Hamming weight of its rows is in $\mathrm{O}(1)$.

Definition 1.29 (Moderate Density Parity Check)**.** An $[n, k]$ *MDPC code* is a linear code for which there exists a moderately sparse parity check matrix, by which we mean that the Hamming weight of its rows is in $\mathrm{O}(\sqrt{n})$.

The two definitions are very much alike. The former codes have great decoding performance but the existence of low weight codewords in their dual is detrimental for its security in a McEliece cryptosystem as was shown in [MRA00]. The latter codes correct this shortcoming at the cost of lower decoding performance.

The Tanner graph is a useful tool to understand and analyze the LDPC decoders. These decoders usually belong to the category of algorithms called message-passing algorithms or the one called bit-flipping algorithms (an overview of these methods will be done in §6.1.1).

While relying on strong security reductions, MDPC-based cryptosystem would require large key sizes. Following a movement initiated by [NTRU] and which has influenced many code- or lattice-based cryptosystems, a great reduction of the key sizes is obtained using an $\mathbb{F}_2[x]/(x^n - 1)$ type ring. In coding theory, this means using quasi-cyclic codes that we define now.

Definition 1.30. A *circulant matrix* is a matrix where each row is rotated one element to the right relative to the preceding row.

Let r be a positive integer. Any $r \times r$ binary circulant matrix $\mathbf{H}$ can be written as

$$\mathbf{H} = \begin{pmatrix} h_0 & h_{r-1} & \ldots & h_2 & h_1 \\ h_1 & h_0 & h_{r-1} & & h_2 \\ \vdots & h_1 & h_0 & \ddots & \vdots \\ h_{r-2} & & \ddots & \ddots & h_{r-1} \\ h_{r-1} & h_{r-2} & \ldots & h_1 & h_0 \end{pmatrix}$$

where $h_0, \ldots, h_{r-1} \in \mathbb{F}_2$.

Proposition 1.31. *The application*

$$\begin{pmatrix} h_0 & h_{r-1} & \ldots & h_2 & h_1 \\ h_1 & h_0 & h_{r-1} & & h_2 \\ \vdots & h_1 & h_0 & \ddots & \vdots \\ h_{r-2} & & \ddots & \ddots & h_{r-1} \\ h_{r-1} & h_{r-2} & \ldots & h_1 & h_0 \end{pmatrix} \mapsto h_0 + h_1 x + \cdots + h_{r-2}x^{r-2} + h_{r-1}x^{r-1}$$

is an isomorphism between the ring of $r \times r$ circulant matrices with coefficients in $\mathbb{F}_2$ and the quotient ring $\mathbb{F}_2[x]/(x^r - 1)$.

Definition 1.32. Let n_0, r, and d be three positive integers. A QC-MDPC code is a code whose parity check matrix

$$\mathbf{H} = \begin{pmatrix} \mathbf{H}_0 & \dots & \mathbf{H}_{n_0-1} \end{pmatrix}$$

consists of n_0 circulant blocks $\mathbf{H}_0, \dots, \mathbf{H}_{n_0-1}$ of size $r \times r$ and row weight d such that $n_0 d = O(\sqrt{n_0 r})$. The code is an $[n_0 r, (n_0-1)r]$ code, it has rate $1 - 1/n_0$.

We refer to r as the *block size*.

Remark 1.33. We can equivalently consider the parity check matrix as a tuple of elements of $\mathbb{F}_2[x]/(x^r-1)$: $(\mathrm{h}_0, \dots, \mathrm{h}_{n_0-1})$.

Similarly to Proposition 1.31, any vector $\mathrm{e} \in \mathbb{F}_2^r$ has an associated polynomial in $\mathbb{F}_2[x]/(x^r-1)$ with the application

$$\begin{pmatrix} e_0 \\ e_1 \\ \vdots \\ e_{r-2} \\ e_{r-1} \end{pmatrix} \mapsto e_0 + e_1 x + \cdots + e_{r-2} x^{r-2} + e_{r-1} x^{r-1} \,.$$

We write n_0 circulant matrices of size $r \times r$ as $\mathbf{H}_0, \dots, \mathbf{H}_{n_0-1}$, an error pattern as $\underline{\mathrm{e}} \in \mathbb{F}_2^{n_0 r}$, and the corresponding syndrome $\underline{\mathrm{s}} \in \mathbb{F}_2^r$:

$$\begin{pmatrix} \mathbf{H}_0 & \dots & \mathbf{H}_{n_0-1} \end{pmatrix} \underline{\mathrm{e}} = \underline{\mathrm{s}} \,.$$

If we write $\mathrm{h}_0, \dots, \mathrm{h}_{n_0-1}$, e, and s their respective associated polynomials in $\mathbb{F}_2[x]/(x^r-1)$, then we have

$$\mathrm{h}_0 \mathrm{e} + \cdots + \mathrm{h}_{n_0-1} \mathrm{e} = \mathrm{s} \,.$$

Although it may offer a little flexibility in a tradeoff between key generation time, decoding time and bandwidth usage, in this document we will not maintain the genericity of the term n_0. We will therefore only focus on double circulant matrices, which will give codes with a rate of $1/2$ *i.e.* $n_0 = 2$. It is the setting of [BIKE], a key encapsulation mechanism[1] based on QC-MDPC codes.

[1] This term is defined in the next chapter.

Chapter 2

Security reduction

Definition 2.1. A *Public Key Encryption* (PKE) scheme is a triple of probabilistic polynomial-time algorithms (KeyGen, Encrypt, Decrypt) of:

- a key generation method that outputs a key pair $(\mathrm{pk}, \mathrm{sk})$;

$$\mathrm{KeyGen}\colon \{0,1\}^\lambda \to \mathcal{K}_{\mathrm{pub}} \times \mathcal{K}_{\mathrm{priv}}\,;$$

- an encryption method that takes a public key and a plaintext m as inputs and outputs the ciphertext c;

$$\mathrm{Encrypt}\colon \mathcal{K}_{\mathrm{pub}} \times \mathcal{M} \to \mathcal{C}\,;$$

- a decryption method that takes a private key and a ciphertext c as inputs and outputs the plaintext m or a failure $\perp$;

$$\mathrm{Decrypt}\colon \mathcal{K}_{\mathrm{priv}} \times \mathcal{C} \to \mathcal{M}\,;$$

where λ is the security parameter, $\mathcal{K}_{\mathrm{pub}}$ is the public key space, $\mathcal{K}_{\mathrm{priv}}$ is the private key space, $\mathcal{M} \ni \perp$ is the message space and $\mathcal{C}$ is the ciphertext space.

Many modern practical usages of cryptography use hybrid cryptosystems (see [SSH; TLS] for examples): they combine a public key cryptosystem with a symmetric key cryptosystem. The former is convenient since it allows establishing a secure channel without the need to pre-share keys and the latter is usually efficient and thus allows a high throughput.

A *Key Encapsulation Mechanism* provides a way for two parties to exchange a common session key, thus solving the first part, establishing a secure channel.

Definition 2.2. A *Key Encapsulation Mechanism* (KEM) is a triple of probabilistic polynomial-time algorithms (KeyGen, Encaps, Decaps) of:

- a key generation method that outputs a key pair $(\mathrm{pk}, \mathrm{sk})$;

$$\mathrm{KeyGen}\colon \{0,1\}^\lambda \to \mathcal{K}_{\mathrm{pub}} \times \mathcal{K}_{\mathrm{priv}}\,;$$

- an encapsulation method that takes a public key as input and outputs a message (typically a session key) m and its encapsulation c;

$$\mathrm{Encaps}\colon \mathcal{K}_{\mathrm{pub}} \to \mathcal{M} \times \mathcal{C}\,;$$

- a decapsulation method that takes a private key and an encapsulated message c as inputs and outputs the message m or a failure $\perp$;

$$\text{Decaps}\colon \mathcal{K}_{\text{priv}} \times \mathcal{C} \to \mathcal{M}\,;$$

where λ is the security parameter, $\mathcal{K}_{\text{pub}}$ is the public key space, $\mathcal{K}_{\text{priv}}$ is the private key space, $\mathcal{M} \ni \perp$ is the message space and $\mathcal{C}$ is the ciphertext space.

2.1 Security games

For a cryptosystem, a reduction of security is a demonstration that if it is broken then some hard problems are broken as well. An important desirable property for a scheme to guarantee the confidentiality of communications is the ciphertext *indistinguishability*.

Indistinguishability is a property that depends on a "game" where an adversary provides two plaintexts and is provided the two corresponding ciphertexts without knowing which is which. If the adversary is not able to match each ciphertext with the right plaintext better than simply choosing randomly then the cryptosystem has the indistinguishability property. Different flavors of this property exist such as the *indistinguishability under chosen plaintext attack* (IND-CPA) and *indistinguishability under chosen ciphertext attack* (IND-CCA).

In Figure 2.1, we give the definitions of the IND-CPA game and the IND-CCA game. In the IND-CPA game, the adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ first chooses two messages $m_0 \neq m_1$. One of the two messages is picked uniformly at random then encrypted and the adversary has to guess which one it was. In the IND-CCA game, the adversary has access to a decryption oracle that can be used for any ciphertext other than the challenge.

Definition 2.3. The *advantage* of an adversary $\mathcal{A}$ to a game G in a cryptosystem K is a measure of the successfullness of the adversary to the game, it is the difference between its probability of winning the game and the probability of winning the game by taking random choices:

$$\text{Adv}_K^G(\mathcal{A}) := \left| \Pr\left[\mathcal{A}\left(G\right) = 1\right] - \frac{1}{2} \right| .$$

2.2 Fujisaki-Okamoto transform

Fujisaki and Okamoto proposed, in [FO99], a hybrid encryption scheme combining a one-way secure scheme and a symmetric encryption primitive that is IND-CCA secure. This was later restated in a more modern way by Dent in [Den03]. Finally, a "modular" analysis was presented in [HHK17], it provides tighter security reductions and takes into account decryption errors (through a notion called δ-correctness). Security reductions are made using the classic *game hopping* technique in the *Random Oracle Model*[1] (ROM).

Post-quantum cryptography community has shown interest in the transform as it is used in most lattice- and code-based submissions to the NIST standardization process.

We focus on the $\mathsf{FO}^{\not\perp}$ transformation, a way to transform a PKE into a KEM with implicit rejection. If the original PKE is IND-CPA secure and δ-correct, it is shown in [HHK17] that the resulting KEM is IND-CCA secure under certain conditions fairly easily obtained.

It requires two hash functions K and H. The former should have outputs in $\mathcal{M}$ and is used to compute the session key. The latter is necessary for a process called

[1] Using the Random Oracle Model means here that the hash functions are to be treated as black boxes which, for any input request, produce a uniformly randomly chosen element of its codomain. They are mathematical functions, so any repetition of an input will always produce the same output.

IND-CPA game for a PKE	IND-CCA game for a KEM
$(\mathtt{pk}, \mathtt{sk}) \leftarrow \mathrm{KeyGen}(\mathtt{rand})$; $b \xleftarrow{\$} \{0,1\}$; $(m_0, m_1, \mathrm{state}) \leftarrow \mathcal{A}_1(\mathtt{pk})$; $c \leftarrow \mathrm{Encrypt}(\mathtt{pk}, m_b)$; $b' \leftarrow \mathcal{A}_2(\mathtt{pk}, c, \mathrm{state})$; **return** $[b' = b]$;	$(\mathtt{pk}, \mathtt{sk}) \leftarrow \mathrm{KeyGen}(\mathtt{rand})$; $b \xleftarrow{\$} \{0,1\}$; $(m, c_0) \leftarrow \mathrm{Encaps}(\mathtt{pk})$; $c_1 \xleftarrow{\$} \mathcal{C}$; $b' \leftarrow \mathcal{A}^{\mathcal{O}^{\mathtt{sk},c}_{\mathrm{dec}}}(m, c_b)$; **return** $[b' = b]$;

IND-CCA game for a PKE	$\mathcal{O}^{\mathtt{sk},c}_{\mathrm{dec}}(c')$

$(\mathtt{pk}, \mathtt{sk}) \leftarrow \mathrm{KeyGen}(\mathtt{rand})$;
$b \xleftarrow{\$} \{0,1\}$;
$(m_0, m_1, \mathrm{state}) \leftarrow \mathcal{A}_1(\mathtt{pk})$;
$c \leftarrow \mathrm{Encrypt}(\mathtt{pk}, m_b)$;
$b' \leftarrow \mathcal{A}_2^{\mathcal{O}^{\mathtt{sk},c}_{\mathrm{dec}}}(\mathtt{pk}, c, \mathrm{state})$;
return $[b' = b]$;
	if $c' = c$ **then**
	  return $\perp$;
	else
	  return $\mathrm{Decrypt}(\mathtt{sk}, c')$;

Figure 2.1: Security games

"derandomization" that transforms a probabilistic scheme into a deterministic one. It should have outputs in $\mathcal{M}$.

Let $\mathrm{PKE}_0 = (\mathrm{KeyGen}_0, \mathrm{Encrypt}_0, \mathrm{Decrypt}_0)$ be a probabilistic PKE. We write $\mathrm{PKE}_1 = (\mathrm{KeyGen}_1, \mathrm{Encrypt}_1, \mathrm{Decrypt}_1)$ the derandomization of PKE_0. It is defined as:

- $\mathrm{KeyGen}_1(\mathtt{seed}) := \mathrm{KeyGen}_0(\mathtt{seed})$,
- $\mathrm{Encrypt}_1(\mathtt{pk}, m) := \mathrm{Encrypt}_0(\mathtt{pk}, m \| \mathrm{H}(m))$,
- $\mathrm{Decrypt}_1(\mathtt{sk}, c) := \{$ $m \| h \leftarrow \mathrm{Decrypt}_0(\mathtt{sk}, c)$; **if** $m \| h \neq \perp$ **and** $h = \mathrm{H}(m)$ **then return** m; **otherwise return** $\perp$; $\}$.

Finally, we obtain a KEM $(\mathrm{KeyGen}, \mathrm{Encaps}, \mathrm{Decaps})$ with

- $\mathrm{KeyGen}(\mathtt{seed}) := \mathrm{KeyGen}_1(\mathtt{seed})$,
- $\mathrm{Encaps}(\mathtt{pk}) := \{$ $m \xleftarrow{\$} \mathcal{M}$; $c \leftarrow \mathrm{Encrypt}_1(\mathtt{pk}, m)$; **return** $(\mathrm{K}(m, c), c)$; $\}$,
- $\mathrm{Decaps}(\mathtt{sk}, c) := \{$ $m \leftarrow \mathrm{Decrypt}_1(\mathtt{sk}, c)$; **if** $m \neq \perp$ **then return** $\mathrm{K}(m, c)$; **otherwise return** $\mathrm{K}(\mathtt{seed}(\mathtt{sk}), c)$; $\}$.

Definition 2.4. A PKE $(\mathrm{KeyGen}, \mathrm{Encrypt}, \mathrm{Decrypt})$ is said to be δ-correct if

$$\mathrm{E}_{(\mathtt{pk},\mathtt{sk}) \leftarrow \mathrm{KeyGen}(\{0,1\}^\lambda)} \left[\max_{m \in \mathcal{M}} \Pr\left[\mathrm{Decrypt}(\mathtt{sk}, c) \neq m \,\middle|\, c \leftarrow \mathrm{Encrypt}(\mathtt{pk}, m) \right] \right] \leq \delta\,.$$

Similarly, a KEM $(\mathrm{KeyGen}, \mathrm{Encaps}, \mathrm{Decaps})$ is said to be δ-correct if

$$\Pr\left[\mathrm{Decaps}(\mathtt{sk}, c) \neq K \,\middle|\, (\mathtt{pk}, \mathtt{sk}) \leftarrow \mathrm{KeyGen}(\{0,1\}^\lambda); (K, c) \leftarrow \mathrm{Encaps}(\mathtt{pk}) \right] \leq \delta\,.$$

The following results can be deduced from [HHK17] and allows the construction of an IND-CCA secure KEM from an IND-CPA secure, δ-correct PKE.

Theorem 2.5 (Theorem 3.2 & Theorem 3.4 in [HHK17]). *If* PKE_1 *is* δ*-correct, then for any IND-CCA adversary* $\mathcal{B}$ *against* KEM*, issuing at most* q_{K} *and respectively* q_{H} *queries to the random oracle* K *and respectively* H *there exists an IND-CPA adversary* $\mathcal{A}$ *against* PKE_1 *running in about the same time as* $\mathcal{B}$ *such that*

$$\mathrm{Adv}^{\mathrm{IND-CCA}}_{\mathrm{KEM}}(\mathcal{B}) \leq q_{\mathrm{H}} \cdot \delta + \frac{2q_{\mathrm{H}} + q_{\mathrm{K}} + 1}{|\mathcal{M}|} + 3\mathrm{Adv}^{\mathrm{IND-CPA}}_{\mathrm{PKE}_1}(\mathcal{A}) \,.$$

Chapter 3

Code-based cryptography

As of today, there are two general approaches to implementing a PKE or a KEM using error correcting codes: (i) those based on Alekhnovich's cryptosystem [Ale03], (ii) those based on McEliece [McE78] or Niederreiter [Nie86] cryptosystems.

The former approach has the advantage of not relying on any trapdoor in the code and its security reduces only to the syndrome decoding problem. An interesting quasi-cyclic variation of this system is [HQC].

In this document, we focus on variants of the latter. For this, the private code must be a code for which there is a trapdoor, namely the ability to decode it efficiently. The problematics (efficiency and security) implied by this requirement is a vast subject and constitutes the main interest of this document, in the specific case of MDPC codes. The primary subject of this document, [BIKE], is a Niederreiter variant.

3.1 McEliece cryptosystem framework

The basic framework of a McEliece cryptosystem is based on the possibility of building a public generator matrix for which encoding is easy but decoding is difficult, and a trapdoor that makes decoding up to a distance t tractable. This is usually done by (i) generating a code for which a good decoder is known, (ii) scrambling it. The scrambled matrix is the public key, and the trapdoor is anything that allows us to come back to the setting of the original code.

Therefore, this framework requires

- a code generation method that outputs a pair $(\mathbf{G}, T)$

$$\text{CodeGen} : \{0,1\}^\lambda \to \mathbb{F}_q^{k\times n} \times \mathcal{T} ;$$

- a decoder that takes the generator matrix $\mathbf{G}$, the trapdoor T and a noisy codeword $\mathrm{y} = \mathrm{c} + \mathrm{e}$ and outputs (with high probability) the codeword c if $|\mathrm{e}|$ is smaller than some constant t

$$\text{Decode} : \mathbb{F}_q^{k\times n} \times \mathcal{T} \times \mathbb{F}_q^n \to \mathbb{F}_q^n ;$$

where $\mathcal{T}$ is the set of trapdoors.

From this we can build a PKE by taking[1] :

[1] We use the inverse $\text{Inv}_{\mathbf{G}}$ of the isomorphism $\mathbb{F}_q^k \to \mathcal{C}$. It can be computed by selecting k positions for which the corresponding columns in $\mathbf{G}$ are linearly indepedent, then extracting the corresponding coordinates (resp. columns) from c (resp. $\mathbf{G}$). If we call the resulting vector c' and the corresponding matrix $\mathbf{G}'$, this inverse is simply $\mathrm{c}'\mathbf{G}'^{-1}$.

- KeyGen(seed) := CodeGen(seed),
- Encrypt($\mathbf{G}$, m) := { e $\xleftarrow{\$}$ $\mathcal{E}_{n,t}$; return m$\mathbf{G}$ + e; },
- Decrypt(($\mathbf{G}, T$), y) := { c ← Decode($\mathbf{G}, T$, y); return $\text{Inv}_{\mathbf{G}}$(c); }.

Here, we denote by $\mathcal{E}_{n,t}$ the set of vectors of $\mathbb{F}_2^n$ with Hamming weight t.

Remark 3.1. The encryption relies on encoding the actual message m, another possibility is to encode the message as the error vector e and choose m ramdomly. The catch with this method is that the error vector must have a Hamming weight at most t. As explained in [Sch72], in the binary case, there exists a simple bijection ϕ between

$$\mathcal{E}_{n,t} := \left\{ \mathrm{e} \in \mathbb{F}_2^n \;\middle|\; |\mathrm{e}| = t \right\} \quad \text{and} \quad \left\{ 0, \dots, \binom{n}{t} - 1 \right\}.$$

Note that $\binom{n}{t} \approx 2^{n\,\mathrm{H}(t/n)}$.

Let $\mathrm{x} \in \mathcal{E}_{n,t}$ and let us write $\{i_1, \dots, i_t\}$ its support with $i_1 < i_2 < \cdots < i_t$. Define $\phi(\mathrm{x}) := \sum_{j=1}^{t} \binom{i_j - 1}{j}$. Its inverse ϕ^{-1} can be computed in polynomial time using an algorithm somewhat similar to a number base conversion algorithm.

3.2 Niederreiter cryptosystem framework

The Niederreiter cryptosystem [Nie86] is dual to the McEliece one. Its framework requires

- a code generation method that outputs a pair $(\mathbf{H}, T)$

$$\text{CodeGen}^{\perp} : \{0,1\}^{\lambda} \to \mathbb{F}_q^{(n-k)\times n} \times \mathcal{T}\,;$$

- a decoder that takes the parity check matrix $\mathbf{H}$, the trapdoor T and a syndrome $\mathrm{y} = \mathbf{H}\mathrm{e}^{\top}$ and outputs the error vector e if |e| is smaller than some constant t

$$\text{Decode}^{\perp} : \mathbb{F}_q^{(n-k)\times n} \times \mathcal{T} \times \mathbb{F}_q^{n-k} \to \mathbb{F}_q^{n}\,;$$

where $\mathcal{T}$ is the set of trapdoors.

From this we can build a PKE by taking:

- KeyGen(seed) := $\text{CodeGen}^{\perp}$(seed),
- Encrypt($\mathbf{H}$, m) := { e ← ϕ^{-1}(m); return $\mathbf{H}\mathrm{e}^{\top}$; },
- Decrypt(($\mathbf{H}, T$), y) := { e ← $\text{Decode}^{\perp}$($\mathbf{H}, T$, y); return ϕ(e); }.

In terms of security, the two systems are equivalent, this was proven in [LDW94]. The Niederreiter setting has a clear advantage in terms of communication bandwith as the ciphertexts have length $(n-k)$ rather than n. Besides that, the difference between the two frameworks will strongly depend on the code actually used to implement it.

3.3 Best known attacks on underlying hard problems

The security of code-based cryptosystem relates to the two following generic problems of decoding linear codes. They were proven to be NP-complete in [BMT78].

Problem 1. *Syndrome Decoding* – $\mathrm{SD}(n, k, t)$
Instance: *A parity check matrix* $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}$*, a syndrome* $\mathrm{s} \in \mathbb{F}_2^{n-k}$*, a target weight* $t > 0$.
Property: *There exists* $\mathrm{e} \in \mathbb{F}_2^n$ *such that* $|\mathrm{e}| = t$ *and* $\mathbf{H}\mathrm{e}^\top = \mathrm{s}$.

Problem 2. *Codeword Finding* – $\mathrm{CF}(n, k, w)$
Instance: *A parity check matrix* $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}$*, a target weight* $w > 0$.
Property: *There exists* $\mathrm{e} \in \mathbb{F}_2^n$ *such that* $|\mathrm{e}| = w$ *and* $\mathbf{H}\mathrm{e}^\top = 0$.

Figure 3.1: Hard problems in code-based cryptography.

Definition 3.2. For any fixed values of n, k and t, we denote $\mathcal{WF}_{\mathcal{A}}(n, k, t)$ the *work factor*, *i.e.* the average cost in binary operations, of algorithm $\mathcal{A}$ to produce a solution to the computational syndrome decoding problem $\mathrm{SD}(n, k, t)$.

Definition 3.3. For a linear code of length n and dimension k, an *information set* is a set $\mathcal{J}$ of k coordinates such that for any vector $(b_j)_{j\in\mathcal{J}}$ there exists a unique codeword c such that $c_j = b_j$ for $j \in \mathcal{J}$.

The best known algorithms for these two problems are variants of the Information Set Decoding (ISD) algorithm defined by Prange in 1962 [Pra62]. This algorithm repeatedly attemps to find an error-free information set. The framework for ISD algorithms is summarized in Figure 3.2.

In the (original) Prange's version, the full row reduction is performed and the idea is to repeat the process until the vector $\mathbf{U}\mathrm{s}$ has a Hamming weight below some target weight w. When it is the case, we can define

$$\mathrm{e} := \begin{pmatrix} (\mathbf{U}\mathrm{s})^\top & \mathbf{0}_k \end{pmatrix} P^\top,$$

then we have $\mathbf{H}\mathrm{e}^\top = \mathrm{s}$ and $|\mathbf{e}| = |\mathbf{U}\mathrm{s}| \leq w$.

An improvement of this algorithm due to Lee and Brickell in [LB88] is to introduce a search step to amortize the cost of the Gaussian elimination. This step consists, for a (small) positive integer p, in searching for a sum of p columns of $\mathbf{H}'$ that is at a distance of $\mathbf{U}\mathrm{s}$ below $w - p$. Since then, this basic technique has received a few improvements. The first idea was to only perform a partial Gaussian elimination to transform the searching step into the smaller problem of finding $\mathrm{e}'' \in \mathbb{F}_2^{k+\ell}$ of weight p such that

$$\mathbf{H}''\mathrm{e}''^\top = \mathbf{U}''\mathrm{s}$$

and then verify that $\mathbf{H}'\mathrm{e}''^\top$ is at distance below $w - p$ of $\mathbf{U}'\mathrm{s}$. We can then complete the error pattern by choosing

$$\mathrm{e} := \begin{pmatrix} \left(\mathbf{H}'\mathrm{e}''^\top + \mathbf{U}'\mathrm{s}\right)^\top & \mathrm{e}'' \end{pmatrix} P^\top$$

The step of finding e'' can be seen as a collision search on which an interesting gain is obtained thanks to the birthday paradox, see [Ste88; Dum91]. Then the algorithm was improved by introducing the representation technique in [MMT11; BJMM12]. Another variant uses nearest neighbours [MO15].

Remark 3.4. As presented here, the algorithm is used to do syndrome decoding, but it can be used to search for low weight codewords as well. To do so, simply consider $\mathrm{s} = 0$ and $p > 0$ in any variant that has a search step (*i.e.* not the original algorithm due to Prange).

Input: Parity check matrix $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}$, syndrome $\mathrm{s} \in \mathbb{F}_2^{(n-k)}$.

Output: An error vector $\mathrm{e} \in \mathbb{F}_2^n$ such that $\mathbf{H}\mathrm{e}^\top = \mathrm{s}$.

- Consider the augmented matrix
$$\begin{pmatrix}\mathbf{H} & \mathrm{s}\end{pmatrix}.$$
- Apply a random permutation $\mathbf{P} \in \mathbb{F}_2^{n\times n}$ on the columns of $\mathbf{H}$
$$\begin{pmatrix}\mathbf{HP} & \mathrm{s}\end{pmatrix}.$$
- Perform a row reduction, multiplying by a matrix $\mathbf{U} \in \mathbb{F}_2^{(n-k)\times(n-k)}$
$$\begin{pmatrix}\mathbf{UHP} & \mathbf{U}\mathrm{s}\end{pmatrix}.$$

with a full row reduction:
$$\begin{pmatrix}\mathbf{1}_{n-k} & \mathbf{H}' & \mathbf{U}\mathrm{s}\end{pmatrix}.$$

with a partial row reduction, for $\ell < n-k$:
$$\begin{pmatrix}\mathbf{1}_{n-k-\ell} & \mathbf{H}' & \mathbf{U}'\mathrm{s} \\ \mathbf{0}_{\ell\times(n-k-\ell)} & \mathbf{H}'' & \mathbf{U}''\mathrm{s}\end{pmatrix}.$$
where
$$\mathbf{U} = \begin{pmatrix}\mathbf{U}' \\ \mathbf{U}''\end{pmatrix}$$
and $\mathbf{U}' \in \mathbb{F}_2^{(n-k-\ell)\times(n-k)}, \mathbf{U}'' \in \mathbb{F}_2^{\ell\times(n-k)}$.

Figure 3.2: One iteration of Information Set Decoding principle

Therefore, any of these algorithms can be used to solve either $\mathrm{SD}(n,k,w)$ or $\mathrm{CF}(n,k,w)$ for some parameters n,k,w. If we write one of these algorithms as $\mathcal{A}$ then both problems have the same workfactor
$$\mathcal{WF}_{\mathcal{A}}(n,k,w).$$

Let us only consider the Prange algorithm to solve $\mathrm{SD}(n,k,t)$ with $t = o(n)$ for now. The dominating factor in the work factor for this algorithm is the average number of trials before finding an error-free information set. It is equal to $\frac{\binom{n}{t}}{\binom{n-k}{t}}$. We have
$$\frac{1}{t}\log_2\left(\frac{\binom{n}{t}}{\binom{n-k}{t}}\right) \sim \frac{1}{t}\log_2\left(\frac{n^t}{(n-k)^t}\right) = c \qquad \text{with } c = -\log_2\left(1-\frac{k}{n}\right)$$
as n goes to infinity, using Stirling's formula. So the work factor is $2^{ct(1+o(1))}$.

For the other variants known at the moment of writing this document, the following result from [CS15] shows that asymptotically, when $t = o(n)$, the work factor is the same.

Proposition 3.5. *[CS15] Let k and t be two functions of n such that $\lim_{n\to\infty} k/n = R$, $0 < R < 1$, and $\lim_{n\to\infty} t/n = 0$. For any algorithm $\mathcal{A}$ among the variants of [Pra62; Ste88; Dum91; BJMM12; MMT11; MO15], we have*
$$\mathcal{WF}_{\mathcal{A}}(n,k,t) = 2^{ct(1+o(1))}, \qquad c = -\log_2(1-R)$$
when n tends to infinity.

The quasi-cyclic structure of a code means that any blockwise circular shift of a codeword is also a codeword and that, given an error pattern, any circular shift of its syndrome corresponds to a blockwise circular shift of the same error pattern. It has been shown in [Sen11] that one can exploit this structure and expect a polynomial gain in the work factor of an ISD algorithm. In practice the work factor for finding low weight codeword in the dual is divided by $(n-k)$ and the work factor for decoding is divided by $\sqrt{n-k}$.

So, the work factor to solve the quasi-cyclic $\mathrm{SD}(n,k,t)$ problem is

$$\mathcal{WF}_{\mathrm{QCSD}}(n,k,t) := \frac{\mathcal{WF}_{\mathcal{A}}(n,k,t)}{\sqrt{n-k}} = 2^{c[1/2+t(1+o(1))]-\log_2(n)/2}$$

and to solve the quasi-cyclic $\mathrm{CF}(n,k,w)$ problem is

$$\mathcal{WF}_{\mathrm{QCCF}}(n,k,w) := \frac{\mathcal{WF}_{\mathcal{A}}(n,k,w)}{n-k} = 2^{c[1+w(1+o(1))]-\log_2(n)}\,.$$

Chapter 4

BIKE

[BIKE] uses QC-MPDC codes in a Niederreiter cryptosystem. Its characteristics are particularly relevant to the NIST Post Quantum Cryptography standardization process[1]. It has (i) small keys, and (ii) efficient, low-complexity algorithms. It is therefore suitable for both software and hardware implementations. The former category received optimization attention using the specialized instructions of modern x86 processors, see [DG19; DGK20a]. The latter has also been studied for microcontrollers, see [HMG13; MG14], as well as Field-Programmable Gate Array (FPGA), see [HMG13; MOG15b; HC17; HWCW19; RMG20]. In these works, improvements or tradeoffs are obtained by focusing on the polynomial multiplication needed for encapsulation and decapsulation, and on the polynomial inversion for the key generation.

Table 4.1: BIKE parameters.

λ	w	t	r^{CPA}	r^{CCA}	ℓ
128	142	134	10 163	12 323	256
192	206	199	19 853	24 659	256
256	274	264	32 749	40 973	256

Let us fix the following parameters:

- r: the block size of a parity check matrix,
- $n = 2r$: the length of a code,
- d: the column weight of a parity check matrix,
- $w = 2d$: the row weight of a parity check matrix,
- t: the weight of an error pattern,
- ℓ: the message space size,
- λ: the security parameter, in bits.

We define the following sets:

- $\mathcal{R}_r \simeq \mathbb{F}_2[x]/(x^r - 1)$: the set of $r \times r$ circulant matrices,
- $\mathcal{R}_{r,d}$: the set of $r \times r$ circulant matrices of row weight d,
- $\mathcal{E}_{n,t}$: the set of vectors of $\mathbb{F}_2^n$ with Hamming weight t.

Table 4.2: BIKE specification.

- KeyGen(`seed`) := {
  $(h_0, h_1) \xleftarrow{\$} \mathcal{R}_{r,d}$;
  $h \leftarrow h_1 h_0^{-1}$;
  $\sigma \xleftarrow{\$} \{0,1\}^\ell$;
  return $(h_0, h_1, \sigma), h$; },
- Encaps(h) := {
  $m \xleftarrow{\$} \{0,1\}^\ell$;
  $(e_0, e_1) \leftarrow H(m)$;
  $c \leftarrow (e_0 + e_1 h, m \oplus L(e_0, e_1))$;
  $K \leftarrow K(m, c)$;
  return K, c; },
- Decaps($(h_0, h_1, \sigma), c$) := {
  $e' \leftarrow \text{Decode}^{\perp}_{\text{MDPC}}(c_0 h_0, h_0, h_1)$;
  $m' \leftarrow c_1 \oplus L(e')$;
  if $e' \neq H(m')$ **then return** $K(\sigma, c)$;
  otherwise return $K(m', c)$; }.

Where H, K and L are three hash functions. H has outputs in $\mathcal{E}_{n,t}$, K and L in $\{0,1\}^\ell$.

BIKE is defined[2] in Table 4.2.

This entire document is devoted to the decoder, so we will not detail its operation in this preliminary chapter. We will however explain the conversion used and what it implies for security, in particular the necessary conditions imposed on the decoder. We will also detail why the block size should be chosen carefully.

4.1 Security

Using notations from §3.2 we can build $\text{PKE}_0 = (\text{KeyGen}_0, \text{Encrypt}_0, \text{Decrypt}_0)$, a Niederreiter cryptosystem using QC-MDPC codes by taking

- $\text{CodeGen}^{\perp}$(`seed`) := {
  $\mathbf{H}_0 \xleftarrow{\$} \mathcal{R}_{r,d}$;
  $\mathbf{H}_1 \xleftarrow{\$} \mathcal{R}_{r,d}$;
  $\mathbf{H} \leftarrow \begin{pmatrix} \mathbf{1}_r & \mathbf{H}_0^{-1}\mathbf{H}_1 \end{pmatrix}$;
  return $(\mathbf{H}, \mathbf{H}_0)$; },
- $\text{Decode}^{\perp}(\mathbf{H}, \mathbf{H}_0, s)$:= {
  $s' \leftarrow \mathbf{H}_0 s$;
  return $\text{Decode}^{\perp}_{MDPC}((\mathbf{H}_0 \quad \mathbf{H}_0\mathbf{H}), s')$; }.

with $\mathcal{K}_{\text{pub}} = \mathcal{R}_r^2$, $\mathcal{K}_{\text{priv}} = \mathcal{R}_{r,d}$, $\mathcal{M} = \mathcal{E}_{n,t}$, $\mathcal{C} = \mathbb{F}_2^r$.

However, applying Niederreiter construction directly in this PKE is not secure. Indeed, the problem is to leave the complete control of the error pattern to the person who encrypts. In the hand of a malicious adversary, this could be used in a reaction attack such as [GJS16] where particular error patterns allow them to gather information about the private key.

Even outside practical concerns, this issue is already taken into account in the analysis of [HHK17]. Indeed, if we want to use the Theorem 2.5 to prove that the scheme is IND-CCA secure, we need to make sure that the PKE is δ-correct with δ really small. Definition 2.4 of δ-correctness concerns the maximum probability of failure on all possible

[1] https://csrc.nist.gov/projects/post-quantum-cryptography

[2] The notation $\xleftarrow{\$}$ means drawing uniformly at random from the right side and assign it to the left side using `seed`.

messages, and it is difficult to prove that no message will be decoded with less success than the average. It is in fact not true, the question of particular problematic error patterns will be discussed in Chapter 16 on the phenomenon known as the "error floor".

To circumvent this problem, BIKE adds randomization to the error pattern. Instead of choosing a random message m and using it directly as an error pattern to compute the syndrome, the message m is first hashed and the hash H(m) is used as the error pattern. The soundness of such a process is guaranteed by a commitment: $\mathrm{m} \oplus \mathrm{L}(\mathrm{H}(\mathrm{m}))$.

Using the more compact polynomial representation, we obtain the specification in Table 4.2.

To apply Theorem 2.5, we have to study the IND-CPA security of the scheme first. It relies on the quasi-cyclic versions of the problems $\mathrm{SD}(n,k,t)$ and $\mathrm{CF}(n,k,w)$. We define the function:

- $\mathrm{QCSD}_{r,t}\left(\left(\mathrm{e}_0,\mathrm{e}_1\right),\mathrm{h},\mathrm{s}\right)$ for $\mathrm{e}_0,\mathrm{e}_1,\mathrm{h},\mathrm{s} \in \mathbb{F}_2[x]/(x^r-1)$.
  It returns 1 if $\left|\mathrm{e}_0\right| + \left|\mathrm{e}_1\right| = t$ and $\mathrm{e}_0 + \mathrm{e}_1\mathrm{h} = \mathrm{s}$, and 0 otherwise.

For any probabilistic polynomial time algorithm $\mathcal{A}$ taking for inputs $\mathrm{h} \in \mathbb{F}_2[x]/(x^r-1)$ and $\mathrm{s} \in \mathbb{F}_2[x]/(x^r-1)$, and producing as output an element of $\left(\mathbb{F}_2[x]/(x^r-1)\right)^2$, we define its advantage concerning the one-wayness of the cipher as:

$$\mathrm{Adv}^{\mathrm{OW}}_{\mathrm{QCSD}_{r,t}}(\mathcal{A}) = \Pr\left[\mathrm{QCSD}_{r,t}(\mathcal{A}(\mathrm{h},\mathrm{e}_0+\mathrm{e}_1\mathrm{h}),\mathrm{h},\mathrm{e}_0+\mathrm{e}_1\mathrm{h}) \,\middle|\, \mathrm{h}\in\mathcal{R}_r,(\mathrm{e}_0,\mathrm{e}_1)\in\mathcal{E}_{n,t}\right].$$

For any probabilistic polynomial time algorithm $\mathcal{D}$ taking as input $\mathrm{h} \in \mathbb{F}_2[x]/(x^r-1)$, and producing 0 or 1 as output, we define its advantage concerning the distinguishability of the parity check matrix as:

$$\mathrm{Adv}^{\mathrm{IND}}_{\mathrm{QCCF}_{r,w}}(\mathcal{D}) = \left|\Pr\left[\mathcal{D}\left(\mathrm{h}_1\mathrm{h}_0^{-1}\right)=1 \,\middle|\, \mathrm{h}_0,\mathrm{h}_1\in\mathcal{R}_{r,w/2}\right] - \Pr\left[\mathcal{D}\left(\mathrm{h}\right)=1 \,\middle|\, \mathrm{h}\in\mathcal{R}_r\right]\right|.$$

Theorem 4.1. *[BIKE, §C.1.2] For any IND-CPA adversary $\mathcal{A}$ against PKE_1 there exists an adversary $\mathcal{A}'$ against $\mathrm{QCSD}_{r,t}$ and a distinghuer $\mathcal{D}$ against $\mathrm{QCCF}_{r,w}$ running in about the same time as $\mathcal{A}$ such that*

$$\mathrm{Adv}^{\mathrm{IND-CPA}}_{\mathrm{PKE}_1}(\mathcal{A}) \leq \frac{1}{2}\mathrm{Adv}^{\mathrm{OW}}_{\mathrm{QCSD}_{r,t}}(\mathcal{A}') + \frac{1}{2}\mathrm{Adv}^{\mathrm{IND}}_{\mathrm{QCCF}_{r,w}}(\mathcal{D}).$$

The number of bits of security of a problem is usually defined as the smallest λ such that for any adversaries $\mathcal{A}$

$$\frac{\mathrm{Time}(\mathcal{A})}{\mathrm{Adv}(\mathcal{A})} \geq 2^{\lambda}$$

where $\mathrm{Time}(\mathcal{A})$ is the running time of $\mathcal{A}$. Note that if $\mathcal{A}$ makes q queries to any oracle then $\mathrm{Time}(\mathcal{A}) > q$.

To summarize Theorem 2.5 & Theorem 4.1, in order to have λ bits of security against an IND-CCA adversary we want:

- $\mathcal{WF}_{\mathrm{QCSD}}(2r,r,t) > 2^{\lambda}$,
- $\mathcal{WF}_{\mathrm{QCCF}}(2r,r,w) > 2^{\lambda}$,
- $2^{\ell} > 2^{\lambda}$,
- $\delta < 2^{-\lambda}$ *i.e.* $\mathrm{DFR}(\texttt{decoder}) < 2^{-\lambda}$.

Remember that we call DFR the decoding failure rate of a decoding algorithm. The best known algorithm to solve the underlying problems are variants of the ISD algorithms

that we discussed in §3.3. So to satisfy the first two points we should have, in a first approximation:

$$2^{1/2+t(1+o(1))-\log_2(n)/2} > 2^\lambda \qquad \text{and} \qquad 2^{1+w(1+o(1))-\log_2(n)} > 2^\lambda$$

for some security parameter λ. Parameters should nevertheless be determined by considering the actual work factors of ISD algorithms.

If only the first three points are verified, the system is still IND-CPA secure. The last point is the main concern of this document and grants IND-CCA security if verified.

Table 4.1 gives the sets of parameters of [BIKE]. Two values for the block size r are provided, r^{CPA} that verifies the first three conditions and r^{CCA} that was the value retained in [BIKE] to have a DFR low enough for IND-CCA security using the method that we detail in Chapter 14.

4.2 Block size

Squaring. If the block size r is an even number, it has been shown in [Lön+16] that an attacker can decrease the cost of the ISD attacks using a squaring technique.

Remember that we are in the context of Proposition 1.31 and we are dealing with polynomials in $\mathbb{F}_2[x]/(x^r-1)$. While, in [Lön+16], only a McEliece variant using QC-MDPC is mentioned, it can easily be translated for the Niederreiter variant and this is what we will do here.

As we are dealing with a field of characteristic 2, squaring a polynomial

$$\sum_{i=0}^{r-1} a_i x^i$$

gives, if r is even,

$$\sum_{i=0}^{r-1} a_i x^{2i} = \sum_{i=0}^{r/2-1} (a_i + a_{i+r/2}) x^{2i} \mod (x^r-1). \tag{4.1}$$

Attacks on the system can be roughly described, for a private key $\mathrm{h} \in \mathbb{F}_2[x]/(x^r-1)$ and a syndrome $\mathrm{s} \in \mathbb{F}_2[x]/(x^r-1)$, as follows.

Key recovery	Message recovery
Find $\mathrm{h}_0, \mathrm{h}_1 \in \mathbb{F}_2[x]/(x^r-1)$ such that	Find $\mathrm{e}_0, \mathrm{e}_1 \in \mathbb{F}_2[x]/(x^r-1)$ such that
$\lvert \mathrm{h}_0 \rvert \leq w$ $\lvert \mathrm{h}_1 \rvert \leq w$ and $\mathrm{h}_0 \mathrm{h} = \mathrm{h}_1$	$\lvert \mathrm{e}_0 \rvert + \lvert \mathrm{e}_1 \rvert \leq t$ and $\mathrm{e}_0 + \mathrm{e}_1 \mathrm{h} = \mathrm{s}$

If we now square all equations, we obtain the following. Note that a squared polynomial has no monomial of odd degree, hence the dimension of the problem is divided by 2.

Key recovery (squared)	Message recovery (squared)
Find $\mathrm{h}_0^2, \mathrm{h}_1^2 \in \mathbb{F}_2[x]/(x^r-1)$ such that	Find $\mathrm{e}_0^2, \mathrm{e}_1^2 \in \mathbb{F}_2[x]/(x^r-1)$ such that
$\lvert \mathrm{h}_0^2 \rvert \leq w_2$ $\lvert \mathrm{h}_1^2 \rvert \leq w_2$ and $\mathrm{h}_0^2 \mathrm{h}^2 = \mathrm{h}_1^2$	$\lvert \mathrm{e}_0^2 \rvert + \lvert \mathrm{e}_1^2 \rvert \leq t_2$ and $\mathrm{e}_0^2 + \mathrm{e}_1^2 \mathrm{h}^2 = \mathrm{s}^2$

Now the weights w_2 and t_2 are not specified. In (4.1), we say that there is a collision if there exists an i such that

$$a_i = a_{i+r/2} = 1\,.$$

Each collision decreases the weight of the squared polynomial by 2 compared to the original polynomial. So the squared versions of the problems are simpler (remember from §3.3 that the complexity of the problems depends mainly on t and w).

Of course, the process can be repeated as many times as the dyadic valuation of r. And each time the process is applied, since the dimension is divided by two, the probability of having a collision increases and therefore the weights[3] w_{2^i} and t_{2^i} decrease as well.

Square roots. Once a solution to the squared problem is found, it remains to lift it to return to the original problem. Say $\mathrm{a}'^2 = \sum_{\substack{i=0 \\ i \text{ even}}}^{r/2-1} a'_i x^{2i}$ is a solution to a problem. For any position i such that $a'_i = 1$ then

$$a'_i = a_i + a_{i+r/2} = 1$$

so either $a_i = 1$ or $a_{i+r/2} = 1$.

A solution to the initial problem is found using an ISD algorithm with the additional leverage provided by this information. To be more precise, if we remember the general structure of such an algorihm explained in Figure 3.2, rather than applying a random permutation $\mathbf{P}$ to the matrix, we restrict ourselves to permutations that do not place the above-mentioned positions in the information set. In other words, these positions will always be permuted to the first $n-k$ (or $n-k-\ell$) positions.

Doing this is equivalent to removing $2 \cdot w_2$ symbols (puncturing the code) so this increases the code rate slightly. And from Proposition 3.5 we know that this will slightly increase the contribution of the factor c in the exponent of the workfactor.

However, if many collisions occurred during the squaring process, the decrease in the weight w (or t) makes this algorithm worthwhile from an attacker's point of view.

As with the squaring process, this process too can be repeated as many times as the dyadic valuation of r.

To summarize, an attacker would perform squarings, then look for solution to the reduced problem, and finally lift the solution by computing square roots. The dominating factor in the complexity of this attack is the first solution to the reduced problem. It is particularly interesting for weak keys: keys with numerous collisions once squared. It has been shown in [Lön+16], for some parameters such that 2^4 divides r, that the logarithm of the cost of the attack divided by the probability of having a weak key is at least 10 bits below the expected security level.

Generalization. The attack was further generalized in [CT19]. This paper adopts a different point of view. It studies codes with a non-trivial automorphism group *i.e.* a group of isometries that leave the code globally invariant. The code is then *folded*: for a given codeword, we consider the sum of its images under the elements of the automorphism group. The result is the same as in the previous paragraphs, the decoding problem is reduced to a smaller dimension, a smaller error weight with the same code rate.

To do the connection with the previous paragraphs, for a quasi-cyclic code with an even block size, the permutation that, in each block, shifts the positions by $r/2$ coordinates is an isometry and along with the identity it forms an automorphism group.

This could be done for any non-trivial divisor r' of r, not just 2. We would then consider shifts in multiples of r/r', giving a group of automorphism of the order r'. To avoid any such attack, r must therefore be a prime number.

[3] Repeated squaring gives the same equations as above, we simply need to change each exponent by 2 by 2^i and the weights w_2 and t_2 by w_{2^i} and t_{2^i} respectively, for the i-th iteration of the process.

In fact, to avoid any other structural attack, in [BIKE], the block size r is chosen so that the polynomial $x^r - 1$ has only two irreducible factors:

$$x + 1 \qquad \text{and} \qquad x^{r-1} + x^{r-2} + \cdots + x + 1\,.$$

Equivalently, it means that 2 is primitive modulo $r - 1$.

It also ensures that any polynomial of $\mathbb{F}_2[x]/(x^r - 1)$ with an odd Hamming weight is invertible, thus avoiding having to integrate any rejection process in the key generation[4].

[4]Remember that in the Niederreiter setting, the first block h_0 has to be inverted to compute the systematic form $\mathrm{h} = \mathrm{h}_1\mathrm{h}_0^{-1}$.

Chapter 5

Syndrome weight and counters in a regular MDPC code

There is a quantity that plays a fundamental role in decoding, notably for bit-flipping which is the type of algorithms used for [BIKE][1]. That quantity is the *counter* of a position, *i.e.* the number of unsatisfied parity check equations involving that position. In this chapter, we study it as a subject in its own right and explain the relation with the syndrome weight.

This chapter is a review of some results of [Cha17] that we will need and refer to regularly throughout this document.

Notation. For this chapter, we consider a (d, w)-regular $[2r, r]$-MDPC code with a (sparse) parity check matrix $\mathbf{H}$ and consider the problem of finding the error vector e of weight t from the syndrome $\mathrm{s} = \mathbf{H}\mathrm{e}^\top$ with the parameters

- r: the block size of the parity check matrix;
- $n = 2r$: the length of the code,
- d: the column weight of the parity check matrix;
- $w = 2d$: the row weight of the parity check matrix,
- t: the weight of the error pattern.

We write[2]

$$\mathbf{H} = \begin{pmatrix} \mathbf{h}_0 & \mathbf{h}_1 & \cdots & \mathbf{h}_{n-1} \end{pmatrix} = \begin{pmatrix} \mathbf{h}_0^\top \\ \mathbf{h}_1^\top \\ \vdots \\ \mathbf{h}_{r-1}^\top \end{pmatrix}$$

where $\mathbf{h}_0, \ldots, \mathbf{h}_{n-1}$ are the columns of $\mathbf{H}$ and $\mathbf{h}_0^\top, \ldots, \mathbf{h}_{r-1}^\top$ are the rows of $\mathbf{H}$.

5.1 Fundamental quantities

Definition 5.1. For any position $j \in \{0, \ldots, n-1\}$, we call *counter*, the quantity

$$\sigma_j(\mathbf{H}, \mathrm{s}) := \left| \mathbf{h}_j \star \mathrm{s} \right| .$$

[1] After all BIKE stands for Bit-Flipping Key Encapsulation.

[2] Note that for a quasi-cyclic code, in the polynomial formalism, if the parity check matrix is represented as $(\mathrm{h}_0, \mathrm{h}_1)$ with $\mathrm{h}_0, \mathrm{h}_1 \in \mathbb{F}_2[x]/(x^r - 1)$ for some block size r, then we have, for $i = 0, 1$

$$\forall j \in \{0, \ldots, r-1\}, \mathbf{h}_{i \cdot r + j} = x^j \mathrm{h}_i .$$

Definition 5.2. The number of equations affected by exactly ℓ errors is

$$E_\ell(\mathbf{H}, \mathrm{e}) := \left| \left\{ i \in \{0, \dots, r-1\} : \left| \mathbf{h}_i^\top \star \mathrm{e} \right| = \ell \right\} \right|.$$

The quantities $\mathbf{H}$, s and e are usually clear from the context. We will omit them and simply write σ_j and E_ℓ.

Proposition 5.3. *The following identities are verified for all* $\mathbf{H}$*, all* e *and* $\mathrm{s} = \mathbf{H}\mathrm{e}^\top$*:*

$$\sum_{\ell \ odd} E_\ell = |\mathrm{s}| \ , \qquad \sum_j \sigma_j = w\,|\mathrm{s}| \ , \qquad \sum_{j \in \mathrm{e}} \sigma_j = \sum_{\ell \text{ odd}} \ell E_\ell \, .$$

Proof. For any $i \in \{1, \dots, r\}$, $s_i = 1$ if and only if $\left| \mathbf{h}_i^\top \star \mathrm{e} \right|$ is odd, implying the first identity.

We can prove the next one by using the definition of σ_j. We will use the fact that for any i and j, we have $i \in \mathbf{h}_j \iff j \in \mathbf{h}_i^\top$. Equivalently, using indicator functions, for any i, j, $\mathbf{1}_{\mathbf{h}_j}(i) = \mathbf{1}_{\mathbf{h}_i^\top}(j)$.

$$\begin{aligned}
\sum_{j=0}^{n-1} \sigma_j &= \sum_{j=0}^{n-1} \left| \mathbf{h}_j \star \mathrm{s} \right| = \sum_{j=0}^{n-1} \sum_{i=0}^{r-1} \mathbf{1}_{\mathbf{h}_j \star \mathrm{s}}(i) \\
&= \sum_{j=0}^{n-1} \sum_{i=0}^{r-1} \mathbf{1}_{\mathbf{h}_j}(i) \mathbf{1}_{\mathrm{s}}(i) = \sum_{j=0}^{n-1} \sum_{i=0}^{r-1} \mathbf{1}_{\mathbf{h}_i^\top}(j) \mathbf{1}_{\mathrm{s}}(i) \\
&= \sum_{i=0}^{r-1} \mathbf{1}_{\mathrm{s}}(i) \left(\sum_{j=0}^{n-1} \mathbf{1}_{\mathbf{h}_i^\top}(j) \right) .
\end{aligned}$$

We can then use the fact that each row of $\mathbf{H}$ has a fixed weight.

Similarly, we can prove the last identity:

$$\begin{aligned}
\sum_{j \in \mathrm{e}} \sigma_j &= \sum_{j, e_j = 1} \left| \mathbf{h}_j \star \mathrm{s} \right| = \sum_{j=0}^{n-1} \sum_{i=0}^{r-1} \mathbf{1}_{\mathbf{h}_j \star \mathrm{s}}(i) \mathbf{1}_e(j) \\
&= \sum_{j=0}^{n-1} \sum_{i=0}^{r-1} \mathbf{1}_{\mathbf{h}_j}(i) \mathbf{1}_{\mathrm{s}}(i) \mathbf{1}_{\mathrm{e}}(j) = \sum_{j=0}^{n-1} \sum_{i=0}^{r-1} \mathbf{1}_{\mathbf{h}_i^\top \star \mathrm{e}}(j) \mathbf{1}_{\mathrm{s}}(i) \\
&= \sum_{i, \left| \mathbf{h}_i^\top \star \mathrm{e} \right| \text{ odd}} \sum_{j=0}^{n-1} \mathbf{1}_{\mathbf{h}_i^\top \star \mathrm{e}}(j) = \sum_{\substack{\ell=0 \\ \ell \text{ odd}}}^{\min(w,t)} \ell E_\ell \, .
\end{aligned}$$

□

5.2 Counters distributions

We can adopt a transposed point of view for a counter, compared to Definition 5.1. First let us see that

$$\mathrm{s} = \begin{pmatrix} \left| \mathbf{h}_0^\top \star \mathrm{e} \right| \bmod 2 \\ \left| \mathbf{h}_1^\top \star \mathrm{e} \right| \bmod 2 \\ \vdots \\ \left| \mathbf{h}_{r-1}^\top \star \mathrm{e} \right| \bmod 2 \end{pmatrix} .$$

We have[3] for any $j \in \{0, \dots, n-1\}$,

$$\sigma_j = \sum_{i \in \mathbf{h}_j} \left(\left| \mathbf{h}_i^\top \star \mathrm{e} \right| \bmod 2 \right) .$$

[3] Remember that we equate a vector with its support.

We now assume that the parity check matrix $\mathbf{H}$ and the error vector e are drawn uniformly at random (but still verifying the constraints on regularity and Hamming weight). This way the syndrome s is a binary random vector and the counters are random variables.

It is natural to assume, for a fixed $j \in \{0, \dots, n-1\}$, that the d random variables defined by $\left(\left|\mathbf{h}_i^\top \star \mathrm{e}\right| \bmod 2\right)$ for i in the support of $\mathbf{h}_j$ are independent. Let us write the following probabilities

$$\pi_0 = \Pr\left[\left(\left|\mathbf{h}_i^\top \star \mathrm{e}\right| \bmod 2\right) = 1 \,\middle|\, j \notin \mathrm{e}, i \in \mathbf{h}_j\right],$$
$$\pi_1 = \Pr\left[\left(\left|\mathbf{h}_i^\top \star \mathrm{e}\right| \bmod 2\right) = 1 \,\middle|\, j \in \mathrm{e}, i \in \mathbf{h}_j\right].$$

Finally, still assuming independence of the random variables, the counters follow one of the following two binomial distributions, depending on whether they concern a position in the support of the error or not:

$$\sigma_j \sim \mathrm{Bin}(d, \pi_0) \qquad \text{if } j \notin \mathrm{e},$$
$$\sigma_j \sim \mathrm{Bin}(d, \pi_1) \qquad \text{if } j \in \mathrm{e}.$$

5.2.1 Average case

With no further assumption on $\mathbf{H}$ or e, we have

$$\pi_0 = \sum_{\ell \text{ odd}} \frac{\binom{w-1}{\ell}\binom{n-w}{t-\ell}}{\binom{n-1}{t}}, \quad \text{and} \quad \pi_1 = \sum_{\ell \text{ even}} \frac{\binom{w-1}{\ell}\binom{n-w}{t-1-\ell}}{\binom{n-1}{t-1}}. \tag{5.1}$$

Proof. If $j \in \mathrm{e}$, let $i \in \mathbf{h}_j$, then $\left(\left|\mathbf{h}_i^\top \star \mathrm{e}\right| \bmod 2\right) = 1$ if and only if among the $(w-1)$ remaining positions $j' \in \mathbf{h}_i^\top \setminus \{j\}$ an even number of them are among the $(t-1)$ remaining errors $j'' \in \mathrm{e} \setminus \{j\}$.

