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

Valentin Vasseur

Mater Cybersecurity
Master of Science in Informatics at Grenoble
Master Informatique / Master Mathématiques & Applications

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

Valentin Vasseur

September 2017

Research project performed at Inria Paris

Under the supervision of:
Nicolas Sendrier, Inria Paris

Defended before a jury composed of:
Jean-Guillaume Dumas
Jean-Louis Roch
Nicolas Sendrier
Vanessa Vitse

September 2017

Abstract

In this work, I look at a variant of the McEliece cryptosystem aiming at reducing the key size. This scheme is based on correcting codes and it relies on a simple probabilistic decoding algorithm. However the failures of the decoding algorithm can be used to retrieve the secrete key. During my internship I tried to reduce that decoding failure rate by introducing as little complexity to the algorithm as possible. Here, I present a few variants of the algorithm and look at how they perform on simulations. Finally I try to find models on the evolution of certain quantities during the algorithm. The knowledge of these evolutions could be used to improve the variants of the algorithm.

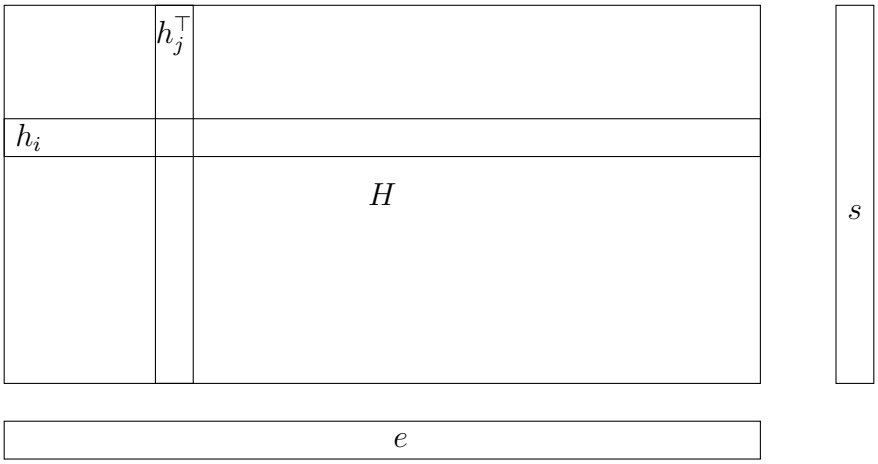
Résumé

Dans ce document, je m'intéresse à une variante du cryptosystème de McEliece visant à réduire la taille des clés. Ce système se base sur la théorie des codes correcteurs et utilise un algorithme de décodage itératif probabiliste simple. Cependant, les échecs au décodage peuvent être utilisés pour récupérer la clé secrète. Pendant mon stage, j'ai essayé de réduire ce taux d'échec au décodage en introduisant aussi peu de compléxité que possible dans l'algorithme. Ici, je présente quelques variantes de cet algorithme et je regarde leurs comportements sur des simulations. Enfin, j'essaie de trouver des modèles sur l'évolution de certaines quantités intervenant dans l'algorithme. La connaissance de ces évolutions pourrait être utilisée pour améliorer les variantes de l'algorithme.

Contents

Nomenclature

In this document, we will focus on the syndrome decoding problem: given $H \in \mathbb{F}_2^{p \times n}$ and $s \in \mathbb{F}_2^p$ we want to find $e \in \mathbb{F}_2^n$ such that $He^\top = s$. We will use the following way of representing the different objects involved.



We will use the variable i to index a row of the matrix H and the variable j a column.

$\mathrm{B}(n, p)$	The binomial distribution with parameters n and p
d	Hamming weight of a column
e	An error vector
G	A generator matrix
H	A parity matrix
h_i	The i-th row of the parity check matrix H
$h_j^\top$	The j-th colum of the parity check matrix H
n	Length of the code (in this document we always have $n = n_0 \cdot p$)
n_0	Number of circulant blocks in the parity check matrix

p Dimension of the code

s A syndrome

w Hamming weight of a row

Proposed parameters in [MTSB13]:

Security	n_0	p	d	t
80	2	4,801	45	84
80	3	3,593	51	53
80	4	3,079	55	42
128	2	9,857	71	134
128	3	7,433	81	85
128	4	6,803	85	68
256	2	32,771	137	264
256	3	22,531	155	167
256	4	20,483	161	137

— 1 —

Context

Most of the cryptosystems used nowadays rely on problems such as the integer factorisation or the discrete logarithm. Those problems could be solved in a polynomial time with a sufficiently powerful quantum computer using Shor's algorithm [Sho97] or a variant [PZ03].