If $j \notin \mathrm{e}$, let $i \in \mathbf{h}_j$, then $\left(\left|\mathbf{h}_i^\top \star \mathrm{e}\right| \bmod 2\right) = 1$ if and only if among the $(w-1)$ remaining positions $j' \in \mathbf{h}_i^\top \setminus \{j\}$ an odd number of them are among the t errors $j'' \in \mathrm{e}$. □

5.2.2 Conditioning the counter distributions with the syndrome weight

Suppose now that we are no longer in the average case with a uniformly drawn error pattern e but that we have a syndrome $\mathrm{s} = \mathbf{H}\mathrm{e}^\top$ such that its weight $|\mathrm{s}| =: S$ is fixed.

Definition 5.4. We write X the quantity

$$X := \left(\sum_{j \in \mathrm{e}} \sigma_j\right) - S = \sum_{\ell > 0} 2\ell E_{2\ell+1}.$$

Suppose also that we know the sum of the counters of erroneous positions:

$$\sum_{j \in \mathrm{e}} \sigma_j = S + X$$

where X is a positive value. And using Proposition 5.3, we have:

$$\sum_{j \notin \mathrm{e}} \sigma_j = (w-1)S - X.$$

Then, using the fact that a random variable following a binomial distribution has values around its mean, we can write:

$$\pi_0 = \frac{(w-1)S - X}{d(n-t)}, \qquad \pi_1 = \frac{S + X}{dt}. \tag{5.2}$$

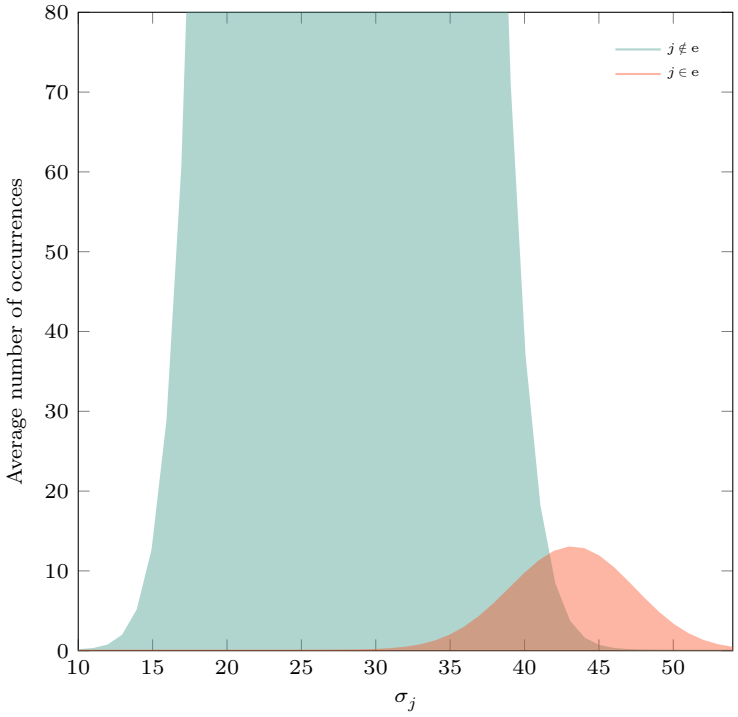
In practice, we often wish to obtain an accurate model for the counter distributions which only depends on S and t. So we must somehow get rid of X. In practice $X = 2E_3 + 4E_5 + \cdots$ is not dominant in the above formula (for relevant parameters) and we can replace it by its expected value.

Proposition 5.5. *The expected value of* $X = \sum_{\ell>0} 2\ell E_{2\ell+1}$ *is*

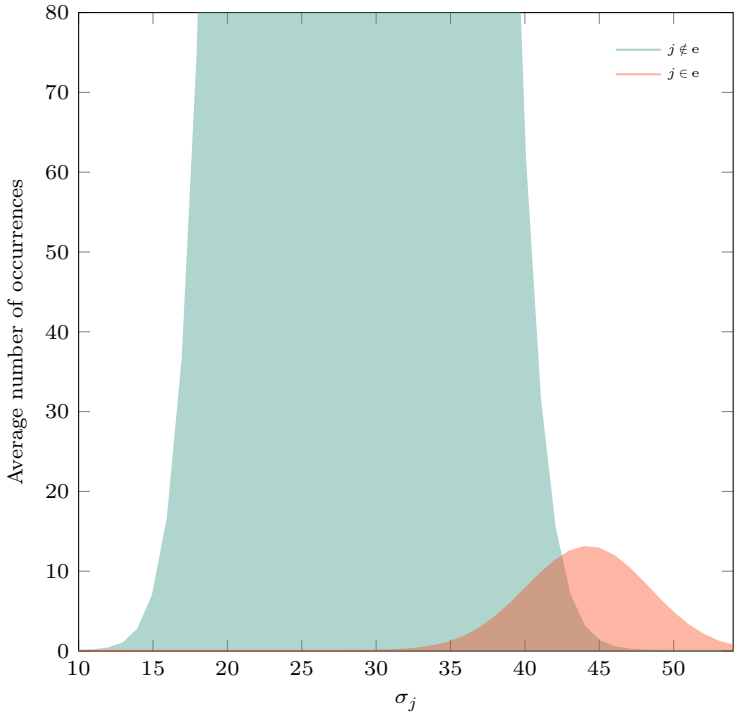
$$\overline{X} = r \sum_{\ell} 2\ell \rho_{2\ell+1} \quad \textit{where} \quad \rho_\ell = \frac{\binom{w}{\ell}\binom{n-w}{t-\ell}}{\binom{n}{t}}.$$

To illustrate the situation, the number of occurrences expected for a counter value with BIKE parameters offering 128-bit security is shown in Figure 5.1.

In practice, the counter distributions with probabilities from (5.2) are really close to those observed from simulation. This gives counters distributions accurate enough to allow improvement of the bit-flipping algorithm with better thresholds as we will see in §6.1.3.1. However, as far as the probabilistic modeling of the algorithm is concerned, we will see in Chapter 11 that this is not good enough. We will show a more accurate model in §11.3.3 that takes into account the regularity of the code.



(a) Average.



(b) Conditioned by $|\mathrm{s}| = 5\,000$ and $X = \sum_{j \in \mathrm{e}} \sigma_j - |\mathrm{s}| = 912$.

Figure 5.1: Counters distributions and threshold for $(r, d, t) = (12\,323, 71, 137)$.

Part II

New bit-flipping decoders for QC-MDPC

Summary of contributions

We design three new decoding algorithms for MDPC codes:

- one that is randomized and sequential: step-by-step,
- one that implements a novel idea consisting in adding a lifetime to each decision taken: Backflip,
- and one that considers an intermediate level of decision: gray decoder.

The first one has a theoretical interest that we will develop in Part III. The other two are at an intermediate level between the hard- and soft-decision algorithms and have great performance with a low complexity. Backflip shows remarkable performance that scales well as the number of iterations increases.

Chapter 6

Introduction

Some decoders are more naturally described as syndrome decoders and others rather as decoders of a noisy codeword. Keeping the notations introduced in Sections 3.1 & 3.2, the algorithms of the first category will have a superscript $\perp$ symbol in their name while the others will not.

In general, a bitflipping-based decoder is best understood in terms of its effects on a syndrome, and a message-passing algorithm carries out calculations related to the positions of a received word. These two categories of decoders will be detailed in this introductory chapter.

In this document, we deal with codes whose parity check matrix consists of two $r \times r$ invertible matrices. The conversions between the two types of decoders are thus explicitly described in the Algorithm 6.1.

Algorithm 6.1: Conversions between decoders and syndrome decoders.

function $\texttt{decoder}^{\perp}(\mathbf{H}, \mathrm{s})$**:**

- **input :** A sparse parity check matrix $\mathbf{H} = (\mathbf{H}_0, \mathbf{H}_1) \in \mathbb{F}_2^{(n-k)\times n}$, a syndrome $\mathrm{s} = \mathbf{H}\mathrm{e}^\top \in \mathbb{F}_2^{n-k}$.
- **output :** An error pattern $\mathrm{e}' \in \mathbb{F}_2^n$ such that $\mathbf{H}\mathrm{e}'^\top = \mathrm{s}$.
- $\mathrm{y} \leftarrow \left(\left(\mathbf{H}_0^{-1}\mathrm{s} \right)^\top \quad 0 \right)$;
- $\mathrm{c} \leftarrow \texttt{decoder}(\mathbf{H}, \mathrm{y})$;
- **return** $\mathrm{y} + \mathrm{c}$;

function $\texttt{decoder}(\mathbf{H}, \mathrm{y})$**:**

- **input :** A sparse parity check matrix $\mathbf{H} = (\mathbf{H}_0, \mathbf{H}_1) \in \mathbb{F}_2^{(n-k)\times n}$, a vector $\mathrm{y} \in \mathbb{F}_2^n$.
- **output :** A codeword $\mathrm{c} \in \mathbb{F}_2^n$ such that $\mathrm{y} = \mathrm{c} + \mathrm{e} \in \mathbb{F}_2^n$.
- $\mathrm{s} \leftarrow \mathbf{H}\mathrm{y}^\top$;
- $\mathrm{e} \leftarrow \texttt{decoder}^{\perp}(\mathbf{H}, \mathrm{s})$;
- **return** $\mathrm{y} + \mathrm{e}$;

6.1 State of the art

Given the close link between an LDPC code and an MDPC code, it is relevant to investigate the decoding of the former. This is what we will do first, before turning our attention to decoders designed specifically for MDPC codes. Finally, we will present the work of [Cha17]

on the choice of thresholds, a central component of bit-flipping algorithms.

6.1.1 LDPC codes

Many variants of decoders were proposed since the invention of LDPC codes. They can all be seen as improvements or specialization of one of the following algorithms:

- bit-flipping;
- belief propagation, also called sum-product algorithm.

The former is a *hard-decision decoder*: a bit is either set to 0 or 1 and there is no way to deal with uncertainty. The latter is a *soft-decision decoder*: its input and all the intermediate states represent a probability or a likelihood.

Soft-decision decoders usually outperform hard-decision ones, but they have a more complex logic and require more memory. Looking for trade-offs between the two types of algorithms is a line of research that has often been considered, and we will label the resulting algorithms as *intermediate*.

6.1.1.1 Hard-decision decoding

Bit-flipping. A very simple iterative decoding algorithm was proposed by Gallager in [Gal63], it is usually referred to as the bit-flipping algorithm. It is based on the following observation: if a position is involved in many unsatisfied equations, it is more likely to be erroneous. Thus, at each iteration of the algorithm, the syndrome is computed and then, for each position, the number of unsatisfied equations in which it is involved is counted and if this count exceeds a certain threshold, the value of the position is flipped. Hopefully, with each iteration, the number of errors is decreased until there are no more errors, the syndrome is then zero. Full descriptions are given in Algorithm 6.2 and Algorithm 6.3. In these descriptions, the threshold is treated as the result of a call to a black box function but, as it is a subject on its own, the actual implementation of such a function will be discussed later in this document. Common choices for the threshold are: a majority vote *i.e.* $T = (\left|\mathbf{h}_j\right| + 1)/2$ or the maximum value reached by the counters $\max_j \left|\mathbf{h}_j \star \mathrm{s}\right|$.

Algorithm 6.2 is said to be *parallel* because at each iteration, the syndrome is recomputed only once after all positions have been checked. On the other hand, Algorithm 6.3 is called *serial* because the syndrome is updated after each flip.

Algorithm 6.2: Parallel bit-flipping.

function `parallel_bitflip`$^{\perp}(\mathbf{H}, \mathrm{s})$**:**
- **input :** A sparse parity check matrix $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}$, a syndrome $\mathrm{s} = \mathbf{H}\mathrm{e}^\top \in \mathbb{F}_2^{n-k}$.
- **output :** An error pattern $\mathrm{e}' \in \mathbb{F}_2^n$ such that $\mathbf{H}\mathrm{e}'^\top = \mathrm{s}$.
- $\mathrm{e}' \leftarrow 0$;
- $\mathrm{s}' \leftarrow \mathrm{s}$;
- $T \leftarrow$ `threshold`$(context)$;
- **while** $\mathrm{s}' \neq 0$ **do**
  - **for** $j \in \{0, \dots, n-1\}$ **do**
    - **if** $\left|\mathbf{h}_j \star \mathrm{s}'\right| \geq T$ **then**
      - $e'_j \leftarrow 1 - e'_j$;
  - $\mathrm{s}' \leftarrow \mathrm{s} - \mathbf{H}\mathrm{e}'^\top$;
- **return** e';

Algorithm 6.3: Serial bit-flipping

```
function serial_bitflip^⊥(H, s):
    input  : A sparse parity check matrix H ∈ F_2^{(n−k)×n},
             a syndrome s = He^⊤ ∈ F_2^{n−k}.
    output : An error pattern e′ ∈ F_2^n such that He′^⊤ = s.
    e′ ← 0;
    s′ ← s;
    T ← threshold(context);
    while s′ ≠ 0 do
        for j ∈ {0, ..., n − 1} do
            if |h_j ⋆ s′| ≥ T then
                e′_j ← 1 − e′_j ;
                s′ ← s − He′^⊤;
    return e′;
```

These algorithms have been studied by Sipser and Spielman in [SS96]. They showed that it converges to the solution given that the Tanner graph has some expanding properties[1] *i.e.* if a relatively small subset of vertices has many connections with the rest of the graph. The threshold they used corresponds to a majority vote *i.e.* $T = \left(\left|\mathbf{h}_j\right| + 1\right)/2$.

Message-passing algorithms. An important class of LDPC decoders are the message-passing algorithms. In these algorithms, the nodes of the Tanner graph of the code are assigned values that are initialized using the received word and then the edges are used to "pass messages" in order to update the node values. Algorithm 6.4 gives the template of such an algorithm that we will use by defining the functions `init`, `check_to_var`, `var_to_check`, `a_posteriori` for the different variants. It uses the function $\mathcal{N}_{\mathbf{H}}$ that returns the set of immediate neighbours of a node in the Tanner graph defined by the parity check matrix $\mathbf{H}$.

Gallager A/B. One algorithm and its variant, often mentioned in the litterature as Gallager A and B algorithms, can be seen as a message-passing version of the bit-flipping decoder. Its description is given it Table 6.1. In contrast to the bit-flipping algorithm, this decoder does not exchange extrinsic information between nodes. In the bit-flipping algorithm, a node sends the same information to all its neighbours (all variable nodes send their current value to all their neighboring check nodes and vice versa). The message-passing algorithm explicitely excludes the information it received from a node from the information it sends to this very node.

The difference between Gallager A and Gallager B is in the way the thresholds T are computed. In Gallager B, thresholds are carefully computed and may change between rounds or nodes. In Gallager A, the threshold is always $T = \deg(v) - 1$ for any variable node v.

6.1.1.2 Soft-decision decoding

Belief propagation. The messages that are passed in the previous algorithm correspond to binary votes depending on the state of the nodes. In the belief propagation algorithm, probabilities or likelihoods are passed. This algorithm was first defined by Gallager

[1] This is explained in more detail in §10.1.2

Algorithm 6.4: Message-passing algorithm.

```
function message_passing(H, y):
    input : The biadjacency matrix of a bipartite graph H ∈ F_2^{(n−k)×n},
            a vector y ∈ F_2^n.
    /* For i ∈ {0, ..., n−k−1} and j ∈ {0, ..., n−1},
       m_{v_j} is a message from the channel,
       m_{c_i→v_j} is a message from check node c_i to variable node v_j,
       m_{v_j→c_i} is a message from variable node v_j to check node c_i.   */
    for j ∈ {0, ..., n−1} do
        m_{v_j} ← init(y_j);
        for c ∈ N_H(v_j) do
            m_{v_j→c} ← m_{v_j}
    /* Loop while the current vector is not a codeword.                    */
    while Hy^⊤ ≠ 0 do
        for i ∈ {0, ..., n−k−1} do
            for v ∈ N_H(c_i) do
                /* Check node to variable nodes.                           */
                m_{c_i→v} ← check_to_var({m_{v'→c_i} | v' ∈ N_H(c_i) \ {v}});
        for j ∈ {0, ..., n−1} do
            for c ∈ N_H(v_j) do
                /* Variable node to check nodes.                           */
                m_{v_j→c} ← var_to_check(m_{v_j}, {m_{c'→v_j} | c' ∈ N_H(v_j) \ {c}});
            y_j ← a_posteriori(m_{v_j}, {m_{c'→v_j} | c' ∈ N_H(v_j)});
    return y;
```

in [Gal63] to decode LDPC codes. In a more general context, the algorithm was redefined by Pearl in [Pea82] and is used in the artificial intelligence field.

There are many equivalent ways to write the algorithm, one can consider probability, likelihood ratio or log-likelihood ratio. Here we will write it using the log-likelihood ratio.

Definition 6.1. Let X be a binary random variable and let Y be a random variable. The *likelihood ratio* is the ratio

$$L(X \mid y) := \frac{\Pr[X = 0 \mid Y = y]}{\Pr[X = 1 \mid Y = y]}.$$

The *log-likelihood ratio* is then simply $\log\big(L\left(X \mid y\right)\big)$.

Suppose that we send a codeword x through a channel, we call y the received vector. Let us now assume that the Tanner graph is a tree, we can thus assume that variable nodes are mutually independent as well as the check nodes. Using Proposition 1.14, it can be shown (see [Gal63]) that

$$\frac{\Pr\left[x_v = 0 \mid y_v, \mathcal{O}_v\right]}{\Pr\left[x_v = 1 \mid y_v, \mathcal{O}_v\right]} = \frac{\Pr\left[x_v = 0 \mid y_v\right]}{\Pr\left[x_v = 1 \mid y_v\right]} \prod_{c \in \mathcal{N}_{\mathbf{H}}(v)} \frac{1 + \prod_{v' \in \mathcal{N}_{\mathbf{H}}(c)} (1 - 2\Pr\left[x_{v'} = 1 \mid y_{v'}, \mathcal{O}_{v'}\right])}{1 - \prod_{v' \in \mathcal{N}_{\mathbf{H}}(c)} (1 - 2\Pr\left[x_{v'} = 1 \mid y_{v'}, \mathcal{O}_{v'}\right])} \tag{6.1}$$

Table 6.1: Gallager A/B.

`init`(y)	:=	$1-2y$;
`check_to_var`$(\mathcal{M})$	:=	$\prod_{m_{v'\to c}\in\mathcal{M}} m_{v'\to c}$;
`var_to_check`$(y,\mathcal{M})$	:=	$s := y + \sum_{m_{c'\to v}\in\mathcal{M}} m_{c'\to v}$; $T \leftarrow$ `threshold`$(context)$; **return** $\begin{cases} y & \text{if } \lvert s\rvert < T \\ \operatorname{sgn}(s) & \text{otherwise;} \end{cases}$
`a_posteriori`$(y,\mathcal{M})$	:=	$s := y + \sum_{m_{c'\to v}\in\mathcal{M}} m_{c'\to v}$; **return** $\begin{cases} \frac{1-y}{2} & \text{if } s=0 \\ \frac{1-\operatorname{sgn}(s)}{2} & \text{otherwise;} \end{cases}$

Table 6.2: Belief propagation.

`init`(y)	:=	$\log\left(L\left(x \mid y\right)\right)$;
`check_to_var`$(\mathcal{M})$	:=	$2\tanh^{-1}\left(\prod_{m_{v'\to c}\in\mathcal{M}} \tanh\left(\frac{m_{v'\to c}}{2}\right)\right)$;
`var_to_check`$(y,\mathcal{M})$	:=	$y + \sum_{m_{c'\to v}\in\mathcal{M}} m_{c'\to v}$;
`a_posteriori`$(y,\mathcal{M})$	:=	$\frac{1-\operatorname{sgn}\left(y+\sum_{m_{c'\to v}\in\mathcal{M}} m_{c'\to v}\right)}{2}$;

where $\mathcal{O}_v$ is the event that the transmitted vector satisfies the parity check equations concerning v for some variable node v.

We can then use the fact that

$$\forall p \in [0,1], 1-2p = \tanh\left(\frac{1}{2}\log\frac{p}{1-p}\right)$$

and

$$\forall x \in (-1,1), \tanh^{-1}(x) = \frac{1}{2}\log\left(\frac{1+x}{1-x}\right)$$

to rewrite the formula using log-likelihood ratios.

Recall that the Tanner graph is assumed to be a tree. Excluding extrinsic information in the conditional probabilities, one can then show that the message-passing algorithm defined by Table 6.2 computes the log-likelihood values for each variable node with conditioning increasing by one more level for each iteration. In (6.1) the expression in the product corresponds to the messages sent from the check nodes to the variable nodes and the whole formula is the converse. Still assuming the Tanner graph is a tree, after enough iterations, the algorithm computes the log-likelihood of each variable node conditioned on the received vector and the values of the nodes in the entire graph.

A quantization method for implementing this algorithm efficiently has been detailed in [CFRU01]. It has been shown that, using this technique, a precision of less than 16 bits has no impact on the performance compared to more precise computations.

Assuming the Tanner graph has no loop of length less than or equal to the number of iterations, the convergence of this algorithm can be studied by a method know as density evolution that will be detailed in §10.1.1.1.

As with the bit-flipping algorithm, message-passing algorithms are characterized by the way they update information. Algorithm 6.4 uses a scheduling method known as *flooding*:

messages from variables (resp. check) nodes are sent all at once, only when all of them have been computed. Other methods have been shown to improve convergence and lower memory requirements by sending messages earlier. One of such scheduling technique is said to be *horizontal*, it processes check nodes one by one, during each iteration a check node immediately sends messages to its (variable nodes) neighbours which in turn immediately send messages to their (check nodes) neighbours. This algorithm is known as layered belief propagation.

6.1.1.3 Intermediate

The belief propagation algorithm usually has better decoding performance while the bit-flipping algorithm has lower computing complexity and memory requirements. Naturally, there has been a search for tradeoffs: a slight increase in complexity or memory for a better decoding performance than the bit-flipping.

Two-bit bit-flipping. In the two-bit bit-flipping algorithm [NV14], the variables nodes are given a strength encoded as a 2-bit value rather than a single bit: 0_s, 0_w, 1_s, 1_w where w indicates a "weak" value and s a "strong" value. Transitions from one strength to another is determined by a table that depends on the counter value. The general structure is described in Algorithm 6.5.

We adopt the convention that the binary vectors are written in lower case and the vectors of $\{0_\mathrm{w}, 0_\mathrm{s}, 1_\mathrm{w}, 1_\mathrm{s}\}$ in upper case. The functions $C\colon \{0,1\} \to \{0_\mathrm{w}, 0_\mathrm{s}, 1_\mathrm{w}, 1_\mathrm{s}\}$ and $c\colon \{0_\mathrm{w}, 0_\mathrm{s}, 1_\mathrm{w}, 1_\mathrm{s}\} \to \{0,1\}$ convert from one to the other. It is suggested that the lifting C is chosen such that $\forall i \in \{0,1\}, C(i) = i_\mathrm{s}$, or $\forall i \in \{0,1\}, C(i) = i_\mathrm{w}$ And the projection is the obvious choice $\forall t \in \{\mathrm{s}, \mathrm{w}\}, C(i_t) = i$.

Algorithm 6.5: Two-bit bit-flipping.

function `twobit_bitflip`$(\mathbf{H}, \mathrm{y})$**:**
- **input :** A sparse parity check matrix $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}$, a vector $\mathrm{y} \in \mathbb{F}_2^n$.
- **output :** A codeword $\mathrm{c} \in \mathbb{F}_2^n$ such that $\mathrm{y} = \mathrm{c} + \mathrm{e} \in \mathbb{F}_2^n$.

$\mathrm{s}^{(0)} \leftarrow \mathbf{H}{\mathrm{y}^{(0)}}^\top$;

$\mathrm{Y}^{(0)} \leftarrow \left(C\left(y_j\right)\right)_{j\in\{0,\dots,n-1\}}$;

$\ell \leftarrow 0$;

while $\mathrm{s}^{(\ell)} \neq 0$ **do**

  $\ell \leftarrow \ell + 1$;

  for $j \in \{0, \dots, n-1\}$ **do**

    $Y_j^{(\ell)} \leftarrow f\left(Y_j^{(\ell-1)}, \chi\left(S^{(\ell)}, j\right)\right)$;

  $\mathrm{y}^{(\ell)} \leftarrow \left(c\left(Y_j^{(\ell)}\right)\right)_{j\in\{0,\dots,n-1\}}$;

  $\mathrm{s}^{(\ell)} \leftarrow \mathbf{H}{\mathrm{y}^{(\ell)}}^\top$;

  for $i \in \{0, \dots, n-k-1\}$ **do**

    $S_j^{(\ell)} \leftarrow \left(\phi\left(s_j^{(\ell-1)}, s_j^{(\ell)}\right)\right)_{j\in\{0,\dots,r-1\}}$;

return $\mathrm{y}^{(\ell)}$;

The check nodes are given a 2-bit value depending on their previous state: i_p, or i_n for $i \in \{0,1\}$ for respectively the previously or newly assigned values. The function

$\phi\colon \{0,1\}^2 \to \{0_{\mathrm{p}}, 0_{\mathrm{n}}, 1_{\mathrm{p}}, 1_{\mathrm{n}}\}$ handles the task of maintaining the 2-bit syndrome values:

$$\forall i \in \{0,1\}, \phi(i,i) := i_{\mathrm{p}} \qquad \text{and} \qquad \phi(1-i,i) := i_{\mathrm{n}}\,.$$

The counters in this model are now tuples rather than integers, they are computed via the function χ:

$$\begin{aligned}
\forall j \in \{0,\dots,n-1\}, \chi_{0_{\mathrm{w}}}(\mathrm{S},j) &= \left|\{i \in \{0,\dots,r-1\} \,|\, S_i = 0_{\mathrm{w}}, h_{i,j} = 1\}\right|, \\
\chi_{0_{\mathrm{s}}}(\mathrm{S},j) &= \left|\{i \in \{0,\dots,r-1\} \,|\, S_i = 0_{\mathrm{s}}, h_{i,j} = 1\}\right|, \\
\chi_{1_{\mathrm{w}}}(\mathrm{S},j) &= \left|\{i \in \{0,\dots,r-1\} \,|\, S_i = 1_{\mathrm{w}}, h_{i,j} = 1\}\right|, \\
\chi_{1_{\mathrm{s}}}(\mathrm{S},j) &= \left|\{i \in \{0,\dots,r-1\} \,|\, S_i = 1_{\mathrm{s}}, h_{i,j} = 1\}\right|.
\end{aligned}$$

The description given in [NV14] is really specific to particular codes: LDPC with column weight 3. Authors suggest two node updating functions f.

The first one f_1 does not use the additional information on the syndrome bits about whether they were previously or newly assigned. It relies on the usual counters. The function is described in the following table showing the output values *vs.* counter values $\chi_{1_{\mathrm{s}}} + \chi_{1_{\mathrm{w}}}$.

	0	1	2	3
0_{w}	0_{s}	1_{w}	1_{s}	1_{s}
0_{s}	0_{s}	0_{s}	0_{w}	1_{s}
1_{w}	1_{s}	0_{w}	0_{s}	0_{s}
1_{s}	1_{s}	1_{s}	1_{w}	0_{s}

What is interesting with this method is that the normal working of the bit-flipping is kept when there is a high degree of confidence in a flip, but otherwise the decision is delayed.

The second suggested function additionally uses the reliability information from the syndrome.

$$f_2(i_v, \chi) = \begin{cases} f_1(i_v, \chi_{1_{\mathrm{s}}} + \chi_{1_{\mathrm{w}}}) & \text{if } (\chi_{0_{\mathrm{s}}}, \chi_{0_{\mathrm{w}}}, \chi_{1_{\mathrm{w}}}) \notin \{(2,0,0),(1,1,0)\}, \\ i_v & \text{if } (\chi_{0_{\mathrm{s}}}, \chi_{0_{\mathrm{w}}}, \chi_{1_{\mathrm{w}}}) = (2,0,0), \\ i_{\mathrm{w}} & \text{if } (\chi_{0_{\mathrm{s}}}, \chi_{0_{\mathrm{w}}}, \chi_{1_{\mathrm{w}}}) = (1,1,0). \end{cases}$$

6.1.2 MDPC codes

6.1.2.1 Bit-flipping

Several variants of Algorithm 6.2 were considered for MDPC decoding. They vary on the way to choose the threshold.

The original publication [MTSB13] suggests using $\max_j \left(\left|\mathbf{h}_j \star \mathrm{s}\right|\right) - \delta$ for some fixed value of δ. A fixed threshold per iteration strategy was explored in [HMG13; MG14; Cho16]. In [HMG13; MG14] they are based on Gallager's analysis of LDPC codes. In [Cho16] they were chosen by iterative experiments. And in [CS16], a novel approach was shown to be relevant: the threshold is chosen as a function of the syndrome weight. Obtaining this function was mainly experimental and many simulation data were needed in this first work, but an analytic formula derived from a theoretical model was presented in the subsequent work [Cha17].

A technique called adaptive auxiliary variable node (AAVN) was proposed in [BJK19]. It adds columns and rows to the parity check matrix in order to remove some cycles of length 4 in the Tanner graph as small cycles are known to be detrimental to decoders. In this work, the threshold is set to $\max_j \left(\left|\mathbf{h}_j \star \mathrm{s}\right|\right)$.

6.1.2.2 Message-passing algorithms

In [BSC16], a "soft" QC-MDPC based McEliece cryptosystem was proposed. In this system, rather than dealing with a vector in $\mathbb{F}_2^n$ and an error vector of fixed weight t, codewords are encoded using binary pulse amplitude modulation (2-PAM) and the error vector is an additive white Gaussian noise (AWGN). Authors suggest using a belief propagation algorithm in the log-likelihood domain to decode.

Using a soft-decision decoder for the original QC-MDPC scheme (with a binary fixed weight error vector) has been considered in [LB18]. They use a "scaled" sum-product algorithm: the messages sent from the check nodes to the variable nodes are multiplied by a factor ω. For the 80 security bits parameters of [MTSB13], they achieve great decoding performance with $\omega = 0.5$.

6.1.3 QC-MDPC decoding thresholds

In Chapter 5, we saw that, using the regularity of a quasi-cyclic code, we can determine a model for the counters distributions. In this model, they follow two binomial distributions: one for the counters concerning errors, and the other for the counters concerning correct positions:

$$\begin{aligned} \sigma_j &\sim \mathrm{Bin}(d, \pi_0) \qquad \text{if } j \notin \mathrm{e}\,, \\ \sigma_j &\sim \mathrm{Bin}(d, \pi_1) \qquad \text{if } j \in \mathrm{e}\,. \end{aligned}$$

The probabilities π_0 and π_1 are determined by one of the following formulas.

Average case: When no further assumption is made on $\mathbf{H}$ or e, we have

$$\pi_0 = \sum_{\ell \text{ odd}} \frac{\binom{w-1}{\ell}\binom{n-w}{|\mathrm{e}|-\ell}}{\binom{n-1}{|\mathrm{e}|}}\,, \quad \text{and} \quad \pi_1 = \sum_{\ell \text{ even}} \frac{\binom{w-1}{\ell}\binom{n-w}{|\mathrm{e}|-1-\ell}}{\binom{n-1}{|\mathrm{e}|-1}}\,. \tag{6.2}$$

When the syndrome weight is known: Conditioning the probabilities for some fixed values of $|\mathrm{s}| = S$ and $X = \sum_{j\in \mathrm{e}} \sigma_j - |\mathrm{s}|$, we have

$$\pi_0 = \frac{(w-1)S - X}{d(n-|\mathrm{e}|)}\,, \quad \text{and} \quad \pi_1 = \frac{S+X}{d\,|\mathrm{e}|}\,. \tag{6.3}$$

The purpose of bit-flipping is, as much as possible, to reduce the error weight at each iteration. Therefore, when this is repeated enough times, the original noisy codeword has been decoded, all the errors have been removed.

To reduce the error weight, the algorithm must flip more errors than correct positions. So a good threshold T is one where, of all the positions with a counter above it, a majority are errors:

$$\sum_{k\geq T} \#\{j \in \mathrm{e} \,|\, \sigma_j = k\} \qquad \geq \qquad \sum_{k\geq T} \#\{j \notin \mathrm{e} \,|\, \sigma_j = k\}\,.$$

In practice, we will take the smallest such T. And since the cardinalities involved in the above inequality are almost impossible to know without having already decoded, we will rely on probabilities. Let us denote

$$p_{\notin \mathrm{e}}(k) = \Pr[\sigma_j = k \,|\, j \notin \mathrm{e}] = \binom{d}{k} \pi_0^k (1-\pi_0)^{d-k}$$

$$\text{and} \qquad p_{\in \mathrm{e}}(k) = \Pr[\sigma_j = k \,|\, j \in \mathrm{e}] = \binom{d}{k} \pi_1^k (1-\pi_1)^{d-k}\,.$$

Then the above inequality translates as:

$$|\mathrm{e}| \sum_{k \geq T} p_{\in \mathrm{e}}(k) \qquad \geq \qquad (n - |\mathrm{e}|) \sum_{k \geq T} p_{\notin \mathrm{e}}(k) \,.$$

The error weight $|\mathrm{e}|$ is not known apart from the first iteration where it is equal to t. For any subsequent iteration, it has to be guessed, for example using the syndrome weight. Since the counters distributions are usually unimodal, a way to choose the threshold T that is more resilient to the imprecision on this guess is to choose the smallest T such that:

$$|\mathrm{e}| \, p_{\in \mathrm{e}}(T) \qquad \geq \qquad (n - |\mathrm{e}|) p_{\notin \mathrm{e}}(T) \,. \tag{6.4}$$

6.1.3.1 Adaptive threshold

Using the probabilities π_0 and π_1 from (6.3), the threshold equation (6.4) gives

$$T = \left\lceil \frac{d \log\left(\frac{1-\pi_1}{1-\pi_0}\right) + \log\left(\frac{|\mathrm{e}|}{n-|\mathrm{e}|}\right)}{\log\left(\frac{1-\pi_1}{1-\pi_0}\right) + \log\left(\frac{\pi_0}{\pi_1}\right)} \right\rceil .$$

This was suggested in [Cha17] and it is very consistent with the thresholds that were empirically determined in [CS16] that were optimized for Algorithm 6.2.

With a few exceptions, these are the thresholds we will use in this document. Moreover, a threshold lower than $(d+1)/2$ is generally detrimental for decoding. Therefore, we will take a threshold that is the maximum between $(d+1)/2$ and T in the above formula.

An example of counters distributions are given, for BIKE parameters offering 128 bits of security, in Figure 6.1a in the average case and Figure 6.1b with the condition that $S = 5\,000$, $X = 912$. Given the low proportion of errors, the chart is zoomed to show the intersection of the two curves. The curves giving the expected number of correct positions are thus clipped.

In the average case, using the threshold $T = 42$ would be relevant as it is after the point of intersection of the two curves thus respecting (6.4).

The event $S = 5\,000$, $X = 912$ happens with probability around 10^{-5}. In this case, the previous, non-adaptive threshold would be too low and may cause the algorithm to flip many correct positions.

6.2 Contributions

By observing the iteration by iteration evolution of a decoder, we can see that one of the most detrimental things that can happen is the addition of many errors. When using a purely threshold-based decoder, such as the classic bit-flipping, the errors that are added are so because they had a high counter. Once they have been flipped, if they have a small counter, they will be harder to correct.

We have designed two new types of intermediate algorithms to prevent this from happening. They all take advantage, each in their own way, of the reliability information obtained from counter values, which would otherwise be lost in a typical bit-flipping algorithm.

- The Backflip decoder that encodes this information over time, limiting the duration of each flip the algorithm does.
- Gray decoders that add a level between flipping a bit and not flipping it, and use flip verification steps on this new level.

These new frameworks introduce new parameters that must be chosen for each set of parameters of the cryptosystem. Contrary to the thresholds, these parameters are not systematically associated, in the current state of knowledge, to a theoretical analysis and are chosen by optimization. We try, however, as much as possible, to give an intuition on the influence of each of these parameters.

We will also discuss the step-by-step decoder which is close to the sequential bit-flipping algorithm described in [SS96]. In comparison, this decoder does not use a majority logic but an adaptive threshold and the positions are sampled more efficiently, with an induced bias that we can explain. The importance of this algorithm will be clarified in Chapter 12 where we build a probabilistic model on its evolution in order to estimate its failure rate.

In previous work, we have studied an adaptation to MDPC codes of the two-bit bit-flip of [NV14], generalized to an arbitrary precision. The additional cost in memory and computation, as well as the ad-hoc nature of its specifications, meant that this work was not pursued and will not be presented in this document. For more information we refer the reader to [Vas17]. In a way, there are similarities between this algorithm and Backflip that we present in Chapter 8. Indeed, one of the principles of Backflip is to keep the flips for a shorter period of time when we are less confident in them, this is also a feature of the two-bit bit-flip.

Benchmark. We will assess the performance of the different algorithms by plotting their DFR *versus* the block size r. As seen in §4.1, the IND-CPA security of a QC-MDPC system is essentially achieved by selecting the appropriate error weight t and row weight w. IND-CCA security additionally requires that the decoder has a sufficiently low failure rate. Increasing the block size r is one way to achieve this property and has little effect on the other IND-CCA security-related system properties.

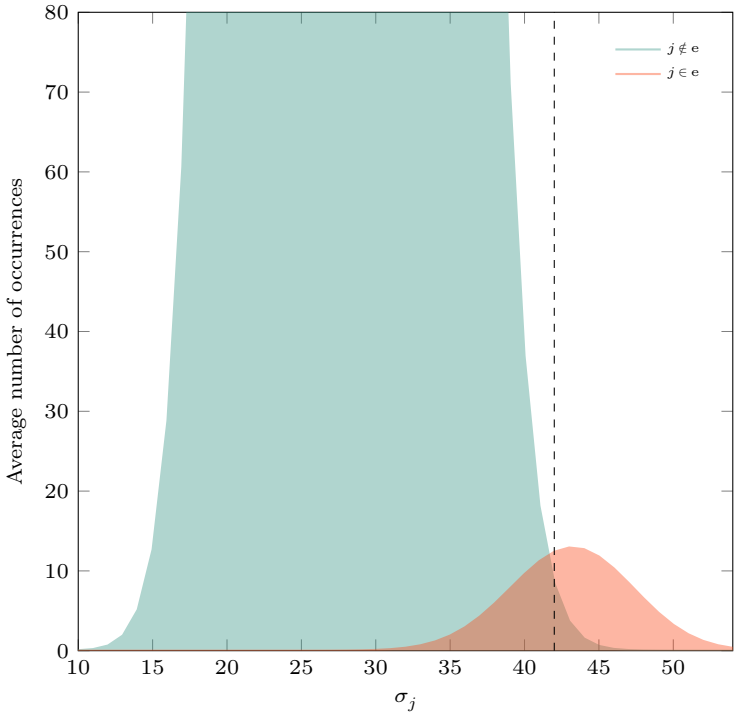
We will use bit-flipping from Algorithm 6.2 and belief propagation from Algorithm 6.4 using Table 6.2 as benchmarks. The former has low complexity but rather poor decoding performance, and for the latter, the opposite applies.

We will refer to the former as "classic" bit-flipping. We use the adaptive thresholds from [Cha17] mentioned in §6.1.3.1. They are adaptive in the sense that they depend on the syndrome weight. The thresholds depend on the probabilities from (5.2), which in turn depend on the parameters d, w, r, n and on certain values that are related to the specific instance we want to decode and the state of the decoder such as the syndrome weight $|\mathrm{s}|$, the error weight $|\mathrm{e}|$ and the value X. The last two values are not information that a decoder can use normally. For this implementation, we replace X by its expected value. And for the error weight, we take the actual value $|\mathrm{e}|$. We know the exact value for our simulation since, contrary to the real world case, we constructed the initial error vector e and we can track any change that increases or decreases its weight. Doing this gives a theoretical curve that does not depend on the quality of the guess of the error weight. In a way, this gives the best possible threshold, it does so by "cheating".

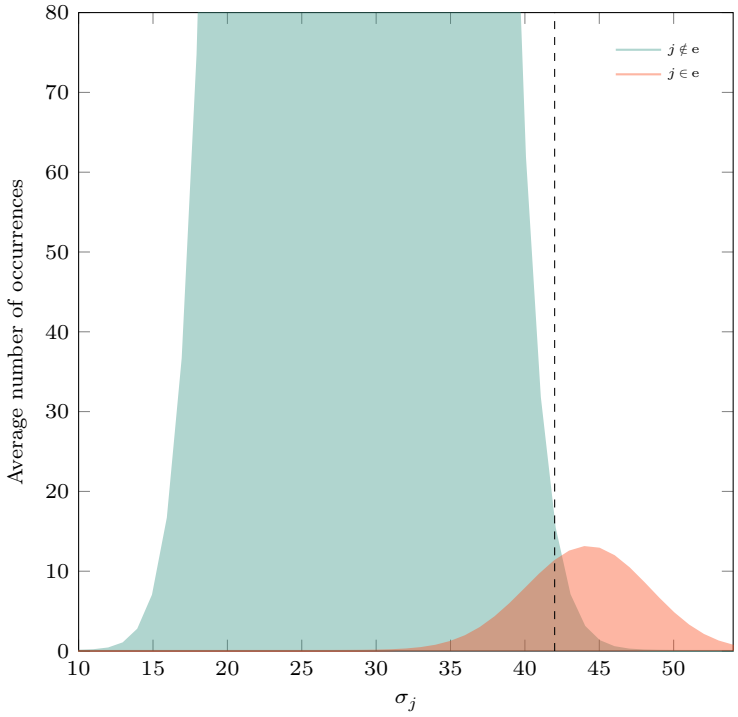
Following [LB18] mentioned in §6.1.2.2, a scaling factor is applied to the messages sent from the check nodes. Here, with BIKE parameters offering 128 bits of security, a scaling factor of $\omega = 0.4$ gave better performance in simulation, and it is therefore this factor that is used for the benchmark.

For the belief propagation curve, we have fewer data points because the computational cost is much higher and the decrease in DFR as block size increases is steeper.

In this document, we will always write algorithms using while loops which can potentially be infinite. In an actual implementation, the number of iterations must be limited and a decoding failure must be explicitly reported.



(a) Average



(b) Conditioned by $|\mathbf{s}| = 5\,000$ and $X = \sum_{j \in \mathbf{e}} \sigma_j - |\mathbf{s}| = 912$

Figure 6.1: Counters distributions and threshold for $(r, d, t) = (12\,323, 71, 137)$

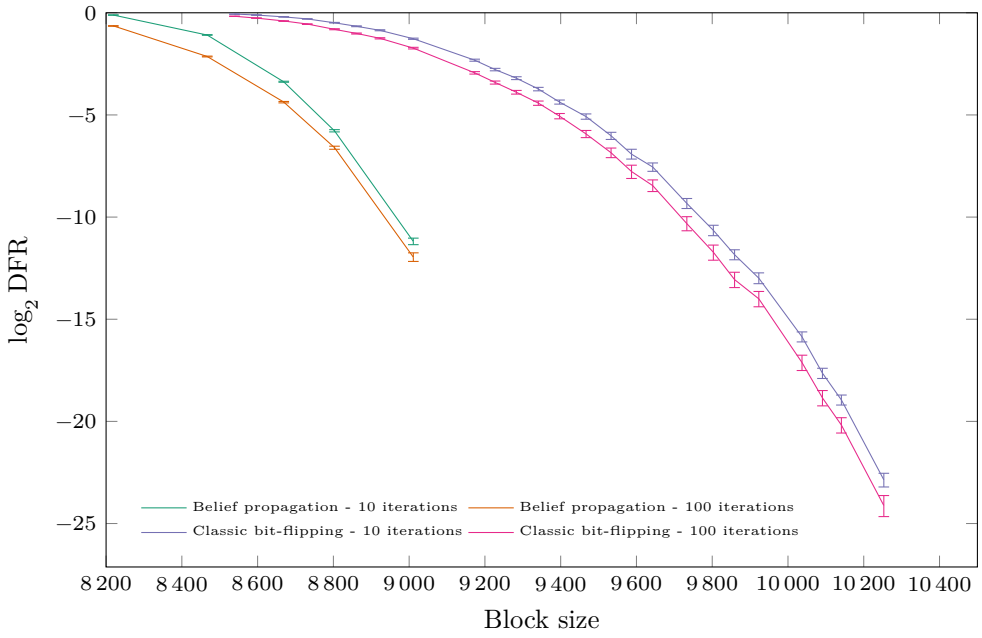


Figure 6.2: DFR *vs.* block size with belief propagation and classic bit-flipping. $(d, t) = (71, 134)$. 99%-confidence intervals.

Chapter 7

Step-by-step

The step-by-step algorithm was originally designed with analyzability in mind. In fact, it is a randomized version of Algorithm 6.3. In comparison, Algorithm 6.2 is parallel: all the counters are computed at once and then all the positions with a counter above the threshold are flipped at once, there is no update of the syndrome between those steps. The state of the decoder (for example in terms of the error weight or the syndrome weight) after one such iteration is hard to predict. In the step-by-step variant that we present here, one position is sampled, its counter is computed and if it is above a threshold it is flipped and the syndrome is updated immediately.

One interesting characteristic of this algorithm is that the range of possible issues after one single flip is rather small: the error weight is adjusted by one unit (positively or negatively) and the syndrome weight by not more than d units. We will see in Chapter 12 that to some extent, one can compute the probability associated to each one of these events. They are in fact the transition probabilities of a Markovian model of the algorithm. We will rely on this model to derive a probability of failure of the algorithm.

Beyond its analysis and although its decoding performance is poorer than, say, classic bit-flipping, it can still be wise to use this algorithm, especially if there are few errors to decode. In this chapter we focus on its design and implementation.

7.1 Definition

The description of the step-by-step decoder is given in Algorithm 7.1.

7.2 Sampling positions

Algorithm 7.1 calls a method to pick one position among the whole set of n positions.

An interesting aspect of a sampling method is the way it affects the counter distributions. The general distributions of the n counters are of course not changed, but as we progress through this section, we will see sampling methods that induce a bias each time greater towards higher counters. As a result, each provides different trade-offs between complexity and the probability of choosing an error.

We will write $p_{\in \mathrm{e}}^{\texttt{sample}}(\sigma)$ the probability that the sampled position is an error and has a counter of σ. Similarly, $p_{\notin \mathrm{e}}^{\texttt{sample}}(\sigma)$ will be the probability that the sampled position is not an error and has a counter of σ.

We present here three different sampling methods.

Algorithm 7.1: Step-by-step bit-flipping

```
function step_by_step_bitflip^⊥(H, s):
    input  : A sparse parity check matrix H ∈ F_2^{(n−k)×n},
             a syndrome s = He^⊤ ∈ F_2^{n−k}.
    output : An error pattern e′ ∈ F_2^n such that He′^⊤ = s.
    e′ ← 0;
    s′ ← s;
    while s′ ≠ 0 do
        T ← threshold(context);
        j ←$ sample(context);
        if |h_j ⋆ s′| ≥ T then
            e′_j ← 1 − e′_j ;
            s′ ← s′ + h_j;
    return e′;
```

7.2.1 Uniform sampling

The simplest possibility that comes to mind is to choose a position uniformly at random. In this case, the general distributions given in §5.2 are only affected by the proportion of errors among the positions:

$$p_{\in \mathrm{e}}^{\mathtt{sample}_0}(\sigma) = \frac{t}{n} \cdot p_{\in \mathrm{e}}(\sigma)\,;$$
$$p_{\notin \mathrm{e}}^{\mathtt{sample}_0}(\sigma) = \frac{n-t}{n} \cdot p_{\notin \mathrm{e}}(\sigma)\,.$$

With this method, the probability of picking an error is $p_{\mathrm{err}}^{\mathtt{sample}_0} = \frac{t}{n}$.

7.2.2 Picking a position in one unsatisfied equation

This method takes advantage of the fact that an unsatisfied equation has at least one error among its positions. The sampling method is therefore

1. randomly pick an unsatisfied equation;
2. randomly pick a position in this equation.

Compared to the previous one, this method induces a bias towards high counters proportional to the counter value. Indeed, a counter is by definition the number of unsatisfied equations in which a position is involved. So the higher the counter is, the better chance it has of being chosen by this method. In terms of probabilities, this translates as:

$$p_{\in \mathrm{e}}^{\mathtt{sample}_1}(\sigma) = \frac{\sigma t}{D_1} \cdot p_{\in \mathrm{e}}(\sigma)\,;$$
$$p_{\notin \mathrm{e}}^{\mathtt{sample}_1}(\sigma) = \frac{\sigma (n-t)}{D_1} \cdot p_{\notin \mathrm{e}}(\sigma)$$

where

$$D_1 = \sum_{\sigma=0}^{d} \sigma \cdot \Big(p_{\in \mathrm{e}}(\sigma) + p_{\notin \mathrm{e}}(\sigma) \Big) = wS\,.$$

With this method, the probability of picking an error is:

$$p_{\text{err}}^{\texttt{sample}_1} = \sum_{\sigma=0}^{d} p_{\in \text{e}}^{\texttt{sample}_1}(\sigma) = \frac{S+X}{wS} \geq \frac{1}{w}\,.$$

7.2.3 Picking a position in two unsatisfied equations

The same idea as the previous one can be generalized to an arbitrary number of equations. In fact, if the sampling method picks a random position involved in exactly k unsatisfied equations, the bias for a counter value σ will be $\sigma(\sigma-1)\cdots(\sigma-k+1)$. Such a sampling method with a large value of k can potentially speed up the process of obtaining a position with a high counter (above the threshold) and thus reduce the number of iterations before a flip.

But increasing k may not always be relevant if one considers that it increases the cost of the sampling process. In this section, however, we focus on the case $k=2$, since its implementation has some interesting links with the spectrum of the parity check matrix discussed in Chapter 15.

To be clear, let us recap the method:

1. randomly pick two different unsatisfied equations;
2. randomly pick a position belonging to both these equations.

Here, the probabilities will be affected quadratically in the counter value.

$$p_{\in \text{e}}^{\texttt{sample}_2}(\sigma) = \frac{\sigma(\sigma-1)\cdot t}{D_2}\cdot p_{\in \text{e}}(\sigma)\,;$$
$$p_{\notin \text{e}}^{\texttt{sample}_2}(\sigma) = \frac{\sigma(\sigma-1)\cdot (n-t)}{D_2}\cdot p_{\notin \text{e}}(\sigma)$$

where

$$D_2 = \sum_{\sigma=0}^{d} \sigma(\sigma-1)\cdot\Big(p_{\in \text{e}}(\sigma)+p_{\notin \text{e}}(\sigma)\Big) = d\cdot(d-1)\cdot(t\pi_1^2+(n-t)\pi_0^2)\,,$$

using the formula for the factorial moments for a binomial distribution.

With this method, the probability of picking an error is:

$$p_{\text{err}}^{\texttt{sample}_2} = \sum_{\sigma=0}^{d} p_{\in \text{e}}^{\texttt{sample}_2}(\sigma) = \frac{t\pi_1^2}{t\pi_1^2+(n-t)\pi_0^2} \geq \frac{1}{1+\frac{t}{n-t}(w-1)^2}\,.$$

As long as $p_{\text{err}}^{\texttt{sample}_1} > \frac{t}{n}$, the probability of picking an error is greater with this method than the previous one

$$p_{\text{err}}^{\texttt{sample}_2} > p_{\text{err}}^{\texttt{sample}_1}\,.$$

Implementing this method for quasi-cyclic matrices. If the same parity check matrix is used more than once, this sampling method can benefit from a precomputation of the intersections between any two of its rows.

In a circulant block of size $r\times r$, say the i_1-th row and the i_2-th row have intersections at positions $j_1,\ldots,j_\ell$. Then if we shift simultaneously the two indices of the rows, the intersecting positions will also be shifted by the same offset: for any δ, the $(i_1+\delta \bmod r)$-th row and the $(i_2+\delta \bmod r)$-th row have intersections at positions $j_1+\delta \bmod r,\ldots,j_\ell+\delta \bmod r$.

Thus, by precomputing all intersecting positions between the first row and any other row, we avoid searching for intersections at each random sampling by using only a small amount of memory. Indeed, if we look at Algorithm 7.2, we can see that $d(d-1)$ intersections have to be computed and stored.

Algorithm 7.2: Sampling a position in two unsatisfied equations.

function `precompute_spectrum`$(r, (\mathrm{h}_0, \mathrm{h}_1))$**:**

input : Block size r, a parity check matrix $(\mathrm{h}_0, \mathrm{h}_1) \in (\mathbb{F}_2[x]/(x^r - 1))^2$.

output : The array of lists spectrum: $(k, \ell) \in \text{spectrum}[\delta]$ *if and only if the rows at indices* 0 *and* δ *of the circulant matrix defined by* h_k *intersect in position* ℓ.

spectrum $\leftarrow \left\{ \emptyset \,\middle|\, i \in \{0, \dots, \lfloor r/2 \rfloor\} \right\}$;

for $k \in \{0, 1\}$ **do**

  foreach $i, j \in \mathrm{Supp}(\mathrm{h}_k)$ **such that** $i < j$ **do**

    $\delta = j - i$;

    if $\delta \leq \lfloor r/2 \rfloor$ **then**

      spectrum$[\delta] \leftarrow$ spectrum$[\delta] \cup \left\{ (k, (r - i) \bmod r) \right\}$;

    $\delta = r - \delta$;

    if $\delta \leq \lfloor r/2 \rfloor$ **then**

      spectrum$[\delta] \leftarrow$ spectrum$[\delta] \cup \left\{ (k, (r - j) \bmod r) \right\}$;

return spectrum;

function $\texttt{sample}_2(r, n, (\mathrm{h}_0, \mathrm{h}_1), \mathrm{s}, \text{spectrum})$**:**

input : Block size r, a parity check matrix $(\mathrm{h}_0, \mathrm{h}_1) \in (\mathbb{F}_2[x]/(x^r - 1))^2$, a syndrome $\mathrm{s} \in \mathbb{F}_2^r$, the precomputed array spectrum $=$ `precompute_spectrum`$(r, (\mathrm{h}_0, \mathrm{h}_1))$.

output : A position involved in at least two unsatisfied equations.

repeat

  Sample (i, j) such that $i < j$ and $s_i = s_j = 1$;

  $\delta \leftarrow j - i$;

  if $\delta \leq \lfloor r/2 \rfloor$ **then**

    start $\leftarrow i$;

  else

    start $\leftarrow j$;

    $\delta \leftarrow r - \delta$;

until spectrum $\neq \emptyset$;

$(k, \ell) \xleftarrow{\$}$ spectrum$[\delta]$;

return $k \cdot r + (\text{start} + \ell) \bmod r$;

Table 7.1: Probability p that a random pair of equations has an empty intersection for [BIKE] parameters and average number of trials $1/p$ before finding one.

Security (bits)	d	Security	r	p	$1/p$
128	71	IND-CPA	10 163	0.63	1.60
		IND-CCA	12 323	0.56	1.80
192	103	IND-CPA	19 853	0.65	1.53
		IND-CCA	24 659	0.58	1.74
256	137	IND-CPA	32 749	0.68	1.47
		IND-CCA	40 973	0.60	1.67

Overhead of this sampling method. Given the sparsity of the parity check matrix of an MDPC code, it is not guaranteed that any pair of equations will have a nonempty intersection. This incurs extra costs as the sampling method might have to restart several times before giving a position as we can see in the loop of Algoritm 7.2.

It is important to notice that, for any circulant matrix, there is a bijection between its row vectors and its column vectors which simply consists in reversing the vectors as can be seen in Definition 1.30. We can thus refer to Chapter 15 in which we study the influence of intersections between the columns of a quasi-cyclic matrix on the DFR. In particular, we can pick some combinatorial results from this chapter. For now, it is only important to know that, in a circulant block, two rows at distance δ from each other have a nonempty intersection if and only if δ is in the spectrum[1] of the circulant block in question.

And using Corollary 15.19, we can say that for a circulant block of size $r \times r$ and column weight d, the probability that a distance is in the spectrum of that block is

$$1 - \frac{\binom{r-d-1}{d-1}}{\binom{r-1}{d-1}}.$$

So, now considering the full double circulant parity check matrix, the probability that two rows have at least one intersection is:

$$1 - \left(\frac{\binom{r-d-1}{d-1}}{\binom{r-1}{d-1}}\right)^2.$$

Table 7.1 shows that with BIKE parameters, the sampling method needs to be restarted, on average, less than once.

7.3 Performance

Performance of the step-by-step are reported in Figure 7.1. They relate to BIKE level 1 parameters (128 bits of security). As for the classic bit-flipping statistics in this document, it is implemented with the adaptive thresholds from [Cha17], $\texttt{threshold}(\textit{context}) = \texttt{threshold}(|\mathrm{s}|, |\mathrm{e}|) = T$ where T is defined as in §6.1.3.1, and using the expected value for X. Here again, to avoid measuring the quality of the guess on the error weight and rather focus on the algorithm itself, the actual error weight $|\mathrm{e}|$ is used in the threshold formula.

Most of the algorithms we study in this paper are parallel: all the counters are computed at once and the syndrome is computed once for each iteration. On the other hand, the step-by-step algorithm is serial and thus an iteration does not represent the same amount

[1] The formal definition of the spectrum is recalled in Definition 15.6 page 143.

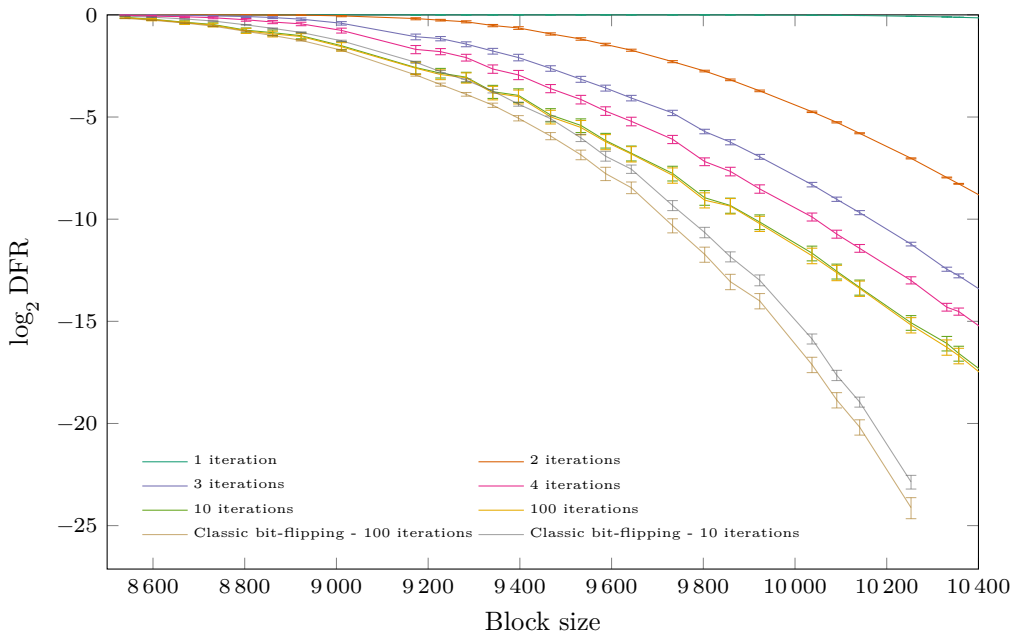


Figure 7.1: DFR *vs.* block size with step-by-step.
$(d, t) = (71, 134)$. 99%-confidence intervals.

of computation in a parallel and in a serial algorithm. In first approximation[2], if we only count the number of counters calculated, a parallel iteration is equivalent to n serial iterations. To make a fair comparison, this is the metric used in figure 7.1.

A point to note is that we are seeing gradual improvements as we increase the number of iterations. However, a plateau is quickly reached. Indeed, 100 iterations do only marginally better than 10 iterations.

7.4 Non-blocking variant

A notable feature of a serial algorithm such as the step-by-step algorithm with thresholds as in §6.1.3.1, compared to a parallel algorithm, is that the weight of the syndrome necessarily decreases with each iteration. Indeed, the chosen thresholds are always greater than $(d+1)/2$. Thus, when we flip a position $j \in \{0, \dots, n-1\}$ with a counter $\sigma_j \geq (d+1)/2$, the weight of the syndrome goes from $|\mathrm{s}|$ to $\left|\mathrm{s} + \mathbf{h}_j\right| = |\mathrm{s}| + d - 2\sigma < |\mathrm{s}| - 1$, using Proposition 1.26. This behaviour is not necessarily found with a parallel algorithm since several columns are flipped at the same time, and therefore, depending on the intersections between these columns, it may happen that these parallel flips increase the weight of the syndrome.

The problem is that the algorithm is then less tolerant to bad flips, those that flip correct positions. If there have been too many bad flips, the error weight will have increased and the syndrome weight will have decreased. As a consequence, all the counters may end up below the threshold. The algorithm will therefore enter a loop where in each iteration it samples a position without ever flipping anything until it fails because the maximum number of iterations has been reached.

[2]In particular, this approximation does not take into account the fact that, on a platform offering vectorized instructions, computing all the counters at the same time may be faster than computing them all individually.

To avoid such a situation we propose a non-blocking variant of the aforementioned step-by-step algorithm. In this variant, presented in Algorithm 7.3, we regularly undo flips in the order they first were performed (flips are stored in a queue).

This cancellation of flips can have two consequences. The first one is that it cancels a good decision, in which case an error is added, and that decision will probably be quickly reversed. The second one, more interesting, is that it cancels a previous bad decision and thus removes an error that may not be removed with a threshold-based approach. The idea of using a queue to keep the list of flips is relevant since the flips of the former type will merely cost us a little time as we move forward in the queue to finally reach the flips of the latter type.

Algorithm 7.3: Non-blocking step-by-step bit-flipping.

```
function nonblocking_step_by_step_bitflip^⊥(H, s):
    input  : A sparse parity check matrix H ∈ F_2^{(n−k)×n},
             a syndrome s = He^⊤ ∈ F_2^{n−k}.
    output : An error pattern e′ ∈ F_2^n such that He′^⊤ = s.
    /* F is a queue containing the flipped position in order.      */
    F ← {};
    /* i is the number of iterations spent without flipping any
       position.                                                   */
    i ← 0;
    e′ ← 0;
    s′ ← s;
    while s′ ≠ 0 do
        T ← threshold(context);
        j ← sample(context);
        if |h_j ⋆ s′| ≥ T then
            if j ∉ F then
                enqueue(F, j);
            else
                remove(F, j);
            e′_j ← 1 − e′_j ;
            s′ ← s′ + h_j;
            i ← 0;
        else
            i ← i + 1;
        if i ≥ max_iter(context) then
            j ← dequeue(F);
            e′_j ← 1 − e′_j ;
            s′ ← s′ + h_j;
            i ← 0;
    return e′;
```

The challenge now is to find the right amount of time to wait before triggering a flip cancellation. In Algorithm 7.3, this is handled by the method `max_iter`.

Constant max_iter(i). We choose a number of iteration and max_iter(i) returns true whenever i is a multiple of this number.

Variable max_iter$(i, |\mathbf{s}|, |F|)$**.** We define this method from a hypothesis testing point of view. First, we write NoFlip^i the event

$$\left\{(j_0, \dots, j_{i-1}) \leftarrow \texttt{sample}(\textit{context})^i, T \leftarrow \texttt{threshold}(\textit{context}), \forall 0 \leq l < i, \left|\mathbf{h}_{j_l} \star \mathrm{s}\right| < T\right\} .$$

This corresponds to the case where the algorithm iterates i times without flipping any position since all sampled positions have a counter below the threshold.

We always suppose that the following hypothesis is true:

$$\mathcal{H}_0 : |\mathrm{e}| = t - |F| \ .$$

In other words, $\mathcal{H}_0$ assumes that all active flips have removed an error.

However, when i iterations have passed without a single flip, we "reject" this hypothesis if the event NoFlip^i has a low probability α (e.g. $\alpha = 0.05$). Rejecting the hypothesis $\mathcal{H}_0$ here signifies that there was at least one flip in the queue F that added an error. In this case, we cancel the first flip in F, *i.e.* the one that was performed the earliest in the execution of the algorithm.

Actual implementation of this algorithm relies on the adaptive threshold T computation explained in §6.1.3.1. In the formula, we can use the syndrome weight $|\mathrm{s}|$, and, under hypothesis $\mathcal{H}_0$, we assume $|\mathrm{e}| = t - |F|$ in the formula.

The probability of NoFlip^i is found assuming independence of the sampled counters, using the sampled counters distributions $p_{\in \mathrm{e}}^{\texttt{sample}_i}(\sigma)$ and $p_{\notin \mathrm{e}}^{\texttt{sample}_i}(\sigma)$ for $\sigma \in \{0, \dots, d\}$ detailed above for $i = 0, 1, 2$. We thus choose the value of $\texttt{max\_iter}(\textit{context})$ to be the smallest integer i such that

$$\left(\sum_{\sigma=0}^{T-1} p_{\in \mathrm{e}}^{\texttt{sample}_i}(\sigma) + p_{\notin \mathrm{e}}^{\texttt{sample}_i}(\sigma)\right)^i < \alpha \, .$$

Here also, in the formula, we can use the syndrome weight $|\mathrm{s}|$, and, under hypothesis $\mathcal{H}_0$, we assume $|\mathrm{e}| = t - |F|$.

Further refinement of this algorithm and its reinsertion into a parallel paradigm resulted in the more performant Backflip algorithm presented in Chapter 8.

Chapter 8

Backflip

In this chapter, we describe a novel way of including soft information in a bit-flipping algorithm. Counters provide information about the reliability of positions. Classic bit-flipping sets a threshold and flips all positions above this threshold. Our algorithm, Backflip, adds a finer granularity by bringing a concept of flip duration. The more reliable a flip is, the longer it remains active.

Decoding is seriously hindered by correct positions which have a high counter. Once flipped, they become errors with a rather low counter. If this phenomenon occurs several times during the same decoding, the algorithm will have effectively produced an instance whose errors are difficult to detect so that any subsequent iteration will not be able to improve the situation. Backflip solves this problem by ensuring that each flip is temporary.

With classic bit-flipping, the only way to reverse a bad decision, and thus to deflip an initially correct position, is if the iterations posterior to these bad flips have sufficiently modified the syndrome. This way, the distributions of the counters have been sufficiently modified so that the counters of incorrectly flipped positions are once again above the threshold. With Backflip, we have a second mechanism that, when decoding starts to stall, progressively cancels the old unreliable flips.

Our algorithm, introduces a new concept for bit-flipping: the duration of a flip, which we call *ttl* for "time to live". The intuition is that the algorithm should be more prone to quickly undo a less reliable flip and conversely to keep a very reliable flip for a longer period of time. Since the higher a counter is, the more likely it is to indicate an error, the ttl should be an increasing function of the counter. It should also be at least equal to one when a counter is above the threshold. We will explore two strategies for the ttl and the thresholds, which, as we will see, are closely related.

As mentioned in §7.4, the idea of unconditionally canceling old flips originated from the non-blocking variant of the step-by-step decoder. In addition to the fact that this algorithm is parallel, it also differs in the fact that it actually uses reliability information.

8.1 Algorithm description

The full description of Backflip is given in Algorithm 8.1. It must keep track of active flips and associate to each one of them their time to live. At the beginning of each iteration, the times to live are decremented and flips that have expired are canceled.

The two functions `threshold` and `ttl` are specified generically and are discussed later in this chapter. The former may depend on any element of the context of the algorithm at runtime, we will see here examples using the current syndrome weight $|s'|$ and the number of active flips $|e'|$. The latter uses a counter, which, remember, gives information about the reliability of the flip.

Algorithm 8.1: Backflip.

```
function backflip^⊥(H, s):
    input  : A sparse parity check matrix H ∈ F_2^{(n−k)×n},
             a syndrome s = He^⊤ ∈ F_2^{n−k}.
    output : An error pattern e′ ∈ F_2^n such that He′^⊤ = s.
    e′ ← 0;
    s′ ← s − He′^⊤;
    /* Time-to-live of flips.                                   */
    D ← 0;
    while s′ ≠ 0 do
        /* Undo flips reaching their time-of-death.             */
        for j ∈ {0, ..., n − 1} do
            D_j ← D_j − 1 ;
            if D_j = 0 then e′_j ← 0 ;
        s′ ← s − He′^⊤;
        T ← threshold(context);
        for j ∈ {0, ..., n − 1} do
            if |h_j ⋆ s′| ≥ T then
                e′_j ← 1 − e′_j;
                D_j ← ttl(|h_j ⋆ s′|)
        s′ ← s − He′^⊤;
    return e′;
```

Overhead compared to classic bit-flipping. First, the algorithm needs additional memory to store the longevity of the flips. An obvious upper bound on the space taken is simply

$$\log_2(\text{max\_ttl}) \cdot n$$

bits. For a constrained hardware implementation, this memory could be reduced by adopting a sparse representation of the array D.

Regarding time complexity, this algorithm needs two syndrome computations for each iteration: one after some flips have been canceled and one to check if the looping condition is verified.

Constant time considerations can be taken into account by setting an upper bound that is sufficiently high for the size of D. Indeed, a decoding for which the number of active flips is, say, a multiple of the original error weight t fails anyway with a probability very close to 1.

8.2 Threshold and time-to-live

The `threshold` and the `ttl` functions are closely linked and their choice must be made jointly in order to achieve good decoding performance. Two strategies are presented here, the first was introduced in [SV20a] and used in [BIKE][1], and the second one takes a different approach that achieves better decoding performance.

[1] Backflip decoder was suggested in version 3 of the specification.

8.2.1 Optimistic threshold strategy

We present here a heuristic for thresholds based on the rationale that being overly optimistic about the weight of errors, *i.e.* assuming that each flip has removed an error, slows down the algorithm when it makes bad decisions. It slows the algorithm down by making fewer new flips, but those that are actually performed are more reliable, and meanwhile, the algorithm will undo some old flips that were potentially bad decisions thus improving the overall situation. Indeed, an optimistic estimation of the error weight would bring it to the initial error weight t subtracted by the number of active flips $|\mathrm{e}'|$. We will see that using the adaptive threshold explained in §6.1.3.1 while underestimating the current error weight increases the threshold.

Threshold selection rule. Here we have $\texttt{threshold}(\mathit{context}) = \texttt{threshold}(|\mathrm{s}|, t - |\mathrm{e}'|) = T$ where T is defined as in §6.1.3.1 with π_0 and π_1 calculated by substituting t for $t - |\mathrm{e}'|$ and S for $|\mathrm{s}'|$:

$$T = \left\lceil \frac{d \log\left(\frac{1-\pi_1}{1-\pi_0}\right) + \log\left(\frac{n-\left(t-|\mathrm{e}'|\right)}{n-|\mathrm{e}|}\right)}{\log\left(\frac{1-\pi_1}{1-\pi_0}\right) + \log\left(\frac{\pi_0}{\pi_1}\right)} \right\rceil . \tag{8.1}$$

Since the current error weight is guessed optimistically, it can happen that $\pi_1 > 1$ in which case the previous equation would not make sense. In this case, we take the limit of (8.1) when π_1 tends to 1, that is d.

Optimistic estimation of t gives a higher threshold. Remember from §6.1.3 that, with adaptive thresholds, we have

$$\pi_0 = \frac{(w-1)\,|\mathrm{s}'| - X}{d(n - |\mathrm{e} + \mathrm{e}'|)}, \qquad \pi_1 = \frac{|\mathrm{s}'| + X}{d\,|\mathrm{e} + \mathrm{e}'|}$$

where $X = \sum_{j \in \mathrm{e}+\mathrm{e}'} \sigma_j - |\mathrm{s}'|$.

Since $X \ll (w-1)\,|\mathrm{s}'|$ and $t \ll n$, underestimating $|\mathrm{e} + \mathrm{e}'|$ will always give an estimate of π_0 close to the true value. However, π_1 will tend to be overestimated.

In this case, the estimated distribution of the correct position counters would be close to reality and the estimated distribution of the error counters would be shifted to the right. Since the threshold is chosen as the smallest value at which the former is below the latter, the underestimation of the error weight would choose a threshold above the optimal.

Time-to-live. We want the ttl to be an increasing function of the counter, we also want it to be nonzero when a counter is above the threshold. We choose a heuristic that verify those properties and also has only a few parameters: an affine function of the difference between the counter and the threshold. We write this difference $\delta = \sigma - T$, then

$$\texttt{ttl}(\sigma) = \max(1, \lfloor a\,\delta + b \rfloor) .$$

To highlight the importance of setting the ttl function, we show in Figure 8.1 its effect on the DFR for different values of a and b. The appropriate value range seems for a to be around 0.5 and b around 1. If we deviate too much from these values, it could be detrimental for the DFR to the point that we can see a plateau.

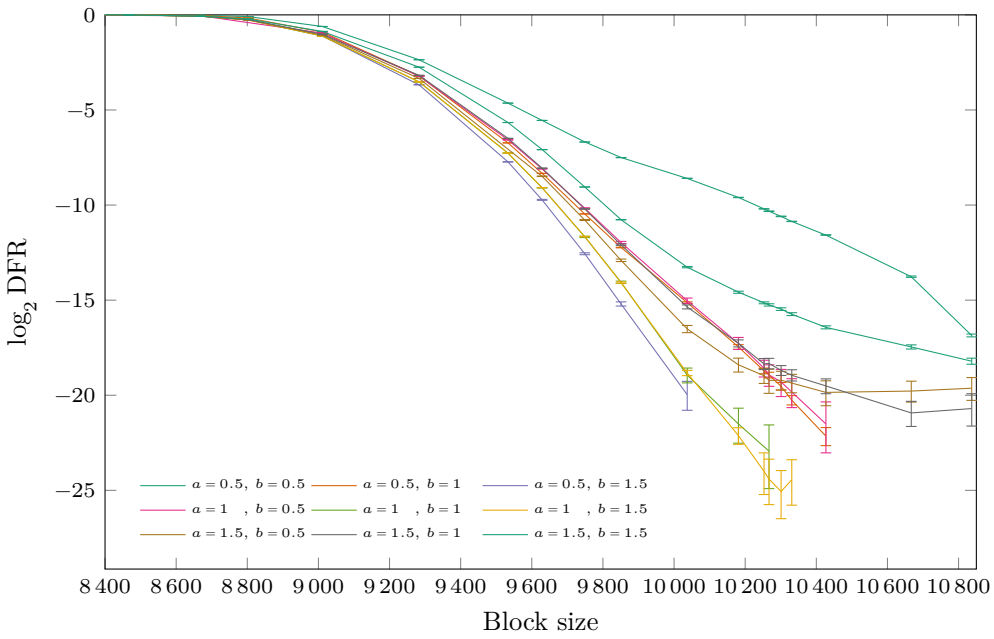


Figure 8.1: DFR *vs.* block size with Backflip (9 iterations) for different affine `ttl` functions $\texttt{ttl}(\sigma) = \max(1, \lfloor a\,\delta + b \rfloor)$).
$(d, t) = (71, 134)$. 99%-confidence intervals.

Optimization. As this strategy is heuristic, its specification is based on an optimization from simulation in the absence of theoretical analysis. In practice, we proceed by using the algorithm under degraded conditions by reducing the block size and limiting the number of iterations so that a DFR can be effectively measured in a reasonable number of simulation. The values for a and b are then determined by applying an optimization technique such as the Nelder-Mead method [NM65] that minimizes the DFR under these degraded conditions. Values obtained for [BIKE] parameters are reported in Table 8.1.

Table 8.1: `ttl` parameters obtained by optimization for BIKE parameters.

Security (bits)	a	b
128	0.45	1.1
192	0.36	1.41
256	0.45	1

Performance. The performance of the algorithm for BIKE parameters for 128 bits of security with the `ttl` function described above, found by optimization, is displayed in Figure 8.2. A notable fact is that, unlike the other bit-flip-derived algorithms seen previously, with this algorithm a gain is observed each time the number of iterations is increased, it appears that we still did not reach a limit after 100 iterations. Another noteworthy point is that it does better in 100 iterations than scaled belief propagation ([LB18] with $\omega = 0.4$), which was our benchmark.

However, we observe an inflection point on the DFR curves corresponding to a small number of iterations.

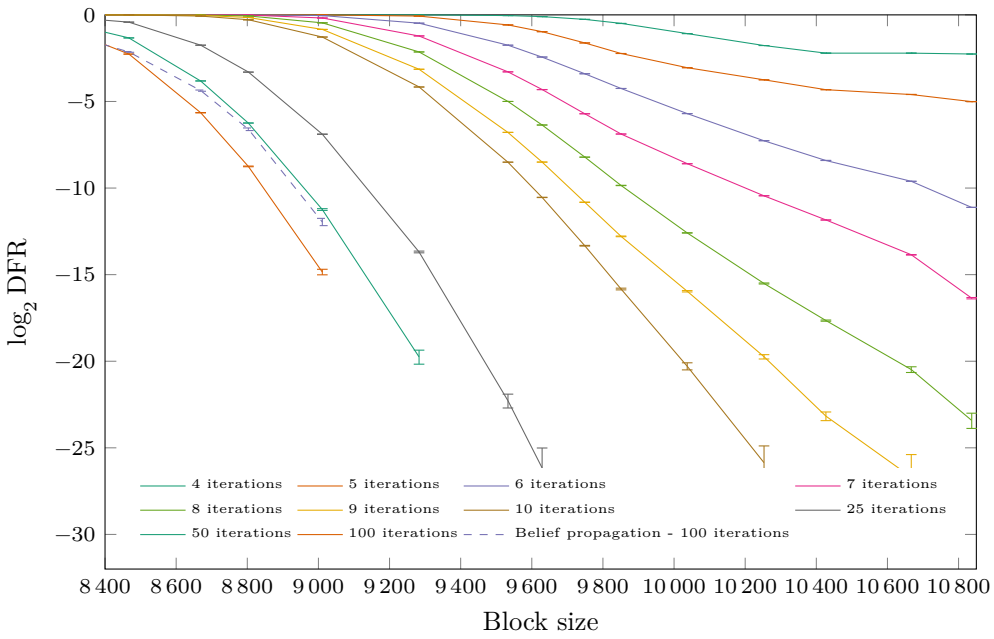


Figure 8.2: DFR *vs.* block size with Backflip. $\texttt{ttl}(\delta) = \max(1, \lfloor 0.45\,\delta + 1.1 \rfloor)$ $(d, t) = (71, 134)$. 99%-confidence intervals.

8.2.2 Multiple thresholds strategy

This strategy follows from the above remark that, when choosing a threshold, two probabilities π_0 and π_1 are estimated. If the current error weight is misestimated, then π_1 is also misestimated, but it has little influence on the estimate of π_0.

The rationale for this threshold selection strategy is therefore to rely only on π_0.

Threshold selection rule. Indeed, rather than finding, as in the previous one, a balance between good and bad decisions that ultimately reduces the error weight, we choose a threshold that simply avoids making bad decisions.

Still in the model explained in §5.2, the expected number of good positions with a counter value of σ is approximately

$$n \binom{d}{\sigma} \pi_0^{\sigma} (1 - \pi_0)^{d-\sigma} .$$

We therefore choose $\texttt{threshold}(\textit{context}) = \texttt{threshold}(|\mathrm{s}|) = T$ where T is the smallest integer such that:

$$n \binom{d}{T} \pi_0^{T} (1 - \pi_0)^{d-T} < \alpha$$

for some fixed value α.

Time-to-live. We apply the same idea, namely that we want to avoid making bad decisions. In fact, the less likely a flip is to add an error, the longer it stays active. To do this, we define several thresholds $T = T_1, T_2, \ldots, T_\ell$ such that for all $i \in \{1, \ldots, \ell\}$, T_i is the

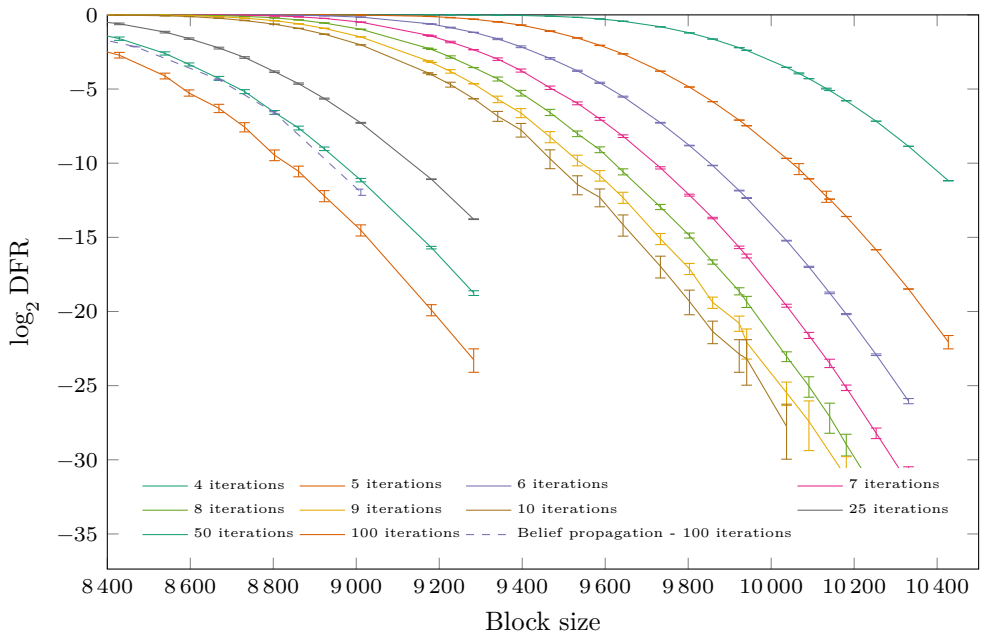


Figure 8.3: DFR *vs.* block size with Backflip with multiple thresholds. $\alpha_1 = 8$, $\alpha_2 = 2$, $\alpha_3 = 1/2$, $\alpha_4 = 1/8$, $\alpha_5 = 1/32$.
$(d, t) = (71, 134)$. 99%-confidence intervals.

smallest integer such that

$$n\binom{d}{T_i}\pi_0^{T_i}(1-\pi_0)^{d-T_i} < \alpha_i$$

for some constants $\alpha = \alpha_1 > \alpha_2 > \cdots > \alpha_\ell > 0$.

Finally, we choose $\texttt{ttl}(\sigma)$ as

$$\texttt{ttl}(\sigma) = \max_i\{i \,|\, \sigma \geq T_i\}\,.$$

Performance. As an example, we provide in Figure 8.3 the DFR curve for BIKE parameters offering 128 bits of security when implementing this method with $\alpha_1 = 8$, $\alpha_2 = 2$, $\alpha_3 = 1/2$, $\alpha_4 = 1/8$, $\alpha_5 = 1/32$, *i.e.* when it follows a geometric progression of common ratio 1/4. As with the other strategy, each additional iteration brings a noticeable gain in the performance. And here again, the algorithm does better than the benchmark. However, interestingly, we do not observe any inflection point in this case.

Chapter 9

Gray decoders

In this chapter we will discuss some bit-flipping variants following a similar framework: the gray decoders. They provide a way to introduce soft information in bit-flipping. Rather than going through all the positions and making the binary decision to flip or not, they add an intermediate level of decision. Positions with a high counter but still below the threshold are neither black nor white but gray. Gray decoders can make immediate decisions on black positions (the probable errors), but for gray positions they may perform reverifications.

Reverifications are a powerful mechanism of gray decoders. They consist in, after performing several flips, immediately recomputing the corresponding counters to verify that they actually remain below the threshold. It allows immediate correction of most bad decisions.

In a bit-flipping algorithm, the first iteration is probably the most important, and it is also the one for which we can extract the most reliable information. Indeed, for the first iteration, the weight of the error is fixed and known, it is the parameter that we called t, and since the error pattern is drawn uniformly at random, it does not suffer from any bias that can sometimes contradict the models on the counters.

Contrary to classic bit-flipping, in gray algorithms, this first information obtained from the counters is exploited a bit more by not reiterating immediately. The positions with a high counter but below the threshold are marked to receive a specific treatment later, before moving on to the next iteration. Thus, compared to classic bit-flipping, some additional operations are performed, they concern a smaller set of positions, with a higher concentration of errors. The high concentration of errors in this set together with the reverifications generally allows to adopt a very simple but very efficient threshold strategy.

9.1 Framework

We will detail different alternatives through the structure of Algorithm 9.1. We consider a classification that will be done by means of counters. Many alternatives can be considered for further processing after flipping the black positions, we will see that the verification method which will be explained here can be used as a building block for a particularly efficient decoding.

9.2 Reverifications

One key aspect of the gray decoders is the reverification steps.

After an iteration of parallel bit-flipping, we can observe two types of behaviour for the counters for flipped positions. The counters corresponding to good flips will be greatly

Algorithm 9.1: Gray bit-flipping.

```
function gray_bitflip^⊥(H, s):
    input  : A sparse parity check matrix H ∈ F_2^{(n−k)×n},
             a syndrome s = He^⊤ ∈ F_2^{n−k}.
    output : An error pattern e′ ∈ F_2^n such that He′^⊤ = s.
    e′ ← 0;
    s′ ← s;
    it ← 1;
    while s′ ≠ 0 do
        G, B ← classify(H, s′);
        for j ∈ B do
            e′_j ← 1 − e′_j ;
        s′ ← s − He′^⊤;
        process(G, B, it);
        it ← it + 1;
    return e′;
```

decreased while those corresponding to bad flips will not. This can be observed in Figure 9.1 for a fixed threshold.

We can thus improve the performance of the algorithm by filtering the flipped positions. After a bit-flipping iteration, we perform a verification step which consists in

- recalculating the counters of the flipped positions,
- among those, flipping those with a counter above a threshold T_v.

In our experiments, it was not necessary to choose a threshold in a sophisticated way and taking $T_v = (d+1)/2$ brought us good results. As we can see in the Figure 9.1 for example, most flipped errors should have a counter below this threshold, and the errors that were added would be above it with a high probability. Some good flips could be reverted and some bad flips could be kept, but overall the situation is better.

The same principle can be applied to the gray positions, *i.e.* those with a counter above some threshold $T_g < T$. The performance of such a method for different block sizes are compared in Figure 9.2. Two observations can be made from this data:

- the reverifications lead to a decrease of $\approx 30\%$ in the error rate compared to the classic bit-flipping, for large block sizes,
- iterating the process more than once is only interesting for smaller block sizes.

9.3 Simple definition

Simply applying the reverification idea to the framework we obtain the algorithm specified in Algorithm 9.2 for some gray threshold function $\texttt{threshold}_g$ and a number I_G of gray iterations. This algorithm appeared in one of our previous works [Vas17]. In this document we will focus on the variants presented in [DGK20b], as they have shown interesting performance, have a constant-time implementation and are now included in the specification [BIKE].

Algorithm 9.2: Simple gray decoder.

```
function classify(H, s'):
    T ← threshold(context);
    T_g ← threshold_g(context);
    for j ∈ {0, ..., n − 1} do
        if |h_j ⋆ s'| ≥ T then
            B ← B ∪ {j};
        else if |h_j ⋆ s'| ≥ T_g then
            G ← G ∪ {j};
    return G, B;
function process(G, B, it):
    for i ∈ {1, ..., I_G} do
        for j  G ∪ B do
            if |h_j ⋆ s'| ≥ (d + 1)/2 then
                e'_j ← 1 − e'_j ;
        s' ← s − He'^⊤;
```

9.4 Variants from Drucker, Gueron & Kostic

In [DGK20b], four variants of gray decoders were defined, their definitions are synthesized in Algorithm 9.3. The authors suggest using $\delta = 3$ as the fixed difference between the black threshold and the gray threshold: $T_g := T - \delta$.

Comparison. To compare the different variants, we consider one iteration at each update of the syndrome that forces the recalculation of the counters. This corresponds to the number of iterations of the *while* loop plus the number of calls to the `iterate` function.

Threshold. Authors of [DGK20b] proposed to use the thresholds used for the One-Round decoder in [BIKE] for versions prior to 3.2. These thresholds were defined as the image of an affine function of the syndrome weight. Indeed, when we restrict the domain to a range of syndrome weight around its average value, the threshold formula approaches an affine function. The actual threshold function was determined using the method of least squares on that range. Functions thus obtained for BIKE parameters offering 128 bits of security but for different block sizes r are presented in Table 9.1.

Furthemore, to avoid a potential problem identified through the study of error floors that we will see in Chapter 16, we set a lower bound on the threshold:

$$\texttt{threshold}(|s'|) = \max\left((d+1)/2, \lfloor a\,|s'| + b \rfloor\right) .$$

In Figure 9.4, we can see the effects that the choice of the threshold function has on the DFR curve. In particular, we can see that it has an influence on the point where the DFR starts to decrease sharply (the *waterfall*) and on the slope beyond this point. It appears that the larger the block size r chosen to affinely approximate the threshold function, the later the waterfall. And with a small block size r the slope will not be as steep.

Performance. The performance of the BGF algorithm for BIKE parameters for 128 bits of security is displayed in Figure 9.5. We can see that a plateau seems to be reached with

Algorithm 9.3: Variants from [DGK20b].

```
function classify(H, s'):
    T ← threshold(context);
    for j ∈ {0, ..., n − 1} do
        if |h_j ⋆ s'| ≥ T then
            B ← B ∪ {j};
        else if |h_j ⋆ s'| ≥ T − δ then
            G ← G ∪ {j};
        return G, B;
function iterate(S):
    for j ∈ S do
        if |h_j ⋆ s'| ≥ (d + 1)/2 then
            e'_j ← 1 − e'_j ;
    s' ← s − He'^⊤;
function process_B(G, B, it):
    iterate(B);
function process_BG(G, B, it):
    iterate(B);
    iterate(G);
function process_BGF(G, B, it):
    if it = 1 then
        iterate(B);
        iterate(G);
function process_BGB(G, B, it):
    iterate(B);
    if it = 1 then
        iterate(G);
```

7 iterations. Any additional iteration only marginally improves performance and doing 100 iterations is almost as good as doing 10 iterations.

9.5 Sorting variant

Algorithm 9.4 also uses the principle of gray decoding but is not based on computing thresholds. The idea is first to sort the positions in descending order of their counters. Then we define a gray area that contains the N_g positions that have the largest counters because they are the most likely to be errors. We then iterate on these positions always keeping the same order defined by the initial counters. For each position we compute the corresponding counter and immediatly flip it and update the syndrome if it is over the majority vote threshold $(d+1)/2$.

As the first positions have the highest counters, they are all the more likely to be errors. This is why this algorithm will, in a first phase, eliminate many errors. The error weight will then be further reduced with reverifications.

As only a few errors are expected to remain after this process, a step-by-step decoder is used as a final step.

Table 9.1: Affine threshold functions for $(d, t) = (71, 134)$.

r	d	a	b
9 800	71	0.00726021	13.5491
9 900	71	0.00717314	13.5541
10 000	71	0.00709255	13.5445
10 100	71	0.00701653	13.5273
10 163	71	0.0069722	13.53
10 200	71	0.006946	13.4979
10 300	71	0.00687955	13.4617
10 400	71	0.00681805	13.414
10 500	71	0.00675966	13.3623

Algorithm 9.4: Sorted bit-flipping.

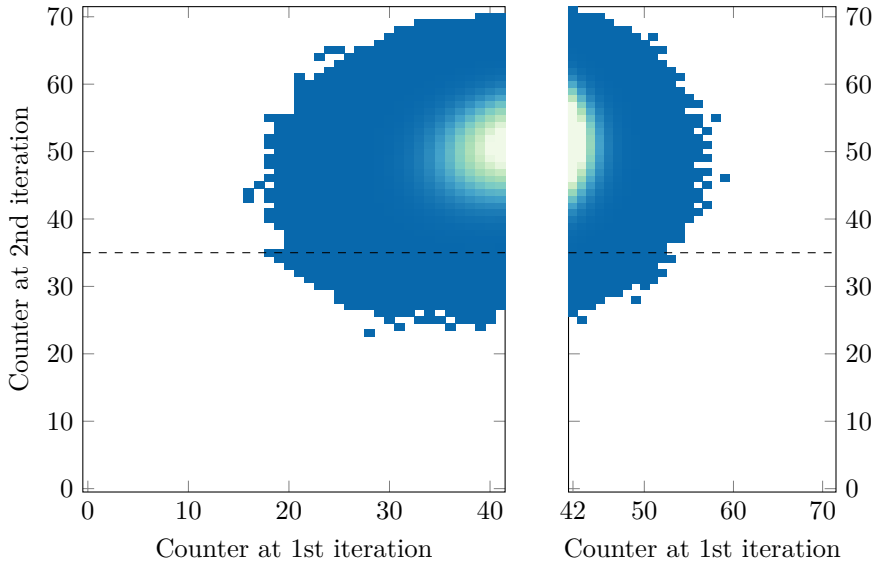
function `sorted_gray_bitflip`$^{\perp}(\mathbf{H}, \mathrm{s})$**:**
- **input :** A sparse parity check matrix $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}$, a syndrome $\mathrm{s} = \mathbf{H}\mathrm{e}^\top \in \mathbb{F}_2^{n-k}$.
- **output :** An error pattern $\mathrm{e}' \in \mathbb{F}_2^n$ such that $\mathbf{H}\mathrm{e}'^\top = \mathrm{s}$.

$\mathrm{e}' \leftarrow 0$;
$\mathrm{s}' \leftarrow \mathrm{s}$;
Sort positions $j_0, j_1, \dots, j_{n-1}$ so that $\forall i < i', \left|\mathbf{h}_{j_i} \star \mathrm{s}\right| \geq \left|\mathbf{h}_{j_{i'}} \star \mathrm{s}\right|$;
for $\ell = 1, \dots, N$ **do**
  for $i = 0, \dots, N_g$ **do**
    if $\left|\mathbf{h}_{j_i} \star \mathrm{s}'\right| \geq (d+1)/2$ **then**
      $e'_j \leftarrow 1 - e'_j$;
      $\mathrm{s}' \leftarrow \mathrm{s}' + \mathbf{h}_j$;
return $\mathrm{e}' +$ `step_by_step_bitflip`$^{\perp}(\mathbf{H}, \mathrm{s}')$;

Figure 9.6 shows the performance of this algorithm with $N_g = n/10$ and $N = 10$. It therefore requires the computation of n counters in the first gray step and a few more that come from the step-by-step algorithm.

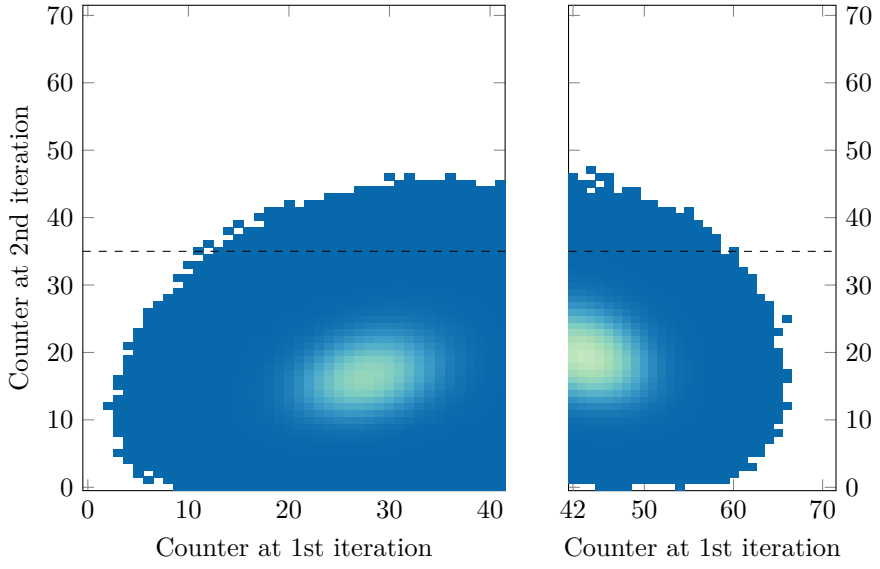
The additional cost of sorting positions must also be taken into account. Finding the N_g positions with the largest counters can be done with a max-heap in time in $O(n \log(N_g))$. Sorting is done in time $O(N_g \log(N_g))$.

Some components of the sorted gray decoder were incorporated in the design of the One-Round algorithm that was chosen in [BIKE] from version 1 to version 3.2.



(a) An error, not flipped

(b) Not an error, flipped



(c) Not an error, not flipped

(d) An error, flipped

Figure 9.1: Evolution of the counters between the first two iterations (fixed threshold $T = 42$).
$(r, d, t) = (12\,323, 71, 134)$ – 10^8 samples.
(White when no data, otherwise the darker the shade the lower the probability.)

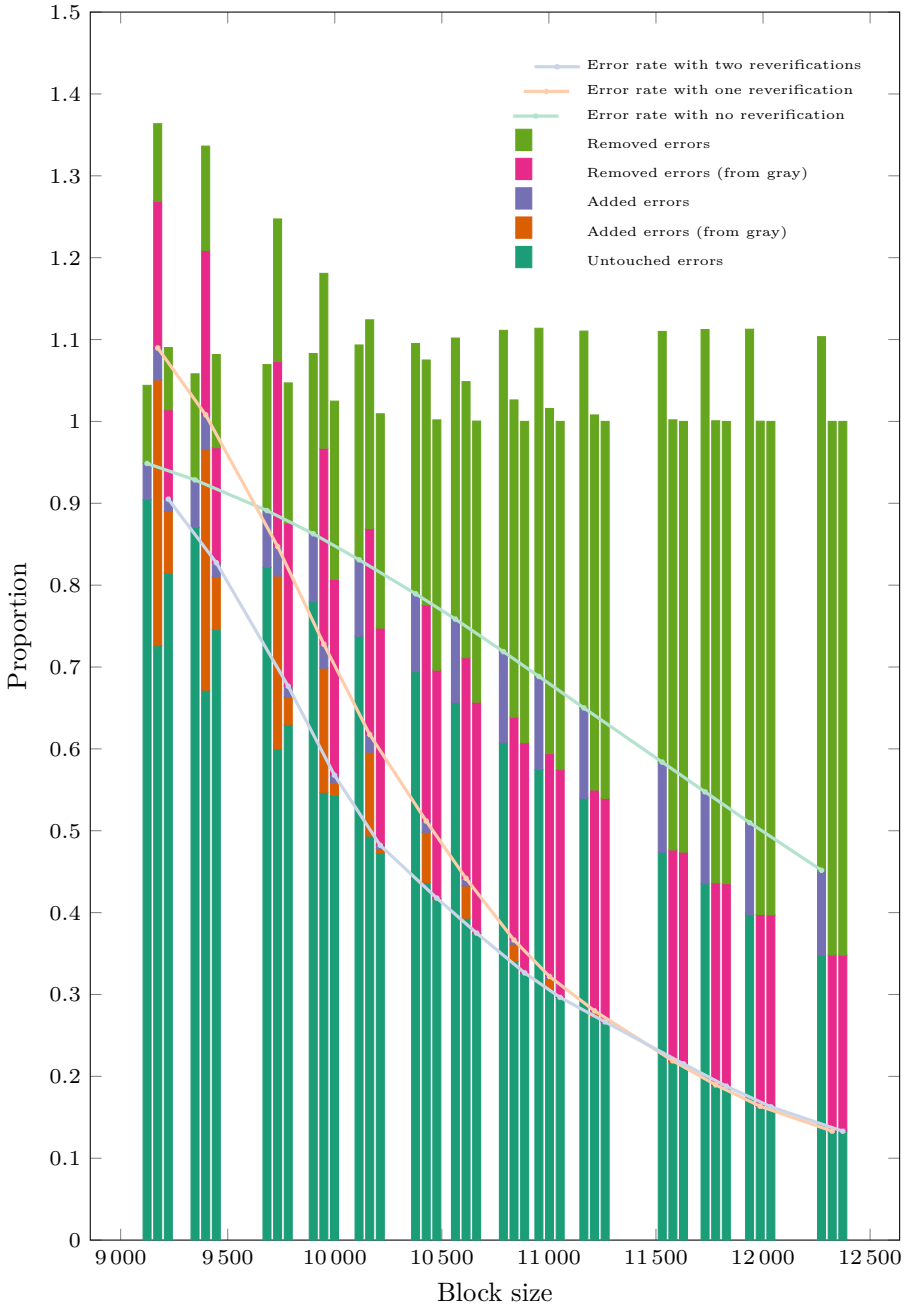


Figure 9.2: Performance of the reverifications with T as in § 6.1.3, $T_g = T - 3$ and $T_v = (d+1)/2$.
$(d, t) = (71, 134)$ – 10^6 samples for each bar.
From left to right, the bars correspond respectively to 0, 1 and 2 reverification steps.

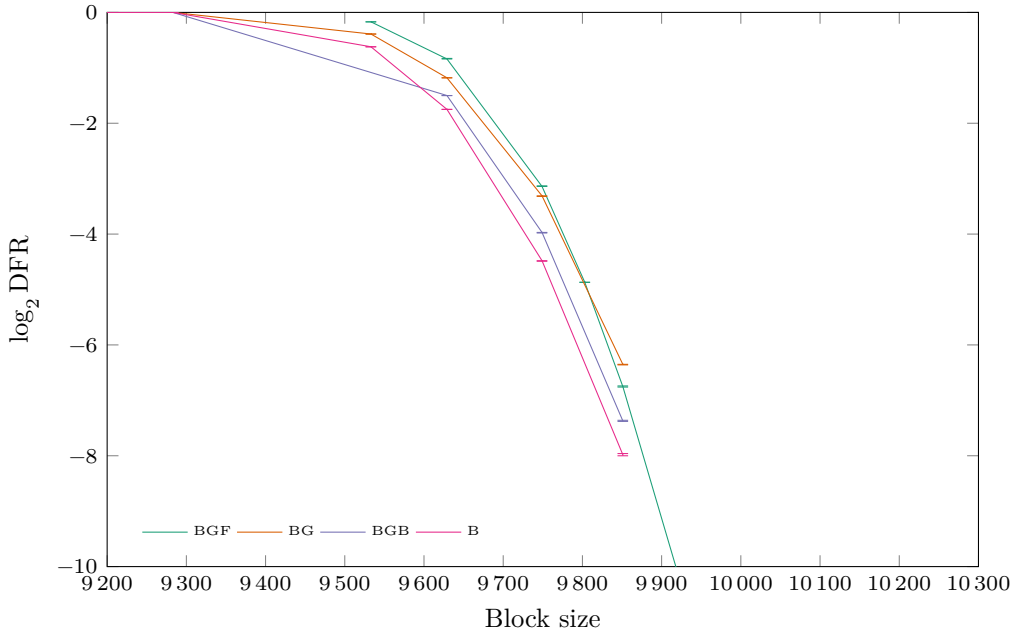


Figure 9.3: DFR *vs.* block size with different gray variants (with at most 9 iterations). $(d, t) = (71, 134)$. 99%-confidence intervals.

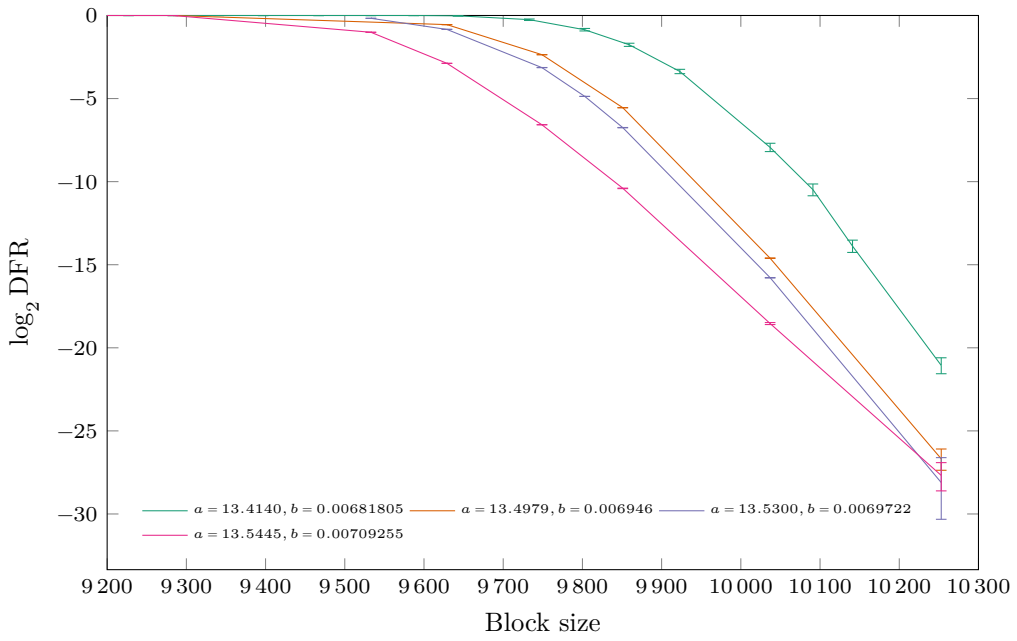


Figure 9.4: DFR *vs.* block size with BGF (9 iterations) for different threshold functions $\texttt{threshold}(|s'|) = \max\left((d+1)/2, \lfloor a\,|s'| + b \rfloor\right)$.
$(d, t) = (71, 134)$. 99%-confidence intervals.

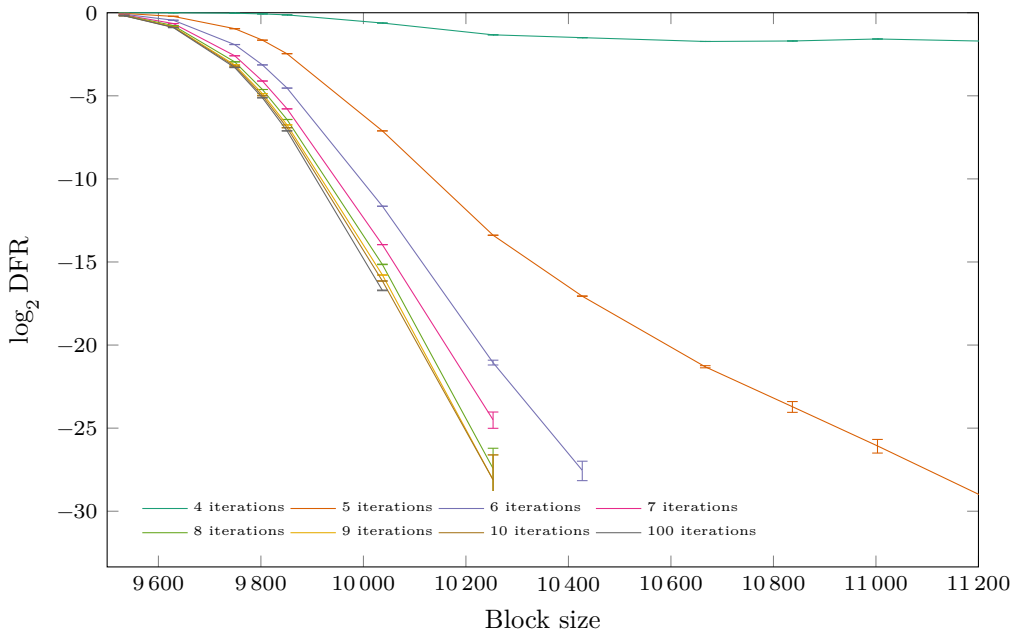


Figure 9.5: DFR *vs.* block size with BGF.
$\texttt{threshold}\left(|s'|\right) = \max\left((d+1)/2, \left\lfloor 0.0069722\,|s'| + 13.53 \right\rfloor\right)$.
$(d,t) = (71, 134)$. 99%-confidence intervals.

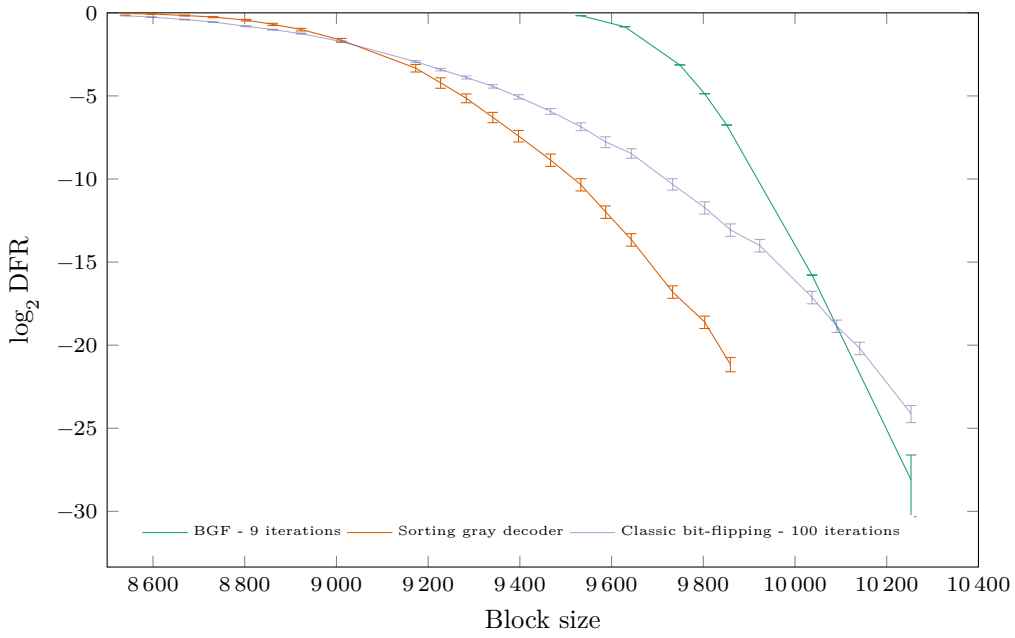


Figure 9.6: DFR *vs.* block size with sorting variant of the gray decoder.
$(d,t) = (71, 134)$. 99%-confidence intervals.

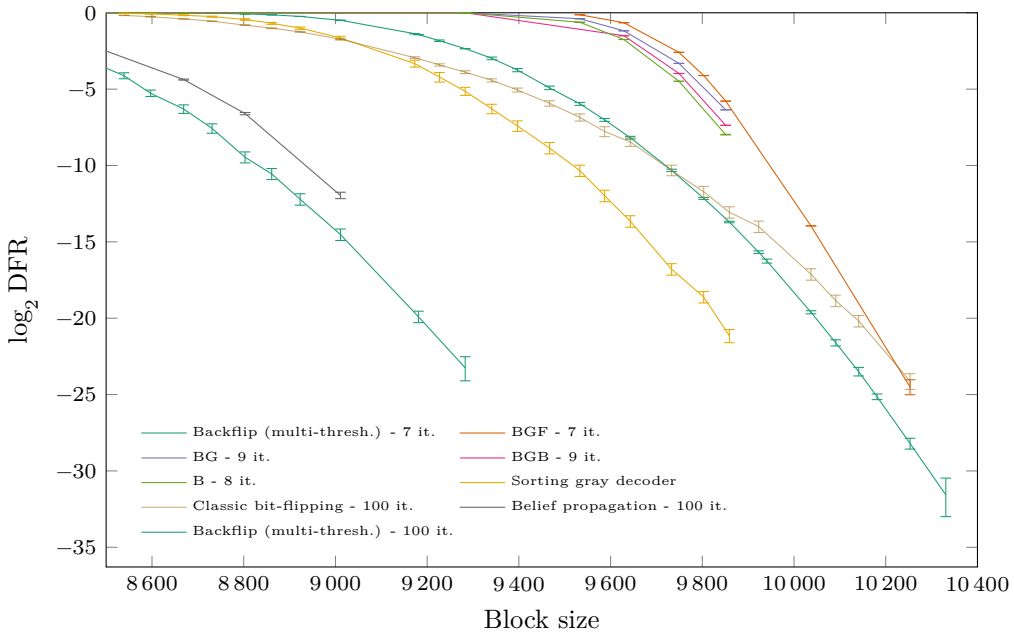


Figure 9.7: DFR *vs.* block size with various decoders (summary). $(d, t) = (71, 134)$. 99%-confidence intervals.

Part III

Analysis of bit-flipping decoders for QC-MDPC

Summary of contributions

We design two new probabilistic models for the bit-flipping decoder:

- one that accurately predicts the distribution of error weights after a single iteration of the algorithm for a regular code,
- one that considers the full decoding process for the (sequential) step-by-step introduced in the Part II.

In contrast to the other models, when compared to the simulation data, we find that they can predict the DFR with very good accuracy. In addition, they are the first ones to integrate the use of an efficient adaptive threshold rule.

The exploitation of these models required ad-hoc optimized implementations in C language. Results for the first model required a particularly important computational effort, since the calculation time, depending on the set of parameters chosen, ranges from half an hour to several hours.