Even if the current state of quantum computing does not allow an attacker to break widespread algorithms such as RSA, DH, ECDH or ECDSA, the threat is real if we want long term secrecy. In response to that possible threat, the NIST [1] will start a process to standardise quantum-resistant cryptographic algorithms (or post-quantum algorithms) in November 2017.

In this document, I will present a post-quantum cryptosystem using error correcting codes (also known as code-based cryptography), a variant of the McEliece cryptosystem [McE78] which uses QC-MDPC codes [MTSB13]. I will specifically focus on the decoding algorithm of those codes for which there are security concerns [GJS16].

1. National Institute of Standards and Technology

— 2 —

Preliminaries

We are interested in decoding a type of linear error correcting code used in a variant of the McEliece cryptosystem. In this chapter we shall therefore recall the definitions and main results about those objects.

2.1 McEliece cryptosystem

Definition 2.1. An $[n, k]$-linear *code* $\mathcal{C}$ is a linear subspace of $\mathbb{F}_q^n$ of dimension k.

We say that such a code is of *length* n and *dimension* k. If $q = 2$ we say that $\mathcal{C}$ is a *binary* code.

The vectors of $\mathcal{C}$ are called *codewords*.

Definition 2.2. A *generator matrix* G of a code $\mathcal{C} \subset \mathbb{F}_q^n$ is a matrix whose rows form a basis of the linear subspace $\mathcal{C}$.

Any vector $v \in \mathbb{F}_q^k$ can be mapped to a codeword $c \in \mathcal{C}$ with:

$$c = vG\,.$$

We say that we *encode* the message v as the codeword c.

A *parity check* H matrix is a matrix such that $c \in \mathcal{C}$ if and only if $Hc^\top = 0$.

For any vector $y \in \mathbb{F}_q^n$, the vector $Hy^\top$ is called the *syndrome* of y.

Definition 2.3. The *Hamming weight* of a vector $v \in \mathbb{F}_q^n$ is the number of its non-zero components:

$$\mathsf{wt}(v) = |\{i \mid v_i \neq 0\}|\,.$$

The *Hamming distance* between two vectors $v, w \in \mathbb{F}_q^n$ is the Hamming weight of their difference:

$$\mathsf{d}(v, w) = \mathsf{wt}(w - v) = |\{i \mid v_i \neq w_i\}|\,.$$

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

$$\Delta(\mathcal{C}) = \min_{\substack{c_1, c_2 \in \mathcal{C} \\ c_1 \neq c_2}} \mathsf{d}(c_1, c_2)\,.$$

Intuitively, correcting codes add redundancy in a message so that the noise which is added during transmission can be removed. A noisy codeword will be at a certain distance of a codeword. If that distance is small enough, we can find the codeword which hopefully corresponds to the transmited message.

Definition 2.4. Let $\mathcal{C}$ be an $[n, k]$-linear code of miminum distance d, $c \in \mathcal{C}$ be a codeword and $e \in \mathbb{F}_q^n$ be an error pattern of weight less than $\frac{d-1}{2}$. We write $v = c + e$.

We say that we *decode* the message v if we find $c \in \mathcal{C}$ knowing v.

If the code $\mathcal{C}$ has a minimum distance of d, we can detect up to $d - 1$ errors and, in theory, correct up to $\left\lfloor \frac{d-1}{2} \right\rfloor$ errors.

In practice, for the decoding process to be fast, we add a structure to the code $\mathcal{C}$. For example, Reed-Solomon codes are based on polynomials over finite fields and decoding can be achieved using a variant of the extended Euclidean algorithm (among other algorithms).

In this document we will talk about MDPC [1] codes which are derived from the LDPC [2] codes. They both use a probabilistic decoding algorithm which requires a sparse parity check matrix.

Definition 2.5. The McEliece cryptosystem uses the family of the Goppa codes [3]. The main idea behind the system is that we can give a way to encode a mesage (the public key $\hat{G}$) without revealing the decoder (the private key (S, G, P)).