Chapter 10

Introduction

10.1 State-of-the-art

10.1.1 LDPC codes

We present three different analyses of the LDPC decoding failure rate. The first one is based on a statistical model developed by Gallager. Many works have been derived from it to adapt it to other channels or algorithms, but the bases of the model are essentially the same (independence of all random variables and assumption that the Tanner graph is a tree). The second analysis by Sipser and Spielman is done on the more general context of expander codes. They show that the simple bit-flipping algorithm with a majority logic will always decode given that the code has a Tanner graph with certain expansion properties. The result is deterministic and relies solely on properties of the graph and not on any statistical model. A link with LDPC codes can be established by observing that a random LDPC code has, with a high probability, a good expansion property. The third analysis by Burshtein enumerates all bad configurations in a regular LDPC code and derives a rough upper bound on the probability that a given code has poor decoding performance under bit-flipping on any of those configurations.

10.1.1.1 Gallager's analysis

In his thesis [Gal63], Gallager analyzes his hard-decision decoding algorithm for regular LDPC codes on the binary symmetric channel. In contrast to a bit-flipping algorithm, this algorithm explicitely excludes extrinsic information between nodes. It is thus closer to a message-passing algorithm.

The analysis relies on the following assumptions:

- the Tanner graph is a tree,
- the values of variable nodes are mutually independent,
- the values of check nodes are mutually independent,
- thresholds depend only on the iteration index.

Let us explain briefly the technique on a regular LDPC code. We keep the convention of this document and write d the variable nodes degree and w the check nodes degree. Suppose that we have a binary symmetric channel with crossover probabilty ϵ. We track the error rate p_i among the variable nodes at iteration $\#i$. At iteration $\#0$, the error rate is exactly the crossover probability

$$p_0 = \epsilon \, .$$

To see what happen on the check nodes side, we refer to Proposition 1.14 in the preliminary chapter. Using this result, we can say that at iteration $\#i$, each check node involves an even number of errors with probability $\frac{1-(1-2p_i)^w}{2}$. Using the threshold T_{i+1}, the probability that an error was received but corrected at iteration $\#(i+1)$ is

$$\epsilon \sum_{s=T_{i+1}}^{d-1} \binom{d-1}{s} \left(\frac{1+(1-2p_i)^w}{2}\right)^s \left(\frac{1-(1-2p_i)^w}{2}\right)^{d-1-s},$$

and on the contrary, the probability that a bit was correctly received but was mistakenly flipped is

$$(1-\epsilon) \sum_{s=T_{i+1}}^{d-1} \binom{d-1}{s} \left(\frac{1-(1-2p_i)^w}{2}\right)^s \left(\frac{1+(1-2p_i)^w}{2}\right)^{d-1-s}.$$

In the end the error rate at iteration $\#(i+1)$ is simply

$$\begin{aligned} p_{i+1} = \epsilon - \epsilon \sum_{s=T_{i+1}}^{d-1} \binom{d-1}{s} \left(\frac{1+(1-2p_i)^w}{2}\right)^s \left(\frac{1-(1-2p_i)^w}{2}\right)^{d-1-s} \\ + (1-\epsilon) \sum_{s=T_{i+1}}^{d-1} \binom{d-1}{s} \left(\frac{1-(1-2p_i)^w}{2}\right)^s \left(\frac{1+(1-2p_i)^w}{2}\right)^{d-1-s}. \end{aligned}$$

One can then choose the thresholds T_i that minimize the probabilities p_i for all i. For a sufficiently large code length and a sufficiently small value of ϵ, p_i converges to 0 as i tends to infinity.

Density Evolution. The method developed by Gallager has then been extended to various channels, for soft-decision algorithms and irregular codes, see for example [RU01; LMSS01]. It is usually referred to as the density evolution method.

10.1.2 Expander codes arguments

There is an important notion in graph theory called the expansion.

Definition 10.1. Consider a bipartite graph of n d-regular nodes on the left and $(d/w)\cdot n$ w-regular nodes on the right. It has *expansion* (α, β) if for any subset S of nodes on the left such that $\left|S\right| \leq \alpha n$ then the set of its neighbors $\mathcal{N}(S)$ satisfies

$$\left|\mathcal{N}(S)\right| \geq \beta \cdot d \cdot \left|S\right|.$$

In [SS96], it has been shown that the bit-flipping algorithm (either sequential or parallel) succeeds as long as the Tanner graph of the code has sufficiently good expansion. The bit-flipping algorithms considered operate according to a majority vote rule, or in other words they use the fixed threshold $(d+1)/2$.

The following proposition tells us that a random regular graph is a good expander.

Proposition 10.2. *Let B be a (d, w) regular bipartite graph of n d-regular nodes on the left and $(d/w)\cdot n$ w-regular nodes on the right. Then with exponentially high probability, for any $\alpha \in (0,1)$, any set S of size $\alpha \cdot n$ has at least*

$$n\left(\frac{d}{w}\left(1-(1-\alpha)^w\right) - \sqrt{\frac{2d\alpha\,\mathrm{H}(\alpha)}{\log_2 e}}\right)$$

neighbors.

Then any reasonably good expander code can be decoded with a bit-flipping algorithm.

Theorem 10.3. *[SS96, Theorem 10 & 11] Let B be a (d, w) regular bipartite graph of n d-regular nodes on the left and $(d/w) \cdot n$ w-regular nodes on the right. If B is a $(\alpha, 3/4 + \epsilon)$ expander, for any $\epsilon > 0$, then the sequential and the parallel bit-flipping algorithms correct up to $\alpha \cdot n/2$ errors. The former does so in a linear number of flips and the latter in $O(\log n)$ iterations.*

10.1.3 Analysis of regular LDPC codes with a bit-flipping algorithm

In [Bur08], Burshtein analyzed the parallel bit-flipping algorithm as defined by Sipser and Spielman in [SS96]. The Tanner graph of an LDPC code is decomposed in several parts:

- the set of unsatisfied equations $\mathcal{J}$,
- the set of satisfied equations $\mathcal{K}$,
- the set of good positions $\mathcal{G}$,
- the set of bad positions $\mathcal{B}$.

The sets $\mathcal{G}$ and $\mathcal{B}$ are then further decomposed depending on whether the counter values are above or below a threshold that corresponds to a majority vote.

They enumerate all the possible configurations of these sets and get a bound on the number of problematic configurations using generating functions. They show that, asymptotically when n goes to infinity, for all codes of length n, a large proportion can decode any error pattern of weight αn for any value $\alpha < \alpha_0$. The limit value α_0 is obtained by finding the value that maximizes a certain function.

In their proof, they start by describing all "possibly bad" configurations, *i.e.* the only configurations in which the algorithm would increase the error weight after an iteration. These configurations are described according to the number of vertices in the aforementioned subsets in the Tanner graph and the number of edges between them. They then determine the probability that a random regular code exists in one of these configurations. The probability of existence of a "bad" code is bounded by simply giving the same weight to any configuration. In the end, the result is very interesting because of its asymptotic significance but less so for practical use. For example, in the case of a regular LDPC code of length 1 000 000, column weight 5 and row weight 10; it can only show that at least 14% of the codes can correct any error configuration with a weight of 13 or less.

10.1.4 MDPC codes

The difference between LDPC codes and MDPC codes is in the degree of the nodes in their Tanner graph. While it is rather small (usually a few units) for the former, for the latter it is closer to one hundred for cryptographically relevant parameters (*i.e.* more than 128 bits of security). So, a typical MDPC code has many short cycles in his Tanner graph (and it cannot be considered to be a tree anymore) and on top of that we add a quasi-cyclic structure. Therefore we cannot completely rely on the same assumptions as Gallager's analysis.

10.1.4.1 Leveraging the syndrome weight for decoding

We have already presented the counters model from [Cha17] in Chapter 5. Determining the error after the first iteration of a parallel bit-flipping is akin to determining the distributions of the number of positions with a counter above the threshold on one side for the positions in error and on the other for the correct positions. With an assumption of mutual independence of all counters, these two distributions are binomial and one can

then calculate the number of error remaining with a convolution. However, it is observed in [Cha17] that this model gives inaccurate predictions both in the case where the counters model uses the syndrome weight and in the case where it does not.

10.1.4.2 Asymptotic analysis

In [Til18b], the decoding of MDPC codes is studied for a code of arbitrarily large length n with the additional restriction that the row weight $w(n)$ is in $\Theta(\sqrt{n})$. Two algorithms are considered. The first is deterministic, and it is shown that, for an MDPC code, with a sufficiently large code length n and with some restriction on the intersections of the columns of the parity check matrix, a majority logic decoder can decode any error pattern of weight $t(n)$ where $t(n) = \Theta\left(\frac{\sqrt{n}\log\log n}{\log n}\right)$. The second algorithm is based on a probabilistic counter model for MDPC codes and uses asymptotic bounds.

Deterministic analysis. The central proposition for this analysis is the following, its proof uses Tanner graph arguments that remind the ones used in [SS96].

Proposition 10.4. *[Til18b, Proposition 1] Consider a code with a parity check matrix for which every column has weight at least d and whose* maximum column intersection *is I. Performing majority-logic decoding based on this matrix (i.e. Algorithm 6.2 with $T = (d+1)/2$) corrects all errors of weight $\leq d/(2I)$.*

Proof. The original proof from the paper considers the Tanner graph and looks at the properties of certain subgraphs according to the degree of their nodes. We can reformulate it to adopt a more algebraic point of view.

Let us write $\mathbf{H}$ an $r \times n$ parity check matrix of column weight at least d as

$$\mathbf{H} = \begin{pmatrix} \mathbf{h}_0 & \mathbf{h}_1 & \cdots & \mathbf{h}_{n-1} \end{pmatrix}$$

where $\mathbf{h}_0, \ldots, \mathbf{h}_{n-1}$ are the columns of $\mathbf{H}$. We suppose that $\mathbf{H}$ has a maximum column intersection of I *i.e.*

$$\forall i, j \in \{0, \ldots, n-1\}, i \neq j \Rightarrow \left|\mathbf{h}_i \star \mathbf{h}_j\right| \leq I\,.$$

We now consider an error pattern e, it gives the syndrome $\mathrm{s} = \sum_{j \in \mathrm{Supp(e)}} \mathbf{h}_j$. Now let us write the counter $\sigma_i := \left|\mathbf{h}_i \star \sum_{j \in \mathrm{Supp(e)}} \mathbf{h}_j\right| = \left|\sum_{j \in \mathrm{Supp(e)}} \mathbf{h}_i \star \mathbf{h}_j\right|$ for position i and separate the cases where i belongs to the support of e from the case where it does not.

- If $i \in \mathrm{Supp(e)}$,

$$\sigma_i = \left|\mathbf{h}_i + \sum_{j \in \mathrm{Supp(e)} \setminus \{i\}} \mathbf{h}_i \star \mathbf{h}_j\right| \geq \left|\mathbf{h}_i\right| - \sum_{j \in \mathrm{Supp(e)} \setminus \{i\}} \left|\mathbf{h}_i \star \mathbf{h}_j\right| \geq d - (|\mathrm{e}| - 1)I\,.$$

- If $i \notin \mathrm{Supp(e)}$,

$$\sigma_i = \left|\sum_{j \in \mathrm{Supp(e)}} \mathbf{h}_i \star \mathbf{h}_j\right| \leq \sum_{j \in \mathrm{Supp(e)}} \left|\mathbf{h}_i \star \mathbf{h}_j\right| \leq |\mathrm{e}|\, I\,.$$

One then easily verifies that when $T = (d+1)/2$ and $|\mathrm{e}| \leq d/(2I)$, we have

$$\forall i \notin \mathrm{Supp(e)}, \forall i' \in \mathrm{Supp(e)}, \quad \sigma_i < T \leq \sigma_{i'}\,.$$

□

This decoder is envisaged in combination with some key filtering. In fact, when n grows, and the column weight is in $O(\sqrt{n})$, it is shown that the maximum column intersections is in $O\left(\frac{\log n}{\log\log n}\right)$ with high probailty. Introducting some filtering in the key generation method would exclude the cases where it is above said bound. And together with the previous proposition, it proves that for some sufficiently large code length n, a majority logic bit-flipping algorithm decodes any error pattern of weight in $O\left(\frac{\sqrt{n}\log\log n}{\log n}\right)$.

Asymptotic statistical model. The model assumes the following properties:

- the counters are mutually independent,
- the counters of erroneous positions follow a binomial distribution:

  $$j \in \mathrm{e} \Rightarrow \sigma_j \sim \mathrm{Bin}(d, \pi_1)$$

  for some "success" probability π_1,
- the counters of correct positions follow a binomial distribution:

  $$j \notin \mathrm{e} \Rightarrow \sigma_j \sim \mathrm{Bin}(d, \pi_0)$$

  for some "success" probability π_0.

In this model the "success" probabilities are determined assuming independence of the syndrome bits and in the context where both the row weight $w(n)$ and the error weight $t(n)$ are in $O(\sqrt{n})$. It is shown that, in this context

$$\forall b \in \{0,1\}, \pi_b = \frac{1}{2} - (-1)^b \epsilon \left(\frac{1}{2} + O\left(\frac{1}{\sqrt{n}}\right)\right)$$

with $\epsilon = e^{-\frac{2wt}{n}}$.

The bit-flipping decoder can decode an error pattern of weight $t(n) = O(\sqrt{n})$ with a failure probability upper bounded by

$$e^{-\Omega\left(n\frac{\log\log n}{\log n}\right)}.$$

Synthesis. In the paper, three different types of decoding scenarios are finally envisaged.

- Do one iteration and choose parameter so that the deterministic bound guarantees a zero probability of failure.
- Do two iterations so that, using the probabilistic model, the error weight after the first iteration is below the deterministic bound.
- Do two iterations so that, using the probabilistic model, the error weight after the first iteration is below some arbitrary bound $(1-\alpha)t(n)$ for some constant $\alpha < 1$.

Numerical applications show that the parameters from [MTSB13] providing 80 bits of security should have their code length multiplied by ~ 427 (row weight also has to be multiplied by $\simeq 45$) with the first method and $\simeq 2$ in the other scenarios.

10.1.4.3 LEDAcrypt

The authors of the QC-MDPC cryptosystem LEDAcrypt described a model for the decoder in their submission [LEDA], it follows a previous article [SBBC19]. As in [Til18b], they adopt a probabilistic approach followed by a deterministic one, but in this case not in an asymptotic context. The probabilistic model is close to the analysis of Gallager discussed previously for LDPC codes.

Statistical models. Two different types of decoders are considered, the in-place decoder and the out-of-place decoder. The former is quite similar to the step-by-step decoder, in the sense that it is sequential, it updates the syndrome after each flip. However, with this one, during an iteration, each one of the n positions is visited exactly once, but in a random order. The latter is a parallel bit-flipping.

In the document, they make the following assumptions:

- the parity check matrix $\mathbf{H}$ is random and regular, it has a column weight d and row weight w,
- the error pattern e is drawn uniformly at random among the vectors of weight t and length n,
- the syndrome bits are mutually independent.

The out-of-place decoder analysis assumes the independence of the counters values and uses it to find the error weight after one iteration. This is essentially similar to the work done in [Cha17], without the conditioning on the syndrome weight.

The analysis seems to some extent appropriate for a single iteration given the examples provided. However, in [LEDA], this analysis is not further iterated. This is undoubtedly due to the discrepancies (which increase with the number of iterations) between this model and the simulation already highlighted in [Cha17].

Based on assumptions reminiscent of what had been done previously in [SV19] that will be presented in Chapter 12, they model the decoder as a Markov chain. However, in [LEDA], they define a Markov chain that only tracks the error weight. Indeed, they assume that the error weight after an iteration depends only on what it was before. The error pattern is always considered to be taken uniformly at random among all the vector of fixed Hamming weight. In addition, they address the fact that, contrary to the step-by-step algorithm, the in-place decoder is not fully randomized, and they consider the worst order possible for visiting the positions.

Combinatorial bound. The reference documentation [LEDA] provides an intersecting improvement over the bound proven in [Til18b]. Given a simply calculated characteristic of the code, it is shown that one can decode any error pattern whose weight is bounded by some constant. Rather than relying on the maximum column intersection, they use a finer characterization of the matrix. Remember that in the proof of Proposition 10.4 we only used the triangle inequality to bound the counter values. We then upper bounded the following quantities by bounding each term of

$$\sum_{j\in \mathrm{Supp(e)}\setminus\{i\}} \left|\mathbf{h}_i \star \mathbf{h}_j\right| \qquad \text{and} \qquad \sum_{j\in \mathrm{Supp(e)}} \left|\mathbf{h}_i \star \mathbf{h}_j\right|$$

individually. This was a very rough upper bound because for any column, all the other columns do not necessarily intersect with it in exactly I rows, the maximum. The finer bound used in [LEDA] is based on the calculation of the $n \times n$ matrix $\mathbf{\Gamma}$ which keeps a record of the number of intersections between all the pairs of columns:

$$\forall i, j \in \{0, \dots, n-1\}, \Gamma_{i,j} = \left|\mathbf{h}_i \star \mathbf{h}_j\right| .$$

Upper bounding the above sums is then a matter of finding the |e| largest elements in the i-th row of $\mathbf{\Gamma}$.

A rather interesting feature of the $\mathbf{\Gamma}$ matrix is that it is symmetric and also quasi-cyclic if $\mathbf{H}$ is quasi-cyclic as well. This allows for a fast computation of $\mathbf{\Gamma}$ that is used for filtering keys in [LEDA].

Synthesis. Similarly to one of the scenarios proposed in [Til18b], they propose a two-iterations decoder modeled, for the first iteration, with the above statistical model, followed by an iteration whose outcome is deterministic. This also implies some sort of key filtering.

Some curves showing the predictions of the DFR in this model compared to simulation is presented in [LEDA]. However, they were done with the parameters offering 80 bits of security of [MTSB13] and not for actual LEDA parameters.

10.2 Contributions

One of the difficulties encountered in the analysis of a QC-MDPC decoder is the fact that many correlations appear and are difficult to take into account. It is often tempting to make independence hypotheses about certain variables involved in the analysis. However, there are at least two factors that can challenge these hypotheses.

- The codes of interest are quasi-cyclic and therefore regular, this regularity adds correlations between the parity check equations compared to a completely random code. Such correlations occur, for example, when dealing with variables related to the syndrome weight or the counters.
- The decoding algorithms are iterative and the operations performed during each iteration further divert the error pattern from a random pattern. For example, bit-flipping flips in priority positions involved in many unsatisfied equations, the resulting error vector will generally give a lower syndrome weight than what is expected for a uniformly random pattern: the error positions are therefore correlated in a way that depends on the parity check equations.

It is rather challenging to find a good model that captures most of the characteristics of the codes and the algorithm without increasing the complexity of the calculations to the point of making them unfeasible. We will see two models that seek, each in its own way, to address these issues.

The first one attempts to take into account the quasi-cyclicity of the code. We will see that it is fairly accurate to predict the outcome of one iteration of a parallel bit-flipping algorithm. It is then tempting to design a decoder that is suitable for this analysis and whose performance would therefore be evaluated within the framework of a very credible model. One disadvantage of this method is that the model assumes that the error pattern is uniformly random and can therefore only be used for the first iteration. The analysis of the subsequent iteration relies on weaker assumptions, which greatly increase the upper bound on the DFR that can be proven.

The second model assumes a Markovian behaviour. In an effort to capture the correlations arising from the iterative nature of the decoder, this model tracks the evolution of the syndrome weight and the error weight of a step-by-step decoder. An interesting result of this model is that it gives an idea of the shape of the DFR curve of a fully iterative decoder for code length that are unreachable with simulations.

Compared to previous works, these two models make it possible to evaluate algorithms with efficient adaptive thresholds that depend on the syndrome weight. When defining the first model, we re-examine the various independence assumptions made in previous work and propose a better approach. This completes [Cha17] in which a discrepancy was already noted.

Chapter 11

One iteration of the parallel decoder with variable thresholds

Our goal, in this chapter, is to analyze one iteration of the parallel bit-flipping decoder with a threshold rule that is a function of the syndrome weight (see Algorithm 11.1). Its *for* loop is perfectly parallel and the decision to flip does not depend on other decisions.

Algorithm 11.1: One iteration of parallel bit-flipping.

function `one_parallel_iteration`$^{\perp}(\mathbf{H}, \mathrm{s}^0)$**:**
- **input :** A sparse parity check matrix $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}$,
  a syndrome $\mathrm{s}^0 = \mathbf{H}(\mathrm{e}^0)^\top \in \mathbb{F}_2^{n-k}$.
- **output :** The error pattern difference after one iteration $\mathrm{e}' \in \mathbb{F}_2^n$.
- $\mathrm{e}' \leftarrow 0$;
- $\mathrm{s}^1 \leftarrow \mathrm{s}^0$;
- $T \leftarrow$ `threshold` $\left(\left|\mathrm{s}^0\right|\right)$;
- **for** $j \in \{0, \dots, n-1\}$ **do**
  - **if** $\left|\mathbf{h}_j \star \mathrm{s}^0\right| \geq T$ **then**
    - $e'_j \leftarrow 1 - e'_j$;
- $\mathrm{s}^1 \leftarrow \mathrm{s}^0 - \mathbf{H}\mathrm{e}'^\top$;
- **return** e'

In our configuration, we have $\left|\mathrm{e}^0\right| = t_0 = t$ and $\mathrm{s}^0 = \mathbf{H}\left(\mathrm{e}^0\right)^\top$. After one iteration, we will have $\mathrm{e}^1 = \mathrm{e}^0 + \mathrm{e}'$ and $\mathrm{s}^1 = \mathbf{H}\left(\mathrm{e}^1\right)^\top$. We also define

$$\text{for } i = 0, 1, \quad t_i := \left|\mathrm{e}^i\right|, \quad S_i := \left|\mathrm{s}^i\right| \quad \text{and} \quad X_i := \left(\sum_{j \in \mathrm{e}^i} \left|\mathbf{h}_j \star \mathrm{s}^i\right|\right) - S_i \,.$$

We will show that when e^0 is drawn uniformly at random and the parity check matrix $\mathbf{H}$ is regular, we can find the joint distribution of S_0 and X_0. The values S_0 and X_0 are really central to our model because they carry a lot of information about the counters distributions. We will then determine the counters distributions more accurately than in Chapter 5. We will see that this accuracy is needed to compute the distribution of t_1. Finally, we propose a possible approach to determine S_1.

It should be noted that S_0 is a piece of information that the algorithm can measure and use but X_0 is not, as it implies already knowing the support of the error pattern e^0

(which is the whole point of decoding). One strength of this model is that it lets us use an adaptive threshold for decoding. Our threshold choices are thus based on the syndrome weight S_0 but not on the value X_0.

Compared to previous analyses reviewed in the state of the art, this model takes into account some correlations implied by the regularity of the quasi-cyclic code.

Convolutions. Numerical computations in this model require the calculation of many convolution powers of probability mass functions. Although the best algorithm to perform this operation is the fast Fourier transform, it is very sensitive to rounding errors if used with floating point arithmetic. This is particularly problematic when dealing with values such as probabilities that can range from 10^{-40} to 1 and when the same magnitude is expected in the result.

In the one-dimensional case, the calculation by direct convolution of a vector of length m with a vector of length n has a complexity of $O(mn)$. Now, the N-th convolution power of a vector of length n can be realized using a fast exponentiation algorithm. In this case, a maximum of $2\log_2(N)$ convolutions are needed between vectors of length at most (respectively at each step): $n, 3n, \ldots, (2^i-1)n, \ldots$ The total complexity of this method is thus $O(N^2n^2)$.

In this chapter, propositions are written using multidimensional random variables for brevity and convenience in writing proofs. The convolutions of these variables can be computed by reducing them to the one-dimensional case using, for example, the following property

$$\Pr\left[\sum_i (X_i, Y_i) = (x, y)\right] = \Pr\left[\sum_i Y_i = y \,\middle|\, \sum_i X_i = x\right] \cdot \Pr\left[\sum_i X_i = x\right].$$

The random variable $\sum_i X_i$ will generally follow a binomial distribution. And the calculation of $\Pr\left[\sum_i Y_i = y \,|\, \sum_i X_i = x\right]$ can be done with one-dimensional convolutions.

11.1 Notations

We adopt the following notations in this chapter.

General.

- The set $\mathcal{E}_{n,t}$ is set of all the error patterns of weight t

$$\mathcal{E}_{n,t} = \{\mathrm{e} \in \{0,1\}^n \mid |\mathrm{e}| = t\},$$

- $\mathcal{H}_{d,w,r\times n}$ is the set of regular $r \times n$ parity check matrices of column weight d and row weight w

$$\mathcal{H}_{d,w,r\times n} = \{\mathbf{H} \in \mathbb{F}_2^{r\times n} \mid \forall i \in \{0,\ldots,r-1\}, \left|\mathbf{h}_i^\top\right| = w, \forall j \in \{0,\ldots,n-1\}, \left|\mathbf{h}_j\right| = d\}.$$

- The indicator function of a set S is the function defined as

$$\mathbf{1}_S(x) := \begin{cases} 1 & \text{if } x \in S, \\ 0 & \text{if } x \notin S. \end{cases}$$

Coding theory.

- We denote the syndrome weight corresponding to a certain error parttern

$$S = |\mathrm{s}| \qquad \mathrm{s} = \mathbf{H}\mathrm{e}^\top, .$$

- For a position $j \in \{0, \dots, n-1\}$, we write its counter $\sigma_j := \left|\mathrm{h}_j \star \mathrm{s}\right|$.
- We write

$$X = \sum_{j \in \mathrm{e}} \sigma_j - S \, .$$

- The definitions of σ_i, E_ℓ, S and X depend on a specific parity check matrix $\mathbf{H}$ and an error pattern e. These dependencies will always be evident from the context and are therefore not explicitly mentioned so as not to clutter the notations.

Probability.

- If p_X and p_Y are respectively the probability mass functions of some independent random variables X and Y then the probability mass function of their sum is the convolution

$$p_{X+Y}(z) = \sum_{x+y=z} p_X(x) \cdot p_Y(y) \, .$$

  We denote it as $p_X * p_Y$.

- The convolution operation is associative and we can thus consider the N-fold iteration of the convolution of a probabily mass function f with itself. We denote it as

$$f^{*N} = \underbrace{f * f * f * \cdots * f * f}_{N} \, .$$

- We indicate the probability of an event E conditioned on (11.1), the equations implied by the regularity of the code, by

$$\Pr_{\mathrm{reg}}[E] \, .$$

- We will need to calculate conditional distributions from a joint distribution f. In this case, we will write N_f the associated normalization factor.

11.2 Mass equations in regular codes

Let $\mathbf{H} \in \mathcal{H}_{d,w,r\times n} \subset \mathbb{F}_2^{r\times n}$ be a quasi-cyclic parity check matrix of column weight d and row weight $w = 2d$. Let $\mathrm{e} \in \mathcal{E}_{n,t} \subset \mathbb{F}_2^n$ be an error pattern of Hamming weight t. We write the syndrome as $\mathrm{s} = \mathbf{H}\mathrm{e}^\top$.

To start our analysis, let us first apply two permutations $\mathbf{P} \in \mathbb{F}_2^{r\times r}$ and $\mathbf{Q} \in \mathbb{F}_2^{n\times n}$ to $\mathbf{H}$, e and s:

$$\mathbf{H}' = \mathbf{P}\mathbf{H}\mathbf{Q}^{-1} \qquad \mathrm{e}' = \mathrm{e}\mathbf{Q}^\top \qquad \mathrm{s}' = \mathbf{P}\mathrm{s}$$

so that we have $\mathrm{s}' = \mathbf{H}'\mathrm{e}'^\top$ in the configuration illustrated in Figure 11.1. This transformation has no impact on the outcome of the parallel decoding process written in Algorithm 11.1 as the counters are unchanged and, since the algorithm is parallel, reordering the positions has no impact on its performance.

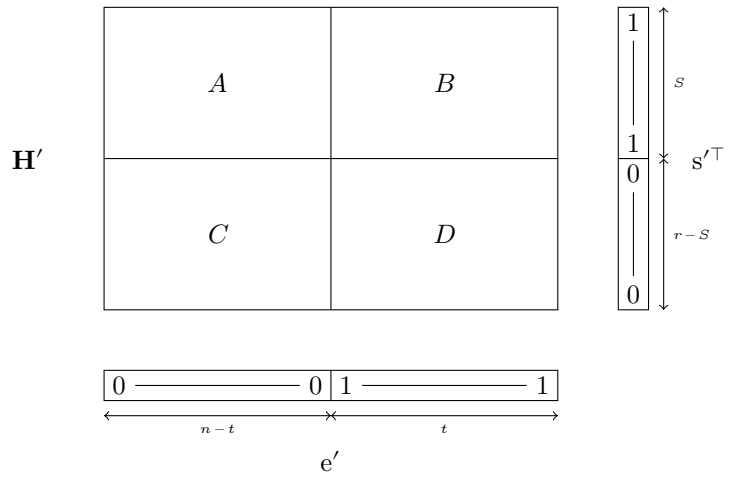


Figure 11.1: Illustration of the reordering of the parity check matrix.

Here A, B, C and D are the number of ones in the respective part of the matrix represented above. A quasi-cyclic code is regular and therefore has a constant row weight w and a constant column weight d. If we know $S = |\mathrm{s}|$ and $X = (\sum_{j\in \mathrm{e}} \sigma_j) - S$, we have the following mass equations

$$\begin{cases} B = S + X \\ A + B = wS \\ A + C = d(n-t) \\ B + D = dt \end{cases} \quad \text{or equivalently} \quad \begin{cases} A = (w-1)S - X \\ B = S + X \\ C = d(n-t) - (w-1)S + X \\ D = dt - S - X\,. \end{cases} \tag{11.1}$$

In the continuation of this chapter, we will define random variables describing each row (or column), then the consideration of these mass equations is essentially a matter of summing these random variables and conditioning the probabilities according to said sums.

Remark 11.1. In previous works, following [Cha17], the value X is usually defined as the sum $E_3 + E_5 + \cdots$ where E_ℓ is, for any positive integer ℓ, the number of equations affected by exactly ℓ errors. Here we take a slightly different point of view, $X = (\sum_{j\in \mathrm{e}} \sigma_j) - S$. The case $X = 0$ is the one where there is exactly one error per unsatisfied equation. In this case, flipping one error will never decrease the counters of the other errors. In a way, X measures the deviation from the case $X = 0$.

Remark 11.2. There is a notable link to be made between this work and Burshtein's work in [Bur08]. The translation between their notation and ours are

$$\begin{aligned} \alpha &= \frac{t}{n}\,, \\ \pi_0 &= \frac{S}{r}\,, \\ \omega_0 &= \frac{\sum_{j\in \mathrm{e}} \sigma_j}{dn} = \frac{S+X}{dn}\,. \end{aligned}$$

In this chapter, we will show formulas to compute the number of good (resp. bad) positions flipped after one iteration. With Burshtein's notation, $\gamma_{1,2}$ is the proportion of positions that are bad but not flipped after one iteration and $\delta_{1,2}$ is the proportion of positions that are good but flipped.

11.3 Modeling the error weight after the first iteration

In this section, we will derive a probabilistic model that allows, among other things, to determine the weight of the error after an iteration. All probabilities will be conditioned by the mass equations in §11.2 and we will see that this approach allows us to get really close to the statistics obtained with decoding simulation for quasi-cyclic codes. We will focus on proving every result by clearly stating the assumptions, and we will motivate the complexity of the model by comparing it to models that do not verify the mass equations.

11.3.1 Estimating the number of errors per equation

We first recall the following proposition, which is rather standard in coding theory.

Proposition 11.3. *Let* $\mathbf{H}$ *be drawn uniformly at random in* $\mathcal{H}_{d,w,r\times n}$ *and* e *drawn uniformly at random in* $\mathcal{E}_{n,t}$*. For any* $i \in \{0,\dots,r-1\}$*, the number of errors involved in the* i*-th equation* $\rho_i := \left|\mathbf{h}_i^\top \star \mathrm{e}\right|$ *follows the distribution below, conditioned on the syndrome bit value* $s_i \in \{0,1\}$*,*

$$\Pr[\rho_i = \ell \mid s_i] = \frac{g_{s_i}(\ell)}{N_{g_{s_i}}}$$

for $\ell \in \{0,\dots,\min(w,t)\}$*, with*

$$g_a(\ell) = \frac{\binom{w}{\ell}\binom{n-w}{t-\ell}}{\binom{n}{t}}\mathbf{1}_{a+2\mathbb{Z}}(\ell)\,, \quad a \in \{0,1\}$$

and N_{g_0} *and* N_{g_1} *are the normalization constants:*

$$N_{g_1} = \sum_{\ell \text{ is odd}} \frac{\binom{w}{\ell}\binom{n-w}{t-\ell}}{\binom{n}{t}}\,, \qquad N_{g_0} = 1 - N_{g_1}\,.$$

Proof. By definition of the matrix-vector product in $\mathbb{F}_2$, since $\mathrm{s} = \mathbf{H}\mathrm{e}^\top$, for any $i \in \{0,\dots,r-1\}$, ρ_i is the number of errors implied in $\mathbf{h}_i^\top$. Since $\mathbf{h}_i^\top$ and e are drawn uniformly at random among vectors of weight respectively w and t, said number follows a hypergeometric distribution. Conditioning the probabilities on the value modulo 2 of this sum is then only a question of separating the distributions according to the parity and then normalizing them. □

In Table 11.1, we can see the decay of these probabilities as ℓ increases with [BIKE] parameters offering 128 bits of security.

11.3.2 Estimating the syndrome weight and the sum of the counters

In the following proposition we give the distribution of the syndrome weight and the sum of the counters when the error is uniformly distributed of weight t and the matrix $\mathbf{H}$ is regular.

Proposition 11.4. [1] *Let* $\mathbf{H}$ *be drawn uniformly at random in* $\mathcal{H}_{d,w,r\times n}$ *and* e *drawn uniformly at random in* $\mathcal{E}_{n,t}$*. The syndrome weight* $S = \left|\mathbf{H}\mathrm{e}^\top\right|$ *and the value* $X = (\sum_{j\in \mathrm{e}} \sigma_j) - S$ *follow the joint distribution for* $k \in \{0,\dots,r\}$ *and* $\ell \in \{0,\dots,dt\}$

$$\Pr_{reg}[(S,X) = (k,\ell)] = \frac{1}{N_\zeta}\,\zeta^{*r}(k, k+\ell, dt-k-\ell) \tag{11.2}$$

[1] The noisy syndrome case (Ouroboros) is covered in Proposition 11.17 of §11.5.

Table 11.1: Numerical applications of Proposition 11.3 for $(r, d, t) = (12\,323, 71, 134)$.

ℓ	$\log_2 \Pr[\rho_i = \ell \mid s_i = 0]$	ℓ	$\log_2 \Pr[\rho_i = \ell \mid s_i = 1]$
0	−0.39	1	−0.14
2	−2.13	3	−3.50
4	−6.53	5	−8.68
6	−12.35	7	−15.03
8	−19.15	9	−22.23
10	−26.72	11	−30.14
12	−34.94	13	−38.65
14	−43.71	15	−47.67
16	−52.98	17	−57.16
18	−62.68	19	−67.08
20	−72.80	21	−77.39
22	−83.30	23	−88.07
24	−94.15	25	−99.09
26	−105.34	27	−110.45
28	−116.85	29	−122.11
30	−128.67	31	−134.08

where ζ is defined on $\{0,1\} \times \{0, \dots, w\}^2$ with, for $a \in \{0, \dots, \min(w,t)\}$,

$$\begin{cases} \zeta(0,0,a) = \frac{\binom{w}{a}\binom{n-w}{t-a}}{\binom{n}{t}} & \text{if } a \text{ is even}, \\ \zeta(1,a,0) = \frac{\binom{w}{a}\binom{n-w}{t-a}}{\binom{n}{t}} & \text{if } a \text{ is odd}. \end{cases}$$

and ζ is zero elsewhere;

$$N_\zeta = \sum_{k,\ell} \zeta^{*r}(k, k+\ell, dt-k-\ell)\,.$$

Proof. For a parity check equation $\mathbf{h}_i^\top$ of $\mathbf{H}$, we consider the three-dimensional random variable $R_i = (s_i, s_i \cdot \ell, (1-s_i)\cdot\ell)$ where ℓ is the number of errors involved in this equation (*i.e.* $\ell = \left|\mathbf{h}^\top{}_i \star \mathbf{e}\right|$) and s_i is the i-th bit of the syndrome, so, by definition $s_i = \ell \bmod 2$. The distribution of this random variable is given by ζ, we have already discussed in the proof of Proposition 11.3 that it follows a hypergeometric distribution.

The sum of r independent random variables of this type follows the distribution given by ζ^{*r}. In this sum, the first coordinate represents the number of unsatisfied equations, the second is the sum of the number of errors in the unsatisfied equations, and the third is the sum of the number of errors in the satisfied equations.

The proposition is obtained by conditioning the probabilities for the coordinates to verify (11.1). □

Remark 11.5. To emphasize the importance of the extra condition in the probabilities, let us look at what happens when we remove it. The probability (11.2) becomes

$$\zeta'^{*r}(S, S+X) \tag{11.3}$$

where ζ' is defined on $\{0,1\} \times \{0, \dots, w\}$ with, for $a \in \{0, \dots, \min(w,t)\}$,

$$\zeta'(a \bmod 2, a) = \frac{\binom{w}{a}\binom{n-w}{t-a}}{\binom{n}{t}}$$

Table 11.2: Comparison of the distribution of S and X between simulation (10^9 samples with quasi-cyclic matrices) and models for $(r, d, t) = (12\,323, 71, 134)$.
Kullback-Leibler divergence is computed between the simulation data $\mathcal{S}$ and the distribution $\mathcal{M}$ obtained with the model specified for each row.

	S		X		$\mathrm{D}_{\mathrm{KL}}(\mathcal{S}\|\mathcal{M})$
	Mean	Var.	Mean	Var.	
Simulation	4 868.832	2 511.872	909.024	1 369.947	-
Proposition 11.4	4 868.996	2 512.022	908.896	1 369.770	$5.069 \cdot 10^{-5}$
(11.3) in Remark 11.5	4 868.831	2 945.150	909.023	1 849.200	1.05198
(11.4) in Remark 11.5	4 868.831	2 945.150	909.023	1 849.200	0.05198

and ζ' is zero elsewhere.

However, the syndromes we consider all correspond to error patterns of the same weight t, and the parity check matrix is column-regular with column weight d. Under these conditions, the syndrome weight necessarily has the same parity as $d \cdot t$ (see Corollary 1.27). Taking this in consideration, we can refine (11.3):

$$\frac{\mathbf{1}_{d\cdot t+2\mathbb{Z}}(S)}{N_{\zeta'}}\zeta'^{*r}(S, S+X) \tag{11.4}$$

with

$$N_{\zeta'} = \sum_{k,\ell} \mathbf{1}_{d\cdot t+2\mathbb{Z}}(k)\zeta'^{*r}(k, k+\ell)\,.$$

A comparison of these different models is given in Table 11.2. We can see that Proposition 11.4 gives a distribution very close to the one obtained by simulation, whereas (11.3) and (11.4) do not. Distribution tails are particularly poorly estimated for the latter two. This suggests that (11.1) does indeed capture most of the correlations between syndrome bits induced by the quasi-cyclicity of the matrix.

11.3.3 Counters distributions

Compared to the distributions obtained by simulation, the counters distributions of the model given in [Cha17] match very well. However, there remains some small bias that affects the model that we will describe later (the distribution of the number of positions above a threshold – the ones that are flipped).

To correct these discrepancies, we can again use the mass equations (11.1) that the regularity of the code implies.

In this subsection, we suppose that the syndrome weight S and the value $X = (\sum_{j\in \mathrm{e}} \sigma_j) - S$ are no longer random variables and their values are known.

Proposition 11.6. [2] *Let* $\mathbf{H}$ *be drawn uniformly at random in* $\mathcal{H}_{d,w,r\times n}$ *and* e *drawn uniformly at random in* $\mathcal{E}_{n,t}$. *We denote the syndrome weight* S *and the value* $X = (\sum_{j\in \mathrm{e}} \sigma_j) - S$, *and suppose that they are known constants. The counters* $\sigma_j = \left|\mathrm{h}_j \star \mathrm{s}\right|$

[2] The noisy syndrome case (Ouroboros) is covered in Proposition 11.18 of §11.5.

follow the distributions for $k \in \{0, \dots, d\}$

$$\begin{aligned}\Pr_{reg}[\sigma_j = k \mid j \in \mathrm{e}, S, X] &= \frac{1}{N_\phi}\, \phi_1^{*S}(k, S+X) \\ &\quad \cdot \phi_0^{*(r-S)}(d-k, dt-(S+X))\,, \\ \Pr_{reg}[\sigma_j = k \mid j \notin \mathrm{e}, S, X] &= \frac{1}{N_\psi}\, \psi_1^{*S}(k, (w-1)S-X) \\ &\quad \cdot \psi_0^{*(r-S)}(d-k, d(n-t)-(w-1)S+X))\,,\end{aligned}$$

probabilities are zero when $k \notin \{0, \dots, d\}$*, and we have, for* $\ell \in \{0, \dots, \min(w,t)\}$ *and* $a \in \{0,1\}$*,*

$$\begin{cases}\phi_a(0,\ell) = \frac{t-\ell}{t} \cdot g_a(\ell)\,, \\ \phi_a(1,\ell) = \frac{\ell}{t} \cdot g_a(\ell)\,,\end{cases} \qquad \begin{cases}\psi_a(0, w-\ell) = \frac{n-t-w+\ell}{n-t} \cdot g_a(\ell)\,, \\ \psi_a(1, w-\ell) = \frac{w-\ell}{n-t} \cdot g_a(\ell)\,.\end{cases}$$

where g_a *is defined in Proposition 11.3 and*

$$\begin{aligned}N_\phi &= \sum_{k=0}^{d} \phi_1^{*S}(k, S+X) \cdot \phi_0^{*(r-S)}(d-k, dt-(S+X))\,, \\ N_\psi &= \sum_{k=0}^{d} \psi_1^{*S}(k, (w-1)S-X) \cdot \psi_0^{*(r-S)}(d-k, d(n-t)-(w-1)S+X))\,.\end{aligned}$$

Proof. First let us suppose that $j \in \mathrm{e}$. For each row i of the parity check matrix $\mathbf{H}$, we consider the 2-dimensional random variables $R'_i = (h_{i,j}, \ell)$ where ℓ is the number of errors implied in the equation at row i: $\ell = \left|\mathbf{h}_i^\top \star \mathrm{e}\right|$. We thus have, for all $\ell \in \{0, \dots, \min(t,w)\}$

$$\Pr[R'_i = (0,\ell)] = \frac{t-\ell}{t} \cdot g_{s_i}(\ell)\,, \qquad \Pr[R'_i = (1,\ell)] = \frac{\ell}{t} \cdot g_{s_i}(\ell)\,.$$

Since we reordered the rows of $\mathbf{H}$, we know that for all $i \in \{0, \dots, S-1\}$, there is an odd number of errors in the corresponding equations ($s_i = 1$), and it is even for all the other rows ($s_i = 0$).

Assuming independence of the random variables R_i, we obtain the distributions of

$$\sum_{i=0}^{S-1} R_i = (\sigma_1, \Sigma_1) \qquad \text{and} \qquad \sum_{i=S}^{r-1} R_i = (\sigma_0, \Sigma_0)$$

with convolutions (it corresponds respectively to ϕ_1^{*S} and $\phi_0^{*(r-S)}$).

The first sum involves the upper right corner of our representation of the parity check matrix in Figure 11.1 while the other one involves the lower right corner. We have mass equations for each one of these parts of the matrix. Here they translate as

$$\begin{aligned}\Sigma_1 &= S + X\,, \\ \Sigma_0 &= dt - (S+X)\,.\end{aligned}$$

We also know that the column weight d is fixed so $\sigma_0, \sigma_1 \in \{0, \dots, d\}$ and $\sigma_0 = d - \sigma_1$. The relevant conditional probabilities are obtained by setting the probabilities to zero when the conditions are not satisfied and then normalizing them through division by N_ϕ.

The same arguments prove the formulas for $j \notin \mathrm{e}$ by doing the substitutions below.

$\in \mathrm{e}$	$\notin \mathrm{e}$
t	$n-t$
ℓ	$w-\ell$
$S+X$	$(w-1)S-X$
ϕ	ψ

□

Remark 11.7. The values obtained with Proposition 11.6 should be compared to the simpler binomial model given by (5.2) established in [Cha17]. Examples are given in Table 11.3. There is only a very slight bias between the binomial model and the distribution obtained with Proposition 11.6. The improvement achieved by the latter could therefore be considered minor and not worth the additional complexity, but the usefulness of this proposition will be more evident in the following subsection when comparing the number of flipped positions predicted by the model with the simulations.

11.3.4 Predicting flips

In this subsection, we still suppose that the syndrome weight S and the value $X = (\sum_{j\in \mathrm{e}} \sigma_j) - S$ are no longer random variables and their values are known, we also suppose that the counters distributions for these values of S and X are known.

We now choose a threshold T. In the model, it can depend on any fixed parameter of the cryptosystem r, n, d, w, t, but it can also depend on S or X. However, it makes little sense to base it on the exact value of X, because such a quantity is not known to a real decoder.

In order to predict the number of errors remaining after one iteration, we want to know the number of positions that have a counter greater than or equal to the threshold T while separating the case where the corresponding position is an error from the case where it is not (in the first case it decreases the error weight while in the other it increases it).

Let us recall the equations from (11.1) concerning the counters (*i.e.* the upper part of the matrix in Figure 11.1:

$$\sum_{j\in \mathrm{e}} \sigma_j = S + X\,, \qquad \sum_{j\notin \mathrm{e}} \sigma_j = (w-1)S - X\,.$$

This dependence between the counters can be taken into account using a technique similar to the previous propositions.

Proposition 11.8. *Let* $\mathbf{H}$ *be drawn uniformly at random in* $\mathcal{H}_{d,w,r\times n}$ *and* e *drawn uniformly at random in* $\mathcal{E}_{n,t}$*. We denote the syndrome weight* S *and the value* $X = (\sum_{j\in \mathrm{e}} \sigma_j) - S$*, and suppose they are known constants. We suppose that the counters distributions* $\Pr_{reg}[\sigma_j \mid j \in \mathrm{e}, S, X]$ *and* $\Pr_{reg}[\sigma_j \mid j \notin \mathrm{e}, S, X]$ *are known (see Proposition 11.6). The number of positions whose counter is greater than or equal to a threshold* T *follows the distributions, for* $k \in \{0,\dots,t\}$ *and* $l \in \{0,\dots,n-t\}$

$$\Pr_{reg}\left[\left|\{j \in \mathrm{e} \,|\, \sigma_j \geq T|\,\}\right| = k \,\middle|\, S, X\right] = \frac{1}{N_{\gamma_1}}\,\gamma_1^{*t}(k, S+X)$$

$$\Pr_{reg}\left[\left|\{j \notin \mathrm{e} \,|\, \sigma_j \geq T|\,\}\right| = l \,\middle|\, S, X\right] = \frac{1}{N_{\gamma_0}}\,\gamma_0^{*(n-t)}(l, (w-1)S-X)$$

where γ_0 *and* γ_1 *are defined on* $\{0,1\}\times\{0,\dots,d\}$ *with*

$$\forall k \in \{0,\dots,d)\}, \begin{cases} \gamma_0(\mathbf{1}_{\geq T}(k), k) = \Pr_{reg}[\sigma_j = k \mid j \notin \mathrm{e}]\,, \\ \gamma_1(\mathbf{1}_{\geq T}(k), k) = \Pr_{reg}[\sigma_j = k \mid j \in \mathrm{e}]\,, \end{cases}$$

Table 11.3: Comparison of the counters distributions between simulation (10^5 samples) and models for $(r, d, t) = (12\,323, 71, 134)$.
Kullback-Leibler divergence is computed between the simulation data $\mathcal{S}$ and the distribution $\mathcal{M}$ obtained on the model specified in the first column.
The probabilities $\Pr_{\text{reg}}[(S, X)]$ are computed according to Proposition 11.4.

		$j \in \mathrm{e}$			$j \notin \mathrm{e}$		
	Mean	Var.	$\mathrm{D}_{\mathrm{KL}}(\mathcal{S}\|\mathcal{M})$	Mean	Var.	$\mathrm{D}_{\mathrm{KL}}(\mathcal{S}\|\mathcal{M})$
$S = 4\,784$ $X = 830$	$\Pr_{\text{reg}}[(S, X)] = 1.001 \cdot 10^{-5}$					
Simulation	27.485	16.748	-	41.896	16.918	-
Proposition 11.6	27.485	16.746	$3.12e-07$	41.894	16.932	$2.3e-06$
Binomial (Rem. 11.7)	27.485	16.845	$1.25e-05$	41.895	17.171	$8.23e-05$
$S = 4\,868$ $X = 820$	$\Pr_{\text{reg}}[(S, X)] = 1.836 \cdot 10^{-5}$					
Simulation	27.969	16.846	-	42.448	16.823	-
Proposition 11.6	27.968	16.854	$4.03e-07$	42.447	16.838	$1.79e-06$
Binomial (Rem. 11.7)	27.969	16.954	$1.5e-05$	42.447	17.069	$7.77e-05$
$S = 4\,868$ $X = 908$	$\Pr_{\text{reg}}[(S, X)] = 3.434 \cdot 10^{-4}$					
Simulation	27.965	16.853	-	43.104	16.707	-
Proposition 11.6	27.965	16.854	$2e-07$	43.102	16.701	$3.11e-06$
Binomial (Rem. 11.7)	27.965	16.948	$1.18e-05$	43.105	16.935	$6.99e-05$
$S = 4\,868$ $X = 998$	$\Pr_{\text{reg}}[(S, X)] = 1.947 \cdot 10^{-5}$					
Simulation	27.961	16.866	-	43.776	16.563	-
Proposition 11.6	27.961	16.855	$4.16e-07$	43.774	16.547	$2.29e-06$
Binomial (Rem. 11.7)	27.962	16.948	$8.88e-06$	43.776	16.787	$6.81e-05$
$S = 4\,932$ $X = 852$	$\Pr_{\text{reg}}[(S, X)] = 4.334 \cdot 10^{-5}$					
Simulation	28.336	16.919	-	43.164	16.670	-
Proposition 11.6	28.336	16.934	$6.44e-07$	43.162	16.688	$3.61e-06$
Binomial (Rem. 11.7)	28.336	17.031	$1.61e-05$	43.164	16.917	$8.18e-05$
$S = 5\,002$ $X = 906$	$\Pr_{\text{reg}}[(S, X)] = 1.002 \cdot 10^{-5}$					
Simulation	28.736	16.997	-	44.090	16.453	-
Proposition 11.6	28.737	17.009	$7.4e-07$	44.089	16.478	$3.73e-06$
Binomial (Rem. 11.7)	28.736	17.109	$1.63e-05$	44.090	16.711	$9.1e-05$

and γ_0 and γ_1 are zero elsewhere,

$$N_{\gamma_1} = \sum_{k=0}^{t} \gamma_1^{*t}(k, S+X)\,, \qquad N_{\gamma_0} = \sum_{k=0}^{n-t} \gamma_0^{*(n-t)}(k, dt-S-X)$$

Proof. First let us suppose that $j \in \mathrm{e}$. We define a two-dimensional random variable $\mathrm{C}_j = (C_{j,1}, C_{j,2})$, $C_{j,2}$ is the counter σ_j of position j and $C_{j,1}$ is such that

$$C_{j,1} = \begin{cases} 0 & \text{if } C_{j,2} < T, \\ 1 & \text{if } C_{j,2} \geq T. \end{cases}$$

The probability mass function of C_j is given by γ_1.

Summing t times the independent random variables C_j for all error position $j \in \mathrm{e}$ gives a random variable whose first coordinate is the number of counters above T and the second coordinate is the sum of all the counters. Its probability mass function is γ_1^{*t}. The relevant conditional probabilities are finally obtained by keeping only the values of $\sum_{j\in\mathrm{e}} \mathrm{C}_j$ such that the second coordinate is $S+X$ and then normalizing the probabilities with a division by N_{γ_1}.

The result is proven for $j \notin \mathrm{e}$ with the substitutions below.

$\in \mathrm{e}$	$\notin \mathrm{e}$
t	$n-t$
$S+X$	$(w-1)S-X$
γ_1	γ_0

□

Remark 11.9. Here again, one could question the necessity of considering the mass equations (11.1).

Without it $|\{j \in \mathrm{e} \,|\, \sigma_j \geq T|\,\}|$ and $|\{j \notin \mathrm{e} \,|\, \sigma_j \geq T|\,\}|$ are two random variables following binomial distributions:

$$|\{j \in \mathrm{e} \,|\, \sigma_j \geq T|\,\}| \sim \mathrm{Bin}(t, p_1)\,, \qquad |\{j \notin \mathrm{e} \,|\, \sigma_j \geq T|\,\}| \sim \mathrm{Bin}(n-t, p_0) \tag{11.5}$$

where

$$p_1 = \Pr_{\mathrm{reg}}[\sigma_j \geq T \mid j \in \mathrm{e}]\,, \qquad p_0 = \Pr_{\mathrm{reg}}[\sigma_j \geq T \mid j \notin \mathrm{e}]\,.$$

Remark 11.10. We should now come back to the previous comments made about the counters distributions in Remark 11.7. For this purpose, substitute, in Proposition 11.8, γ_0 for γ_0' and γ_1 for γ_1' with

$$\forall k \in \{0, \dots, d)\}, \begin{cases} \gamma_0'(\mathbf{1}_{\geq T}(k), k) = \binom{d}{k} \pi_1^k (1-\pi_1)^{d-k}\,, \\ \gamma_1'(\mathbf{1}_{\geq T}(k), k) = \binom{d}{k} \pi_0^k (1-\pi_0)^{d-k}\,, \end{cases} \tag{11.6}$$

remember from (5.2) that

$$\pi_1 = \frac{S+X}{dt}, \text{ and } \pi_0 = \frac{(w-1)S-X}{d(n-t)}\,.$$

Comparison between these models is provided in Table 11.4. We can observe that compared to Remark 11.9, our model makes a better prediction of the variance. And compared to Remark 11.10, our model makes a better prediction of the mean.

Table 11.4: Comparison between simulation (10^5 samples) and models for $(r, d, t) = (12\,323, 71, 134)$.
t_1^+ is the number of added errors, t_1^- is the number of removed errors, and the error weight after one iteration is $t_1 = t + t_1^+ - t_1^-$.
Kullback-Leibler divergence is computed between the simulation data $\mathcal{S}$ and the distribution $\mathcal{M}$ obtained on the model specified in the first column.
The probabilities $\Pr_{\text{reg}}[(S, X)]$ are computed according to Proposition 11.4.

	t_1^+			t_1^-			t_1		
	Mean	Var.	$\mathrm{D}_{\mathrm{KL}}(\mathcal{S}\|\mathcal{M})$	Mean	Var.	$\mathrm{D}_{\mathrm{KL}}(\mathcal{S}\|\mathcal{M})$	Mean	Var.	$\mathrm{D}_{\mathrm{KL}}(\mathcal{S}\|\mathcal{M})$
$S = 4\,784$ $X = 830$	$\Pr_{\text{reg}}[(S,X)] = 1.001 \cdot 10^{-5}$								
Simulation	9.158	9.115	-	72.491	11.931	-	70.667	20.956	-
Proposition 11.8 & 11.11	9.157	9.102	$1.245 \cdot 10{-}4$	72.513	11.986	$2.240 \cdot 10{-}4$	70.642	21.095	$3.972 \cdot 10{-}4$
(11.5) in Remark 11.9	9.159	9.153	$1.248 \cdot 10{-}4$	72.483	33.276	$2.785 \cdot 10{-}1$	70.675	42.432	$1.451 \cdot 10{-}1$
(11.6) in Remark 11.10	9.496	9.447	$8.955 \cdot 10{-}3$	72.487	11.987	$1.916 \cdot 10{-}4$	71.008	21.434	$4.389 \cdot 10{-}3$
$S = 4\,868$ $X = 820$	$\Pr_{\text{reg}}[(S,X)] = 1.836 \cdot 10^{-5}$								
Simulation	5.888	5.908	-	66.729	11.990	-	73.158	17.746	-
Proposition 11.8 & 11.11	5.910	5.888	$2.059 \cdot 10{-}4$	66.728	12.023	$2.176 \cdot 10{-}4$	73.182	17.904	$2.593 \cdot 10{-}4$
(11.5) in Remark 11.9	5.911	5.906	$2.007 \cdot 10{-}4$	66.717	33.494	$2.792 \cdot 10{-}1$	73.194	39.409	$1.803 \cdot 10{-}1$
(11.6) in Remark 11.10	6.143	6.119	$7.946 \cdot 10{-}3$	66.744	12.021	$2.234 \cdot 10{-}4$	73.400	18.139	$2.653 \cdot 10{-}3$
$S = 4\,868$ $X = 908$	$\Pr_{\text{reg}}[(S,X)] = 3.434 \cdot 10^{-4}$								
Simulation	5.885	5.890	-	75.311	11.944	-	64.574	17.716	-
Proposition 11.8 & 11.11	5.889	5.866	$1.625 \cdot 10{-}4$	75.337	11.962	$2.944 \cdot 10{-}4$	64.555	17.832	$2.803 \cdot 10{-}4$
(11.5) in Remark 11.9	5.892	5.895	$1.482 \cdot 10{-}4$	75.295	32.980	$2.739 \cdot 10{-}1$	64.596	38.886	$1.756 \cdot 10{-}1$
(11.6) in Remark 11.10	6.125	6.095	$7.042 \cdot 10{-}3$	75.290	11.970	$2.949 \cdot 10{-}4$	64.831	18.066	$2.966 \cdot 10{-}3$
$S = 4\,868$ $X = 998$	$\Pr_{\text{reg}}[(S,X)] = 1.947 \cdot 10^{-5}$								
Simulation	5.900	5.923	-	83.895	11.848	-	56.006	17.725	-
Proposition 11.8 & 11.11	5.871	5.845	$3.133 \cdot 10{-}4$	83.966	11.882	$5.381 \cdot 10{-}4$	55.905	17.744	$6.489 \cdot 10{-}4$
(11.5) in Remark 11.9	5.871	5.869	$3.107 \cdot 10{-}4$	83.897	31.374	$2.549 \cdot 10{-}1$	55.976	37.230	$1.588 \cdot 10{-}1$
(11.6) in Remark 11.10	6.102	6.080	$5.108 \cdot 10{-}3$	83.861	11.891	$2.886 \cdot 10{-}4$	56.241	17.974	$2.481 \cdot 10{-}3$
$S = 4\,932$ $X = 852$	$\Pr_{\text{reg}}[(S,X)] = 4.334 \cdot 10^{-5}$								
Simulation	19.260	19.184	-	88.539	11.785	-	64.721	31.092	-
Proposition 11.8 & 11.11	19.312	19.113	$3.606 \cdot 10{-}4$	88.597	11.832	$4.429 \cdot 10{-}4$	64.715	30.948	$3.896 \cdot 10{-}4$
(11.5) in Remark 11.9	19.316	19.309	$3.676 \cdot 10{-}4$	88.511	30.055	$2.380 \cdot 10{-}1$	64.805	49.358	$6.734 \cdot 10{-}2$
(11.6) in Remark 11.10	19.939	19.725	$1.731 \cdot 10{-}2$	88.463	11.827	$5.795 \cdot 10{-}4$	65.476	31.564	$1.346 \cdot 10{-}2$
$S = 5\,002$ $X = 906$	$\Pr_{\text{reg}}[(S,X)] = 1.002 \cdot 10^{-5}$								
Simulation	4.882	4.928	-	75.247	11.890	-	63.635	16.757	-
Proposition 11.8 & 11.11	4.899	4.883	$1.693 \cdot 10{-}4$	75.256	11.947	$2.883 \cdot 10{-}4$	63.644	16.835	$3.723 \cdot 10{-}4$
(11.5) in Remark 11.9	4.900	4.902	$1.465 \cdot 10{-}4$	75.216	33.001	$2.764 \cdot 10{-}1$	63.685	37.883	$1.874 \cdot 10{-}1$
(11.6) in Remark 11.10	5.097	5.077	$6.797 \cdot 10{-}3$	75.210	11.969	$3.729 \cdot 10{-}4$	63.885	17.037	$3.066 \cdot 10{-}3$

11.3.5 Error weight after one iteration

The two random variables that count the number of flipped positions, separated according to whether they are erroneous or not, can be considered independent. The error weight after an iteration is then simply determined by a convolution of their respective distributions.

Proposition 11.11. *Let* $\mathbf{H}$ *be drawn uniformly at random in* $\mathcal{H}_{d,w,r\times n}$ *and* e *drawn uniformly at random in* $\mathcal{E}_{n,t}$*. We denote the syndrome weight* S *and the value* $X = (\sum_{j\in \mathrm{e}} \sigma_j) - S$*, and suppose they are known constants. We suppose that the distributions* $\Pr_{reg}\left[|\{j \in \mathrm{e} \,|\, \sigma_j \geq T|\}| = k \middle| S, X\right]$ *and* $\Pr_{reg}\left[|\{j \notin \mathrm{e} \,|\, \sigma_j \geq T|\}| = l \middle| S, X\right]$ *are known (see Proposition 11.8). The number of errors remaining after one iteration of the bit-flipping algorithm with threshold* T*, is, for* $\delta \in \{-t_0, \dots, n - t_0\}$

$$\Pr_{reg}[t_1 = t_0 + \delta \,|\, S, X] = \sum_{-k+l=\delta} \Pr_{reg}\left[|\{j \in \mathrm{e} \,|\, \sigma_j \geq T|\}| = k \middle| S, X\right] \cdot \Pr_{reg}\left[|\{j \notin \mathrm{e} \,|\, \sigma_j \geq T|\}| = l \middle| S, X\right].$$

11.3.6 Unconditional probability of the error weight after the first iteration

We can now summarize all the propositions of this section in a single distribution that we will compare to the simulation, namely the error weight after an iteration t_1.

Here are the outlines of the nested conditioning on the probabilities that we are going to achieve.

- First we determine, using Proposition 11.4, the joint distribution of S_0 and X_0:

$$\Pr_{\mathrm{reg}}[(S_0, X_0)].$$

- Using Proposition 11.6 & 11.8, for every values of S_0 and X_0 such that $\Pr_{\mathrm{reg}}[(S_0, X_0)]$ is not negligible, and depending on a threshold function, we determine the distribution of the number of added (t_1^+) or removed (t_1^-) errors:

$$\Pr_{\mathrm{reg}}\left[|\{j \in \mathrm{e} \,|\, \sigma_j \geq T|\}| = t_1^- \middle| S_0, X_0\right],$$
$$\Pr_{\mathrm{reg}}\left[|\{j \notin \mathrm{e} \,|\, \sigma_j \geq T|\}| = t_1^+ \middle| S_0, X_0\right].$$

In the end, we can find the distribution we are looking for

$$\Pr_{\mathrm{reg}}[t_1] = \sum_{S_0, X_0} \Pr_{\mathrm{reg}}[(S_0, X_0)] \sum_{t_0 + t_1^+ - t_1^- = t_1} \Pr_{\mathrm{reg}}\left[|\{j \in \mathrm{e} \,|\, \sigma_j \geq T|\}| = t_1^- \middle| S_0, X_0\right] \Pr_{\mathrm{reg}}\left[|\{j \notin \mathrm{e} \,|\, \sigma_j \geq T|\}| = t_1^+ \middle| S_0, X_0\right].$$

A comparison between the distribution of t_1 obtained from this model and the one obtained with the simulations is provided in Figure 11.2. Due to the rapid decay of probabilities in the tails, it is presented on a binary logarithmic scale. We can observe that the model is really accurate and that it succeeds in predicting the probabilities with a precision up to a fraction of a decimal digit in binary logarithm. We note that the same accuracy is obtained with any other block size or threshold function.

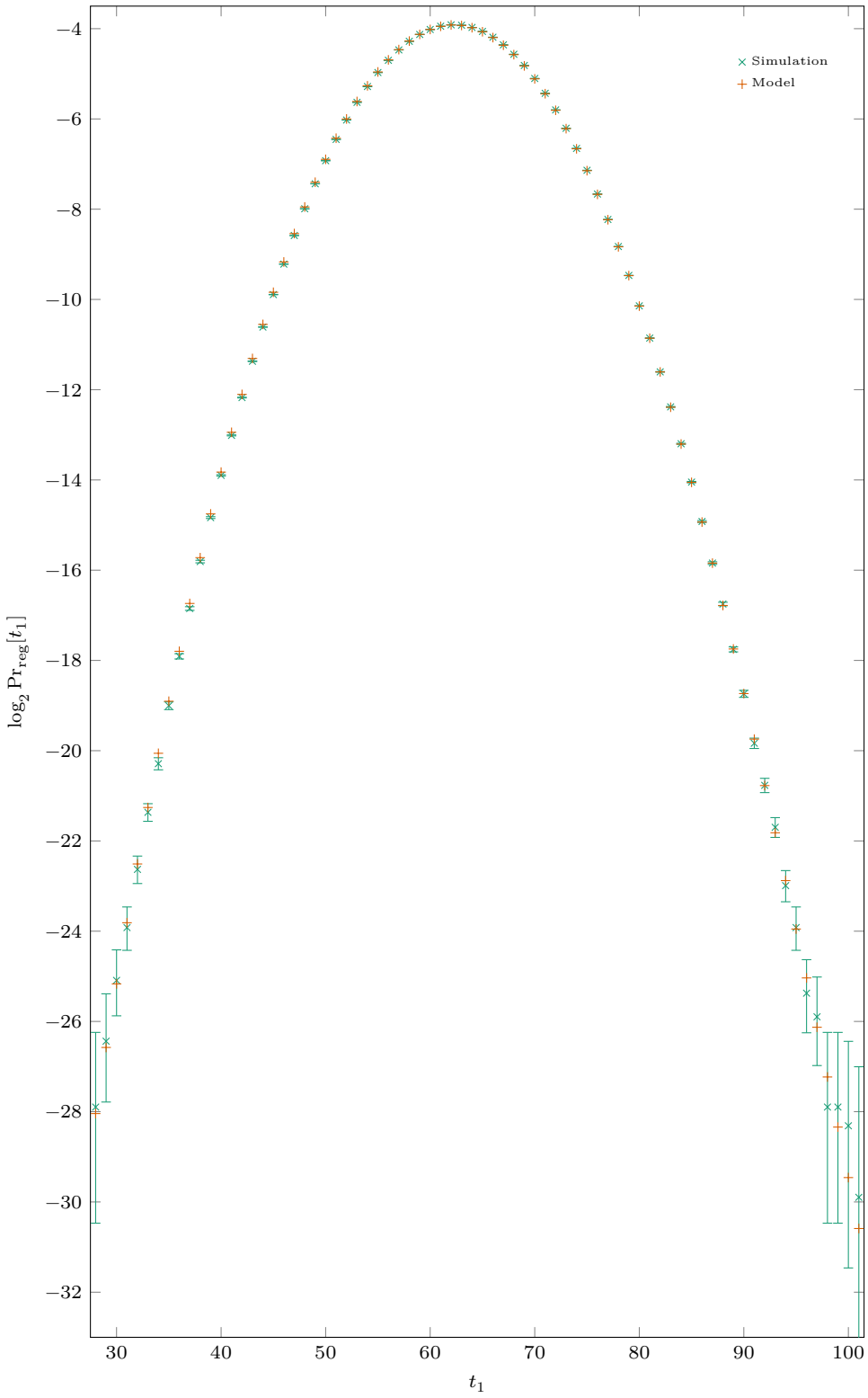


Figure 11.2: Error weight distribution after one iteration. $(r, d, t) = (12\,323, 71, 134)$. Simulation data are for 10^9 samples. 99%-confidence intervals.

11.4 A two-iteration decoder with a DFR analysis

In the previous section, we assessed the accuracy of the model to predict the error weight after one iteration. Using the same idea as in [Til18b] or [LEDA], we designed a decoder whose first iteration is analyzed by the probabilistic model we have just established, followed by a deterministic property which guarantees the successful decoding of an error pattern with a weight below a certain bound. The deterministic property that we use is an intermediate between [Til18b] and [LEDA] and require some key filtering. The reason for this intermediate result is that we can estimate the additional cost of filtering by using results on the distance spectrum proven in Chapter 15.

In this section, we will explore three strategies (for some security parameter λ):

- Do one iteration and choose parameters so that the probalistic model predicts a DFR below $2^{-\lambda}$.
- Do two iterations so that, using the probabilistic model, the error weight after the first iteration is below the deterministic bound with probability above $(1-2^{-\lambda})$.
- Take a reciprocal point of view and discuss the number of errors that must be unconditionally corrected in the second iteration for a given block size to obtain a DFR below $2^{-\lambda}$.

11.4.1 One iteration

Decoding in one iteration means that the error weight t_1 after one iteration is zero. With this decoder, we have

$$\mathrm{DFR}(r) = \sum_{t_1>0} \Pr_{\mathrm{reg}}[t_1]\,.$$

However, with this algorithm, we lose the advantage of an iterative algorithm that has the possibility to make mistakes that are rectified later. The impact on the block size is quite significant since it would have to be multiplied by at least 40 to obtain a sufficiently low DFR. For example, the predicted DFR for $(d,t) = (71,134)$ are shown in Figure 11.3. Obtaining a DFR below 2^{-128} for the 128-bits of security parameters of [BIKE] $(d,t) = (71,134)$ requires a block size $r > 540\,000$.

11.4.2 Two iterations

We can design an algorithm that first performs a parallel iteration of the bit-flip algorithm using a threshold depending on the syndrome weight, and then a second iteration with a fixed threshold. The first one is quite efficient, corrects many errors and has an accurate probabilistic model. The second one is sufficient to correct the residual errors, and its soundness is deterministic and does not require any assumptions about the error pattern or the syndrome.

Let us first recall the definition of the maximum column intersection and prove proposition that is a slight improvement (it can correct up to one more error) of [Til18b, Proposition 1] or a particular case of [LEDA, §3.4].

Definition 11.12. [Til18b, Definition 2] Let $\mathbf{H}$ be a binary parity check matrix. The number of intersections between two different columns j and j' is equal to the number of rows i for which $h_{i,j} = h_{i,j'} = 1$. The *maximum column intersection* of $\mathbf{H}$ is the maximum for any pairs of columns:

$$\max_{j,j'} \left| \mathbf{h}_j \star \mathbf{h}_{j'} \right| .$$

Remark 11.13. In a quasi-cyclic code with a parity check matrix represented as $\mathrm{h} = (\mathrm{h}_0, \mathrm{h}_1) \in (\mathbb{F}_2[x]/(x^r-1))^2$, any column can be written as $x^k \mathrm{h}_\ell$ for some $k \in \{0, \dots, r-1\}$

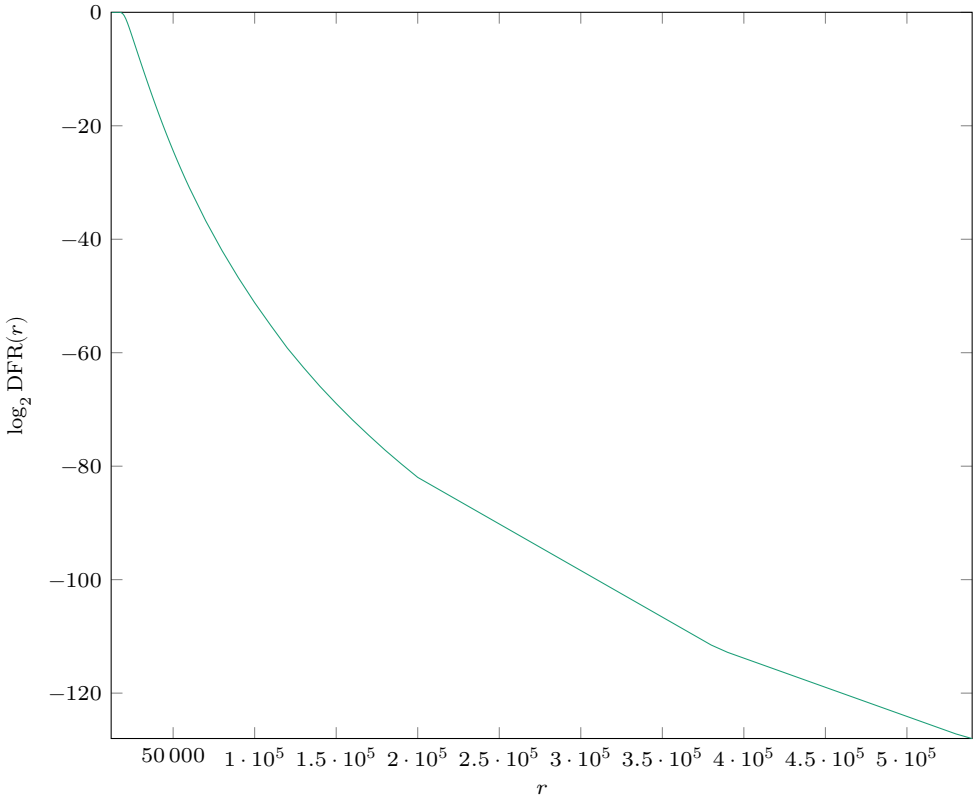


Figure 11.3: Predicted DFR for a one-iteration decoder, for $(d, t) = (71, 134)$

and $\ell \in \{0,1\}$. Since shifting two columns by the same offset does not change their number of intersections, we can rewrite the maximum column intersection formula as

$$I := \max_{\substack{k,k' \in \{0,\dots,r-1\} \\ \ell,\ell' \in \{0,1\}}} \left| x^k \mathrm{h}_\ell \star x^{k'} \mathrm{h}_{\ell'} \right| = \max(I_0, I_1, I_{01}) \,.$$

where

$$I_0 := \max_{k \in \{0,\dots,r-1\}} \left| \mathrm{h}_0 \star x^k \mathrm{h}_0 \right| , \qquad I_1 := \max_{k \in \{0,\dots,r-1\}} \left| \mathrm{h}_1 \star x^k \mathrm{h}_1 \right|$$

are the maximum column intersection inside each block and

$$I_{01} := \max_{k \in \{0,\dots,r-1\}} \left| \mathrm{h}_0 \star x^k \mathrm{h}_1 \right|$$

concerns the intersections between the two blocks.

Proposition 11.14. *Consider a code with a parity check matrix for which every column has weight d and whose maximum column intersection is I. A bit-flipping iteration (Algorithm 11.1) with $T = \lfloor (d+I+1)/2 \rfloor$) corrects all errors of weight $\leq \lfloor (d+I-1)/2I \rfloor$.*

Proof. Let us consider the counters $\sigma_j = \left| \mathrm{h}_j \star \mathrm{s} \right|$ for all the positions $j \in \{0, \dots, n-1\}$, where e is an error pattern of weight t and $\mathbf{H}$ is a parity check matrix of column weight d and maximal column intersection I. From [Til18b], we have, if $j \in \mathrm{e}$,

$$\sigma_j \geq d - I(t-1) \,,$$

and, if $j \notin \mathrm{e}$,

$$\sigma_j \leq It \,.$$

The error pattern e can be corrected if there exists T such that

$$It < T \leq d - I(t-1) \,.$$

The largest t that verifies this inequality is $\left\lfloor \frac{d+I-1}{2I} \right\rfloor$ and we can verify that we can take $T = \left\lfloor \frac{d+I+1}{2} \right\rfloor$. □

We will consider the two-iteration bit-flipping as described in Algorithm 11.2. With Proposition 11.14, we now have an upper bound on its DFR:

$$\mathrm{DFR}(r) \leq \sum_{t_1 > \lfloor (d+I-1)/2I \rfloor} \Pr_{\mathrm{reg}}[t_1] \,.$$

For any maximum column intersection I, we write

$$r_{I,\lambda} := \min \left\{ r \,\middle|\, \sum_{t_1 > \lfloor (d+I-1)/2I \rfloor} \Pr_{\mathrm{reg}}[t_1] < 2^{-\lambda} \right\} ,$$

the minimum block size that one should use to have a DFR below $2^{-\lambda}$ with Algorithm 11.2. Examples of values are given in Table 11.5 for [BIKE] parameters.

Algorithm 11.2: Two iterations of the parallel bit-flipping algorithm.

```
function two_parallel_iterations^⊥(H, s, I):
    input  : A sparse parity check matrix H ∈ F_2^{(n−k)×n},
             a syndrome s = He^⊤ ∈ F_2^{n−k},
             the maximum number of intersections I between columns of H.
    output : The error pattern difference after two iterations e′ ∈ F_2^n.
    e′ ← 0;
    s′ ← s;
    T ← threshold(|s|);
    for j ∈ {0, ..., n − 1} do
        if |h_j ⋆ s| ≥ T then
            e′_j ← 1 − e′_j ;
    s′ ← s − He′^⊤;
    for j ∈ {0, ..., n − 1} do
        if |h_j ⋆ s′| ≥ ⌊(d + I + 1)/2⌋ then
            e′_j ← 1 − e′_j ;
    return e′;
```

Table 11.5: Example values for $r_{I,\lambda}$ with $(d, t) = (71, 134)$ and $\lambda = 128$.

I	$\lfloor (d+I-1)/2I \rfloor$	$r_{I,128}$
1	35	19 778
2	18	24 490
3	12	28 577
4	9	32 408
5	7	36 649
6	6	39 778
7	5	44 104
8–10	4	50 430
11–14	3	60 654
15–23	2	80 728
24–70	1	134 408
71	0	539 795

Key filtering and overhead. One can use the Algorithm 15.3 in §15.6 to filter out keys whose maximum column intersection is too high. We can estimate the probability that the key generation function draws a bad key and the number of trials before finding a good key follows a geometric distribution. We can consider this number of trials as an overhead factor on the key generation. However, the additional costs only concern the entropy generation and this does not represent the major part of the cost in a Niederreiter scheme such as BIKE, where the most expensive operation is a polynomial (or matrix) inversion.

Remember that in Remark 11.13 we have separated the maximum column intersection I between its intrablocks parts I_0, I_1 and its interblock part I_{01}. Therefore, we have:

$$\Pr[I \leq m] = \Pr[I_0 \leq m \text{ and } I_1 \leq m \text{ and } I_{01} \leq m]\,.$$

The values I_0 and I_1 concern independent blocks and are therefore independent random

Table 11.6: Overhead due to key filtering for a maximum column intersection value I and block size r with $(d, t) = (71, 134)$.

I	r	Avg. keygen overhead
2	24 490	$+1.5 \cdot 10^{+26}\%$
3	28 577	$+429.26\%$
4	32 408	$+3.24\%$
5	36 649	$+0.04\%$

variables, so we can write:

$$\Pr[I \leq m] = \Pr[I_0 \leq m] \cdot \Pr[I_1 \leq m] \cdot \Pr[I_{01} \leq m \,|\, I_0 \leq m, I_1 \leq m] \,.$$

For ease of calculation, and only at the cost of a slight inaccuracy, we will also assume the independence with the interblock part I_{01}: $\Pr[I_{01} \leq m \,|\, I_0 \leq m, I_1 \leq m] = \Pr[I_{01} \leq m]$.

We will use results from Chapter 15 and thus the proofs of the different assertions can be found there. In particular, it is shown in this chapter that there is a link between the column intersections inside a circulant block and its *distance spectrum*. The relevant combinatorial result from Corollary 15.17 is reported here without justification.

Proposition 11.15. *For given integers m, $0 \leq m < d$, and δ, $1 \leq \delta \leq \lfloor r/2 \rfloor$, define*

$$\mathcal{N}_m := \frac{r}{d-m} \binom{d-1}{d-m-1} \binom{r-d-1}{d-m-1} ,$$

$$\pi_m = \frac{\mathcal{N}_m}{\binom{r}{d}} = \frac{r}{d-m} \frac{\binom{d-1}{d-m-1}\binom{r-d-1}{d-m-1}}{\binom{r}{d}} , \qquad \pi'_m = \frac{\binom{d}{m}\binom{r-d}{d-m}}{\binom{r}{d}} .$$

Then

$$\Pr[I_0 \leq m] = \Pr[I_1 \leq m] = \left(\sum_{k \leq m} \pi_k \right)^{\lfloor r/2 \rfloor} , \quad \Pr[I_{01} \leq m] = \left(\sum_{k \leq m} \pi'_k \right)^{r} .$$

In Algorithm 15.3, for a threshold $\tau \in \{1, \ldots, d\}$, the probability of accepting a key is $\Pr[I \leq m]$. Since all the key generations are independent, the number of trials before finding a suitable key follows a geometric distribution. In Table 11.6 we give the mean values for different parameter sets. Filtering keys to have a maximum column intersection of 3 appears to be an interesting tradeoff, and for any greater value it has a negligible influence on the key generation time.

11.4.3 Decoding performance requirements after the first iteration

With our model, we can estimate the greatest error weight after one iteration that happens with a non-negligible probability (*i.e.* larger than $2^{-\lambda}$). Results are given in Figure 11.4. The figure shows that, for a block size of about 21 000, we need to successfully decode with a very high probability any error pattern whose weight does not exceed a few tens. Such a performance does not seem unattainable in view of our experimentations. For example, the error patterns responsible for the error floors discussed in Chapter 16 are the worst error patterns that we know of in terms of decoding, but they have a larger weight.

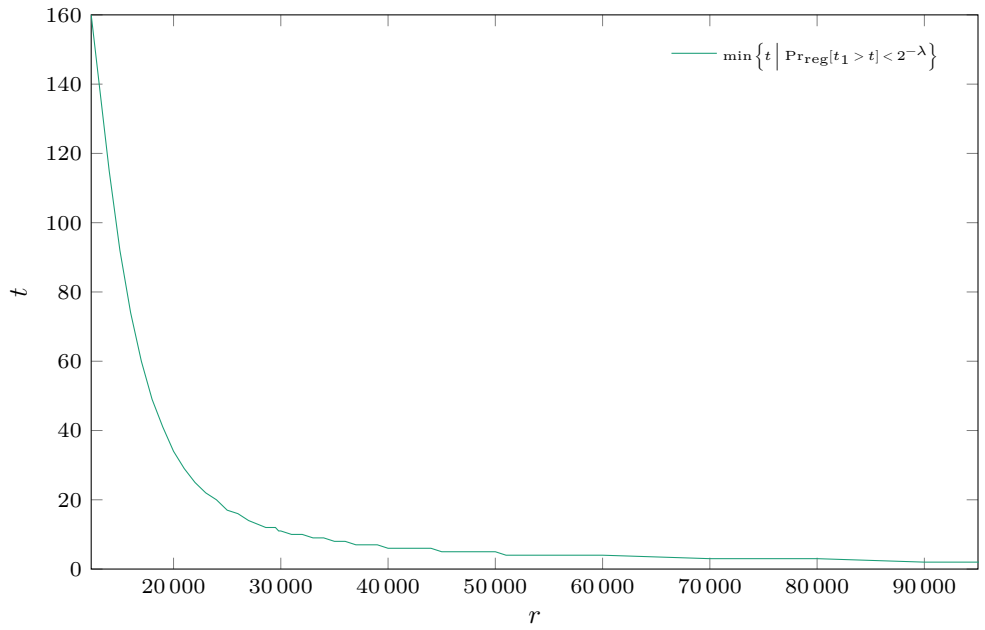


Figure 11.4: Number of errors that needs to be decoded after one iteration to have a DFR below $2^{-\lambda}$, for $(d, t) = (71, 134)$

11.5 Noisy syndrome decoding

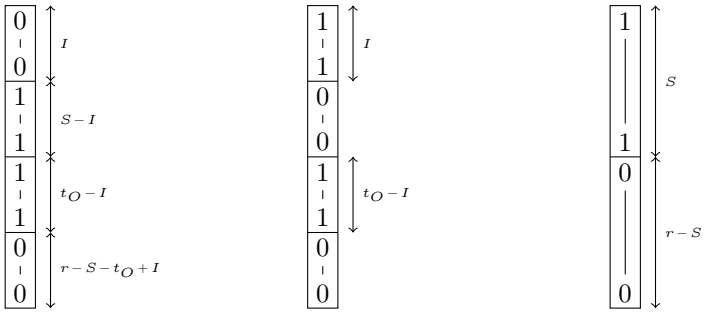
Ouroboros is a key exchange protocol that uses MDPC codes, see [DGZ17]. In comparison to the MDPC-McEliece scheme, Ouroboros relies only on one security assumption (Quasi-Cyclic Syndrome Decoding, QCSD) instead of two (QCSD and Quasi-Cyclic Codeword Finding), see BIKE-3 variants of [BIKE] for the versions earlier than 3.2. It also differs in the decoder: in Ouroboros one needs to decode a noisy syndrome *i.e.* $\mathrm{s} = \mathbf{H}\mathrm{e}^\top + \mathrm{e}'$. This requires only a small modification of a bit-flipping algorithm to decode. It is a matter of stopping the algorithm when the syndrome has a small weight rather than a zero weight.

The analysis of an iteration also requires some adaptation of the model. Basically, contrary to the noiseless problem, a one in the syndrome does not mean that there is an odd number of errors in the corresponding equation (and conversely a zero in the syndrome does not mean an even number of errors). To give an idea of the impact of the added noise, with BIKE-3 (Ouroboros), $\left|\mathrm{e}'\right| = \left|\mathrm{e}\right|/2$.

In fact, the additional noise on the syndrome only affects the weight of the syndrome and the counters distributions, the other propositions of the previous section then apply without further adaptation. In this section we show how the noise on the syndrome is to be taken into account in the relevant propositions.

Since the number of equations affected is known and fixed in the parameters of the scheme, we know exactly how many syndrome bits will have their parity flipped. We still use the notation S for the syndrome weight $S = \left|\mathbf{H}\mathrm{e}^\top + \mathrm{e}'\right|$, we write the syndrome noise weight $t_O := \left|\mathrm{e}'\right|$. Now, we need an additional random variable to account for the number of unsatisfied equations affected by the noise on the syndrome $I = \left|\mathrm{e}' \star \mathrm{s}\right|$. The noiseless syndrome weight is then

$$\left|\mathrm{s} + \mathrm{e}'\right| = S + t_0 - 2I\,.$$



Remark 11.16. We already know that if s is drawn uniformly at random in $\mathcal{E}_{r,S}$ and e' drawn uniformly at random in $\mathcal{E}_{r,t_O}$, the number of intersection between them follows a hypergeometric distribution, we write

$$f(S,I) = \frac{\binom{S+t_O-2I}{t_O-I}\binom{r-S-t_O+2I}{I}}{\binom{r}{t_O}}$$

the probability of the vector e' having I intersections with s.

When e and s have I intersections, there are exactly I unsatisfied equations with an even number of errors, $t_O - I$ satisfied equations with an odd number of errors, the other equations behave as expected.

Proposition 11.17. *Let* $\mathbf{H}$ *be drawn uniformly at random in* $\mathcal{H}_{d,w,r\times n}$, e *drawn uniformly at random in* $\mathcal{E}_{n,t}$ *and* e' *drawn uniformly at random in* $\mathcal{E}_{r,t_O}$. *The syndrome weight* $S = \left|\mathrm{e}' + \mathbf{H}\mathrm{e}^\top\right|$, *the value* $X = (\sum_{j\in \mathrm{e}} \sigma_j) - S$, *and the value* $I = t_O - \left|\mathrm{e}' \star \mathrm{s}\right|$ *follow the joint distribution*

$$\Pr_{reg}[(S,X,I) = (k,\ell,o)] = \frac{1}{N_{\zeta,\overline{\zeta}}}\, f(k,o)\, \left(\zeta^{*(r-t_O)} * \overline{\zeta}^{*t_O}\right)(k, k+\ell, dt-k-\ell)$$

with, for $a \in \{0,\dots,\min(w,t)\}$,

$$\begin{cases} \overline{\zeta}(1,a,0) = \frac{\binom{w}{a}\binom{n-w}{t-a}}{\binom{n}{t}} & \text{if } a \text{ is even}, \\ \overline{\zeta}(0,0,a) = \frac{\binom{w}{a}\binom{n-w}{t-a}}{\binom{n}{t}} & \text{if } a \text{ is odd}, \end{cases} \qquad \begin{cases} \zeta(0,0,a) = \frac{\binom{w}{a}\binom{n-w}{t-a}}{\binom{n}{t}} & \text{if } a \text{ is even}, \\ \zeta(1,a,0) = \frac{\binom{w}{a}\binom{n-w}{t-a}}{\binom{n}{t}} & \text{if } a \text{ is odd}. \end{cases}$$

(compared to ζ *the first component of* $\overline{\zeta}$ *is flipped) and*

$$N_{\zeta,\overline{\zeta}} = \sum_{k,\ell,o} f(k,o)\, \left(\zeta^{*(r-t_O)} * \overline{\zeta}^{*t_O}\right)(k, k+\ell, dt-k-\ell)\,.$$

Proposition 11.18. *Let* $\mathbf{H}$ *be drawn uniformly at random in* $\mathcal{H}_{d,w,r\times n}$, e *drawn uniformly at random in* $\mathcal{E}_{n,t}$ *and* e' *drawn uniformly at random in* $\mathcal{E}_{r,t_O}$. *We denote the syndrome weight* S, *the value* $X = (\sum_{j\in \mathrm{e}} \sigma_j) - S$ *and the value* $I = t_O - \left|\mathrm{e}' \star \mathrm{s}\right|$, *and suppose they*

Table 11.7: Comparison of the distribution of S, X and I between simulation (10^9 samples with quasi-cyclic matrices) and the model for $(r, d, t, t_O) = (12\,269, 67, 154, 77)$.
The Kullback-Leibler divergence between the simulation data $\mathcal{S}$ and the distribution obtained from Proposition 11.17 is $9.015 \cdot 10^{-4}$.

	S		X		I	
	Mean	Var.	Mean	Var.	Mean	Var.
Simulation	5 030.068	2 618.891	1 080.049	1 643.046	45.520	18.596
Proposition 11.17	5 030.225	2 618.815	1 079.920	1 642.820	45.519	18.596

Table 11.8: Comparison of the counters distributions between simulation (10^5 samples) and models for $(r, d, t, t_O) = (12\,269, 67, 154, 77)$.
Kullback-Leibler divergence is computed between the simulation data $\mathcal{S}$ and the distribution $\mathcal{M}$ obtained on the model specified in the first column.
The probabilities $\Pr_{\text{reg}}[(S, X, I)]$ are computed according to Proposition 11.17.

			$j \in \text{e}$		$j \notin \text{e}$	
	Mean	Var.	$\text{D}_{\text{KL}}(\mathcal{S}\|\mathcal{M})$	Mean	Var.	$\text{D}_{\text{KL}}(\mathcal{S}\|\mathcal{M})$
$S = 5\,031$	$X = 1\,080$	$I = 46$	$\Pr_{\text{reg}}[(S, X, I)] = 2.842 \cdot 10^{-5}$			
Simulation	27.397	16.107	-	39.682	15.979	-
Proposition 11.18	27.398	16.107	$3.75 \cdot 10^{-7}$	39.681	15.982	$4.31 \cdot 10^{-7}$

are known constants. The counters $\sigma_j = \left|\mathbf{h}_j \star \text{s}\right|$ *follows the distributions for* $k \in \{0, \dots, d\}$

$$\begin{aligned}
\Pr_{reg}[\sigma_j = k \mid j \in \text{e}, S, X, I] &= \frac{1}{N'(\phi)} \left(\phi_0^{*I} * \phi_1^{*(S-I)}\right)(k, S+X) \\
&\quad \cdot \left(\phi_1^{*(t_O-I)} * \phi_0^{*(r-S-t_O+I)}\right)(d-k, dt-(S+X)), \\
\Pr_{reg}[\sigma_j = k \mid j \notin \text{e}, S, X, I] &= \frac{1}{N'(\psi)} \left(\psi_0^{*I} * \psi_1^{*(S-I)}\right)(k, (w-1)S-X) \\
&\quad \cdot \left(\psi_1^{*(t_O-I)} * \psi_0^{*(r-S-t_O+I)}\right)(d-k, d(n-t)-(w-1)S+X),
\end{aligned}$$

(the functions ϕ_s *and* ψ_s *are defined as in Proposition 11.6) with*

$$\begin{aligned}
N'(\phi) &= \sum_{k=0}^{d} \left(\phi_0^{*I} * \phi_1^{*(S-I)}\right)(k, S+X) \\
&\quad \cdot \left(\phi_1^{*(t_O-I)} * \phi_0^{*(r-S-t_O+I)}\right)(d-k, dt-(S+X)), \\
N'(\psi) &= \sum_{k=0}^{d} \left(\psi_0^{*I} * \psi_1^{*(S-I)}\right)(k, (w-1)S-X) \\
&\quad \cdot \left(\psi_1^{*(t_O-I)} * \psi_0^{*(r-S-t_O+I)}\right)(d-k, d(n-t)-(w-1)S+X).
\end{aligned}$$

Comparisons between the model and simulation are given in Table 11.7 & 11.8.

11.6 Going further to predict the syndrome weight after the first iteration

Concerning the bounds used in the previous section for the two-iteration decoding, although they do not require additional assumptions about the error pattern, they do characterize worst-case scenarios. In particular, the worst possible error patterns produce syndromes with unusual weights. Obtaining the joint distribution of the error weight and the syndrome weight after an iteration would allow for a finer analysis of the DFR since outliers would be weighted by their extremely low probability. Knowledge of this distribution also appears necessary if we plan to consider algorithms performing more than two iterations.

In this section, we outline the possibility of predicting the syndrome weight after one iteration. The difficulty with the ensuing model is that it is more complex and requires too much computational time.

Similarly to the mass equations of Section 11.2, we can apply permutations to the parity check matrix and the vectors to have more precise equations. To evaluate the syndrome changes (a one that becomes a zero and *vice versa*), as before, we need to take into account some mass equations. We suppose that the number of correct positions to be flipped t^+ and the number of errors to be removed t^- is known. We also suppose that the sum Σ^+ of the t^+ counters of the correct positions to be flipped, and the sum Σ^- of the t^- counters of errors to be removed are fixed.

Again, applying two permutations $\mathbf{P'} \in \mathbb{F}_2^{r\times r}$ and $\mathbf{Q'} \in \mathbb{F}_2^{n\times n}$ to $\mathbf{H}$, e and s:

$$\mathbf{H''} = \mathbf{P'}\mathbf{H}\mathbf{Q'}^{-1} \qquad \mathrm{e''} = \mathrm{e}\mathbf{Q'}^{\top} \qquad \mathrm{s''} = \mathbf{P'}\mathrm{s}$$

so that we have $\mathrm{s''} = \mathrm{e''}\mathbf{H''}^{\top}$ in the configuration illustrated in Figure 11.5.

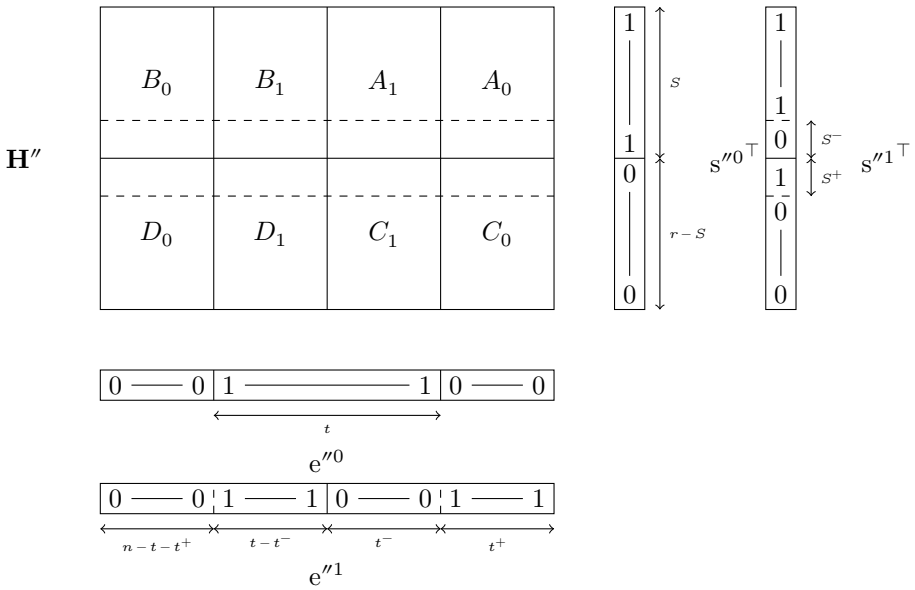


Figure 11.5: New reordering of the parity check matrix.

Now, the equations are

$$\begin{aligned} A_1 &= \Sigma^- \,, & A_0 &= \Sigma^+ \,, \\ A_1 + B_1 &= S + X \,, & A_0 + B_0 &= (w-1)S - X \,, \\ A_0 + C_0 &= dt^+ \,, & A_1 + C_1 &= dt^- \,, \\ A_0 + B_0 + C_0 + D_0 &= d(n-t) \,, & A_1 + B_1 + C_1 + D_1 &= dt \,. \end{aligned}$$

With some independence assumptions, the distributions of Σ^+ and Σ^- can be determined as well as the repartition of the ones for each row of the parity check matrix.

A derivation, albeit more complex, similar to the one presented here for the error weight after the first iteration can be established for the syndrome weight after the first iteration using these mass equations. The distributions thus calculated on a few particular starting values for S_0 and X_0 seemed to be in line with the simulations. However, the cost of these few calculations did not bode well for a complete model.

Chapter 12

Markovian model of the step-by-step algorithm

In this chapter, unlike the previous one, we make coarser assumptions about the decoder's behaviour. However, these assumptions are reasonable and allow us to model a complete iterative decoder. We will still rely on some results of the previous chapter.

12.1 Notations

We adopt the following notations in this chapter.

General.

- The set $\mathcal{E}_{n,t}$ is set of all the error patterns of weight t

$$\mathcal{E}_{n,t} = \{\mathrm{e} \in \{0,1\}^n \mid |\mathrm{e}| = t\}\,,$$

- $\mathcal{H}_{d,w,r\times n}$ is the set of regular $r \times n$ parity check matrices of column weight d and row weight w

$$\mathcal{H}_{d,w,r\times n} = \{\mathbf{H} \in \mathbb{F}_2^{r\times n} \mid \forall i \in \{0,\dots,r-1\}, \left|\mathbf{h}_i^\top\right| = w, \forall j \in \{0,\dots,n-1\}, \left|\mathbf{h}_j\right| = d\}\,.$$

Coding theory.

- We denote the syndrome weight corresponding to a certain error parttern

$$S = |\mathrm{s}| \qquad \mathrm{s} = \mathbf{H}\mathrm{e}^\top,.$$

- For a position $j \in \{0,\dots,n-1\}$, we write its counter $\sigma_j := \left|\mathrm{h}_j \star \mathrm{s}\right|$.
- We write

$$X = \sum_{j\in \mathrm{e}} \sigma_j - S\,.$$

- The definitions of σ_i, E_ℓ, S and X depend on a specific parity check matrix $\mathbf{H}$ and an error pattern e. These dependencies will always be evident from the context and are therefore not explicitly mentioned so as not to clutter the notations.

12.2 Algorithm supported by the model

Remember the definition of step-by-step bit-flipping, Algorithm 7.1 already discussed in Chapter 7.

One of its key features is that it is easy to track the error weight and the syndrome weight at each step if we know whether the flipped position is an error or not as well as the value of its counter. Indeed, by Proposition 1.26, when the position j with the counter σ_j is flipped the weight of the syndrome becomes $|\mathrm{s}| + \left|\mathbf{h}_j\right| - 2\sigma_j$.

In the following section, we establish the assumptions that we make to estimate the probabilities associated with the different events that affect the outcome of the algorithm. We will see that it describes, in fact, a Markov chain. We then define some additional assumptions that are made mainly out of convenience as they greatly simplify the calculations in the model. In another section we detail the transition probabilities of the Markov chain. And finally we compare the predicted DFR in this model to the one obtained by simulations of Algorithm 7.1 for a great range of block size r.

Necessity of the randomization. The algorithm is fully randomized to allow us, firstly to use a slightly more efficient sampling method than going through all the positions, and secondly to avoid any bias when doing so.

12.3 Assumptions

Let $\mathbf{H}$ be drawn uniformly at random in $\mathcal{H}_{d,w,r\times n}$ and e be a vector of weight t. We write the syndrome $\mathrm{s} := \mathbf{H}\mathrm{e}^\top$. We write $t := |\mathrm{e}|$ and $S := |\mathrm{s}|$.

First, we assume that the counters behave as written in [Cha17].

Assumption 1. *The counters $\sigma_j = \left|\mathbf{h}_j \star \mathrm{s}\right|$ are independent and follow binomial distributions depending on wether j is in the support of the error pattern* e *or not:*

$$\sigma_j \sim \mathrm{Bin}(d, \pi_0) \text{ if } j \notin \mathrm{e}\,, \quad \sigma_j \sim \mathrm{Bin}(d, \pi_1) \text{ if } j \in \mathrm{e}\,.$$

with

$$\pi_1 = \frac{S + \overline{X}}{dt}\,, \qquad \pi_0 = \frac{(w-1)S - \overline{X}}{d(n-t)}$$

and (see Proposition 12.2)

$$\overline{X} = \xi\,\mathrm{E}[X \mid S, t]$$

for some constant ξ.

Remark 12.1. We have seen in the previous chapter that while the counter distributions above are extremely close to the observations that there remains a small bias (see §11.3.3).

We write S_i and t_i the random variables accounting for respectively the syndrome weight and the error weight at the i-th iteration of the step-by-step algorithm. We also define the event L_i that the decoder is in a blocked state at the i-th iteration, this happens when there is no position with a counter greater than or equal to the threshold:

$$L_i := \{\forall j, \sigma_j^{(i)} < T\}\,.$$

where $\sigma_j^{(i)}$ is the counter associated to position j at the i-th iteration.

Assumption 2. *The stochastic process in Algorithm 7.1 is a time-homogeneous Markov chain,* i.e. *the sequence $(S_i, t_i)_i$ is such that for all $i \geq 1$*

$$\begin{aligned}\Pr\Big[(S_{i+1}, t_{i+1}) = (a_{i+1}, b_{i+1}) \,\Big|\, \neg L_i, (S_i, t_i) = (a_i, b_i), \dots, \neg L_0, (S_0, t_0) = (a_0, b_0)\Big] \\ = \Pr\Big[(S_i, t_i) = (a_{i+1}, b_{i+1}) \,\Big|\, \neg L_{i-1}, (S_{i-1}, t_{i-1}) = (a_i, b_i)\Big]\,.\end{aligned}$$

Proposition 12.2. *Assuming that the random variables that indicate the number of errors* $\left|\mathbf{h}_i^\top \star \mathrm{e}\right|$ *for some equation indices* i *are independent and if* $\mathrm{e} \in \mathcal{E}_{n,t}$*, the expectation of* $X := \sum_{\ell>0} 2\ell E_{2\ell+1}$ *knowing* $S = \left|\mathbf{H}\mathrm{e}^\top\right|$ *is*

$$\mathrm{E}[X \mid S, t] = S \cdot \frac{\sum_\ell 2\ell\rho_{2\ell+1}}{\sum_\ell \rho_{2\ell+1}} \text{ where } \rho_\ell = \frac{\binom{w}{\ell}\binom{n-w}{t-\ell}}{\binom{n}{t}}.$$

Proof. For any row $i \in \{0, \dots, r-1\}$, we define R_i the random variable defined as follows

$$R_i = \begin{cases} 0 & \text{if } s_i = 0, \\ \left|\mathbf{h}_i^\top \star \mathrm{e}\right| - 1 & \text{otherwise.} \end{cases}$$

If we know the error weight t, then $\mathrm{E}[R_i \mid s_i = 1] = \frac{\sum_\ell 2\ell\rho_{2\ell+1}}{\sum_\ell \rho_{2\ell+1}}$. Assuming independence, the expected sum of S of these random variables would be equal to S multiplied by the expected value of a single one. □

Remark 12.3. The model derived from these two assumptions gradually degenerates as the number of iterations grows, but remains relatively accurate in the first few iterations of Algorithm 7.1, that is even when e is not uniformly distributed.

Markov chain. These two assumptions allow us to model the step-by-step decoder as a finite state machine (FSM) with a state (S, t) with S the syndrome weight and t the error weight. Assumption 2 compels us to separate the situation where the decoder is blocked from the one where it is not. Indeed, transitioning to a blocked state concerns the whole "population" of counters while the other transitions concern the sampling method. As a result, the transitions of the finite state machine follow the configuration shown in Figure 12.1.

An error pattern is successfully decoded in f iterations if the FSM follows a path that leads to the state $S = 0, t = 0$:

$$(S_0, t_0) \to (S_1, t_1) \to \cdots \to (S_{f-1}, t_{f-1}) \to (0, 0).$$

To estimate the probability of success of the decoder (from which we easily deduce the DFR[1]) in the model defined by the two assumptions, we want to compute the probability of these events for any possible starting state (S_0, t_0). The possible starting states are constrained by the structure of the code and the weight of the initial error pattern.

Validity of the Markovian model. In Assumption 1, we introduced a factor ξ next to $\mathrm{E}[X \mid S, t]$. This factor compensates for the fact that the process is not strictly Markovian. The problem lies in the term X. The decoder flips high counters in order to decrease the syndrome weight. If we focus on an unsatisfied equation that becomes satisfied in this process, if it contained more than one error, they also see their counters decreasing. The error pattern obtained during decoding is therefore not random and the distribution of the counters is biased. They are lower than what is expected for a given syndrome weight S and error weight t. In order to take this bias into account, a factor ξ, close to 1 is added.

[1] In fact, to ensure sufficient accuracy with floating point operations, an actual implementation of the model would rather calculate the complement of the probability of success: the probability of reaching a blocked state.

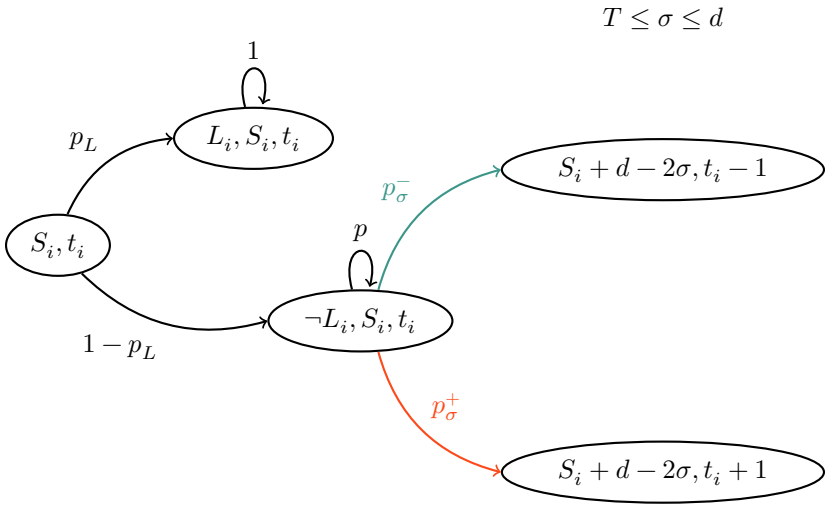


Figure 12.1: Diagram of the transitions profile at the i-th iteration starting from a state where the syndrome weight is S_i and the error weight is t_i. The edges are assigned probabilities that we establish in section 12.5 (note that they depend on S_i and t_i).

12.4 DFR estimation within the model

Additional assumptions. Although it is not strictly necessary for the calculation of the DFR in this model, for this work we specify the decoder so that

1. the threshold T is always greater than or equal to $(d+1)/2$,
2. the number of iterations is not limited.

The first assumption ensures that, in the algorithm, any flip reduces the weight of the syndrome by at least 1. Indeed, if a position j with the counter $\sigma_j \geq (d+1)/2$ is flipped at iteration $\#i$, the syndrome weight becomes

$$S_{i+1} = S_i + d - 2\sigma_j \leq S_i - 1\,.$$

Note that if the algorithm is implemented with a mechanism that exits the loop and outputs a decoding failure when none of the n counters hits the threshold, the probability that it will never terminate is zero. In this sense, the second assumption is still legitimate for a concrete algorithm (albeit conflicting with any goal of guaranteeing a constant-time execution).

These two additional assumptions significantly reduce the time complexity of the model computations. They guarantee that there is no loop in the Markov chain. The former ensures that transitions from a state (S,t) can only go in the direction that lowers its first component S. The latter allows another representation of the Markov chain without any self-loop (apart from those with a probability of 1). This is tantamount to considering an equivalent algorithm whose main loop would be

- sample positions at random until you find one with a counter above the threshold,
- flip it.

We can thus calculate for each state (S,t) the probability that it leads to a failure once and for all as long as we do it for S in ascending order.

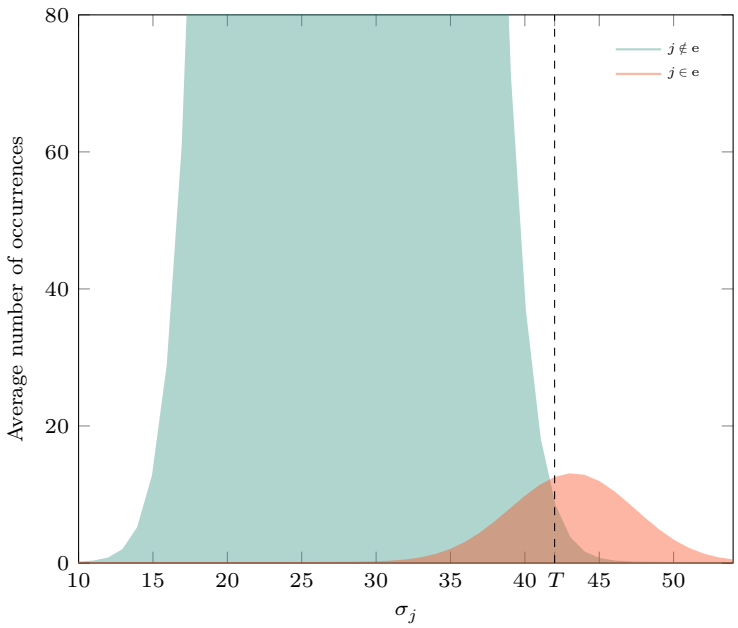


Figure 12.2: Example of a threshold choice for $(r, d, t) = (12\,323, 71, 137)$ and $S = 4\,868$ ($\overline{X} = 908.87$)

Threshold. The choice of a good threshold has been discussed in Part II. In this chapter we choose the smallest value T such that

$$\Pr[\sigma_j \geq T, j \in \mathrm{e}] \geq \Pr[\sigma_j \geq T, j \notin \mathrm{e}]\,.$$

In other words, if we plot, with on the x-axis the counter value and on the y-axis the expected number of positions that have this counter value, the two curves concerning respectively the errors and the correct positions, the threshold is the intersection of the two curves (see Figure 12.2).

With Assumption 1, we have

$$T := \max\left(\frac{d+1}{2}, \left\lceil \frac{d\log\left(\frac{1-\pi_1}{1-\pi_0}\right) + \log\left(\frac{t}{n-t}\right)}{\log\left(\frac{1-\pi_1}{1-\pi_0}\right) + \log\left(\frac{\pi_0}{\pi_1}\right)} \right\rceil\right).$$

One edge case should be treated separately. If $\pi_1 > 1$, take $T = d$. This can happen because the value of $\overline{X}$ is only an estimation based on an expected value.

With this theoretical model, the decoder needs to know the value of t, *i.e.* the actual number of remaining errors. A more realistic threshold value can be used, based, for example, on an estimate of the weight of the error as a function of the syndrome weight (see [CS16]).

12.5 Transition probabilities

Let us break down probabilities tied to the edges in the diagram of Figure 12.1. We consider one iteration of the loop in Algorithm 7.1. A threshold T has been computed.

1. For any position j, the counter $\sigma = \left|\mathbf{h}_j \star \mathrm{s}\right|$ is below the threshold. The decoder is in a blocked state. This happens with probability p_L.

2. There exists at least one counter greater than or equal to the threshold, this happens with probability $1 - p_L$.

   Now suppose position j has been sampled, the corresponding counter value is denoted $\sigma = \left|\mathbf{h}_j \star \mathrm{s}\right|$. From this point, there are three different types of transition probabilities:

   (a) if $\sigma \geq T$ and $j \in \mathrm{e}$, then $(S,t) \to (S + d - 2\sigma, t-1)$, with probability p^-_σ,
   (b) if $\sigma \geq T$ and $j \notin \mathrm{e}$, then $(S,t) \to (S + d - 2\sigma, t+1)$, with probability p^+_σ,
   (c) if $\sigma < T$, then $(S,t) \to (S,t)$, with probability p.

Counters probabilities notation. For ease of reading and for the sake of genericity, we adopt the following mathematical notations for the probabilities of the counters, for all $\sigma \in \{0, \dots, d\}$

$$f^1_{S,t}(\sigma) := \Pr\left[\left|\mathbf{h}_j \star \mathrm{s}\right| = \sigma \;\middle|\; |\mathrm{e}| = t, |\mathrm{s}| = S, j \in \mathrm{e}\right],$$
$$f^0_{S,t}(\sigma) := \Pr\left[\left|\mathbf{h}_j \star \mathrm{s}\right| = \sigma \;\middle|\; |\mathrm{e}| = t, |\mathrm{s}| = S, j \notin \mathrm{e}\right].$$

Under Assumption 1

$$f^1_{S,t}(\sigma) = \binom{d}{\sigma} \pi_1^\sigma (1-\pi_1)^{d-\sigma}, \qquad f^0_{S,t}(\sigma) = \binom{d}{\sigma} \pi_0^\sigma (1-\pi_0)^{d-\sigma}.$$

12.5.1 Blocked state

Let us first treat the case where the entire "population" of positions has counters below the threshold. In this case, the algorithm would simply be caught in an infinite loop.