$\mathcal{C} \leftarrow \text{Goppa}(n, k, t)$ G generator matrix for $\mathcal{C}$ S random $k \times k$ invertible matrix P random $n \times n$ permutation matrix	$\hat{G} = SGP$ $\longrightarrow$	
		$m \in \{0,1\}^k$ $c' = m\hat{G}$ $e \leftarrow \{0,1\}^n$ $\lvert e \rvert = t$
$\hat{c} = c'P^{-1}$ $\hat{m} = \text{Decode}(\hat{c})$ $m = \hat{m}S^{-1}$	$\longleftarrow$ $c = c' + e$	

Here we denote by $\text{Goppa}(n, k, t)$ the family of Goppa codes of length n, dimension k and capable of correcting t errors.

The security of the McEliece cryptosystem relies on the difficulty of decoding a random linear code. The best algorithms to solve this problem are all variations of the Prange algorithm [Pra62].

1. Moderate Density Parity Check
2. Low Density Parity Check
3. The definition of a Goppa code is out of the scope of this document.

2.2 MDPC-McEliece

Goppa codes are interesting in such a scheme because they provide a high error correction capacity and they have a parity check matrix which is hardly distinguishable from a random matrix. But their main issue is the key size which prevents some lightweight or embedded usages. In [Aug+15], the proposed parameters for 128 bits of post-quantum security for the McEliece cryptosystem are $n = 6,960$, $k = 5,413$, $t = 119$, this gives a public key size of $k \times (n - k) = 8,373,911$ bits.

Definition 2.6. As their names suggest, LDPC [4] and MDPC [5] codes have respectively a parity check matrix of low and moderate density. Here the density of the matrix is defined by the weight of its rows.

For LDPC, rows have a constant weight (usually less than 10). For MDPC they have a weight which scales in $O(\sqrt{n})$ where n is the length of the code.

LDPC codes are widely used in telecommunications. A cryptographic scheme has been proposed using LDPC codes [BCGM07] in a McEliece variant where they replace the Goppa codes. However a weakness was later found [BC07] which could break the system. The proposed fix used matrices with a specific structure which has been broken in [OTD10]. Finally in [BBC08] a more general structure was proposed fixing these issues. As in the McEliece cryptosystem, (S, G, P) would be the private key, and the public key would be $\hat{G} = SGP$ with G the generator matrix of an LDPC code. In [BBC08], P has to be an invertible matrix of moderate weight.

In [MTSB13], authors took a different approach. They directly generate a parity check matrix of moderate density. Such a modification circumvents all the previous attacks on the LDPC-McEliece system. This generates codes with a worse decoding capability but it is still good enough to use the usual decoding algorithms. Note that this system does not require the matrices P and S as in the original McEliece system.

Definition 2.7. The MDPC-McEliece cryptosystem works as follows.

H random $k \times n$ matrix with rows of weight w $G = (I_k \vert \tilde{G})$ a generator matrix corresponding to H	$\xrightarrow{\quad G \quad}$	
		$m \in \{0,1\}^{n-k}$ $c' = mG$ $e \leftarrow \{0,1\}^n$ $\vert e \vert = t$
$m = \text{Decode}(\hat{c})$	$\xleftarrow{\quad c = c' + e \quad}$	

4. Low Density Parity Check
5. Moderate Density Parity Check

As for the original McEliece system, for 128 bits of post-quantum security, MDPC-McEliece would require huge key size. But it becomes interesting if we use a quasi-cyclic version of the MDPC codes.

Definition 2.8. A *circulant* matrix is a matrix such that it is uniquely determined by its first row. Indeed each row is the cyclic permutation with an offset of one of the row just above it.

$$\begin{pmatrix} m_0 & m_1 & \cdots & m_{n-1} & m_n \\ m_n & m_0 & m_1 & \cdots & m_{n-1} \\ & \ddots & \ddots & \ddots & \\ m_2 & \cdots & m_n & m_0 & m_1 \\ m_1 & m_2 & \cdots & m_n & m_0 \end{pmatrix}$$

The set of circulant matrices of size $n \times n$ over a field $\mathbb{K}$ is isomorphic to $\mathbb{K}[X]/(X^n - 1)$.