Proposition 12.4. *Under Assumption 1, we have*

$$\begin{aligned} p_L :=& \Pr\left[(S,t) \to (L,S,t)\right] = \Pr\left[\forall j \in \{0,\dots,n-1\}, \sigma_j < T \,\middle|\, S,t\right] \\ &= \left(\sum_{\sigma<T} f^1_{S,t}(\sigma)\right)^t \cdot \left(\sum_{\sigma<T} f^0_{S,t}(\sigma)\right)^{n-t}. \end{aligned}$$

12.5.2 Transitions from a non-blocked state

The transition probabilities from a non-blocked state depend on the counters distribution as well as the sampling method. For all $\sigma \in \{T, \dots, d\}$,

$$p^-_\sigma := \Pr\left[j \leftarrow \texttt{sample}(\textit{context}), j \in \mathrm{e}, \left|\mathbf{h}_j \star \mathrm{s}\right| = \sigma\right],$$
$$p^+_\sigma := \Pr\left[j \leftarrow \texttt{sample}(\textit{context}), j \notin \mathrm{e}, \left|\mathbf{h}_j \star \mathrm{s}\right| = \sigma\right].$$

The probability of staying in the same state is then:

$$p = \sum_{\sigma<T} (p^-_\sigma + p^+_\sigma).$$

In Chapter 7, we have defined three sampling methods. In the order we treat them, each method induces a bias towards large counters that is stronger than the previous one.

Uniform sampling. The probabilities for the method of simply choosing a position $j \in \{0, \dots, n-1\}$ uniformly at random are obtained by simply weighting the distributions of the counters by the error rate or its complement:

$$\forall \sigma \in \{T, \dots, d\} \qquad p_\sigma^- = \frac{t}{n} f_{S,t}^1(\sigma)\,, \qquad p_\sigma^+ = \frac{(n-t)}{n} f_{S,t}^0(\sigma)\,.$$

Picking a position in one unsatisfied equation. With the method consisting in choosing an unsatisfied equation uniformly at random then, still uniformly at random, one position involved in the equation, counters distributions are affected by a factor linear in σ.

$$\forall \sigma \in \{T, \dots, d\} \qquad p_\sigma^- = \frac{\sigma t}{wS} \cdot f_{S,t}^1(\sigma)\,, \qquad p_\sigma^+ = \frac{\sigma(n-t)}{wS} \cdot f_{S,t}^0(\sigma)\,.$$

Picking a position in two unsatisfied equations. When the same principle is applied with two unsatisfied equations, counters distributions are affected by a factor quadratic in σ.

$$\forall \sigma \in \{T, \dots, d\} \qquad p_\sigma^- = \frac{\sigma(\sigma-1)t}{D} \cdot f_{S,t}^1(\sigma)\,, \qquad p_\sigma^+ = \frac{\sigma(\sigma-1)(n-t)}{D} \cdot f_{S,t}^0(\sigma)$$

where $D = d(d-1)(t\pi_1^2 + (n-t)\pi_0^2)$.

Infinite number of iterations. As explained in §12.4, not setting a limit to the number of iterations has advantages in terms of computational complexity. In this case, we denote the probabilities with a prime symbol (p'_L, p'^-_σ, p'^+_σ, p').

The probability of blocking is unchanged: $p'_L = p_L$. One advantage of considering an infinite number of iterations is that it removes loops so $p' = 0$. The other probabilities are multiplied by $\sum_{i \geq 0} p^i = 1/(1-p)$:

$$p'^-_\sigma = \frac{p_\sigma^-}{1-p}\,, \qquad p'^+_\sigma = \frac{p_\sigma^+}{1-p}\,.$$

12.6 Results

First, we place ourselves in the context of the [BIKE] cryptosystem. We explore two different approaches to decoding that can make use of this model.

The first one is to use exclusively the step-by-step decoder. The second one consists in performing a first pass with another algorithm, then decoding the residual errors with a step-by-step decoder.

In any case, it is desirable to know the starting states of the step-by-step decoder. That is to say, in the first case, it is necessary to know the joint distribution of the syndrome and the error weight of a uniformly random error pattern, and in the second case, it is necessary to know this distribution for the output of the first pass. Then, the DFR is calculated by weighting the probability that each starting state causes a failure by its occurrence frequency at the input of the step-by-step decoder.

For the first case, it was discussed in the previous chapter, but the second one is more complicated. The problem lies in the syndrome weight which is difficult to evaluate, for example for a parallel decoder as we saw in the previous chapter. We will therefore see the problem in a reciprocal way: we will determine the set of states in which it is necessary to arrive with the first pass to be certain (for example, with a probability of $1 - 2^{-128}$) that the step-by-step decoder will be successful. The issue is then shifted to finding a decoder that brings us to one of these states with a high probability.

12.6.1 Using the step-by-step decoder only

Starting distribution. We wish to have the joint distribution of (S_0, t_0). As we are in the context of the [BIKE] cryptosystem, the initial error weight is set to $t_0 = t$.

Since the error pattern is the result of a pseudorandom expansion of a seed, it can be seen as drawn uniformly at random in $\mathcal{E}_{n,t}$ (in the random oracle model). Therefore, Proposition 11.4 covers the joint distribution of (S_0, t_0). It accurately takes into account the fact that the parity check matrix is quasi-cyclic[2] as shown in the previous chapter.

Computation Results. In Figure 12.3 we show the DFR predicted by the model for three different values of ξ so that we have a lower and an upper bound of the simulation curve. The values of ξ are close as they are within 0.02 of each other.

12.6.2 Using the step-by-step decoder for residual error correction

In Figure 12.4, we give an example of the DFR predicted by this Markov model. All parameters are fixed $(r, d) = (12\,323, 71)$ but a wide range of values for the syndrome weight and the error weight is considered. What is represented are the contours of the DFR: for each Iso-λ line, the states inside the curve lead to a failure with a predicted probability less than $2^{-\lambda}$. As an indication, the syndrome weight for a uniformly random error model is also represented as an overlay with the mean value and the range of expected values with a probability greater than 2^{-128} for a given t. However, the output of a real decoder is not uniformly random at all, and the same data would usually be lower.

[2]Strictly speaking, it takes into account the mass equations that stem from the regularity of the code

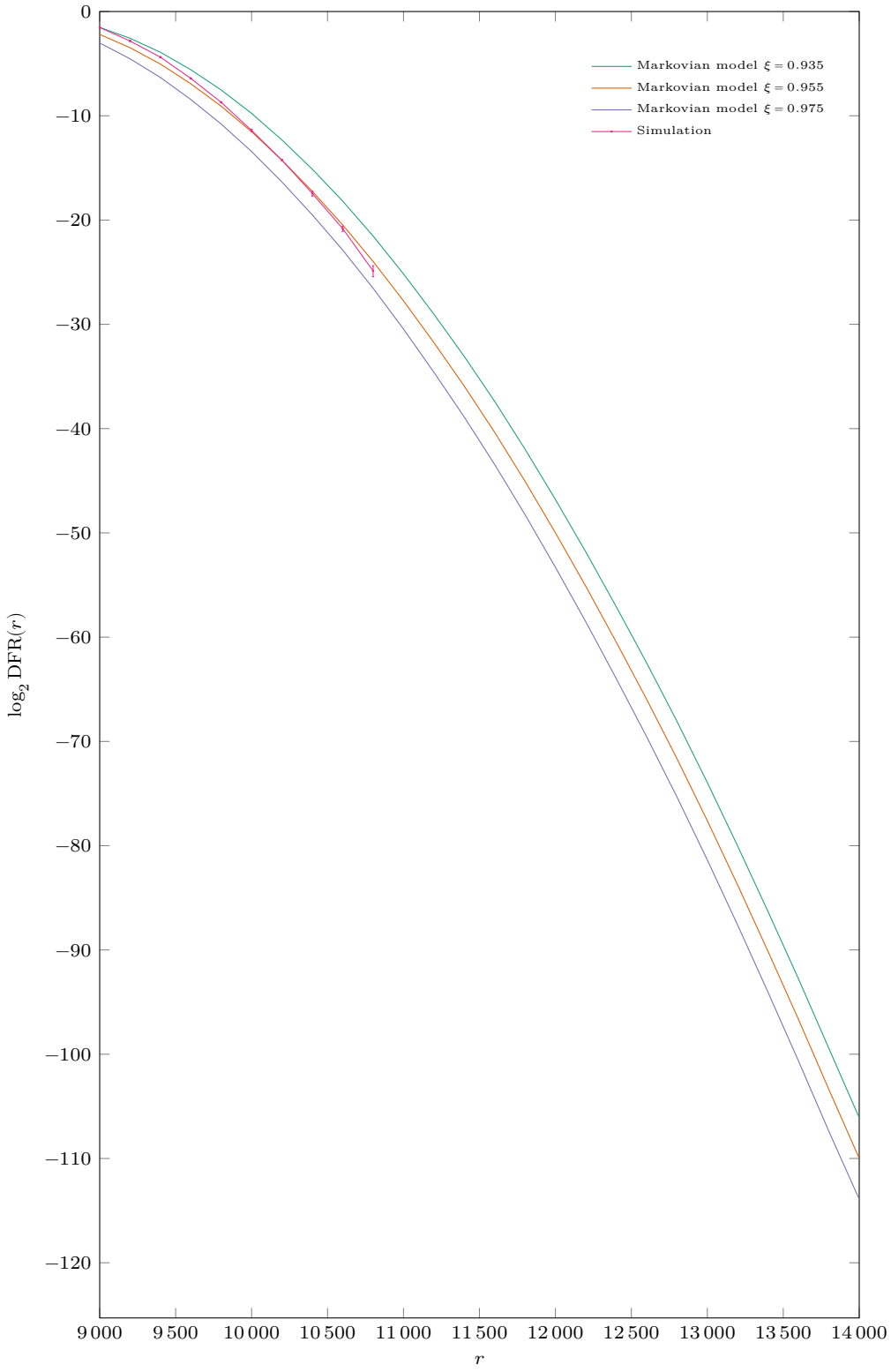


Figure 12.3: DFR *vs.* block size.

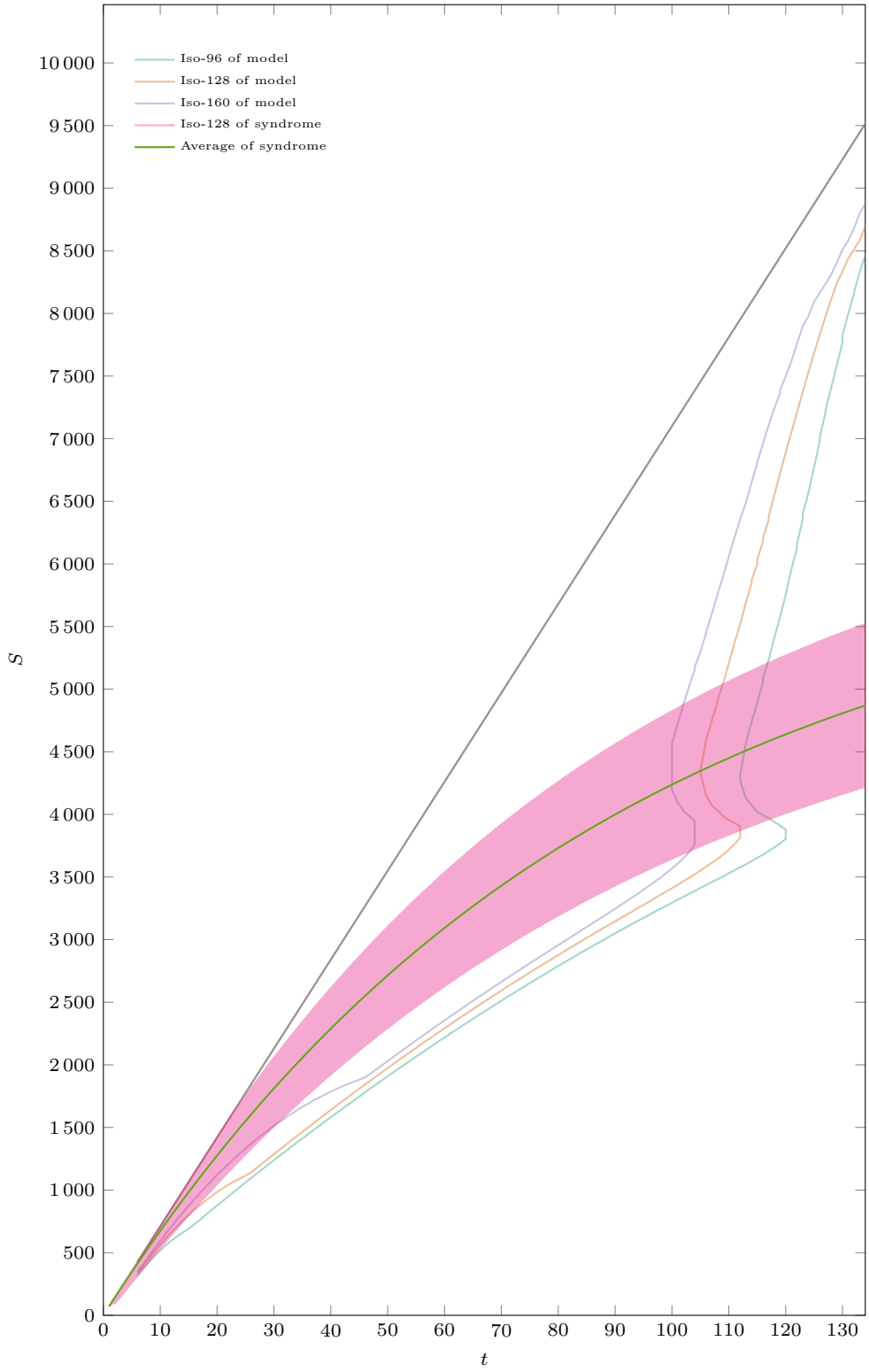

Iso-96 of model
Iso-128 of model
Iso-160 of model
Iso-128 of syndrome
Average of syndrome
S
t
0
500
1 000
1 500
2 000
2 500
3 000
3 500
4 000
4 500
5 000
5 500
6 000
6 500
7 000
7 500
8 000
8 500
9 000
9 500
10 000
0
10
20
30
40
50
60
70
80
90
100
110
120
130

Figure 12.4

Part IV

Practical DFR estimation

Summary of contributions

Based on the knowledge of the decoding acquired from models and simulations, we establish an assumption about the behaviour of the DFR curve.

- We apply it to establish a framework for extrapolating DFR from simulation data. It is necessary to use the inferred results rigorously, which we do by means of confidence intervals for which we derive formulas.
- We then make sure that the assumption is not violated by particular structures in the code. We measure the effect of parity check matrices with many column intersections on the decoder and compute precisely, proving new combinatorial properties of the spectrum, their density.
- Finally, we exhibit near-codewords, which are found in any QC-MDPC code. We see how they negatively influence decoding and estimate their impact on the average DFR.

Chapter 13

Introduction

LDPC codes share many properties with MDPC codes and have received a lot of attention in the last decades in the context of telecommunication. However, applications of coding theory to cryptography and to telecommunication have different requirements. The latter usually focuses on finding one *good* code that fits some channel and hardware constraints. For example IEEE 802.11 standard defines several common parity check matrices for its different standardized rates [802.11, §20.3.8]. With a code-based public key encryption scheme such as BIKE, each key generation involves choosing a new random code. The IND-CCA security of such a scheme requires that the family of the generated codes has a low average decoding failure rate (DFR).

13.1 State-of-the-art

13.1.1 Designing good LDPC code

Construction of LDPC codes usually relies on choosing a random parity check matrix of fixed degree then rejecting matrices with a certain structure. In [MN97] a construction avoids more than one overlap between columns, another avoids short cycles in the Tanner graph. Irregular codes construction first optimizes the density evolution as a function of the degree distribution polynomials $\lambda = \sum \lambda_i x^{i-1}$ and $\rho = \sum \rho_i x^{i-1}$ where λ_i and ρ_i are the fraction of edges incident to a variable (resp. check) node of degree i, see [RSU01]. A technique known as progressive edge growth guarantees a Tanner graph whose smallest short cycle (its length is called the *girth* of the graph) is greater than the method above [HEA01].

13.1.2 Error floors in LDPC codes

Estimation of the error floors for LDPC codes has been the focus of many research papers, most of them involve particularities of the channel, the structure of the code or a specific decoder. The usual suspects are subsets of the Tanner graph with nodes of unusual degrees: the *trapping sets* and the *stopping sets*. A (u, v) trapping set consists in u variable nodes inducing a graph where v check nodes have an odd degree. A stopping set is a subset of the variable nodes inducing a graph where no check node has degree 1.

One can define the decisional problem of determining if there is a stopping set of size less than t for a given linear code. And similarly we can define the problem of determining if there exists a trapping set of size less than t such that k check nodes have degree 1 in the induced subgraph for a given k and a linear code. A reduction of the former to the latter was established in [WKP09] and the former was proven to be NP-complete in [KS06]. The NP-completeness was proven for generic codes only, but it is pointed out in [WKP09] that it still holds if we add the constraint that the parity check matrix is sparse.

Error floors for LDPC codes over the binary erasure channel (BEC) were analysed in [Di+02; Ric03]. The analysis was extended to the additive white Gaussian noise channel (AWGN) and the binary symmetric channel (BSC) in [Ric03]. The method involves searching for candidate trapping sets in the Tanner graph then estimating their influence on the decoding failure rate.

13.1.3 DFR and spectrum of QC-MDPC codes

It was shown in [GJS16] that there is a correlation between the decoding failure probability and the resemblance between an error pattern and the secret parity check matrix.

Here the resemblance is defined via the *distance spectrum*. It is the set of all the distances between two ones in a vector, it is counted blockwise and circularly. For example, say $r = 11$ and $\mathrm{e} = (\mathrm{e}_0, \mathrm{e}_1) \in \left(\mathbb{F}_2^r\right)^2$ with $\mathrm{e}_0 = (0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0)$ then the distance spectrum for this block is the set $\{2, 4, 5\}$.

The key observation of the paper is that the decoder's probability of failure depends on the number of common distances between the spectrum of the key and the spectrum of the error pattern. Authors observed that, when a distance is in the spectrums of both of them, the decoder is slightly less likely to fail. They thus set up a reaction attack in which an attacker sends many ciphertexts to an oracle that returns whether the decryption (decoding) was successfull. As the attacker knows the plaintext associated to a ciphertext, they can extract its distance spectrum. With enough samples, each distance can be classified, and one can tell with great certitude whether it belongs to the parity check matrix spectrum or not. Authors then showed that the sparse parity check matrix (the private key) can be recovered with an algorithm akin to a depth-first search.

They used Algorithm $\mathcal{B}$ from [MOG15a] *i.e.* a parallel bit-flipping algorithm with a precomputed table of thresholds b_i for each iteration. But similar behaviour can be observed with any other algorithm.

We will see that the distance spectrum is a powerful tool that shows up in other security related aspects such as the weak keys for QC-MDPC.

13.1.4 Weak keys in a QC-LDPC cryptosystem

The second round version of LEDACrypt [LEDA] is a cryptosystem that uses QC-LDPC codes. It shares some similarities with BIKE in the fact that it uses a QC-MDPC code, but it has the additional propriety that the parity check matrix $\mathbf{H}$ can be factored as $\mathbf{H} = \mathbf{H}'\mathbf{Q}$ where $\mathbf{H}'$ and $\mathbf{Q}$ are sparse, quasi-cyclic matrices of dimension respectively $p \times (p \cdot n_0)$ and $(p \cdot n_0) \times (p \cdot n_0)$ for some parameters n_0 and p. In [APRS20], authors exhibited sets of weak keys. The product structure of the parity check matrix induces a strong bias in the bits positions of the low-weight codewords. This bias can be exploited to speed up the information set decoding algorithms[1] (ISD). While the typical ISD algorithm chooses a random permutation on the columns for each iteration, they first consider an information set consisting of consecutive positions in each block. They have shown that a single iteration of the Prange algorithm can break a significant proportion of the (thus weak) keys. From this fundamental idea, they have achieved improvements by exploiting some isomorphisms provided by the structure of the product of quasi-cyclic matrices. In the end, they showed that the cost of the attack divided by the density of these weak keys is much lower than the alleged security of the system.

The latest specification of LEDACrypt corrects that flaw by removing the product structure and thus using indeed a QC-MDPC code.

[1] See §3.3.

13.1.5 Weak keys in QC-MDPC cryptosystems

Taking QC-MDPC codes, some weak keys were studied in [BDLO16], their particular algebraic properties allow retrieval of the secret key by solving a *rational reconstruction problem*. The rational reconstruction problem consists, for two polynomials f and g in $\mathbb{K}[x]$ where $\mathbb{K}$ is a field and the polynomials are such that $0 < \deg(\mathrm{f}) < \deg(\mathrm{g})$, in finding, for a positive integer $r \leq \deg(\mathrm{g})$, two polynomials ϕ and ψ of $\mathbb{K}[x]$ such that

$$\begin{cases} \deg(\psi) < r\,, \\ \deg(\phi) < \mathrm{def}(\mathrm{g}) - r\,, \\ \gcd(\phi, \psi) = 1\,, \\ \frac{\psi}{\phi} = \mathrm{f} \mod \mathrm{g}\,. \end{cases}$$

A polynomial-time algorithm to solve this problem is the extended euclidean algorithm.

In [BDLO16], authors define a set of weak keys which are, for the case of BIKE, the subset of keys of the form $\mathrm{h} = (\mathrm{h}_0, \mathrm{h}_1)$ where $\deg(\mathrm{h}_0) + \deg(\mathrm{h}_1) < r$. Considering the quasi-cyclic structure of the code, they further extend this set by using isomorphisms.

In the end, the density of those keys is small, of order $2^{-\lambda}$ where λ the security parameters of the system (*e.g.* $\lambda = 128$), and do not represent a direct threat.

13.2 Contributions

Concerning the decoding, there are three qualities that are desirable for a QC-MDPC scheme: (i) a small block size, (ii) a low complexity and (iii) a low proven DFR. It is a rather challenging to have all three at the same time.

The main issue with proving a low DFR lies in the fact that the algorithms are iterative. The distributions of the relevant information to the algorithm can be computed for a uniformly random error pattern with a uniformly random parity check matrix. After one iteration the error pattern can no longer be considered uniformly random, some bits were flipped based on their counters which are information that are correlated to other bits. These correlations are hard to characterize and therefore the analysis for the first iteration cannot be iterated for the next ones.

In addition, the decoders that can be analysed often have rather poor performance, either because they are sequential or because they do not take advantage of the gain offered by being iterative.

The better performance achieved by the latest developments in decoding algorithms offers the promise of smaller parameter sizes, but they are even less amenable to analysis. Therefore, in this part we consider the use of simulation to estimate the DFR for these recent evolutions.

In the first chapter, we formulate an assumption, supported by the analytical results, that we use to extrapolate the DFR from simulations. Using values obtained by simulation requires great rigor with regard to margins of error. We thus cover the confidence intervals for the DFR obtained from the simulation data and for the extrapolated value.

Now, let us rewrite the definition of the decoding failure rate in the context of a Niederreiter cryptosystem with a sparse parity check matrix set $\mathcal{H}$ and the set of admissible error patterns $\mathcal{E}$. Let $\mathcal{W}$ be any subset of $\mathcal{H}$ and $\mathcal{D}$ be a decoder.

$$\begin{aligned} \mathrm{DFR}_{\mathcal{D}} &= \Pr\left[\mathcal{D}(\mathbf{H}, \mathbf{H}\mathrm{e}^\top) \neq \mathrm{e} \,\middle|\, \mathbf{H} \in \mathcal{H}, \mathrm{e} \in \mathcal{E}\right] \\ &= \frac{|\mathcal{W}|}{|\mathcal{H}|}\,\mathrm{DFR}_{\mathcal{D},\mathcal{W}} + \left(1 - \frac{|\mathcal{W}|}{|\mathcal{H}|}\right)\mathrm{DFR}_{\mathcal{D},\mathcal{H}\setminus\mathcal{W}} \end{aligned}$$

with

$$\mathrm{DFR}_{\mathcal{D},\mathcal{W}} = \Pr\left[\mathcal{D}(\mathbf{H}, \mathbf{H}\mathrm{e}^\top) \neq \mathrm{e} \,\middle|\, \mathbf{H} \in \mathcal{W}, \mathrm{e} \in \mathcal{E}\right]$$

and

$$\mathrm{DFR}_{\mathcal{D},\mathcal{H}\setminus\mathcal{W}} = \Pr\left[\mathcal{D}(\mathbf{H}, \mathbf{H}\mathrm{e}^\top) \neq \mathrm{e} \,\middle|\, \mathbf{H} \in \mathcal{H} \setminus \mathcal{W}, \mathrm{e} \in \mathcal{E}\right] .$$

In the context of decoding failures, a set of weak keys is a set $\mathcal{W}$ such that

$$\frac{|\mathcal{W}|}{|\mathcal{H}|}\,\mathrm{DFR}_{\mathcal{D},\mathcal{W}} > 2^{-\lambda} .$$

If this inequality is satisfied, but there is an imbalance between the factors so that $\frac{|\mathcal{W}|}{|\mathcal{H}|}$ is rather small but $\mathrm{DFR}_{\mathcal{D},\mathcal{W}}$ is rather large, then the estimates based on simulations could be skewed. In Chapter 15 we define and evaluate the influence of some subsets of keys on the DFR. We show a new property that links the spectrum of a key with the number of intersections between two columns in the parity check matrix. The weak keys that we define have an unusual spectrum (or equivalently an unusual number of columns intersections), we give combinatorial formulas to count them and evaluate their influence on the DFR with simulations.

Similarly, we can focus on the other side of the product $\mathbf{H}\mathrm{e}^\top$ and look at specific error patterns e. In this case, the phenomenon that we want to avoid is known as the error floor in coding theory.

For all sparse parity check matrix $\mathbf{H}$, we associate $\mathcal{E}_{\mathbf{H}}$ a subset of $\mathcal{E}$. Let $\mathcal{D}$ be a decoder.

$$\mathrm{DFR}_{\mathcal{D}} = \frac{1}{|\mathcal{H}|}\sum_{\mathbf{H}\in\mathcal{H}}\left(\frac{|\mathcal{E}_{\mathbf{H}}|}{|\mathcal{E}|}\,\mathrm{DFR}_{\mathcal{D},\mathbf{H}}^{\mathcal{E}_{\mathbf{H}}} + \left(1 - \frac{|\mathcal{E}_{\mathbf{H}}|}{|\mathcal{E}|}\right)\mathrm{DFR}_{\mathcal{D},\mathbf{H}}^{\mathcal{E}\setminus\mathcal{E}_{\mathbf{H}}}\right)$$

with

$$\mathrm{DFR}_{\mathcal{D},\mathbf{H}}^{\mathcal{E}_{\mathbf{H}}} = \Pr\left[\mathcal{D}(\mathbf{H}, \mathbf{H}\mathrm{e}^\top) \neq \mathrm{e} \,\middle|\, \mathrm{e} \in \mathcal{E}_{\mathbf{H}}\right]$$

and

$$\mathrm{DFR}_{\mathcal{D},\mathbf{H}}^{\mathcal{E}\setminus\mathcal{E}_{\mathbf{H}}} = \Pr\left[\mathcal{D}(\mathbf{H}, \mathbf{H}\mathrm{e}^\top) \neq \mathrm{e} \,\middle|\, \mathrm{e} \in \mathcal{E} \setminus \mathcal{E}_{\mathbf{H}}\right] .$$

If for any sparse parity check matrix $\mathbf{H}$ we can find a set of error patterns $\mathcal{E}_{\mathbf{H}}$ that hinder the decoding performance, it is problematic if

$$\frac{1}{|\mathcal{H}|}\sum_{\mathbf{H}\in\mathcal{H}}\frac{|\mathcal{E}_{\mathbf{H}}|}{|\mathcal{E}|}\,\mathrm{DFR}_{\mathcal{D},\mathbf{H}}^{\mathcal{E}_{\mathbf{H}}} = \mathrm{E}_{\mathbf{H}\in\mathcal{H}}\left[\frac{|\mathcal{E}_{\mathbf{H}}|}{|\mathcal{E}|}\,\mathrm{DFR}_{\mathcal{D},\mathbf{H}}^{\mathcal{E}_{\mathbf{H}}}\right] > 2^{-\lambda} .$$

In Chapter 16, we will see that the quasi-cyclic structure together with the sparseness of the parity check matrix of a QC-MDPC imply the existence of several error patterns called "near-codewords". When used to decode a pattern close to one of these near-codewords (using the Hamming distance), a decoder is more likely to fail. Once again, similar to the weak keys situation, it is a matter of counting the problematic patterns and evaluating their influence on decoding by means of simulations.

Chapter 14

A DFR extrapolation framework

Several works and simulation show that, excluding a phenomenon such as the error floor discussed in Chapter 16, the function $r \mapsto \log \mathrm{DFR}_{\mathcal{D}}(r)$ is always decreasing and, at worst, it is an affine function. Similar to how security reduction often relies on the complexity of the best known algorithm to solve a particular problem (*e.g.* information set decoding for most code-based cryptographic schemes), we make a decoding assumption that we can use to estimate the DFR of a particular decoding algorithm.

14.1 Notations

We adopt the following notations in this chapter.

- The decoding failure rate for a QC-MDPC, with a set of keys $\mathcal{H}$ and a set of messages $\mathcal{E}$ for a decoder $\mathcal{D}$ is written as

$$\mathrm{DFR}^{\mathcal{E}}_{\mathcal{D},\mathcal{H}} = \Pr\left[\mathcal{D}(\mathbf{H}, \mathbf{H}\mathrm{e}^\top) \neq \mathrm{e} \,\middle|\, \mathbf{H} \in \mathcal{H}, \mathrm{e} \in \mathcal{E}\right] .$$

- For ease of reading, when omitted, the set $\mathcal{E}$ is by default the whole set of admissible messages

$$\mathcal{E} = \mathcal{E}_{n,t} = \{\mathrm{e} \in \{0,1\}^n \mid |\mathrm{e}| = t\} .$$

- When omitted, the set $\mathcal{H}$ is by default the whole set of admissible keys

$$\begin{aligned}\mathcal{H} &= \mathcal{H}_{d,w,r\times n} \\ &= \left\{\mathbf{H} \in \mathbb{F}_2^{r\times n} \,\middle|\, \forall i \in \{0,\dots,r-1\}, \left|\mathbf{h}_i^\top\right| = w, \forall j \in \{0,\dots,n-1\}, \left|\mathbf{h}_j\right| = d\right\} ,\end{aligned}$$

- The column weight d, the row weight w, and the error weight t will be obvious from the context and we will write

$$\mathrm{DFR}(r) = \mathrm{DFR}^{\mathcal{E}_{2r,t}}_{\mathcal{D},\mathcal{H}_{d,2d,r\times 2r}} .$$

14.2 The decoder security assumption

We defined a Markovian model for a simple variant of bit-flipping, the step-by-step decoder[1]. This decoder corrects fewer errors than other bit-flipping variants, however it uses the same ingredients: computing counters and flipping the corresponding positions if they are

[1] The algorithm and the analysis are discussed in Chapters 7 & 12.

above some threshold. We fix the values of the parameters d and t, then a model can be generated and a DFR computed for arbitrary large values of the block size r. We observe that in the range of interest for r, the function $r \mapsto \log \mathrm{DFR}(r)$ is first strictly concave and eventually decreases linearly with r.

[Til18a] also explores the asymptotic behaviour of QC-MDPC decoding. The asymptotic formula it provides for the DFR cannot be used directly because the setting is different (d and t vary with r), and also the conditions under which it can be proven are not relevant for decoders and parameters of practical interest. However, the indication provided by the formula is consistent with the previous remark, the dominant term in the expression decreases linearly with r.

Experimentally, when plotting the DFR (obtained by simulation) *vs.* the block size r in a logarithmic scale, one observes a concave curve. This was observed for all the variants of bit-flipping decoding.

As there is enough evidence from the state of the art to back this claim, we suggest formulating a decoding assumption on which we will base our estimation of the DFR. We denote $\mathcal{D}$ a decoder: it is a family of decoding algorithms that can be applied to any QC-MDPC codes with fixed (d, t) and variable r.

Assumption 3. *For a given decoder $\mathcal{D}$, and a given security level λ, the function $r \mapsto \log \mathrm{DFR}_{\mathcal{D}}(r)$ is concave if $\mathrm{DFR}_{\mathcal{D}}(r) \geq 2^{-\lambda}$.*

Under this assumption, a conservative estimation on the DFR can be made from the measure of the DFR for two different block sizes r_1 and r_2. Indeed, let $r_1 < r_2 < r_3$, then by definition of the concavity of a function we have

$$\log \mathrm{DFR}_{\mathcal{D}}(r_2) \geq \frac{r_3 - r_2}{r_3 - r_1} \log \mathrm{DFR}_{\mathcal{D}}(r_1) + \frac{r_2 - r_1}{r_3 - r_1} \log \mathrm{DFR}_{\mathcal{D}}(r_3)$$

as $\frac{r_3 - r_2}{r_3 - r_1}, \frac{r_2 - r_1}{r_3 - r_1} \in [0, 1]$ and $\frac{r_3 - r_2}{r_3 - r_1} r_1 + \frac{r_2 - r_1}{r_3 - r_1} r_3 = r_2$.

This inequality can be rewritten as in the following proposition.

Proposition 14.1. *Under Assumption 3, an upper bound of the DFR for a block size r_3 can be extrapolated from the DFR for block sizes r_1 and r_2, with $r_1 < r_2 < r_3$ using the following inequality:*

$$\log \mathrm{DFR}_{\mathcal{D}}(r_3) \leq \log \mathrm{DFR}_{\mathcal{D}}(r_1) + \frac{\log \mathrm{DFR}_{\mathcal{D}}(r_2) - \log \mathrm{DFR}_{\mathcal{D}}(r_1)}{r_2 - r_1}(r_3 - r_1)\,.$$

Another way to say it is that the secant line crossing the graph of $r \mapsto \log \mathrm{DFR}_{\mathcal{D}}(r)$ at r_1 and r_2 upper bounds $\log \mathrm{DFR}_{\mathcal{D}}(r_3)$ for any $r_3 > r_2$. It is an equality when $r \mapsto \log \mathrm{DFR}_{\mathcal{D}}(r)$ is affine, and this was the observed asymptotic behaviour in the Markovian model. However, in the range of values for r usually used in simulation, the function has a faster decrease and r_1 and r_2 should be as high as the simulation and the desired precision allow.

For instance in Figure 14.1, suppose the low curve (blue) is giving the $\log_2(\mathrm{DFR})$ and we are able to make accurate simulation as long as the DFR is above 2^{-25} (black dots). Taking the tangent at the last point gives us the red line from which we derive an upper bound r' for a block size with a DFR below 2^{-128} as well as an upper bound 2^{-s} for the DFR for a given block size r.

14.3 Security of the system with respect to the block size

The security of a QC-MDPC based cryptosystem relates to the generic problem of decoding linear codes. Remember, from §3.3, the result that is implied by Proposition 3.5 and [Sen11], stating that the security of the system depends on

$$\frac{\mathcal{WF}_{\mathcal{A}}(n, k, w)}{n - k} = 2^{c[1 + w(1 + o(1))] - \log_2(n)} \quad \text{and} \quad \frac{\mathcal{WF}_{\mathcal{A}}(n, k, t)}{\sqrt{n - k}} = 2^{c[1/2 + t(1 + o(1))] - \log_2(n)/2}$$

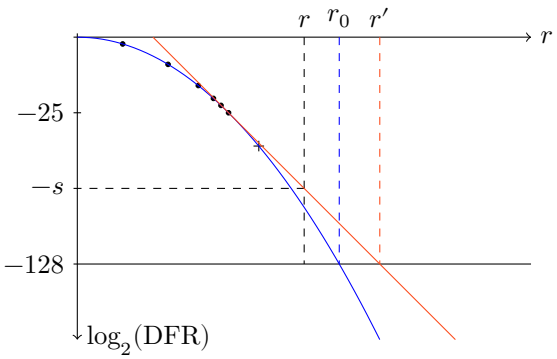


Figure 14.1: DFR tangent extrapolation

with $c = -\log_2(1-k/n)$, for any algorithm $\mathcal{A}$ among the variants of [Pra62; Ste88; Dum91; BJMM12; MMT11; MO15].

With BIKE parameters, n does not exceed 2^{16} and the code rate is always $R = k/n = 1/2$ so $c = 1$. Consequently, the methodology we develop in this chapter to decrease the DFR for an algorithm by increasing the code length has very little impact on this aspect of the security of the system.

14.4 Confidence interval

The failure rate $\mathrm{DFR}^{\mathcal{E}}_{\mathcal{D},\mathcal{H}}(r)$ is the probability of the Bernoulli trial written in Algorithm 14.1 returning False. In this section we will denote this DFR as p for conciseness.

Algorithm 14.1: Bernoulli trial for decoding failure.

input : Block size r, message space $\mathcal{E}$, key space $\mathcal{H}$, decoder $\mathcal{D}$.
$\mathrm{e} \xleftarrow{\$} \mathcal{E}$;
$\mathbf{H} \xleftarrow{\$} \mathcal{H}$;
$\mathrm{s} \leftarrow \mathbf{H}\mathrm{e}^\top$;
$\mathrm{e}' \leftarrow \mathcal{D}(\mathbf{H}, \mathbf{H}, \mathrm{s})$;
return $[\mathrm{e}' = \mathrm{e}]$;

Repeating this procedure N times and counting the number of failures F consitutes a random variable following a binomial distribution $\mathrm{Bin}(N, p)$. The ratio F/N gives an estimate of the actual probability p and, using a normal approximation, the standard deviation of this estimate is proportional to

$$\frac{\sqrt{F(N-F)}}{N\sqrt{N}}\,.$$

To be more precise, let us recall the definition of a confidence interval.

Definition 14.2. Let $\alpha \in (0,1)$. Suppose we have an observable vector $\mathrm{x} = (x_1, \ldots, x_n)$ from a distribution that depends on an unknown parameter $\theta \in \Theta$. A *confidence interval* with a *confidence level* $(1-\alpha)$ for θ is obtained from a function $\mathcal{I}$ that satisfies:

$$\forall \theta \in \Theta, \quad \Pr\left[\theta \in \mathcal{I}(\mathrm{x}, \alpha) \,\middle|\, \theta\right] \geq 1 - \alpha\,.$$

The actual value of the left-hand side of this inequality is called the *coverage probability*. So the confidence level is the infimum of the coverage probabilities for all $\theta \in \Theta$.

The aforementioned normal approximation is a common way to roughly estimate a confidence interval. Some more advanced techniques were designed so that the coverage probability is closer to the desired $(1-\alpha)$.

14.5 A first estimation

Here we will recall the Clopper-Pearson interval also known as the exact method as it gives a coverage probability that is always above $(1-\alpha)$. It is a conservative method for computing a confidence interval.

14.5.1 Clopper-Pearson interval

First, we need to set the framework for our problem. We repeat N times a Bernoulli process with probability of failure θ. The observed number of failures is a random variable that we denote $X \in \{0, \dots, N\}$. X follows a binomial distribution $\mathrm{Bin}(N, \theta)$:

$$\Pr[X = x] = \begin{cases} \binom{N}{x}\theta^x(1-\theta)^{N-x} & \text{if } x \in \{0, \dots, N\}; \\ 0 & \text{otherwise.} \end{cases}$$

The probability θ is the unknown parameter that we want to estimate.

This method uses the cumulative probabilities of the binomial distribution $\Pr[X \geq x]$ and $\Pr[X \leq x]$ for some number of failures x and some undeternmined probability θ:

$$\Pr[X \geq x] = \sum_{k=x}^{N} \binom{N}{k}\theta^k(1-\theta)^{N-k} \qquad \Pr[X \leq x] = \sum_{k=0}^{x} \binom{N}{k}\theta^k(1-\theta)^{N-k}\,.$$

The confidence interval is a range of values for θ such that for a number of failures F observed among N samples is not abnormal *i.e.* for any probability θ in the interval, the observation happens with probability at least $(1-\alpha)$.

Given the monotonicity of the cumulative probabilities with respect to θ, the confidence interval can be written as $(\theta_-; \theta_+)$ with

$$\theta_- = \inf_\theta \left\{ \theta \,\middle|\, \Pr[X \geq F] > \frac{\alpha}{2} \right\}, \qquad \theta_+ = \sup_\theta \left\{ \theta \,\middle|\, \Pr[X \leq F] > \frac{\alpha}{2} \right\}.$$

Another common way to write it is:

Proposition 14.3 (Clopper-Pearson [CP34]). *A $(1-\alpha)$-confidence interval for a failure probability p when F failures have been observed out of N experiments is (θ_-, θ_+) where*

$$\theta_- = B^{-1}(\alpha/2, F, N-F+1) \qquad \theta_+ = B^{-1}(1-\alpha/2, F+1, N-F)\,.$$

With B^{-1} the inverse of $x \mapsto B(x, a, b)$, the incomplete beta function:

$$B(x, a, b) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} \int_0^x t^{a-1}(1-t)^{b-1}\,dt\,.$$

14.5.2 A first estimation of confidence intervals for extrapolations

Now let us come back to our extrapolation problem. First, say we know the true probabilities $p_1 = \mathrm{DFR}(r_1)$ and $p_2 = \mathrm{DFR}(r_2)$. If we suppose that $r \mapsto \log \mathrm{DFR}_{\mathcal{D}}(r)$ is an affine function (the extreme case of Assumption 3) then the failure probability $p_3 = \mathrm{DFR}(r_3)$ for a block size $r_3 > r_2 > r_1$ is

$$p_3 = p_1^{-A} p_2^{1+A} \tag{14.1}$$

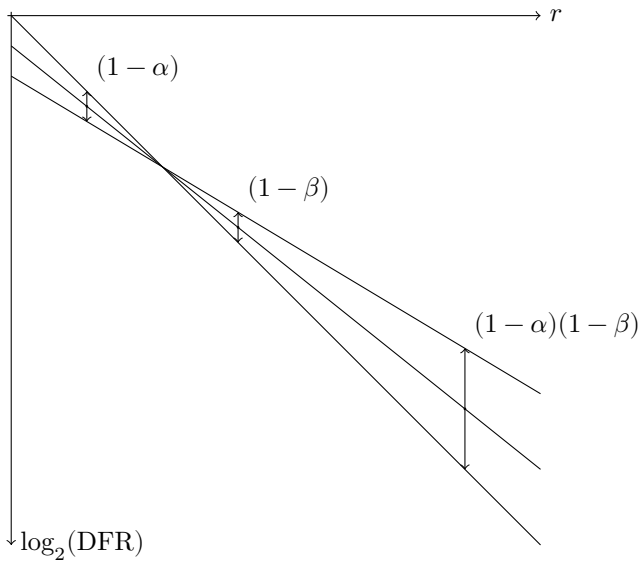


Figure 14.2: Rough estimation of the confidence interval.

with $A = \frac{r_3 - r_2}{r_2 - r_1} > 0$.

In reality, for a block size r_i, we only know that F_i failures were observed out of N_i samples with $i = 1, 2$. If p_1 has a $(1-\alpha)$-confidence interval $(p_1^-; p_1^+)$ and p_2 has a $(1-\beta)$-confidence interval $(p_2^-; p_2^+)$, then p_3 has a confidence interval $(p_3^-; p_3^+)$

$$p_3^- = (p_1^+)^{-A}(p_2^-)^{1+A}; \qquad p_3^+ = (p_1^-)^{-A}(p_2^+)^{1+A}$$

with a coverage probability at least $(1-\alpha)(1-\beta)$. This is illustrated in Figure 14.2.

In practice, for a given confidence level, one can find a narrower confidence interval than this one.

14.6 Using posterior probability

In this section we detail how to compute narrower confidence interval using posterior probabilities.

We see the failure probability of a Bernoulli trial as a random variable $\Theta_i \in [0, 1]$ for i= 1, 2. Under the condition that $\Theta_i = \theta_i$, X_i follows a binomial distribution for $i = 1, 2$:

$$X_i \,|\, \Theta_i = \theta_i \sim \mathrm{Bin}(N_i, \theta_i)\,.$$

We define the random variable $\Theta_3 \in [0, 1]$ as

$$\Theta_3 = \Theta_1^{-A}\Theta_2^{1+A}$$

for some constant $A > 0$.

Our goal is, with no prior knowledge on Θ_3, to find its distribution if we observe $X_i = F_i$ for $i = 1, 2$.

Let us first recall two properties about random variables (see [Spr79] and [BT02] for proofs). In this section, we write f_X the probability density functions of any random variable X.

Proposition 14.4. *Let X and Y be two independent positive random variables whose probability density functions are respectively f_X and f_Y then the ratio $Z = X/Y$ is a random variable with the following probability density function:*

$$f_{X/Y}(z) = \int_0^\infty f_Y(y) f_X(zy) y \, dy \, .$$

Proposition 14.5. *Let X be a real valued random variable and let $g : \mathbb{R} \to \mathbb{R}$ be an invertible monotonic differentiable measurable function whose inverse is h. Let Y be the random variable defined by $Y = g(X)$. We write f_X and f_Y the probability density functions of respectively X and Y. Then*

$$f_Y(y) = |h'(y)| f_X(h(y)) \, .$$

Corollary 14.6. *For any random variable X and any real number $A \neq 0$*

$$f_{X^A}(x) = \left| \frac{1}{A} \right| x^{1/A-1} f_X(x^{1/A}) \, .$$

With no prior knowledge on the failure probability distributions we have the following proposition.

Proposition 14.7. *For $i = 1, 2$, taking the uniform prior $f_{\Theta_i}(\theta_i) = 1$, under the observation $X_i = x_i$, Θ_i follows a beta distribution $\mathrm{B}(x_i + 1, N_i - x_i + 1)$:*

$$f_{\Theta_i \,|\, X_i = x_i}(\theta) = \frac{1}{B(x_i + 1, N_i - x_i + 1)} \theta^{x_i} (1 - \theta)^{N_i - x_i}$$

where $B(a, b) = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}$.

Proof. Bayes' theorem states that:

$$f_{\Theta_i \,|\, X_i = x_i}(\theta) = \frac{\Pr(X_i = x_i \,|\, \Theta_i) f_{\Theta_i}(\theta)}{\int_0^1 \Pr(X_i = x_i \,|\, \Theta_i) f_{\Theta_i}(\theta) d\theta} \, .$$

□

Applying Corollary 14.6, we can determine the distributions of Θ_1^A and Θ_2^{1+A}. Then, applying Proposition 14.4

$$f_{\Theta_3 \,|\, X_1 = x_1, X_2 = x_2}(z) = \frac{z^{1/(1+A)-1}}{A(1+A)} \int_0^1 f_{\Theta_1 \,|\, X_1 = x_1}(x^{1/A}) f_{\Theta_2 \,|\, X_2 = x_2}\left((zx)^{1/(1+A)}\right) x^{1/A+1/(1+A)-1} \, dx \, .$$

Finally, since the integrand is non-negligible only in a narrow range of values and since we are interested in the logarithm of the DFR, we can apply a change of variable and Proposition 14.5 to obtain the following proposition.

Proposition 14.8. *Let $\Theta_i \in [0, 1]$ be a real-valued random variable for $i = 1, 2$. Let X_i be random variables such that under the condition that $\Theta_i = \theta_i$, X_i follows a binomial distribution for $i = 1, 2$:*

$$X_i \,|\, \Theta_i = \theta_i \sim \mathrm{Bin}(N_i, \theta_i) \, .$$

Let $\Theta_3 \in [0, 1]$ be the random variable defined as

$$\Theta_3 = \Theta_1^{-A} \Theta_2^{1+A}$$

for some constant $A > 0$.

$$f_{\log(\Theta_3)\,|\,X_1=x_1,X_2=x_2}(s) = \frac{e^{s(x_2+1)/(1+A)}}{K} \int_{-\infty}^{0} e^{t(\frac{x_1+1}{A}+\frac{x_2+1}{1+A})} (1-e^{\frac{t}{A}})^{N_1-x_1}(1-e^{\frac{s+t}{1+A}})^{N_2-x_2}\,dt\,.$$

with

$$K = A(1+A)B(x_1+1, N_1-x_1+1)B(x_2+1, N_2-x_2+1)\,.$$

We can now define the $(1-\alpha)$ confidence interval $(\log\theta_3^-; \log\theta_3^+)$ on the extrapolated value $\log\theta_3$ where $\log\theta_3^-$ is such that

$$\int_{-\infty}^{\log\theta_3^-} f_{\log(\theta_3)}(s)\,ds = \alpha/2$$

and $\log\theta_3^+$ is such that

$$\int_{\log\theta_3^+}^{0} f_{\log(\theta_3)}(s)\,ds = \alpha/2\,.$$

Proposition 14.9. *Let p_1 and p_2 be the failure probabilities of two independent Bernoulli trials. Suppose, for $i = 1, 2$, that we observe F_i failures out of N_i samples of each Bernoulli trial, and suppose that*

$$p_3 = p_1^{-A} p_2^{1+A}\,.$$

Then a $(1-\alpha)$-confidence interval for p_3 is $(\exp(\ell p_3^-), \exp(\ell p_3^+))$ where ℓp_3^- is the greatest value such that

$$\frac{1}{K}\int_{-\infty}^{\ell p_3^-}\int_{-\infty}^{0} e^{s\left(\frac{F_2+1}{1+A}\right)} e^{t\left(\frac{F_1+1}{A}+\frac{F_2+1}{1+A}\right)} (1-e^{\frac{t}{A}})^{N_1-F_1}(1-e^{\frac{s+t}{1+A}})^{N_2-F_2}\,dt\,ds < \frac{\alpha}{2}$$

and ℓp_3^+ is the smallest value such that

$$\frac{1}{K}\int_{\ell p_3^+}^{\infty}\int_{-\infty}^{0} e^{s\left(\frac{F_2+1}{1+A}\right)} e^{t\left(\frac{F_1+1}{A}+\frac{F_2+1}{1+A}\right)} (1-e^{\frac{t}{A}})^{N_1-F_1}(1-e^{\frac{s+t}{1+A}})^{N_2-F_2}\,dt\,ds < \frac{\alpha}{2}$$

where

$$K = A(1+A)B(F_1+1, N_1-F_1+1)B(F_2+1, N_2-F_2+1)\,.$$

Example 14.10. In Table 14.1 we compare the confidence intervals $(\mathrm{DFR}^-, \mathrm{DFR}^+)$ for extrapolated DFR at $r_3 = 12\,323$ for two simulations data:

- BGF (9 iterations): for $r_1 = 10\,037$, $F_1 = 66\,391$, $N_1 = 3\,747\,161\,784$, $r_2 = 10\,253$, $F_2 = 5$, $N_2 = 1\,445\,221\,866$,
- Backflip 7 iterations : $r_1 = 10\,181$, $F_1 = 394$, $N_1 = 14\,576\,092\,619$, $r_2 = 10\,253$, $F_1 = 111$, $N_1 = 34\,283\,154\,045$.

We can see a difference of a few bits between confidence intervals from the simple method and Proposition 14.9.

14.7 Choosing parameters

As discussed in §4.1, the security of QC-MDPC system depends mainly on the column weight d and the error weight t, achieving a low DFR is a matter of choosing the right block size r.

Table 14.1: Confidence intervals for some decoding simulation data.

		Simple method		Proposition 14.9	
Algorithm	log(DFR)	log(DFR^-)	log(DFR^+)	log(DFR^-)	log(DFR^+)
BGF 9 iterations	-146.20	-172.30	-129.11	-164.21	-130.31
Backflip 7 iterations	-116.22	-134.13	-99.00	-128.13	-104.57

To achieve IND-CPA security, the parameter t and d should be chosen to match a certain security level λ. To have IND-CCA security, we also need to make sure the DFR is less than $2^{-\lambda}$. In order to find the smallest r block size having this property, we will run decoding simulations.

Under Assumption 3, the DFR can be estimated from two sets of simulations. The only varying parameter between those two sets is the block size r. Another thing to consider is how we balance the computational effort between these two sets. We assume that the block size has a negligible influence on the effort necessary to run one simulation, and we cap the total number of simulations by a value N. In total, there are four parameters to be determined: r_1, r_2, N_1, N_2 ($N = N_1 + N_2$).

We then run N_i simulations with block size r_i and count the number of decoding failures F_i for $i = 1, 2$. Using Proposition 14.9 we can then choose the smallest r such that an upper bound on the DFR for this r is below $2^{-\lambda}$.

However, it is difficult to choose relevant values for r_1, r_2, N_1, N_2 using an analytical formula. Such a formula would also require knowing $r \mapsto \text{DFR}_{\mathcal{D}}(r)$ but we only assume that it is concave.

To overcome this issue, we can rely on a heuristic. First we can compute a rough estimate of $r \mapsto \text{DFR}_{\mathcal{D}}(r)$ with a polynomial interpolation and use it in a nonlinear optimization to determine the values of r_1, r_2, N_1 and N_2 that minimize the upper bound given in Proposition 14.9. In practice, the nonlinear optimization method described in [NM65] converges quickly.

The complete heuristic is detailed in Algorithm 14.2.

Algorithm 14.2: Heuristic for finding r_1, r_2, N_1 and N_2.

input : Initial error weight t, column weight d, decoder $\mathcal{D}$, total number of tests $N_{\max}$, precision of measures ϵ.

$N_{\text{total}} \leftarrow 0$;
$i \leftarrow 0$;
$\Delta \leftarrow w^2/200$;
$S \leftarrow \{\}$;
repeat
  $r \leftarrow w^2/2 + i \times \Delta$;
  $F \leftarrow 0$;
  $N \leftarrow 0$;
  repeat
    if `Bernoulli`$(r, \mathcal{D}) =$ False **then**
      $F \leftarrow F + 1$
    $N \leftarrow N + 1$;
    $N_{\text{total}} \leftarrow N_{\text{total}} + 1$;
    $(p_-, p_+) \leftarrow CI_{\text{binomial}}(F, N)$;
    if $\log p_+ - \log p_- < \epsilon$ **then**
      exit loop;
  until $N_{\text{total}} \geq N_{\max}$;
  $S \leftarrow S \cup \{(r, F/N)\}$;
  $i \leftarrow i + 1$;
until $N_{\text{total}} \geq N_{\max}$;
$p_{\text{quad}} \leftarrow$ Quadratic interpolation of S;
$(r_1, r_2, N_1) \leftarrow \underset{r_1, r_2, N_1}{\arg\min}\, CI^+_{\text{extra}}(N_1 p_{\text{quad}}(r_1), N_1, (N - N_1)p_{\text{quad}}(r_2), N - N_1))$;
`/*` CI^+_{extra} `is the upper bound of the confidence interval given in Proposition 14.9.`
`In practice the argmin is obtained with a nonlinear optimization method.` `*/`
return $(r_1, r_2, N_1, N - N_1)$;

Chapter 15

Weak keys: Subsets of parity check matrices

Short cycles in the Tanner graph and high number of intersections between columns in the parity check matrix are detrimental for the decoding capabilities of the decoder.

In this chapter, we start from the definition of the distance spectrum due to [GJS16]. We show new properties of this spectrum that can be used to enumerate them. We make an observation on the counters distribution for parity check matrices with unusual spectrum and see how that can be detrimental to the decoding performance. Then, we construct three sets of weak keys for a QC-MDPC system, these keys have an unusual spectrum and thus a DFR higher than average. We eventually assess their impact on the security of the system by estimating their DFR with simulation and using upper bounds on their cardinality.

15.1 QC-MDPC Codes

15.1.1 Definition and polynomial representation

Let $\mathbf{H} = (\mathbf{H}_0, \mathbf{H}_1)$ be the parity check matrix of a QC-MDPC code. Equivalently, the parity check matrix can be written as a tuple of polynomials $(\mathrm{h}_0, \mathrm{h}_1) \in \left(\mathbb{F}_2[x]/(x^r-1)\right)^2$, using the following isomorphism.

Proposition 15.1 (Recall)**.** *The application*

$$\mathbf{H} \mapsto h_{0,0} + h_{1,0}x + \cdots + h_{r-2,0}x^{r-2} + h_{r-1,0}x^{r-1}$$

is an isomorphism between the ring of circulant $r \times r$ matrices with coefficients in $\mathbb{F}_2$ and the quotient ring $\mathbb{F}_2[x]/(x^r-1)$.

If we denote h_i the polynomial corresponding to first column of the block $\mathbf{H}_i$ for $i = 0, 1$, and if we write the row vector $(\mathrm{e}_0, \mathrm{e}_1)$ as polynomials then the product

$$(\mathbf{H}_0, \mathbf{H}_1)(\mathrm{e}_0, \mathrm{e}_1)^\top$$

is the column vector represented by the polynomial $\mathrm{e}_0\mathrm{h}_0 + \mathrm{e}_1\mathrm{h}_1$.

The following isomorphism is important in our analysis as it allows us to reduce properties on any distance δ to the case $\delta = 1$.

Proposition 15.2. *For all $\delta \in \mathbb{Z}_r^\times$, the endomorphism ϕ_δ of $(\mathbb{F}_2[x]/(x^r-1), +, \times)$ induced by*

$$x \mapsto x^\delta$$

is an isomorphism and an isometry for the Hamming distance.

Remark 15.3. The inverse of ϕ_δ is $\phi_{\delta^{-1}}$.

Remark 15.4. Previous works on weak keys for QC-LDPC or QC-MDPC codes also mention this isometry: [APRS20, §4.3] & [BDLO16, §6]. Together with the circular shifts (multiplications by x^i for all $i \in \{0, \dots, r-1\}$), it allows a quadratic gain on the size of a set of keys while holding similar characteristics.

15.1.2 Decoding

To understand the rationale behind our constructions of weak keys, let us first adopt an intuitive point of view of the bit-flipping decoder. Remember Algorithm 6.2, the bit-flipping algorithm. The counter $\left|\mathbf{h}_j \star \mathrm{s}\right|$ of a position j is the number of parity check equations (rows of $\mathbf{H}$) involving that position and which are unsatisfied. If the number of unsatisfied parity check equations is high, the coordinate on that position is likely to be erroneous.

Finding e from the syndrome $\mathrm{s} = \mathbf{H}\mathrm{e}^\top$ and the sparse parity check matrix $\mathbf{H}$ such that $|\mathrm{e}| = t$ is possible by exploiting the bias in the counters. Figure 5.1 gives the number of positions with given counter values, the smaller Gaussian shape curve on the right stands for the erroneous positions. The decoder knows the counters but not the errors' location (*i.e.* it knows only the sum of the two curves). So Algorithm 6.2 chooses a sensible threshold T and flips all positions with a counter above it, the syndrome is recomputed, then the counters again. This process is repeated and the error weight usually decreases after each iteration until all errors have been removed and the syndrome is zero.

Remark 15.5. When the block size decreases or the error weight increases, this usually results in a decrease of the error counters. When this happens, the small Gaussian curve in Figure 5.1 will move to the left and error detection will become more difficult—even if the threshold is adjusted—because they will be overwhelmed by the sheer mass of correct positions. Any effect that moves the small curve to the left or the large curve to the right therefore has a negative effect on the decoder behaviour. As we will see later, this is precisely what happens with weak keys.

15.2 Notations

We adopt the following notations in this chapter.

- The decoding failure rate for a QC-MDPC, with a set of keys $\mathcal{H}$ and a set of messages $\mathcal{E}$ for a decoder $\mathcal{D}$ is written as
$$\mathrm{DFR}^{\mathcal{E}}_{\mathcal{D},\mathcal{H}} = \Pr\left[\mathcal{D}(\mathbf{H}, \mathbf{H}\mathrm{e}^\top) \neq \mathrm{e} \,\middle|\, \mathbf{H} \in \mathcal{H}, \mathrm{e} \in \mathcal{E}\right] .$$
- For ease of reading, when omitted, the set $\mathcal{E}$ is by default the whole set of admissible messages
$$\mathcal{E} = \mathcal{E}_{n,t} = \{\mathrm{e} \in \{0,1\}^n \mid |\mathrm{e}| = t\} .$$
- When omitted, the set $\mathcal{H}$ is by default the whole set of admissible keys
$$\begin{aligned}\mathcal{H} &= \mathcal{H}_{d,w,r\times n} \\ &= \left\{\mathbf{H} \in \mathbb{F}_2^{r\times n} \,\middle|\, \forall i \in \{0,\dots,r-1\}, \left|\mathbf{h}_i^\top\right| = w, \forall j \in \{0,\dots,n-1\}, \left|\mathbf{h}_j\right| = d\right\} ,\end{aligned}$$
- The column weight d, the row weight w, and the error weight t will be obvious from the context and we will write
$$\mathrm{DFR}(r) = \mathrm{DFR}^{\mathcal{E}_{2r,t}}_{\mathcal{D},\mathcal{H}_{d,2d,r\times 2r}} .$$

15.3 Distance Spectrum

In [GJS16] the notion of spectrum for a circulant matrix was introduced. We recall its definition and significance here and show its close relashionship with the intersections between two columns in the same circulant block.

Definition 15.6. We define the distance between two positions $i, j \in \{0, \dots, r-1\}$ in a circulant block as

$$\mathrm{d}(i,j) = \min((r+j-i) \bmod r, (r+i-j) \bmod r) .$$

When calculating the distances between the positions of the support of $\mathrm{h} \in \mathbb{F}_2[x]/(x^r-1)$, some distances may appear several times. The number of occurrences of a distance δ is called its *multiplicity*:

$$\mu(\delta, \mathrm{h}) = \left| \left\{ (i,j) \;\middle|\; 0 \le i \le j < r, h_i = h_j = 1 \text{ and } \mathrm{d}(i,j) = \delta \right\} \right| .$$

The *spectrum* of h is defined as the set of all distances with their multiplicity:

$$\mathrm{Sp}(\mathrm{h}) = \left\{ (\delta, \mu(\delta, \mathrm{h})) \,\middle|\, \delta \in \{0, 1, \dots, \lfloor r/2 \rfloor\} \right\} .$$

The *spectral polynomial* of h is defined as:

$$\mathrm{s}(\mathrm{h}) = \sum_{\delta=1}^{\lfloor r/2 \rfloor} \mu(\delta, \mathrm{h})\, x^\delta .$$

It is shown in [GJS16] that the knowledge, even partial, of the distance spectrum of a sparse polynomial allows its complete recovery. It is also shown that the statistical analysis of error patterns leading to failures of the QC-MDPC decoder provides information on the secret key spectrum and eventually allows a key recovery attack.

15.3.1 New properties of the distance spectrum

Proposition 15.7. *Let* $\mathrm{h} \in \mathbb{F}_2[x]/(x^r-1)$. *We write* $\mathrm{h}^\top = \phi_{-1}(\mathrm{h}) = \sum_{i \in \mathrm{Supp}(\mathrm{h})} x^{-i}$ *for any* $\mathrm{h} \in \mathbb{F}_2[x]/(x^r-1)$, *this operation corresponds to transposing the circulant block. We have*

$$\mathrm{h}\mathrm{h}^\top = |\mathrm{h}| + \mathrm{s}(\mathrm{h}) + \mathrm{s}(\mathrm{h})^\top$$

where the above product of polynomials is considered in $\mathbb{Z}[x]/(x^r-1)$.

Proof. By expanding the product then decomposing the sum, we obtain

$$\begin{aligned}
\mathrm{h}\mathrm{h}^\top &= \left(\sum_{i \in \mathrm{Supp}(\mathrm{h})} x^i \right) \left(\sum_{j \in \mathrm{Supp}(\mathrm{h})} x^{-j} \right) \\
&= \sum_{\substack{i,j \in \mathrm{Supp}(\mathrm{h}) \\ i=j}} x^{i-j} + \sum_{\substack{i,j \in \mathrm{Supp}(\mathrm{h}) \\ i>j}} x^{i-j} + \sum_{\substack{i,j \in \mathrm{Supp}(\mathrm{h}) \\ i<j}} x^{i-j} \\
&= \sum_{\substack{i,j \in \mathrm{Supp}(\mathrm{h}) \\ i=j}} x^{i-j} + \sum_{\substack{i,j \in \mathrm{Supp}(\mathrm{h}) \\ i<j}} x^{\min(i-j, r+i-j)} + \sum_{\substack{i,j \in \mathrm{Supp}(\mathrm{h}) \\ i<j}} x^{\max(i-j, r+i-j)} \\
&= |\mathrm{h}| + \mathrm{s}(\mathrm{h}) + \mathrm{s}(\mathrm{h})^\top .
\end{aligned}$$

□

There is a one-to-one correspondence between the spectrum of a polynomial h and the number of common bits between h and its δ-shift $x^\delta \mathrm{h}$.

Corollary 15.8. *For all* $\mathrm{h} \in \mathbb{F}_2[x]/(x^r-1)$ *and all* $\delta \in \{0,1,\dots,\lfloor r/2 \rfloor\}$*, we have*

(i) $\mu(\delta,\mathrm{h}) = \left|\mathrm{h} \star x^\delta \mathrm{h}\right|$*;*

(ii) if $\delta \neq 0$*,* $\mu(1,\mathrm{h}) = \mu\left(\delta, \phi_\delta(\mathrm{h})\right)$*.*

Proof. Considering coefficients indices modulo r:

$$\mu(\delta,\mathrm{h}) = (\mathrm{h}\mathrm{h}^\top)_\delta = \sum_{i=0}^{r-1} h_i h_{i-\delta} = \left|\mathrm{h} \star x^\delta \mathrm{h}\right| .$$

The second identity easily derives from $\phi_\delta(x\mathrm{h}) = x^\delta \phi_\delta(\mathrm{h})$ and the fact that ϕ_δ is an isometry for the Hamming distance. □

Remark 15.9. $\mathrm{Sp}(\mathrm{h}) = \mathrm{Sp}(x^\ell \mathrm{h})$ for all $\ell \in \{0,\dots,r-1\}$.

Proposition 15.10. *The application*

$$\begin{aligned} \mathrm{Sp}(\mathrm{h}) &\to \mathrm{Sp}(\phi_a(\mathrm{h})) \\ (\delta, m) &\mapsto (\delta', m) \end{aligned}$$

with $\delta' = \min(a\delta \bmod r, r - (a\delta \bmod r))$*, is a bijection.*

Proof. $\phi_\delta(\mathrm{h}\mathrm{h}^\top) = \phi_\delta(\mathrm{h})\phi_\delta(\mathrm{h})^\top$ □

15.3.2 Distance spectrum statistics

Definition 15.11. Let $\mathrm{h} \in \mathbb{F}_2[x]/(x^r-1)$. Viewed as a binary vector, we assume h starts with 0 and ends with 1, using run-length encoding it can be uniquely described as a sequence of strictly positive integers $(z_1, o_1, z_2, o_2, \dots, z_s, o_s)$: it starts with z_1 zeros followed by o_1 ones, followed by z_2 zeros, *etc.*

We will write $\mathrm{h} \sim (z_1, o_1, z_2, o_2, \dots, z_s, o_s)$.

Proposition 15.12. *If* $\mathrm{h} \sim (z_1, o_1, z_2, o_2, \dots, z_s, o_s)$*, then*

$$\mu(1,\mathrm{h}) = \sum_{i=1}^{s} (o_i - 1) .$$

Proof. Counting the multiplicity of $\delta = 1$ in h is simply counting the number of times there are two consecutive ones. In a block of o_i consecutive ones, this happens $(o_i - 1)$ times. □

The following proposition reduces the problem of fixing a multiplicity in a vector to splitting $|\mathrm{h}|$ ones into s nonempty segments interleaved with s nonempty segments of zeros of total length $(r - |\mathrm{h}|)$.

Proposition 15.13. *Let* $\mathrm{h} \in \mathbb{F}_2[x]/(x^r-1)$*. Suppose* $\mathrm{h} \sim (z_1, o_1, z_2, o_2, \dots, z_s, o_s)$*, then*

$$\begin{cases} o_1 + o_2 + \cdots + o_s = |\mathrm{h}| \ ; \\ z_1 + z_2 + \cdots + z_s = r - |\mathrm{h}| \end{cases}$$

and

$$\mu(1,\mathrm{h}) = m \quad \textit{if and only if} \quad s = |\mathrm{h}| - m .$$

Corollary 15.14. *There are exactly* $\binom{d-1}{d-m-1}\binom{r-d-1}{d-m-1}$ *polynomials* $\mathrm{h} \in \mathbb{F}_2[x]/(x^r-1)$ *of weight* d *starting with a zero and ending with a one such that* $\mu(1,\mathrm{h}) = m$*.*

Proof. The equivalence of the previous proposition gives $s = d - m$, and patterns following the two other conditions are counted using the "stars and bars" principle. □

To count the number of general patterns h such that $\mu(1, \mathrm{h}) = m$, circular shifts of patterns starting with a zero and ending with a one have to be considered. However not all shifts are possible as we need to avoid counting several times the same configuration. For example shifting $\mathrm{h} \sim (z_1, o_1, z_2, o_2, \dots, z_s, o_s)$ by $(z_1 + o_1)$ positions would give $x^{-(z_1+o_1)}\mathrm{h} \sim (z_2, o_2, \dots, z_s, o_s, z_1, o_1)$.

For the sake of clarity, let us generalize our $\sim$ notation for any pattern h.

Definition 15.15. Let $\mathrm{h} \in \mathbb{F}_2[x]/(x^r - 1)$ and let ℓ be the smallest integer such that $x^{-\ell}\mathrm{h}$ starts with a 0 and ends with a 1. We write

$$\mathrm{h} \sim (z_1, o_1, \dots, z_s, o_s)^\ell$$

if and only if

$$x^{-\ell}\mathrm{h} \sim (z_1, o_1, \dots, z_s, o_s)\,.$$

Proposition 15.16. *For any pattern* $\mathrm{h} \in \mathbb{F}_2[x]/(x^r - 1)$ *of weight* d *such that* $\mu(1, \mathrm{h}) = m$, *there is a unique representation*

$$\mathrm{h} \sim (z_1, o_1, \dots, z_s, o_s)^\ell \quad \textit{with} \quad \begin{cases} s = d - m\,; \\ o_1 + \cdots + o_s = d\,; \\ z_1 + \cdots + z_s = r - d\,; \\ \ell \in \{0, \dots, z_1 + o_1 - 1\}\,. \end{cases}$$

Corollary 15.17. *For given integers* m, $0 \le m < d$, *and* δ, $1 \le \delta \le \lfloor r/2 \rfloor$, *there are*

$$\mathcal{N}_m := \frac{r}{d-m}\binom{d-1}{d-m-1}\binom{r-d-1}{d-m-1}$$

polynomials $\mathrm{h} \in \mathbb{F}_2[x]/(x^r - 1)$ *of weight* d *such that* $\mu(\delta, \mathrm{h}) = m$.

Proof. When $\delta = 1$ the result derives from Proposition 15.16.

- First, if $m < d-1$, all values $z_1, \dots, z_s$ and $o_1, \dots, o_s$ are at least 1, so $z_1 \in \{1, \dots, r - d - (s-1)\}$ and $o_1 \in \{1, \dots, d - (s-1)\}$. If z_1 and o_1 are fixed, Corollary 15.14 tells us that there are $\binom{d-o_1-1}{d-m-2}\binom{r-d-z_1-1}{d-m-2}$ ways to choose the values of $z_2, \dots, z_s$ and $o_2, \dots, o_s$. Accounting for the different $(z_1 + o_1)$ possible values of ℓ, we obtain

$$\sum_{z_1=1}^{r-2d+m+1}\sum_{o_1=1}^{m+1}(z_1+o_1)\binom{d-o_1-1}{d-m-2}\binom{r-d-z_1-1}{d-m-2} = \frac{r}{d-m}\binom{d-1}{d-m-1}\binom{r-d-1}{d-m-1}\,.$$

- Now if $m = d-1$, then the only relevant patterns are the r shifts of the pattern consisting of z_1 consecutive zeros followed by o_1 ones.

The generalization to any value of δ derives from the identity $\mu(1, \mathrm{h}) = \mu\big(\delta, \phi_\delta(\mathrm{h})\big)$ of Corollary 15.8. □

Remark 15.18. The probability that two binary vectors of length r and weight d drawn uniformly at random have m intersections is

$$\frac{\binom{d}{m}\binom{r-d}{d-m}}{\binom{r}{d}} = \frac{d}{d-m}\frac{r-d}{r}\frac{\mathcal{N}_m}{\binom{r}{d}}\,.$$

Thus, in a quasi-cyclic matrix, for $m > \left\lfloor \frac{d^2}{r} \right\rfloor$, a column is less likely to have exactly m of intersections with another column in its own circulating block than with a column from other blocks. In fact, with BIKE parameters, $\left\lfloor \frac{d^2}{r} \right\rfloor = 0$.

Corollary 15.19. *Let r and d be positive integers and let $\mathrm{h} \in \mathbb{F}_2[x]/(x^r-1)$ be a pattern of weight d. For any integer δ such that $1 \leq \delta \leq \lfloor r/2 \rfloor$, the probability that δ has a nonzero mutiplicity in the spectrum of h is*

$$1 - \frac{\mathcal{N}_0}{\binom{r}{d}} = 1 - \frac{\binom{r-d-1}{d-1}}{\binom{r-1}{d-1}}\,.$$

Corollary 15.20. *We assume the independence of the multiplicities of the spectrum. Let m be an integer such that $0 \leq m < d$, δ be such that $1 \leq \delta \leq \lfloor r/2 \rfloor$ and $\mathrm{h} \in \mathbb{F}_2[x]/(x^r-1)$,*

$$\pi_m = \Pr[\mu(\delta, \mathrm{h}) = m] = \frac{\mathcal{N}_m}{\binom{r}{d}}\,,$$
$$p_{\geq m} = \Pr\left[\max_{\delta \in \{1,\ldots,\lfloor r/2 \rfloor\}} \mu(\delta, \mathrm{h}) \geq m\right] = 1 - (1-\pi_m)^{\lfloor r/2 \rfloor}\,,$$

and

$$p_{=m} = \Pr\left[\max_{\delta \in \{1,\ldots,\lfloor r/2 \rfloor\}} \mu(\delta, \mathrm{h}) = m\right] = p_{\geq m} - p_{\geq m+1}\,.$$

Example. We give in Table 15.1, for $(r, d) = (12\,323, 71)$, the probabilities of having a certain multiplicy in the spectrum of a circulant block. We observe it is typically low. Note that the independence of the multiplicities for all distances is not true in general. However, this approximation is really close to the values observed in simulation.

15.3.3 Reconstructing the secret key from the spectrum

Proposition 15.7 brings a new approach to the problem of reconstructing a key from its spectrum used in [GJS16]. Knowing $\mathrm{Sp}(\mathrm{h})$, one can write the following system of equations in r boolean variables $h_0, \ldots, h_{r-1} \in \{0, 1\}$.

$$\begin{cases} d = \mu(0, \mathrm{h}) = h_0^2 + \cdots + h_{r-1}^2 = h_0 + \cdots + h_{r-1} \\ \mu(1, \mathrm{h}) = h_0h_1 + h_1h_2 + \cdots + h_{r-1}h_0 \\ \quad\vdots \quad = \quad \vdots \\ \mu\left(\lfloor (r-1)/2 \rfloor, \mathrm{h}\right) = h_0h_{\lfloor \frac{r-1}{2} \rfloor} + h_1h_{\lfloor \frac{r-1}{2} \rfloor + 1} + \cdots + h_{r-1}h_{\lfloor \frac{r-1}{2} \rfloor - 1} \end{cases}$$

Such a system is said to be *pseudo-boolean* as it involves linear operations in $\mathbb{Z}$ of boolean values. Most satisfiability modulo theories (SMT) solvers implement a way to solve them. Nevertheless, the resolution of this system can greatly benefit from simplification.

Indeed, say, for some δ, $\mu(\delta, \mathrm{h}) = 0$. Then

$$h_0h_\delta + h_1h_{\delta+1} + \cdots + h_{r-1}h_{\delta-1} = 0$$

Table 15.1: Numerical application of the multiplicity probabilities given in Corollary 15.20 for $(r, d) = (12\,323, 71)$.

m	π_m	$p_{\geq m}$	$p_{=m}$
0	0.667	1.0	0.0
1	0.272	1.0	0.0
2	0.0539	1.0	0.0
3	0.00692	1.0	0.0186
4	0.000647	0.981	0.73
5	$4.69 \cdot 10^{-5}$	0.251	0.234
6	$2.75 \cdot 10^{-6}$	0.0168	0.016
7	$1.34 \cdot 10^{-7}$	0.000827	0.000793
8	$5.55 \cdot 10^{-9}$	$3.42 \cdot 10^{-5}$	$3.3 \cdot 10^{-5}$
9	$1.98 \cdot 10^{-10}$	$1.22 \cdot 10^{-6}$	$1.18 \cdot 10^{-6}$
10	$6.13 \cdot 10^{-12}$	$3.78 \cdot 10^{-8}$	$3.68 \cdot 10^{-8}$
11	$1.67 \cdot 10^{-13}$	$1.03 \cdot 10^{-9}$	$1.01 \cdot 10^{-9}$
12	$4.05 \cdot 10^{-15}$	$2.49 \cdot 10^{-11}$	$2.44 \cdot 10^{-11}$
13	$8.74 \cdot 10^{-17}$	$5.39 \cdot 10^{-13}$	$5.28 \cdot 10^{-13}$
14	$1.69 \cdot 10^{-18}$	$1.04 \cdot 10^{-14}$	$1.02 \cdot 10^{-14}$

can be written as the conjunctive normal form (CNF)

$$\overline{h_0 h_\delta} \wedge \overline{h_1 h_{\delta+1}} \wedge \cdots \wedge \overline{h_{r-1} h_{\delta-1}} \equiv (\overline{h_0} \vee \overline{h_\delta}) \wedge (\overline{h_1} \vee \overline{h_{\delta+1}}) \wedge \cdots \wedge (\overline{h_{r-1}} \vee \overline{h_{\delta-1}}) \,.$$

To further simplify the system, one can use the fact that at least one multiplicity is nonzero and a lot of variables can be removed. This comes from the fact that the equation system has at least $2r$ solutions: if one vector is a solution then so are its circular shifts and the transpositions of its circular shifts. Therefore, if we know that $\mu\left(\delta', \mathrm{h}\right) > 0$ then we can fix $h_0 = h_{\delta'} = 1$ and the above CNF gives $h_\delta = h_{r-\delta} = h_{\delta+\delta'} = h_{\delta'-\delta} = 0$. So each zero multiplicity in the spectrum can fix the value of at most four variables.

Reconstructing a block with $r = 12\,323$ and $d = 71$ from this simplified system takes less than a second using the SMT solver z3. In comparison, the algorithm proposed in [GJS16] is a depth-first search and does not use the information provided by the multiplicities.

Also, this modelisation tells us that if h is irreducible, the key reconstruction from the spectrum only has two solutions: h and $\mathrm{h}^\top$.

15.4 Weak keys: Constructions and properties

15.4.1 IND-CCA security and weak keys for KEMs

Remember from Definition 2.4 that a KEM (KeyGen, Encaps, Decaps) is said to be δ-correct if

$$\Pr\left[\mathrm{Decaps}(\mathtt{sk}, c) \neq K \,\middle|\, (\mathtt{pk}, \mathtt{sk}) \leftarrow \mathrm{KeyGen}(\{0,1\}^\lambda); (K, c) \leftarrow \mathrm{Encaps}(\mathtt{pk})\right] \leq \delta \,.$$

And remember that for [BIKE], since the encapsulation includes a randomization, δ is the average DFR for a specific decoder, for any key and any error vector.

We can summarize Theorem 2.5 in the case of BIKE, for any IND-CCA adversary $\mathcal{B}$ against BIKE issuing at most q queries to any random oracle with a decoder $\mathcal{D}$ for a set of keys $\mathcal{H}$, with

$$\mathrm{Adv}^{\text{IND-CCA}}_{\text{BIKE}}(\mathcal{B}) \leq q \cdot \mathrm{DFR}_{\mathcal{D},\mathcal{H}} + \epsilon$$

where ϵ encompasses the advantage related to the underlying difficult problems and therefore does not depend at all on the decoder used for the system. This ϵ is dealt with in the usual manner: proper semantically secure transform and system parameters selection according to the computational assumptions.

If one wants to challenge the fact that the decoder actually offers the required security, *i.e.* its failure rate is less than $2^{-\lambda}$, one could present a set of weak keys $\mathcal{W} \subset \mathcal{H}$. The average DFR for these keys $\mathrm{DFR}_{\mathcal{D},\mathcal{W}}$ would have to be higher than $\mathrm{DFR}_{\mathcal{D},\mathcal{H}}$ but its density would also have to be high enough to contribute significantly to the average DFR, *i.e.* the set $\mathcal{W}$ has to be such that

$$\frac{|\mathcal{W}|}{|\mathcal{H}|}\,\mathrm{DFR}_{\mathcal{D},\mathcal{W}} > 2^{-\lambda}\,.$$

In the following sections we will construct three categories of weak keys and evaluate the left-hand side of the above inequality.

15.4.2 Type I

In [DGK19], weak keys are specified as $(\mathrm{h}_0, \mathrm{h}_1)$ with

$$\mathrm{h}_0 = (1 + x + ... + x^{f-1}) + \mathrm{h}'_0$$

such that $\left|\mathrm{h}'_0\right| = d - f$ and $\left|(1 + x + ... + x^{f-1}) \star \mathrm{h}'_0\right| = 0$ for f in a range from 0 to 40. Authors observe that the correcting capability deteriorates as f grows. Values as high as 40 always lead to a decoding failure in simulation.

The reason for this degradation comes from the fact that, compared to a typical key, a weak key admits column pairs with a larger intersection in its private parity check matrix. This can be seen by computing the spectral polynomial of h_0:

$$\begin{aligned}\mathrm{s}(\mathrm{h}_0) = \mathrm{h}_0{\mathrm{h}_0}^\top &= (1 + x + ... + x^{f-1})(1 + x^{-1} + ... + x^{-(f-1)}) + \mathrm{s}' \\ &= f + (f-1)x + ... + x^{f-1} + \mathrm{s}'\end{aligned}$$

where s' has nonnegative coefficients. So any column $x^j\mathrm{h}_0$ has at least $(f-1)$ intersections with its two neighbours $x^{j\pm1}\mathrm{h}_0$, at least $(f-2)$ intersections with $x^{j\pm2}\mathrm{h}_0$, *etc.*

The typical maximum column intersection of BIKE keys is small: for $(r, d) = (12\,323, 71)$ about a quarter of the keys have a maximal column intersection greater than 5 (see Table 15.1). More intersections between columns mean higher correlations between their counters. In Figure 15.1 we measure the difference between the weak keys defined above with parameter $f = 20$ and random keys. With a weak key, an erroneous position tends to have lower counter when its immediate neighbour is erroneous. Conversely, still with a weak key, a non-erroneous position tends to have higher counter when its neighbour is erroneous.

Intuitively, this means that (i) neighbours that are both erroneous tend to hide each other and (ii) an erroneous position will contaminate its correct neighbours. Both effects negatively impact the (threshold-based) decoder (see Remark 15.5). Indeed, a higher counter on average for correct positions implies that more of them are above the threshold and are thus being flipped, adding errors. A lower counter for errors implies that more of them are below the threshold and are thus left unchanged, not decreasing the error weight.

As the decoding degradation is explained by the abnormal distribution of multiplicities in the spectrum of a block, we can generalize the construction of the weak keys. Using Remark 15.9 and Proposition 15.10, from one key defined as in [DGK19] we derive many more with the same multiplicity distribution (up to a permutation of the distance values in the spectrum).

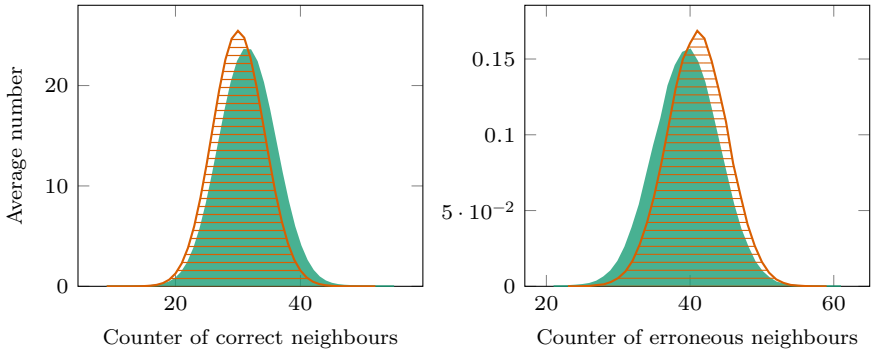


Figure 15.1: Counter values of the neighbours of an error for typical keys (horizontal lines) and weak keys with $f = 20$ (filled).
$(r, d, t) = (12\,323, 71, 134)$.

Definition 15.21. We call weak key of Type I and parameter f, a key $\mathrm{h} = (\mathrm{h}_0, \mathrm{h}_1)$ such that

$$\mathrm{h}_i = \phi_\delta(x^\ell(1 + x + \ldots + x^{f-1}) + \mathrm{h}'_i)$$

for some $i \in \{0, 1\}$, $\ell \in \{0, \ldots, r-1\}$, with $|\mathrm{h}'_i| = d - f$ and $|\mathrm{h}_i| = d$.

Algorithm 15.1 gives a generation algorithm for Type I weak keys.

Algorithm 15.1: Type I weak keys generation.

input : Block size r, column weight d, an integer f.
output : $\mathrm{h} \in \mathbb{F}_2[x]/(x^r - 1)$ with $|\mathrm{h}| = d$ and f δ-consecutive positions.
$(p_1, p_2, \ldots, p_f) \leftarrow (0, 1, \ldots, f-1)$;
Sample $(d - f)$ values $(p_{f+1}, \ldots, p_d)$ from $\{f, \ldots, r-1\}$;
$\delta \xleftarrow{\$} \{1, \ldots, \lfloor r/2 \rfloor\}$;
$\ell \xleftarrow{\$} \{0, \ldots, r-1\}$;
$\mathrm{h} \leftarrow 0$;
for $k \in \{1, \ldots, d\}$ **do**
  /* Coordinate transformation to directly compute $\phi_\delta(x^\ell \mathrm{h})$. */
  $h_{\delta(\ell + p_k)} \leftarrow 1$;
return h;

Proposition 15.22. *We denote $\mathcal{W}_I(f)$ the set of weak keys of Type I of parameter f with blocks of weight d and length r.*

$$|\mathcal{W}_I(f)| \leq 2r \left\lfloor \frac{r}{2} \right\rfloor \binom{r-f}{d-f}.$$

In Definition 15.21, the constructed keys are such that $\mu(\delta, \mathrm{h}_i) \geq f-1$, $\mu(2\delta, \mathrm{h}_i) \geq f-2$, $\ldots$, $\mu((f-1)\delta, \mathrm{h}_i) \geq 1$.

15.4.3 Type II

Instead of having several high multiplicities at the same time, Type II weak keys only increase the multiplicity of a single distance. We will see that they have a lower impact on DFR for a given multiplicity, but a higher density.

Definition 15.23. We call weak key of Type II and parameter m, a key $\mathrm{h} = (\mathrm{h}_0, \mathrm{h}_1)$ such that $\mu\left(\delta, \mathrm{h}_i\right) = m$ for some $i \in \{0, 1\}$ and some distance $\delta \in \{1, \dots, \lfloor r/2 \rfloor\}$.

Thanks to Corollary 15.17 of §15.3.2 we may obtain an upper bound for the number of Type II weak keys.

Proposition 15.24. *Let m be an integer such that $0 \leq m < d$. We denote $\mathcal{W}_{II}(m)$ the set of patterns h of weight d and length r for which one distance at least of its spectrum has multiplicity m. Then*

$$\left|\mathcal{W}_{II}(m)\right| \leq 2 \left\lfloor \frac{r}{2} \right\rfloor \frac{r}{d-m} \binom{d-1}{d-m-1} \binom{r-d-1}{d-m-1}.$$

We only have an upper bound because, for a given m, the sets $\{\mathrm{h} \in \mathbb{F}_2[x]/(x^r - 1) \mid \mu\left(\delta, \mathrm{h}\right) = m\}$ are not disjoint when δ varies. But in practice, when m is above the typical values (4 or 5), the intersections are very small and the bound is very tight. Algorithm 15.2 derives from the combinatorial analysis of §15.3.2 and gives a generation algorithm for Type II weak keys. Its correctness is guaranteed by Proposition 15.16 and Corollary 15.8.

Algorithm 15.2: Type II weak keys generation.

input : Block size r, column weight d, an integer m.
output : $\mathrm{h} \in \mathbb{F}_2[x]/(x^r - 1)$ with $|\mathrm{h}| = d$ and $\exists \delta, \mu\left(\delta, \mathrm{h}\right) = m$.

$s \leftarrow d - m$;
$a_0 \leftarrow 0$; $a_s \leftarrow d$;
$b_0 \leftarrow 0$; $b_s \leftarrow r - d$;
Sample $(s-1)$ values $(a_1, \dots, a_{s-1})$ from $\{1, \dots, d-1\}$;
Sample $(s-1)$ values $(b_1, \dots, b_{s-1})$ from $\{1, \dots, r-d-1\}$;
`/* Componentwise subtraction.` `*/`
$(o_1, \dots, o_s) \leftarrow (a_1, \dots, a_s) - (a_0, \dots, a_{s-1})$;
$(z_1, \dots, z_s) \leftarrow (b_1, \dots, b_s) - (b_0, \dots, b_{s-1})$;
$\delta \xleftarrow{\$} \{1, \dots, \lfloor r/2 \rfloor\}$;
$\ell \xleftarrow{\$} \{0, \dots, z_1 + o_1 - 1\}$;
$\mathrm{h} \leftarrow 0$;
$i \leftarrow -\ell$;
for $j \in \{1, \dots, s\}$ **do**
  $i \leftarrow i + z_j$;
  for $k \in \{0, \dots, o_j - 1\}$ **do**
    `/* Coordinate transformation to directly compute` $\phi_\delta(x^\ell \mathrm{h})$. `*/`
    $h_{\delta(i+k)} \leftarrow 1$;
  $i \leftarrow i + o_j$;
return h;

Remark 15.25. With BIKE, the suggested decoders are parallel: the syndrome is only computed once for each iteration and flips are chosen independently of the order in which positions are considered in an iteration. Therefore, the decoders are such that

$$\mathcal{D}((\phi_\delta(\mathrm{h}_0), \phi_\delta(\mathrm{h}_1)), \phi_\delta(\mathrm{s})) = \phi_\delta(\mathcal{D}((\mathrm{h}_0, \mathrm{h}_1), \mathrm{s})).$$

This means that the set of patterns for which a distance δ has a multiplicity m has the same DFR as the set of patterns for which a distance $\delta' \neq \delta$ has a multiplicity m. Put another way, for a given multiplicity m, for all distances $\delta \in \{1, \dots, \lfloor r/2 \rfloor\}$ the constructed sets have exactly the same contribution to the average DFR.

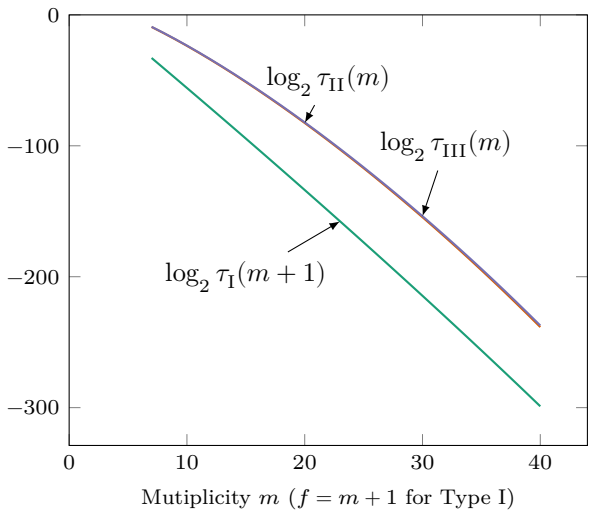


Figure 15.2: Density of weak keys versus multiplicity (log scale) for $(r, d) = (12\,323, 71)$. Type III keys are slightly denser, but their count is very close to the count for Type II.

15.4.4 Type III

Weak keys of Type I and II have properties that concern only one block of the parity check matrix. We can also define weak keys that have many intersections between two columns of two different blocks.

Definition 15.26. We call weak key of Type III and parameter m, a key $\mathrm{h} = (\mathrm{h}_0, \mathrm{h}_1)$ such that $\left|\mathrm{h}_0 \star x^\ell \mathrm{h}_1\right| = m$ for some $\ell \in \{0, \dots, r-1\}$.

Proposition 15.27. *We denote $\mathcal{W}_{III}(m)$ the set of weak keys of Type III of parameter m with blocks of weight d and length r.*

$$\left|\mathcal{W}_{III}(m)\right| \leq r \binom{d}{m} \binom{r-d}{d-m}$$

15.4.5 Statistics

In Figure 15.2 we give the density of all types of weak keys for $(r, d) = (12\,323, 71)$. We denote for all $m > 0$ and for type $\in \{\mathrm{I}, \mathrm{II}, \mathrm{III}\}$

$$\tau_{\mathrm{type}}(m) = \frac{\left|\mathcal{W}_{\mathrm{type}}(m)\right|}{\binom{r}{d}}.$$

We shift the Type I curve ($m = f - 1$) to align the multiplicities. As we will observe in §15.5, the Type I keys have a worse effect on decoding for a given multiplicity. We observe also that for large multiplicity (roughly above $f = 21$ for Type I, and above $m = 27$ for Type II and III) the density is small enough to make those keys harmless (assuming a target of 128 bits of security), regardless of their impact on decoding.

15.5 DFR estimations

To estimate the DFR for weak keys, we will rely on the framework developed in Chapter 14 based on Assumption 3. To be more precise, we assume in this section that for any subset

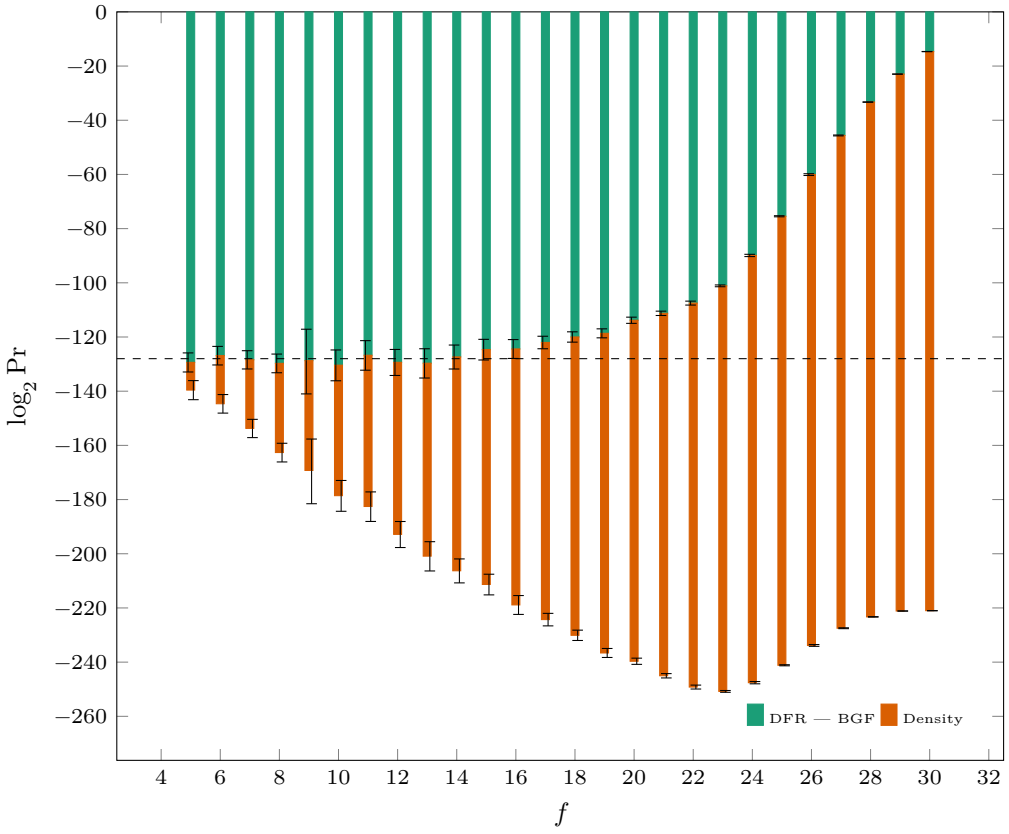


Figure 15.3: Extrapolated (for $r = 12\,323$) DFR *vs.* number of consecutive ones f with Type I weak keys and BGF (7 iterations). $(d, t) = (71, 134)$. 99%-confidence intervals.

of keys $\mathcal{W} \subset \mathcal{H}$, the function $r \mapsto \mathrm{DFR}_{\mathcal{D},\mathcal{W}}(r)$ is concave. In other words, the high distance multiplicity will not have a stronger effect when the block size gets larger, but will affect the decoder similarely for any block size.

In Figure 15.3 & 15.4 & 15.5, we give simulation results for all the types of weak keys previously defined with several values for f or m using BGF with 7 iterations. We can see that no set of weak keys seems to be any threat as the product of their density by their average DFR is always below $2^{-\lambda}$, $\lambda = 128$.

We can observe that Type I keys with $f \geq 10$ and Type II and III with $m \geq 14$ have a negligible influence on the average DFR since their densities multiplied by their DFR are well below $2^{-\lambda}$. For the other weak keys, with lower mutiplicity, the estimated DFR for weak keys is within the confidence interval of the average DFR obtained with random keys. This means that for $f < 10$ and $m < 14$ we did not observe in our experiment a measurable difference in the decoder's DFR between weak keys and random keys.

While all types of weak keys that we established have at least a pair of columns with $(f-1)$ (for Type I) or m (for Type II or III) intersections, some differences explain the different DFR. In a weak key of Type I and parameter f, a column $x^j \mathrm{h}_i$ has at least $(f-1)$ intersections with its two neighbours $x^{j\pm\delta}\mathrm{h}_i$, at least $(f-2)$ intersections with $x^{j\pm 2\delta}\mathrm{h}_i$, etc. In a weak key of Type II and parameter m, a column $x^j \mathrm{h}_i$ has exactly m intersections with its two neighbours $x^{j\pm\delta}\mathrm{h}_i$. And in a weak key of Type III and parameter m, a column $x^j \mathrm{h}_0$ has exactly m intersections with a single column $x^{j'}\mathrm{h}_1$.

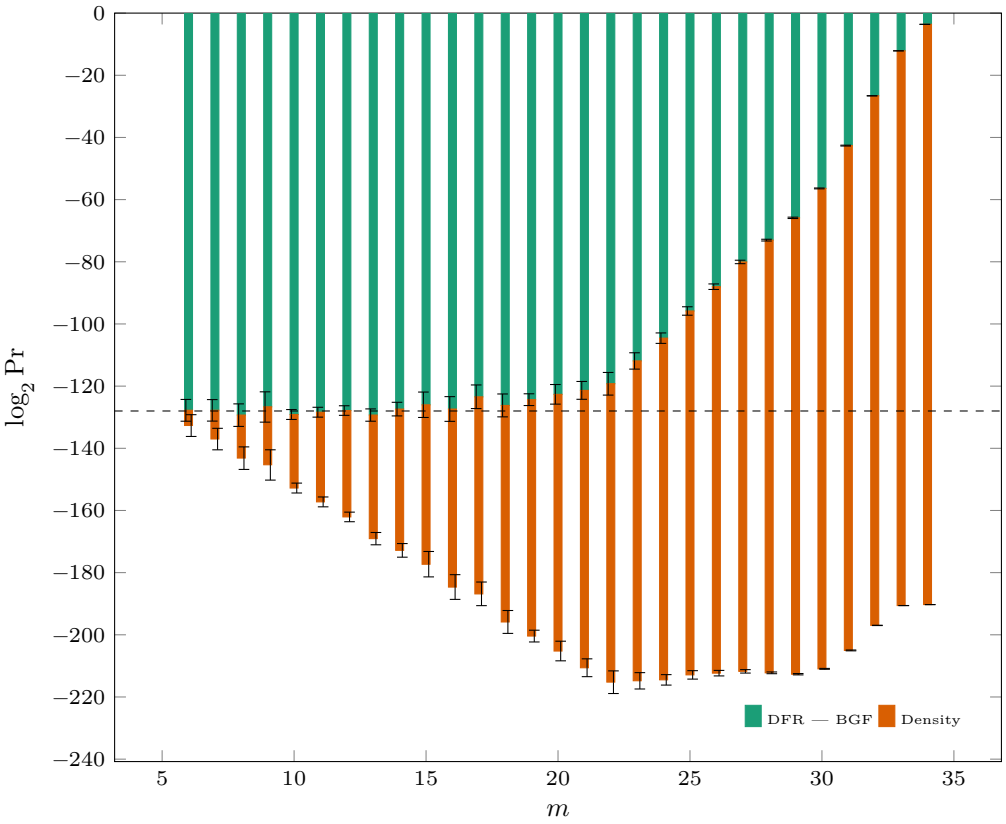


Figure 15.4: Extrapolated (for $r = 12\,323$) DFR *vs.* multiplicity m with Type II weak keys and BGF (7 iterations). $(d, t) = (71, 134)$. 99%-confidence intervals.

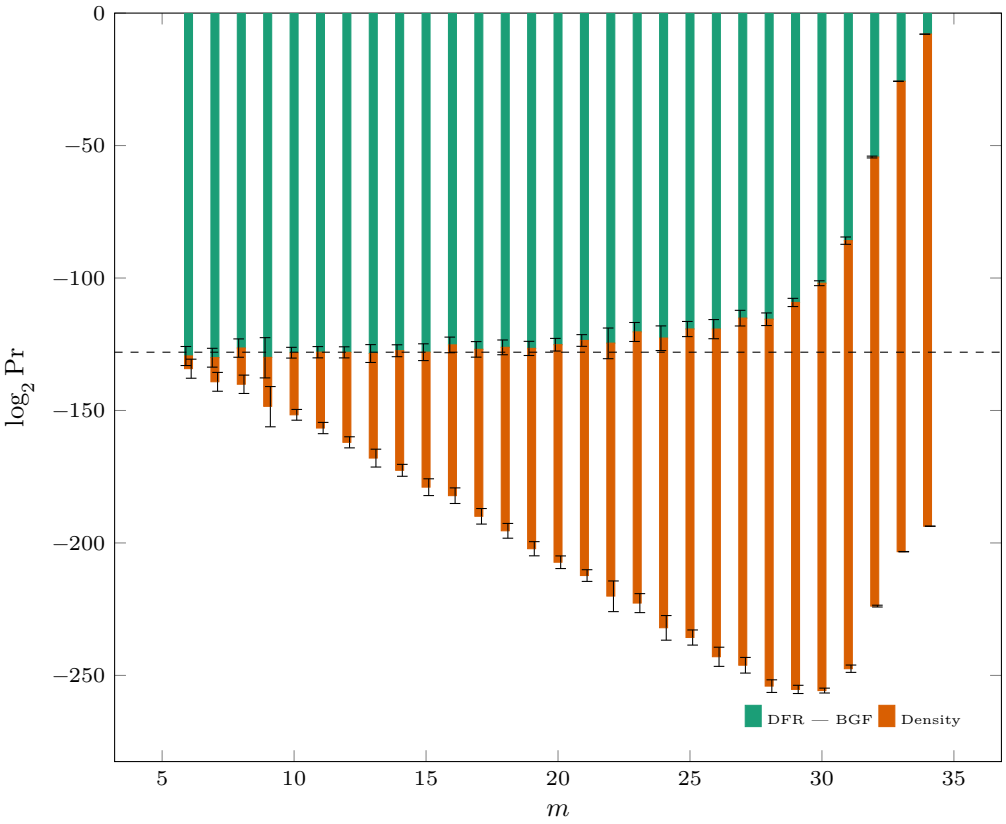


Figure 15.5: Extrapolated (for $r = 12\,323$) DFR *vs.* multiplicity m with Type III weak keys and BGF (7 iterations). $(d, t) = (71, 134)$. 99%-confidence intervals.

15.6 Filtering weak keys

Algorithm 15.3 provides a way of filtering weak keys defined in this paper. It filters Type II weak keys (which include Type I) by computing the spectrum of each block and rejecting when a multiplicity is too high (over a threshold). Type III weak keys are filtered by computing the intersection between every column from the first block with every column from the second block. From the statistics of §15.4 by setting a threshold to, say, 10, this algorithm rejects less than one key out of several million.

Algorithm 15.3: Rejection algorithm for weak keys.

input : Block size r, column weight d, a threshold τ,
a key $\mathrm{h}_0 = x^{j_{0,1}} + x^{j_{0,2}} + \cdots + x^{j_{0,d}}, \mathrm{h}_1 = x^{j_{1,1}} + x^{j_{1,2}} + \cdots + x^{j_{1,d}}$.

for $i \in \{0, 1\}$ **do**
  $\mathrm{S} \leftarrow 0$;
  for $k \in \{1, \ldots, d\}$ **do**
    for $\ell \in \{k+1, \ldots, d\}$ **do**
      /* d is the distance of Definition 15.6. */
      $\delta \leftarrow \mathrm{d}(j_{i,k}, j_{i,\ell})$;
      $S_\delta \leftarrow S_\delta + 1$;
  if $\max_{\delta \in \{1,\ldots,\lfloor r/2 \rfloor\}} S_\delta \geq \tau$ **then**
    /* Filter Type II. */
    return *Reject*;
for $k \in \{1, \ldots, d\}$ **do**
  for $\ell \in \{1, \ldots, d\}$ **do**
    if $\left|\mathrm{h}_0 \star x^{j_{0,k} - j_{1,\ell}} \mathrm{h}_1\right| \geq \tau$ **then**
      /* Filter Type III. */
      return *Reject*;
return *Accept*;

Chapter 16

Error floors: Subsets of error patterns

LDPC codes and Turbo codes are known to suffer from a phenomenon called error floor (see [Gar+01; Ric03]). When the signal-to-noise ratio increases, the decoding performance of these codes first undergoes a sharp decrease in a region called the waterfall region, then there is a sudden change in slope and the performance flattens out (see Figure 16.1) This latter phenomenon is called the error floor.

Just like LDPC codes, QC-MDPC codes are not spared from error floor phenomena. However, the techniques used to estimate the error floor of LDPC codes usually rely on enumerating subgraphs of the Tanner graph of the code. The Tanner graph is denser for an MDPC code and these methods are usually not practical.

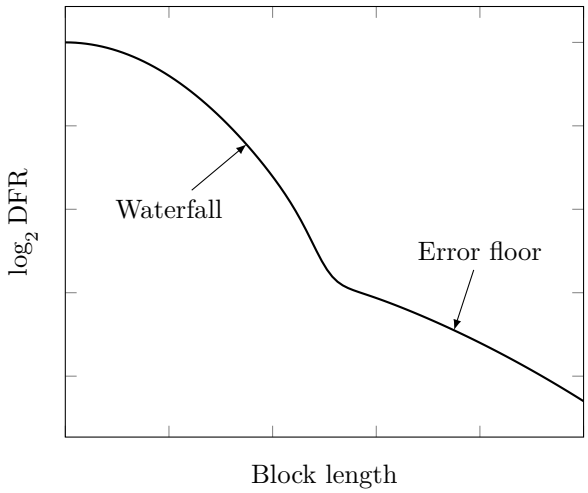


Figure 16.1: Typical DFR curve.

To initiate the discussion of the techniques used in this chapter, let us first consider the case of a minimum distance decoder.

Minimum distance decoding. We assume that we decode an error pattern of weight t in a QC-MDPC code with $\mathcal{D}$ a minimum distance decoder (*i.e.* it outputs the smallest possible error pattern e which, when added to the input, gives a codeword).

Recall that we have defined the DFR as the probability that the decoder outputs an error vector that is different from the one that was used to compute the syndrome:

$$\mathrm{DFR}^{\mathcal{E}}_{\mathcal{D},\mathcal{H}} = \Pr\left[\mathcal{D}(\mathbf{H}, \mathbf{H}\mathrm{e}^\top) \neq \mathrm{e} \,\middle|\, \mathbf{H} \in \mathcal{H}, \mathrm{e} \in \mathcal{E}\right] .$$

With a minimum distance decoder, we have a decoding failure $\mathcal{D}(\mathbf{H}, \mathbf{H}\mathrm{e}^\top) \neq \mathrm{e}$ when

$$\mathrm{e} = \mathrm{c} + \mathrm{e}' \qquad \text{or equivalently} \qquad \mathrm{e}' = \mathrm{c} + \mathrm{e} \tag{16.1}$$

with c a codeword and $\left|\mathrm{e}'\right| \leq t$.

We know that

$$\left|\mathrm{e}'\right| = |\mathrm{c}| + |\mathrm{e}| - 2\,|\mathrm{e} \star \mathrm{c}| ,$$

which means that

$$\frac{|\mathrm{c}|}{2} \leq |\mathrm{e} \star \mathrm{c}| .$$

If e, e′ and c are taken uniformly at random, the probability for this inequality to be satisfied is

$$\sum_{i=\lceil |\mathrm{c}|/2 \rceil}^{\min(|\mathrm{c}|,t)} \frac{\binom{|\mathrm{c}|}{i}\binom{n-|\mathrm{c}|}{t-i}}{\binom{n}{t}} . \tag{16.2}$$

If we write the parity check matrix $\mathbf{H} \in \mathcal{H}$ of a QC-MDPC as

$$\mathbf{H} = \begin{pmatrix} \mathbf{H}_0 & \mathbf{H}_1 \end{pmatrix}$$

then we have a generator matrix $\mathbf{G} \in \mathcal{H}$

$$\mathbf{G} = \begin{pmatrix} \mathbf{H}_1^\top & \mathbf{H}_0^\top \end{pmatrix} .$$

So, taking each row of $\mathbf{G}$, we see that there are r codewords of weight w, they are all obtained by circularly shifting the first row. As w is well below the Gilbert-Varshamov bound, it is safe to assume that there are no smaller codewords in the general case.

Moreover, most of the parameters used to instantiate a QC-MDPC system will take into account the slight asymmetry in the cost of the key and message attacks respectively (because of [Sen11]) by having $w > t$. Therefore, in (16.2), since the sum is zero when $w > 2t$, the sum of two codewords with weight w will probably not trigger a decoding failure because they have a weight slightly less than $2w$. In the end we are only concerned by the r codewords of weight w.

Given the previous remark and the fast decay in the summand when i grows in (16.2), we can approximate the probability of having a bad error pattern with

$$r\frac{\binom{w}{\lceil w/2 \rceil}\binom{n-w}{t-\lceil w/2 \rceil}}{\binom{n}{t}} . \tag{16.3}$$

In order to put into perspective the rarity of such a configuration, let us give an asymptotic formula for (16.3) when $r \to \infty$:

$$2^{A - B\log_2(r)} \tag{16.4}$$

where

$$A = \log_2\left(\binom{w}{\lceil w/2 \rceil}\frac{t!}{(t-\lceil w/2 \rceil)!}\right) - \left\lceil \frac{w}{2} \right\rceil \quad \text{and} \quad B = \left(\left\lceil \frac{w}{2} \right\rceil - 1\right) .$$

Numerical application of this formula are given in Table 16.1. We can see that it is always negligible compared to $2^{-\lambda}$ where λ is the security parameter.

Table 16.1: Numerical application for (16.4) with BIKE parmeters.

λ	w	t	A	B	r	$A - B\log_2(r)$
128	142	134	535.496	70	12 323	−415.738
192	206	199	838.289	102	24 659	−649.874
256	274	264	1 171.659	136	40 597	−910.376

In this chapter we will see that in practice, as the decoder is not minimum distance and is typically a bit-flipping decoder, we can slightly relax the construction of e' in (16.1). Indeed, we will see that c need not necessarily be a codeword (it can be a near-codeword) and that we can choose $\left|\mathrm{e}'\right| > t$. Although these new conditions do not necessarily trigger a decoding failure, they do have a negative impact on decoding performance to an extent that we will quantify using simulations.

In a QC-MDPC code, the quasi-cyclicity of the code endows it with a polynomial structure. Together with the fact that by definition it has a sparse parity check matrix, and it is defined in a field of characteristic 2, we shall see that many near-codewords can be found. We will build subsets of error patterns that are spheres around codewords or near-codewords for a certain radius. We will see that, using a bit-flipping decoder, they have a higher DFR. The density of these subsets multiplied by their DFR is a lower bound of the average DFR. Since the slope is rather flat when r increases, we refer to the curve of this lower bound as the error floor.

16.1 Notations

We adopt the following notations in this chapter.

- The decoding failure rate for a QC-MDPC, with a set of keys $\mathcal{H}$ and a set of messages $\mathcal{E}$ for a decoder $\mathcal{D}$ is written as
$$\mathrm{DFR}^{\mathcal{E}}_{\mathcal{D},\mathcal{H}} = \Pr\left[\mathcal{D}(\mathbf{H}, \mathbf{H}\mathrm{e}^\top) \neq \mathrm{e} \,\middle|\, \mathbf{H} \in \mathcal{H}, \mathrm{e} \in \mathcal{E}\right] .$$
- For ease of reading, when omitted, the set $\mathcal{E}$ is by default the whole set of admissible messages
$$\mathcal{E} = \mathcal{E}_{n,t} = \{\mathrm{e} \in \{0,1\}^n \mid |\mathrm{e}| = t\} .$$
- When omitted, the set $\mathcal{H}$ is by default the whole set of admissible keys
$$\begin{aligned}\mathcal{H} &= \mathcal{H}_{d,w,r\times n} \\ &= \left\{\mathbf{H} \in \mathbb{F}_2^{r\times n} \,\middle|\, \forall i \in \{0,\dots,r-1\}, \left|\mathbf{h}_i^\top\right| = w, \forall j \in \{0,\dots,n-1\}, \left|\mathbf{h}_j\right| = d\right\} ,\end{aligned}$$
- The column weight d, the row weight w, and the error weight t will be obvious from the context and we will write
$$\mathrm{DFR}(r) = \mathrm{DFR}^{\mathcal{E}_{2r,t}}_{\mathcal{D},\mathcal{H}_{d,2d,r\times 2r}} .$$

16.2 Structured patterns in QC-MDPC codes

16.2.1 Low weight codewords

There is an isometry between codewords of a QC-MDPC code of rate 1/2 and those of its dual. The following definition describes the set of codewords of weight w for a QC-MDPC.

Definition 16.1. Let $\mathrm{h} = \left(\mathrm{h}_0, \mathrm{h}_1\right) \in \left(\mathbb{F}_2[x]/(x^r-1)\right)^2$ be the parity check matrix of a QC-MDPC code of row weight w.

We write $\mathcal{C}$ the set of codewords of weight w:

$$\mathcal{C} = \left\{ \left(x^s \mathrm{h}_1, x^s \mathrm{h}_0\right) \,\middle|\, s \in \{0, \dots, r-1\}) \right\} .$$

The sum of any two codewords of weight w gives the set of codewords $2\mathcal{C}$. And any $c \in 2\mathcal{C}$ has weight $2w - \epsilon_0 - \epsilon_1$ where the distribution of ϵ_i is given by π_m in Corollary 15.20 for $m = \epsilon_i$ and $i = 0, 1$. These codewords have too great a weight to be relevant here. Indeed, we will construct error patterns of weight t that are close to codewords. With the usual BIKE parameters, we already have $w > t$, doubling the size of the codewords can only increase the distance to the error patterns of fixed weight t.

16.2.2 Near-codewords

Definition 16.2. Let $\mathbf{H}$ be the parity check matrix of a linear code. A (u, v) near-codeword is an error pattern e of weight u such that $\left|\mathbf{H}\mathrm{e}^\top\right| = v$.

We say that the variable nodes corresponding to the support of such an error pattern constitute a (u, v) trapping set as defined in [Ric03].

Let us write a circulant block in its polynomial representation $\mathrm{h} = \sum_{i \in \mathrm{Supp(h)}} x^i \in \mathbb{F}_2[x]/(x^r-1)$. Applying the Frobenius endomorphism we obtain:

$$\mathrm{h}^2 = \sum_{i \in \mathrm{Supp(h)}} x^{2i} .$$

Since $\forall i \in \{0,1\}, \left|\mathrm{h}_i^2\right| = \left|\mathrm{h}_i\right| = d$, we have identified (d, d) near-codewords present in any QC-MDPC code.

In the following definition, we describe all the (d, d) near-codewords based on this template, *i.e.* those of each circulant block and all their circular shifts.

Definition 16.3. Let $\mathrm{h} = \begin{pmatrix} \mathrm{h}_0 & \mathrm{h}_1 \end{pmatrix} \in \mathbb{F}_2[x]/(x^r-1)^2$ be the parity check matrix of a QC-MDPC code of row weight w.

We write $\mathcal{N}$ the following set of (d, d) near-codewords $\mathrm{e} = (\mathrm{e}_0, \mathrm{e}_1)$

$$\mathcal{N} = \left\{ \left(x^s \mathrm{h}_0, 0\right) \,\middle|\, s \in \{0, \dots, r-1\} \right\} \cup \left\{ \left(0, x^s \mathrm{h}_1\right) \,\middle|\, s \in \{0, \dots, r-1\} \right\} .$$

There are $2r$ such (d, d) near-codewords.

We will also consider the set $2\mathcal{N}$ of the sums of two (d, d) near-codewords as they are $(2d - \epsilon_0, 2d - \epsilon_1)$ near-codewords (for some small values ϵ_0 and ϵ_1) and we usually have $2d = w \approx t$

16.3 Error patterns impeding decoding

We have already covered the case where one uses a minimum distance decoder in a QC-MDPC scheme. But such a decoder is too challenging to efficiently implement, and we use a bit-flipping or a belief propagation algorithm. We will here focus on variants of the former.

In any bit-flipping variant, the syndrome is used to compute the counters that are then used to decide whether to flip a bit or not. In this section we will use those two data as a benchmark to assess the influence of low weight codewords and near-codewords on the decoder.

We keep the same construction as before and choose an error pattern e' as

$$\mathrm{e} = \mathrm{c} + \mathrm{e}'$$

with c in $\mathcal{C}$, $\mathcal{N}$ or $2\mathcal{N}$ and $|\mathrm{e}| = t$, but this time we relax the condition on the weight of e'. We have

$$\left|\mathrm{e}'\right| = |\mathrm{c}| + |\mathrm{e}| - 2\,|\mathrm{e} \star \mathrm{c}| \ .$$

For a set $\mathcal{S}$ that is either $\mathcal{C}$, $\mathcal{N}$ or $2\mathcal{N}$, we define the set of error patterns e that are near $\mathcal{S}$. We set the distance to these sets via the constant $\ell = |\mathrm{e} \star \mathrm{c}|$.

Definition 16.4. Let $S \subset (\mathbb{F}_2[x]/(x^r-1))^2$, we write

$$\mathcal{A}_{t,\ell}(S) = \bigcup_{v \in S} \{u \in (\mathbb{F}_2[x]/(x^r-1))^2 \mid |u| = t, |u \star v| = \ell\} \, .$$

Algorithm 16.1 provides a way to construct these sets.

Algorithm 16.1: Weak error patterns generation.

function `weak_error`$(r, t, \ell, (\mathrm{h}_0, \mathrm{h}_1), S)$**:**
- **input :** Block size r, error weight t, an integer ℓ, a key $(\mathrm{h}_0, \mathrm{h}_1) \in (\mathbb{F}_2[x]/(x^r-1))^2$, a set $S \in \{\mathcal{C}, \mathcal{N}, 2\mathcal{N}\}$.
- **output :** A vector c that has ℓ intersections with an element of S.
- $\mathrm{c} \leftarrow$ `sample`$_S(\mathrm{h}_0, \mathrm{h}_1)$;
- $s \xleftarrow{\$} \{0, \dots, r-1\}$;
- $(p_0, \dots, p_{\ell-1}) \xleftarrow{\$}$ Sample ℓ values from Supp(c);
- $(p_\ell, \dots, p_{t-1}) \xleftarrow{\$}$ Sample $(t-\ell)$ values from $\{0, \dots, r-1\} \setminus$ Supp(c);
- $\mathrm{c} \leftarrow 0$;
- **for** $k \in \{0, \dots, t-1\}$ **do**
  - $c_{s+p_k} \leftarrow 1$;
- **return** c;

function `sample`$_{\mathcal{C}}(\mathrm{h}_0, \mathrm{h}_1)$**:**
- **return** $(\mathrm{h}_1, \mathrm{h}_0)$;

function `sample`$_{\mathcal{N}}(\mathrm{h}_0, \mathrm{h}_1)$**:**
- $(\mathrm{c}_0, \mathrm{c}_1) \leftarrow (0, 0)$;
- $i \xleftarrow{\$} \{0, 1\}$;
- $\mathrm{c}_i \leftarrow \mathrm{h}_i$;
- **return** $(\mathrm{c}_0, \mathrm{c}_1)$;

function `sample`$_{2\mathcal{N}}(\mathrm{h}_0, \mathrm{h}_1)$**:**
- $s \xleftarrow{\$} \{0, \dots, r-1\}$;
- **return** `sample`$_{\mathcal{N}}(\mathrm{h}_0, \mathrm{h}_1) + x^s$`sample`$_{\mathcal{N}}(\mathrm{h}_0, \mathrm{h}_1)$;

These sets can also be seen as unions of spheres of radius $w + t - 2\ell$ and centers in S.

Proposition 16.5. *Suppose that there exists a weight w such that $\forall u \in S, |u| = w$, then $\forall v \in \mathcal{A}_{t,\ell}(S), |v - u| = w + t - 2\ell$ and*

$$\left|\mathcal{A}_{t,\ell}(S)\right| \leq |S| \binom{w}{\ell} \binom{n-w}{t-\ell} \, .$$

In this case, we write

$$\mathcal{D}_{t,w+t-2\ell}(S) = \left|S\right| \frac{\binom{w}{\ell}\binom{n-w}{t-\ell}}{\binom{n}{t}},$$

an upper bound on the density of the set $\mathcal{A}_{t,\ell}(S)$.

Remark 16.6. For any ℓ and S, we can rewrite $\mathcal{A}_{t,\ell}(S)$ as

$$\mathcal{A}_{t,\ell}(S) = \bigcup_{s\in S} \bigcup_{\substack{c\subset s\\ |c|=\ell}} \{c\} + \{v \;\big|\; |v| = t-\ell, |c\star v| = 0\}.$$

It is a union of spheres, and they might intersect. However, this is highly unlikely with the parameters usually considered for QC-MDPC schemes. Indeed, suppose that for some $c, c' \in S$ there exists a u such that $u = c + v = c' + v'$ where $|c| = \left|c'\right| = \ell$ and $|v| = \left|v'\right| = t - \ell$. Let us rewrite it as $c + c' = v + v'$. We always have $\left|c + c'\right| \leq 2\ell$, and

$$\Pr[\left|v + v'\right| \leq 2\ell] \leq \sum_{k=t-2\ell}^{t-\ell} \frac{\binom{t-\ell}{k}\binom{n-w-t+\ell}{t-\ell-k}}{\binom{n-w}{t-\ell}}.$$

Multiplying the right-hand side by $\left|S\right|^2 \binom{w}{\ell}^2$ gives a really rough upper bound on the proportion of duplicates in Proposition 16.5. This is negligible for $\ell < 45$ with $(r, d, t) = (12\,323, 71, 134)$ and $S \in \{\mathcal{C}, \mathcal{N}, 2\mathcal{N}\}$. In this case, the upper bound $\mathcal{D}_{t,w+t-2\ell}(S)$ is really close to the actual density.

In Figure 16.2 we give the density of all the sets of error patterns described in the previous section for $(r, d, t) = (12\,323, 71, 134)$. We can see, for each set, the range of distance of interest for our analysis *i.e.* those that have a density greater that 2^{-128}.

Influence on decoding. Let $\mathcal{S}$ be $\mathcal{C}$, $\mathcal{N}$ or $2\mathcal{N}$. The proximity of an error pattern to an element of $\mathcal{S}$ has an influence on the counters. Let us write one of these elements $\mathrm{c} \in \mathcal{S} \in \left(\mathbb{F}_2[x]/(x^r - 1)\right)^2$ and an error pattern $\mathrm{e} \in \left(\mathbb{F}_2[x]/(x^r - 1)\right)^2$. Let us separate the positions in c and e depending on whether they are mutual or not.

We can write

$$\mathrm{p} = \mathrm{e} \star \mathrm{c}, \qquad \mathrm{e}^\perp = \mathrm{e} - \mathrm{p}, \qquad \mathrm{c}^\perp = \mathrm{c} - \mathrm{p},$$

then

$$\mathrm{e} = \mathrm{e}^\perp + \mathrm{p}, \qquad \mathrm{c} = \mathrm{c}^\perp + \mathrm{p}.$$

Since the distance between e and c is given by the formula

$$|\mathrm{e}| + |\mathrm{c}| - 2\,|\mathrm{p}|,$$

and assuming the elements of $\mathcal{S}$ all have the same weight, a closest element of $\mathcal{S}$ to e is one that maximizes $\ell = |\mathrm{p}|$.

We now suppose that c is one of those. In Table 16.2, we give statistics for different sets $\mathcal{S}$ and different values of ℓ, namely the syndrome weight and the distribution of the counters classified according to whether they concern a position belonging to $\mathrm{e}^\perp$, $\mathrm{h}^\perp$, p or none of these. Remember that for a bit-flipping iteration to be efficient, the counters of positions in the support of e have to be higher than the others. What we can observe from the table is that as ℓ increases, the counters of positions in the support of $\mathrm{e}^\perp$ or $\mathrm{c}^\perp$ increase, while those of p and the others decrease. In other words, the algorithm is more likely to mistakenly add errors by flipping positions in the support of $\mathrm{c}^\perp$ and miss the errors in $\mathrm{p} \subset \mathrm{e}$. It is also interesting to point out that this influence on the counters as well as on the syndrome weight can even be observed for small values of ℓ.

Note that as all the sets $\mathcal{C}$, $\mathcal{N}$ or $2\mathcal{N}$ are stable by blockwise circular shifts, any error pattern has at least one intersection with one of them, and is highly likely to have two. Therefore, depending on the set $\mathcal{S}$, the set $\mathcal{A}_{t,\ell}(S)$ might be empty for lower values of ℓ.

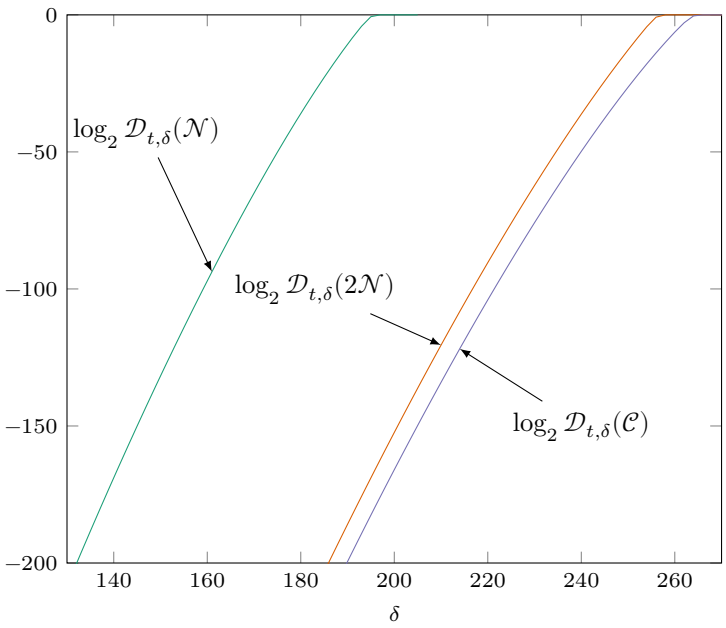


Figure 16.2: Upper bound on the probability that an error pattern of weight t is at a certain distance δ from $\mathcal{N}$, $2\mathcal{N}$ or $\mathcal{C}$.
For $\ell = |\mathrm{e} \star \mathrm{c}|$, the distance is $\delta = |\mathrm{c}| + t - 2\ell$ where $|\mathrm{c}|$ is d, w or w respectively for $\mathcal{N}$, $2\mathcal{N}$ or $\mathcal{C}$.
$(r, d, t) = (12\,323, 71, 134)$.

16.4 Lower bound on the DFR with simulations

Using Algorithm 16.1, we can evaluate the DFR on the sets $\mathcal{A}_{t,\ell}(S)$ for any decoding algorithm, any ℓ, and $S \in \{\mathcal{C}, \mathcal{N}, 2\mathcal{N}\}$. Figure 16.3 & 16.4 & 16.5 gives DFR estimates for various decoders. We can observe that the product of the density of the sets $\mathcal{A}_{t,\ell}(\mathcal{C})$, $\mathcal{A}_{t,\ell}(\mathcal{N})$, $\mathcal{A}_{t,\ell}(2\mathcal{N})$ multiplied by the obtained DFR is increasing. But we eventually reach the limits of what can be obtained with simulation. It would not be surprising if as the distance increases, the DFR converges to its average value (which is expected to be below $2^{-\lambda}$).

We can also see in those figures that the near-codewords in $\mathcal{N}$ seem to have the most influence on the decoder in comparison to the other sets $\mathcal{C}$ or $2\mathcal{N}$.

16.5 Comments

Not that the DFR obtained are not the results of extrapolations and this construction gives directly a lower bound on the DFR. The (small) imprecision on these values comes from the fact that we only compute a bound on the density of the considered sets and the fact that the DFR is estimated with simulation. As discussed in the previous section, the former is really tight and, for the latter, estimations are given with their 99% confidence interval.

While decoders are slowed down by the closeness of problematic patterns, we observe that failures would not be avoided by simply increasing the number of iterations. Error patterns are either decoded in a few iterations or they are never decoded.

The technique described here for generating error patterns close to $\mathcal{N}$ provides a good way to test decoders' borderline cases. For example, although the threshold selection

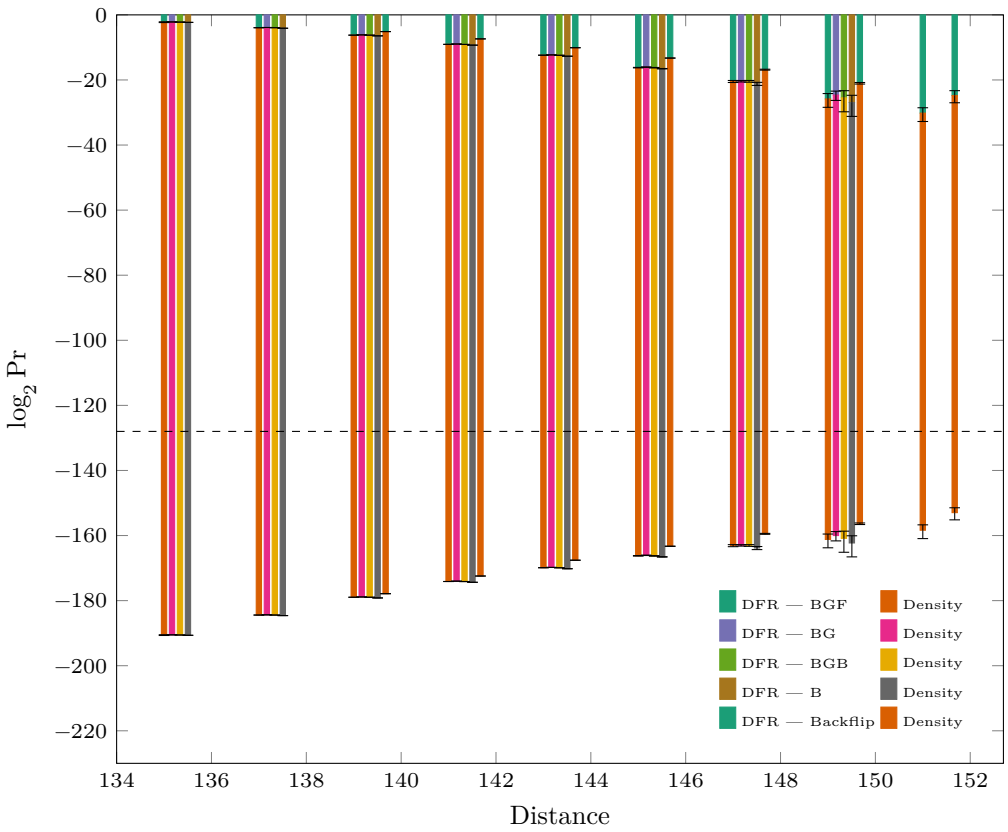


Figure 16.3: DFR *vs.* distance to $\mathcal{N}$ with various gray decoders (7 iterations for BGF, 8 for B and 9 for the others) and Backflip with multiple thresholds (7 iterations). $(r, d, t) = (12\,323, 71, 134)$. 99%-confidence intervals.

rules for BlackGray decoders are not explicitly described in [DGK20b], the additional implementation of [BIKE] (for versions prior to and including 3.2) implemented a BlackGray decoder and used a simple affine function of the syndrome weight. Our early work on error floors showed that this resulted in the existence of a subset of error patterns whose average DFR multiplied by the density was well above $2^{-\lambda}$, thus threatening the IND-CCA security of the system. In this case, the solution was to set a minimum value for the threshold at $(d+1)/2$.

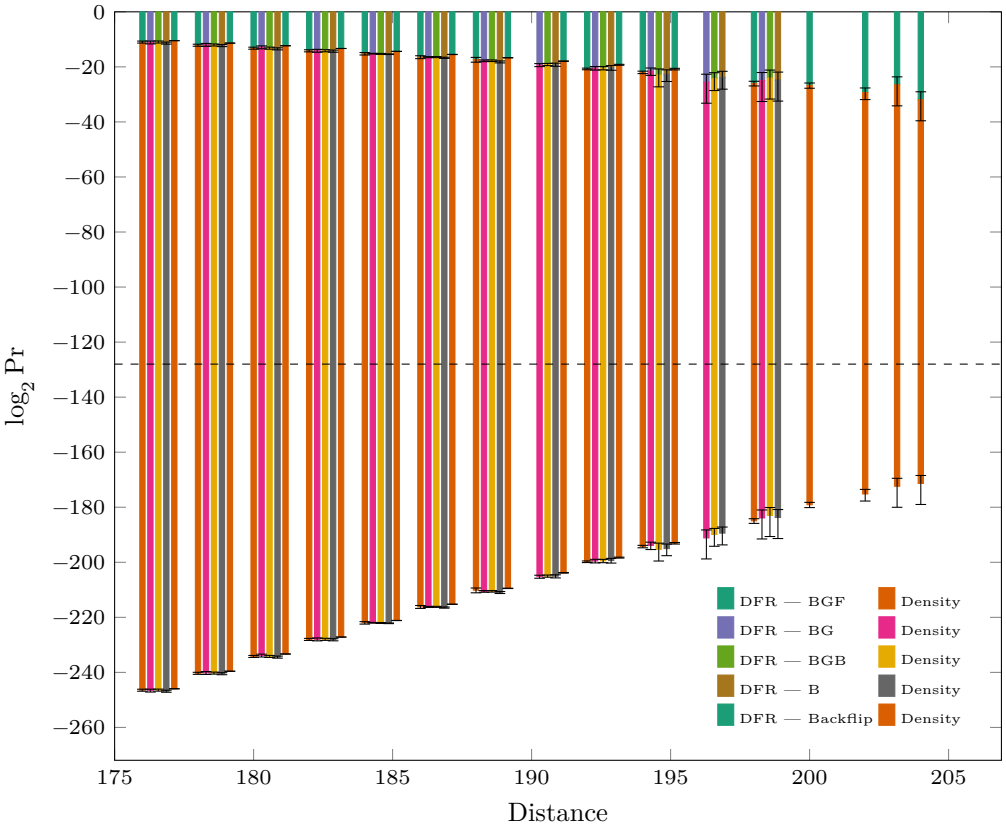


Figure 16.4: DFR *vs.* distance to $2\mathcal{N}$ with various gray decoders (7 iterations for BGF, 8 for B and 9 for the others) and Backflip with multiple thresholds (7 iterations). $(r, d, t) = (12\,323, 71, 134)$. 99%-confidence intervals.

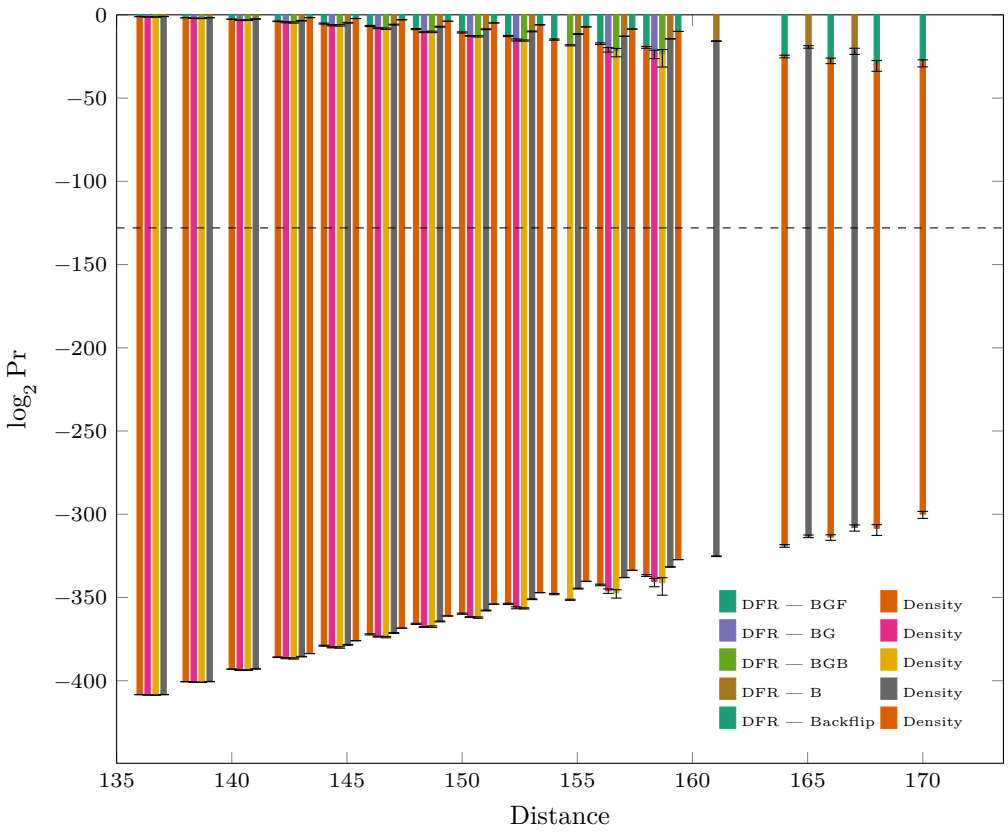


Figure 16.5: DFR *vs.* distance to $\mathcal{C}$ with various gray decoders (7 iterations for BGF, 8 for B and 9 for the others) and Backflip with multiple thresholds (7 iterations). $(r, d, t) = (12\,323, 71, 134)$. 99%-confidence intervals.

Table 16.2: Influence of codewords and near-codewords on the syndrome weight and the counters distributions for $(r, d, t) = (11\,779, 71, 134)$.

ℓ	Syndrome weight		Counters							
			$\in \mathrm{e}^\perp$		$\in \mathrm{p}$		$\in \mathrm{c}^\perp$		Other	
	Mean	Var	Mean	Var	Mean	Var	Mean	Var	Mean	Var
Average case			($\in$ e)							
-	4 740.826	2 436.014	42.590	17.032	-	-	-	-	28.496	17.045
$\mathcal{S} = \mathcal{N}$										
3	4 750.178	2 447.685	42.629	16.986	41.897	16.997	29.399	17.034	28.550	17.049
4	4 740.988	2 474.073	42.610	17.016	41.982	16.979	29.316	17.056	28.495	17.044
5	4 737.927	2 438.134	42.614	17.027	41.864	17.043	29.404	17.028	28.476	17.041
10	4 724.176	2 441.586	42.698	17.015	40.921	16.975	30.342	17.020	28.390	17.025
30	4 557.218	2 331.555	43.731	16.769	37.008	16.774	34.229	16.775	27.373	16.803
40	4 388.433	2 212.589	44.765	16.516	34.588	16.535	36.677	16.522	26.351	16.553
50	4 138.419	2 032.129	46.288	16.080	31.413	16.114	39.908	16.052	24.841	16.127
60	3 775.992	1 779.213	48.498	15.339	26.942	15.375	44.499	15.360	22.658	15.399
70	3 249.590	1 392.664	51.697	14.018	20.312	14.066	51.333	13.981	19.496	14.108
$\mathcal{S} = 2\mathcal{N}$										
7	4 752.599	2 449.347	42.630	16.968	42.236	16.938	29.190	17.047	28.564	17.052
8	4 741.606	2 448.857	42.624	17.003	42.095	17.014	29.226	17.035	28.497	17.042
9	4 737.578	2 448.165	42.627	17.020	41.986	16.993	29.300	17.029	28.472	17.039
10	4 735.105	2 449.520	42.637	17.017	41.873	17.022	29.389	17.044	28.456	17.036
30	4 654.694	2 405.722	43.133	16.925	40.041	17.168	31.092	17.192	27.960	16.933
50	4 475.758	2 310.459	44.238	16.657	38.243	17.160	32.889	17.178	26.869	16.683
70	4 155.064	2 106.282	46.207	16.120	35.868	16.951	35.277	16.937	24.923	16.154
90	3 611.695	1 745.599	49.530	14.956	32.098	16.281	39.105	16.274	21.640	15.016
110	2 689.331	1 088.797	55.151	12.267	25.314	14.254	46.023	14.168	16.087	12.408
130	1 082.673	214.484	64.897	5.562	12.161	7.660	59.483	7.311	6.458	5.843
$\mathcal{S} = \mathcal{C}$										
5	4 743.816	2 439.823	42.606	17.008	42.282	17.033	28.940	17.071	28.512	17.047
6	4 739.387	2 462.484	42.607	17.010	42.168	17.022	29.013	17.083	28.484	17.044
7	4 737.776	2 453.670	42.614	17.017	42.076	17.017	29.106	17.092	28.474	17.041
8	4 736.235	2 452.988	42.619	17.024	41.981	17.070	29.200	17.098	28.464	17.039
30	4 653.611	2 393.044	43.144	16.916	40.012	17.184	31.132	17.199	27.953	16.933
50	4 471.595	2 304.922	44.265	16.655	38.168	17.163	32.969	17.171	26.843	16.678
70	4 144.716	2 105.496	46.267	16.105	35.726	16.941	35.423	16.945	24.860	16.135
90	3 589.383	1 718.329	49.668	14.901	31.815	16.259	39.391	16.222	21.504	14.963
110	2 642.406	1 058.521	55.436	12.109	24.740	14.157	46.614	14.061	15.804	12.251
130	983.970	184.843	65.497	5.057	10.924	7.260	60.756	6.885	5.865	5.355

Conclusion

We first studied a sequential bit-flipping algorithm that updates the syndrome and recalculates its threshold after each flip. It was designed primarily to be stochastic and suitable for a probabilistic analysis. However, it has led us to design algorithms based on a novel idea which consists in regularly cancelling old flips. This idea has been concretized in a more efficient way with Backflip. Backflip is particularly efficient, especially when it performs a large number of iterations. Interestingly, it performs better than the soft-decision decoding algorithm to which we compared it. This work was published in [SV20a] and Backflip was suggested in [BIKE] for the second round of the NIST strandardization process. Finally, we discussed the efficiency of gray decoders.

We then studied two probabilistic models of decoders. For the first one, we managed to incorporate a feature that has not been fully taken into account so far in the analyses of decoding algorithms, namely the quasi-cyclicity of the code and the regularity that it implies. Our model accurately predicts the behaviour of an iteration of parallel bit-flipping, and gives us the distribution of the error weight after one iteration. The second model concerns the step-by-step decoder mentioned above. This model is able to handle the complete algorithm execution not just one iteration. Compared to the previous model, it is less accurate in this case. An interesting application for this algorithm and its analysis is to use it at the end of decoding when there is not many errors left to decode. This is what can be done with an iteration of a gray decoder for example. This work on the Markovian model has been published in [SV19].

Finally, we presented a DFR extrapolation framework. It is based on an assumption that followed the conclusions obtained from the DFR models. We have rigorously determined the confidence intervals associated with these extrapolations. We then constructed some sets of weak keys that were justified by structural properties of quasi-cyclic codes. This work is the object of a preprint [SV20b]. Then we shed light on the existence of near codewords due to this structure, and assessed their influence on decoding. This provides a lower bound on the DFR and ensures that extrapolations are not wrong. These extrapolation techniques and checks against weak keys and error floors were used extensively for the selection of [BIKE]'s IND-CCA parameters.

Perspectives. We have presented efficient algorithms, but many aspects of their implementations are based on simulations and optimizations. It would be interesting to have a better understanding and justification of ttl functions or the thresholds used to define the gray positions.

The proximity to near codewords emerges as crucial in understanding the decoding instances that fail. In particular, it has an influence on the syndrome weight and counters distributions. Models could greatly benefit from conditioning these distributions to the distance to a near codeword.

Errors appear in model predictions when we try to iterate it beyond one iteration because of the correlations that appear when decoding. It would be interesting to see to what extent these correlations can be related to the near codewords we found, *i.e.* how

much the algorithm tends to increase the number of bits in common with a row of the parity check matrix.

Concerning weak keys, with those of Type I, each column has a large intersection with several columns, and with those of Type II and III, each column has a large intersection with only one (or two) other columns. There is a range of possibilities between these two extremes that it would be interesting to investigate. For example, we could consider keys that have two distances with a high multiplicity, or three, *etc.*

Bibliography

[802.11] "IEEE Standard for Information technology—Telecommunications and information exchange between systems Local and metropolitan area networks—Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications". In: *IEEE Std 802.11-2016 (Revision of IEEE Std 802.11-2012)* (Dec. 14, 2016), pp. 1–3534. DOI: 10.1109/IEEESTD.2016.7786995.

[Ala+20] Gorjan Alagic, Jacob Alperin-Sheriff, Daniel Apon, David Cooper, Quynh Dang, John Kelsey, Yi-Kai Liu, Carl Miller, Dustin Moody, Rene Peralta, Ray Perlner, Angela Robinson, and Daniel Smith-Tone. "Status Report on the Second Round of the NIST Post-Quantum Cryptography Standardization Process". In: (July 2020). DOI: 10.6028/NIST.IR.8309.

[Ale03] Michael Alekhnovich. "More on average case vs approximation complexity". In: *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.* 2003, pp. 298–307. DOI: 10.1109/SFCS.2003.1238204.

[APRS20] Daniel Apon, Ray A. Perlner, Angela Robinson, and Paolo Santini. "Cryptanalysis of LEDAcrypt". In: *Advances in Cryptology - CRYPTO*. Ed. by Daniele Micciancio and Thomas Ristenpart. Vol. 12172. LNCS. Springer, 2020, pp. 389–418. DOI: 10.1007/978-3-030-56877-1_14.

[BDLO16] Magali Bardet, Vlad Dragoi, Jean-Gabriel Luque, and Ayoub Otmani. "Weak Keys for the Quasi-Cyclic MDPC Public Key Encryption Scheme". In: *AFRICACRYPT 2016*. Ed. by David Pointcheval, Abderrahmane Nitaj, and Tajjeeddine Rachidi. Vol. 9646. LNCS. Springer, 2016, pp. 346–367. DOI: 10.1007/978-3-319-31517-1_18.

[BHLV17] Daniel J Bernstein, Nadia Heninger, Paul Lou, and Luke Valenta. "Post-quantum RSA". In: *Post-Quantum Cryptography (PQCrypto)*. Ed. by Tanja Lange and Tsuyoshi Takagi. Vol. 10346. LNCS. Utrecht, Netherlands: Springer, 2017, pp. 311–329. DOI: 10.1007/978-3-319-59879-6_18.

[BIKE] Carlos Aguilar Melchor, Nicolas Aragon, Paulo S L M Barreto, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Ghosh Santosh, Shay Gueron, Tim Güneysu, Rafael Misoczki, Edoardo Persichetti, Nicolas Sendrier, Jean-Pierre Tillich, Valentin Vasseur, and Gilles Zémor. *BIKE*. NIST Round 3 submission for Post-Quantum Cryptography. Aug. 2020. URL: https://bikesuite.org.

[BJK19] Irina E Bocharova, Thomas Johansson, and Boris D Kudryashov. "Improved iterative decoding of QC-MDPC codes in the McEliece public key cryptosystem". In: *IEEE International Symposium on Information Theory (ISIT)*. 2019, pp. 1882–1886. DOI: 10.1109/ISIT.2019.8849839.

[BJMM12] Anja Becker, Antoine Joux, Alexander May, and Alexander Meurer. "Decoding Random Binary Linear Codes in $2^{n/20}$: How $1 + 1 = 0$ Improves Information Set Decoding". In: *Advances in Cryptology - EUROCRYPT*. LNCS. Springer, 2012. DOI: 10.1007/978-3-642-29011-4_31.

[BMT78] Elwyn Berlekamp, Robert McEliece, and Henk van Tilborg. "On the inherent intractability of certain coding problems". In: *IEEE Transactions on Information Theory* 24.3 (May 1978), pp. 384–386. DOI: 10.1109/TIT.1978.1055873.

[BSC16] Marco Baldi, Paolo Santini, and Franco Chiaraluce. "Soft McEliece: MDPC code-based McEliece cryptosystems with very compact keys through real-valued intentional errors". In: *IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2016, pp. 795–799. DOI: 10.1109/ISIT.2016.7541408.

[BT02] Dimitri P. Bertsekas and John N. Tsitsiklis. *Introduction to probability*. English. Belmont, Mass.: Athena Scientific, 2002. ISBN: 978-1886529403.

[Bur08] David Burshtein. "On the Error Correction of Regular LDPC Codes Using the Flipping Algorithm". In: *IEEE Transactions on Information Theory* 54.2 (2008), pp. 517–530. DOI: 10.1109/TIT.2007.913261.

[CFRU01] Sae-Young Chung, G David Forney, Thomas J Richardson, and Rüdiger Urbanke. "On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit". In: *IEEE Communications Letters* 5.2 (Feb. 2001), pp. 58–60. ISSN: 1558-2558. DOI: 10.1109/4234.905935.

[Cha17] Julia Chaulet. "Étude de cryptosystèmes à clé publique basés sur les codes MDPC quasi-cycliques". French. PhD thesis. University Pierre et Marie Curie, Mar. 2017. URL: https://tel.archives-ouvertes.fr/tel-01599347.

[Cho16] Tung Chou. "QcBits: Constant-Time Small-Key Code-Based Cryptography". In: *CHES 2016*. Ed. by Benedikt Gierlichs and Axel Y. Poschmann. Vol. 9813. LNCS. Springer, 2016, pp. 280–300. DOI: 10.1007/978-3-662-53140-2_14.

[CP34] Charles J Clopper and Egon S Pearson. "The use of confidence or fiducial limits illustrated in the case of the binomial". In: *Biometrika* 26.4 (Dec. 1934), pp. 404–413. ISSN: 0006-3444. DOI: 10.1093/biomet/26.4.404.

[CS15] Rodolfo Canto Torres and Nicolas Sendrier. "Analysis of Information Set Decoding for a Sub-linear Error Weight". In: *Post-Quantum Cryptography (PQCrypto)*. Ed. by Tsuyoshi Takagi. 2015, pp. 144–161. DOI: 10.1007/978-3-319-29360-8_10,.

[CS16] Julia Chaulet and Nicolas Sendrier. "Worst case QC-MDPC decoder for McEliece cryptosystem". In: *IEEE International Symposium on Information Theory (ISIT)*. IEEE Press, 2016, pp. 1366–1370. DOI: 10.1109/ISIT.2016.7541522.

[CT19] Rodolfo Canto Torres and Jean-Pierre Tillich. "Speeding up decoding a code with a non-trivial automorphism group up to an exponential factor". In: *IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2019, pp. 1927–1931. DOI: 10.1109/ISIT.2019.8849628.

[Den03] Alexander W Dent. "A Designer's Guide to KEMs". In: *9th IMA International Conference on Cryptography and Coding*. Ed. by Kenneth G. Paterson. Vol. 2898. LNCS. Cirencester, UK: Springer, Dec. 2003, pp. 133–151. DOI: 10.1007/978-3-540-40974-8_12.

[DG19] Nir Drucker and Shay Gueron. "A toolbox for software optimization of QC-MDPC code-based cryptosystems". In: *Journal of Cryptographic Engineering (JCEN)* 9.4 (2019), pp. 341–357. DOI: 10.1007/s13389-018-00200-4.

[DGK19] Nir Drucker, Shay Gueron, and Dusan Kostic. *On constant-time QC-MDPC decoding with negligible failure rate*. Cryptology ePrint Archive, Report 2019/1289. 2019. URL: https://eprint.iacr.org/2019/1289.

[DGK20a] Nir Drucker, Shay Gueron, and Dusan Kostic. "Fast Polynomial Inversion for Post Quantum QC-MDPC Cryptography". In: LNCS 12161 (2020). Ed. by Shlomi Dolev, Vladimir Kolesnikov, Sachin Lodha, and Gera Weiss, pp. 110–127. DOI: 10.1007/978-3-030-49785-9_8.

[DGK20b] Nir Drucker, Shay Gueron, and Dusan Kostic. "QC-MDPC Decoders with Several Shades of Gray". In: *Post-Quantum Cryptography (PQCrypto)*. Ed. by Jintai Ding and Jean-Pierre Tillich. Vol. 12100. LNCS. Springer, 2020, pp. 35–50. DOI: 10.1007/978-3-030-44223-1_3.

[DGZ17] Jean-Christophe Deneuville, Philippe Gaborit, and Gilles Zémor. "Ouroboros: A Simple, Secure and Efficient Key Exchange Protocol Based on Coding Theory". In: *Post-Quantum Cryptography (PQCrypto)*. Ed. by Tanja Lange and Tsuyoshi Takagi. Vol. 10346. LNCS. Utrecht, Netherlands: Springer, 2017, pp. 18–34. DOI: 10.1007/978-3-319-59879-6_2.

[Di+02] Changyan Di, David Proietti, I Emre Telatar, Thomas J Richardson, and Rüdiger L Urbanke. "Finite-length analysis of low-density parity-check codes on the binary erasure channel". In: *IEEE Transactions on Information Theory* 48.6 (2002), pp. 1570–1579. DOI: 10.1109/TIT.2002.1003839.

[Dum91] Ilya Dumer. "On minimum distance decoding of linear codes". In: *Fifth Joint Soviet–Swedish International Workshop on Information Theory*. Ed. by Grigori A Kabatianskii. Moscow, Russia, 1991, pp. 50–52.

[FO99] Eiichiro Fujisaki and Tatsuaki Okamoto. "Secure Integration of Asymmetric and Symmetric Encryption Schemes". In: *CRYPTO'99*. Ed. by Michael J. Wiener. Vol. 1666. LNCS. Santa Barbara, CA, USA: Springer, Aug. 1999, pp. 537–554. DOI: 10.1007/3-540-48405-1_34.

[Gal63] Robert G. Gallager. *Low Density Parity Check Codes*. Cambridge, MA, USA: M.I.T. Press, 1963.

[Gar+01] Roberto Garello, Franco Chiaraluce, Paola Pierleoni, Marco Scaloni, and Sergio Benedetto. "On error floor and free distance of turbo codes". In: *IEEE International Conference on Communications (ICC)*. Vol. 1. IEEE, 2001, 45–49 vol.1. DOI: 10.1109/ICC.2001.936270.

[GJS16] Qian Guo, Thomas Johansson, and Paul Stankovski. "A Key Recovery Attack on MDPC with CCA Security Using Decoding Errors". In: *Advances in Cryptology - ASIACRYPT*. Ed. by Jung Hee Cheon and Tsuyoshi Takagi. Vol. 10031. LNCS. 2016, pp. 789–815. ISBN: 978-3-662-53886-9. DOI: 10.1007/978-3-662-53887-6_29.

[HC17] Jingwei Hu and Ray CC Cheung. "Area-time efficient computation of niederreiter encryption on QC-MDPC codes for embedded hardware". In: *IEEE Transactions on Computers* 66.8 (2017), pp. 1313–1325. DOI: 10.1109/TC.2017.2672984.

[HEA01] Xiao-Yu Hu, Evangelos Eleftheriou, and Dieter M Arnold. "Progressive edge-growth Tanner graphs". In: *IEEE Global Communications Conference (GLOBECOM)*. Vol. 2. San Antonio, TX, USA: IEEE, Nov. 2001, 995–1001 vol.2. DOI: 10.1109/GLOCOM.2001.965567.

[HHK17] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. "A modular analysis of the Fujisaki-Okamoto transformation". In: *Theory of Cryptography Conference*. Springer. 2017, pp. 341–371. DOI: 10.1007/978-3-319-70500-2_12.

[HMG13] Stefan Heyse, Ingo von Maurich, and Tim Güneysu. "Smaller Keys for Code-Based Cryptography: QC-MDPC McEliece Implementations on Embedded Devices". In: *International Conference on Cryptographic Hardware and Embedded Systems (CHES)*. Ed. by Guido Bertoni and Jean-Sébastien Coron. Vol. 8086. LNCS. Springer, 2013, pp. 273–292. DOI: 10.1007/978-3-642-40349-1_16.

[HQC] Carlos Aguilar Melchor, Nicolas Aragon, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jurjen Bos, Jean-Christophe Deneuville, Philippe Gaborit, Edoardo Persichetti, Jean-Marc Robert, Pascal Véron, and Gilles Zémor. *HQC*. NIST Round 3 submission for Post-Quantum Cryptography. July 2020. URL: https://pqc-hqc.org/.

[HWCW19] Jingwei Hu, Wen Wang, Ray CC Cheung, and Huaxiong Wang. "Optimized Polynomial Multiplier Over Commutative Rings on FPGAs: A Case Study on BIKE". In: *International Conference on Field-Programmable Technology (ICFPT)*. IEEE. 2019, pp. 231–234. DOI: 10.1109/ICFPT47387.2019.00035.

[Joz01] Richard Jozsa. "Quantum factoring, discrete logarithms, and the hidden subgroup problem". In: *Computing in Science & Engineering* 3.2 (2001), pp. 34–43. DOI: 10.1109/5992.909000.

[KS06] K Murali Krishnan and Priti Shankar. "On the Complexity of finding stopping set size in Tanner Graphs". In: *Conference on Information Sciences and Systems (CISS)*. 2006, pp. 157–158. DOI: 10.1109/CISS.2006.286453.

[LB18] Gianluigi Liva and Hannes Bartz. "Protograph-based Quasi-Cyclic MDPC Codes for McEliece Cryptosystems". In: *IEEE International Symposium on Turbo Codes & Iterative Information Processing (ISTC)*. Hong Kong, China: IEEE, Dec. 2018, pp. 1–5. DOI: 10.1109/ISTC.2018.8625356.

[LB88] Pil J. Lee and Ernest F. Brickell. "An Observation on the Security of McEliece's Public-Key Cryptosystem". In: *Advances in Cryptology - EUROCRYPT*. Vol. 330. LNCS. Springer, 1988, pp. 275–280. DOI: 10.1007/3-540-45961-8_25.

[LDW94] Yuan Xing Li, Robert H. Deng, and Xin Mei Wang. "On the equivalence of McEliece's and Niederreiter's public-key cryptosystems". In: *IEEE Transactions on Information Theory* 40.1 (1994), pp. 271–273. DOI: 10.1109/18.272496.

[LEDA] Marco Baldi, Alessandro Barenghi, Franco Chiaraluce, Gerardo Pelosi, and Paolo Santini. *LEDAcrypt*. Revision 3.0 of the submission to the NIST post-quantum cryptography call. May 2020. URL: https://www.ledacrypt.org/documents/LEDAcrypt_v3.pdf.

[LMSS01] Michael G Luby, Michael Mitzenmacher, Mohammad Amin Shokrollahi, and Daniel A Spielman. "Improved low-density parity-check codes using irregular graphs". In: *IEEE Transactions on Information Theory* 47.2 (Feb. 2001), pp. 585–598. DOI: 10.1109/18.910576.

[Lön+16] Carl Löndahl, Thomas Johansson, Masoumeh Koochak Shooshtari, Mahmoud Ahmadian-Attari, and Mohammad Reza Aref. "Squaring attacks on McEliece public-key cryptosystems using quasi-cyclic codes of even dimension". In: *Designs, Codes and Cryptography* 80.2 (2016), pp. 359–377. DOI: 10.1007/s10623-015-0099-x.

[McE78] Robert J. McEliece. "A Public-Key System Based on Algebraic Coding Theory". In: DSN Progress Report 42-44. Jet Propulsion Lab, 1978, pp. 114–116. URL: https://ipnpr.jpl.nasa.gov/progress_report2/42-44/44N.PDF.

[MG14] Ingo von Maurich and Tim Güneysu. "Towards Side-Channel Resistant Implementations of QC-MDPC McEliece Encryption on Constrained Devices". In: *Post-Quantum Cryptography (PQCrypto)*. Vol. 8772. LNCS. Springer, 2014, pp. 266–282. DOI: 10.1007/978-3-319-11659-4_16.

[MMT11] Alexander May, Alexander Meurer, and Enrico Thomae. "Decoding random linear codes in $O(2^{0.054n})$". In: *Advances in Cryptology - ASIACRYPT*. Ed. by Dong Hoon Lee and Xiaoyun Wang. Vol. 7073. LNCS. Springer, 2011, pp. 107–124. DOI: 10.1007/978-3-642-25385-0_6.

[MN97] David JC MacKay and Radford M Neal. "Near Shannon limit performance of low density parity check codes". In: *Electronics Letters* 33.6 (Mar. 13, 1997), pp. 457–458. ISSN: 0013-5194. DOI: 10.1049/el:19970362.

[MO15] Alexander May and Ilya Ozerov. "On Computing Nearest Neighbors with Applications to Decoding of Binary Linear Codes". In: *Advances in Cryptology - EUROCRYPT*. Ed. by E. Oswald and M. Fischlin. Vol. 9056. LNCS. Springer, 2015, pp. 203–228. DOI: 10.1007/978-3-662-46800-5_9.

[MOG15a] Ingo von Maurich, Tobias Oder, and Tim Güneysu. "Implementing QC-MDPC McEliece Encryption". In: *ACM Transactions on Embedded Computing Systems* 14.3 (Apr. 2015), 44:1–44:27. ISSN: 1539-9087. DOI: 10.1145/2700102.

[MOG15b] Ingo Von Maurich, Tobias Oder, and Tim Güneysu. "Implementing QC-MDPC McEliece encryption". In: *ACM Transactions on Embedded Computing Systems (TECS)* 14.3 (Apr. 2015), pp. 1–27. DOI: 10.1145/2700102.

[MRA00] Chris Monico, Joachim Rosenthal, and Amin A. Shokrollahi. "Using low density parity check codes in the McEliece cryptosystem". In: *IEEE International Symposium on Information Theory (ISIT)*. Sorrento, Italy: IEEE, 2000, p. 215. DOI: 10.1109/ISIT.2000.866513.

[MTSB13] Rafael Misoczki, Jean-Pierre Tillich, Nicolas Sendrier, and Paulo S. L. M. Barreto. "MDPC-McEliece: New McEliece variants from Moderate Density Parity-Check codes". In: *IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2013, pp. 2069–2073. DOI: 10.1109/ISIT.2013.6620590.

[Nie86] Harald Niederreiter. "Knapsack-type cryptosystems and algebraic coding theory". In: *Problems of Control and Information Theory* 15.2 (1986), pp. 159–166. URL: http://real-j.mtak.hu/7997/1/MTA_ProblemsOfControl_15.pdf.

[NM65] John A Nelder and Roger Mead. "A Simplex Method for Function Minimization". In: *The Computer Journal* 7.4 (Jan. 1965), pp. 308–313. DOI: 10.1093/comjnl/7.4.308.

[NTRU] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. "NTRU: A Ring-Based Public Key Cryptosystem". In: *Algorithmic Number Theory Symposium (ANTS)*. Ed. by Joe Buhler. Vol. 1423. LNCS. Portland, OR, USA: Springer, June 1998, pp. 267–288. DOI: 10.1007/BFb0054868.

[NV14] Dung Viet Nguyen and Bane Vasic. "Two-Bit Bit Flipping Algorithms for LDPC Codes and Collective Error Correction". In: *IEEE Transactions on Communications* 62.4 (Apr. 2014), pp. 1153–1163. DOI: 10.1109/TCOMM.2014.021614.130884.

[Pea82] Judea Pearl. "Reverend Bayes on Inference Engines: A Distributed Hierarchical Approach". In: *Proceedings of the Second AAAI Conference on Artificial Intelligence*. AAAI'82. AAAI Press, 1982, pp. 133–136. URL: https://www.aaai.org/Library/AAAI/1982/aaai82-032.php.

[Pra62] Eugene Prange. "The use of information sets in decoding cyclic codes". In: *IRE Transactions on Information Theory* 8.5 (Sept. 1962), pp. 5–9. DOI: 10.1109/TIT.1962.1057777.

[Ric03] Tom Richardson. "Error Floors of LDPC Codes". In: *41st Annual Allerton Conference on Communication, Control, and Computing*. 2003, pp. 1426–1435.

[RMG20] Jan Richter-Brockmann, Johannes Mono, and Tim Güneysu. *Folding BIKE: Scalable Hardware Implementation for Reconfigurable Devices*. Cryptology ePrint Archive, Report 2020/897. 2020. URL: https://eprint.iacr.org/2020/897.

[RSU01] Thomas J Richardson, Mohammad Amin Shokrollahi, and Rüdiger L Urbanke. "Design of capacity-approaching irregular low-density parity-check codes". In: *IEEE Transactions on Information Theory* 47.2 (Feb. 2001), pp. 619–637. DOI: 10.1109/18.910578.

[RU01] Thomas J Richardson and Rüdiger L Urbanke. "The capacity of low-density parity-check codes under message-passing decoding". In: *IEEE Transactions on Information Theory* 47.2 (2001), pp. 599–618. DOI: 10.1109/18.910577.

[SBBC19] Paolo Santini, Massimo Battaglioni, Marco Baldi, and Franco Chiaraluce. "Hard-decision iterative decoding of LDPC codes with bounded error rate". In: *IEEE International Conference on Communications (ICC)*. IEEE, May 2019, pp. 1–6. DOI: 10.1109/ICC.2019.8761536.

[Sch72] J Schalkwijk. "An algorithm for source coding". In: *IEEE Transactions on Information Theory* 18.3 (1972), pp. 395–399. DOI: 10.1109/TIT.1972.1054832.

[Sen11] Nicolas Sendrier. "Decoding One Out of Many". In: *Post-Quantum Cryptography (PQCrypto)*. Vol. 7071. LNCS. 2011, pp. 51–67. DOI: 10.1007/978-3-642-25405-5_4.

[Sho99] Peter W Shor. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer". In: *SIAM Review* 41.2 (Jan. 1999), pp. 303–332. DOI: 10.1137/S0036144598347011.

[Spr79] Melvin Dale. Springer. *The Algebra of random variables*. English. New York: Wiley, 1979. ISBN: 978-0471014065.

[SS96] Michael Sipser and Daniel A. Spielman. "Expander codes". In: *IEEE Transactions on Information Theory* 42.6 (1996), pp. 1710–1722. DOI: 10.1109/18.556667.

[SSH] Tatu Ylönen and Chris Lonvick. "The Secure Shell (SSH) Transport Layer Protocol". In: *RFC* 4253 (2006), pp. 1–32. DOI: 10.17487/RFC4253.

[Ste88] Jacques Stern. "A method for finding codewords of small weight". In: *Coding Theory and Applications*. Ed. by Gérard Cohen and Jacques Wolfmann. Vol. 388. LNCS. Springer, 1988, pp. 106–113. DOI: 10.1007/BFb0019850.

[SV19] Nicolas Sendrier and Valentin Vasseur. "On the Decoding Failure Rate of QC-MDPC Bit-Flipping Decoders". In: *Post-Quantum Cryptography (PQCrypto)*. Ed. by Jintai Ding and Rainer Steinwandt. Vol. 11505. LNCS. Chongqing, China: Springer, May 2019, pp. 404–416. DOI: 10.1007/978-3-030-25510-7_22.

[SV20a] Nicolas Sendrier and Valentin Vasseur. "About Low DFR for QC-MDPC Decoding". In: *Post-Quantum Cryptography (PQCrypto)*. Ed. by Jintai Ding and Jean-Pierre Tillich. Vol. 12100. LNCS. Paris, France: Springer, Apr. 2020, pp. 20–34. DOI: 10.1007/978-3-030-44223-1_2.

[SV20b] Nicolas Sendrier and Valentin Vasseur. *On the existence of weak keys for QCMDPC decoding*. Cryptology ePrint Archive, Report 2020/1232. 2020. URL: https://eprint.iacr.org/2020/1232.

[Tan81] R. Tanner. "A recursive approach to low complexity codes". In: *IEEE Transactions on Information Theory* 27.5 (1981), pp. 533–547. DOI: 10.1109/TIT.1981.1056404.

[Til18a] Jean-Pierre Tillich. "The Decoding Failure Probability of MDPC Codes". In: *IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2018, pp. 941–945. DOI: 10.1109/ISIT.2018.8437843.

[Til18b] Jean-Pierre Tillich. *The decoding failure probability of MDPC codes*. Sept. 2018. URL: https://arxiv.org/abs/1801.04668.

[TLS] Eric Rescorla. "The Transport Layer Security (TLS) Protocol Version 1.3". In: *RFC* 8446 (2018), pp. 1–160. DOI: 10.17487/RFC8446.

[Vas17] Valentin Vasseur. "Cryptographie post-quantique : étude du décodage des codes QC-MDPC". MA thesis. Université Grenoble Alpes, Sept. 2017. URL: https://hal.inria.fr/hal-01664082.

[WKP09] Chih-Chun Wang, Sanjeev R Kulkarni, and H Vincent Poor. "Finding All Small Error-Prone Substructures in LDPC Codes". In: *IEEE Transactions on Information Theory* 55.5 (2009), pp. 1976–1999. DOI: 10.1109/TIT.2009.2015993.