Definition 2.9. A QC-MDPC [6] code is a code whose parity check matrix H consists in n_0 blocks which are $p \times p$ circulant matrices where p is a prime number. Those blocks have the same row weight and column weight d. The row weight of the parity check matrix is therefore $w = n_0 d$.

$$H = \left(\begin{array}{c|c|c} H_0 \circlearrowright & \cdots & H_{n_0} \circlearrowright \end{array} \right)$$

As for circulant matrices, quasi-cyclic matrices are entirely described by their first row. For the QC-MDPC-McEliece cryptosystem, this yields public keys of size $32,771$ bits for 128 bits of post-quantum security.

The security of the system relies on the two following problems:

— decoding t errors in a quasi-cyclic $[n, n-p]$ linear code;

— deciding whether the code spanned by some block circulant $p \times n$ matrices has a minimum distance less than w.

Both problems are NP-hard in the non quasi-cyclic case. They are conjectured hard in the quasi-cyclic case and we can choose the parameters n, p, w, t such that it takes at least 2^κ operations to solve them for κ-bit security.

6. Quasi Cyclic Moderate Density Parity Check

2.3 Decoding algorithms for the MDPC codes

Decoding from a random linear code is usually hard. For MDPC (or LDPC), there are efficient algorithms, they use the fact that the parity check matrix is sparse.

Those decoders can be split into two categories:

— hard decision algorithms: they operate on data consisting of 0 and 1;

— soft decision algorithms: they can use all the values in-between.

Soft decision algorithms have better correction capabilities but are more complex to implement. Hard decision algorithms are better suited for an embedded implementation at the expense of the correction capability.

In the next following sections, we will detail a hard decision algorithm — the bitflipping algorithm — and a soft decision algorithm — the belief propagation algorithm.

For convenience, we will sometimes consider a vector $v \in \mathbb{F}_2^n$ as the set of indices of its nonzero values:

$$v = \{i \in \{1, \dots, n\} \mid v_i \neq 0\} .$$

We will frequently need to use the cardinality of the intersection of two vectors. For this, we will use different equivalent points of view:

$$|h \cap v| = \langle h, v \rangle_{\mathbb{Z}} = \sum_k h_k v_k .$$

We will call

— m the transmitted message,

— $c = mG$ the corresponding codeword,

— e the error pattern,

— y the received noisy codeword ($y = c \oplus e$).

We recall that the syndrome $s = Hy^\top$ is zero if and only if y is a codeword. So we have

$$s = Hy^\top = Hc^\top \oplus He^\top = He^\top = \bigoplus_{j \in e} h_j^\top .$$

We will say that the elements in e are erroneous positions or errors. The goal of the decoding process is finding the vector e.

For any $i \in \{1, \dots, p\}$ we say that the i-th equation is satisfied if $|h_i \cap y|$ (or equivalently $|h_i \cap e|$) is even and unsatisfied if it is odd. The equation can be written as:

$$\sum_{i=0}^{n} h_{ji} e_i = 0 \mod 2 .$$

We have achieved the decoding process when we have a vector y such that its syndrome is zero *i.e.* when all the equations are satisfied.

2.3.1 Bit flipping

The bitflipping algorithm was first proposed by Gallager in [Gal63] to decode LDPC codes. It relies on the fact that each bit of the syndrome tells us if the corresponding equation is satisfied or not. Thus if a position is involved in many unsatisfied equations, it is probably an error.

The algorithm relies on the computation, for each position j, of a counter with the formula $\sigma_j = \langle h_j^\top, s \rangle_{\mathbb{Z}}$. This counter is the number of unsatisfied equations involving the position j.

The algorithm is iterative and on each iteration it will compute all the σ_j values and flip the positions for which the counter is over a certain threshold. When we have flipped all the errors, y is a codeword and so its syndrome is zero.

Algorithm 1 The bitflipping algorithm

procedure BITFLIPPING(y, H) ▷ $y = (y_1, \ldots, y_n) \in \{0,1\}^n$
  while $Hy^\intercal \neq 0$ **do** ▷ $H = (h_1, \ldots, h_n) \in \{0,1\}^{k \times n}$
    $s \leftarrow Hy^\intercal$
    for $j = 1, \ldots, n$ **do**
      if $\sigma_j = \langle s, h_j^\top \rangle_{\mathbb{Z}} \geq \text{threshold}$ **then**
        $y_j \leftarrow 1 - y_j$
  return y

It is really simple but the hard part is choosing a threshold. In the LDPC case, the column weight is as big as a few units but for MDPC it can be as big as 161 and a bad choice of threshold can make the decoding process fail (this will further be discussed in chapter 4).

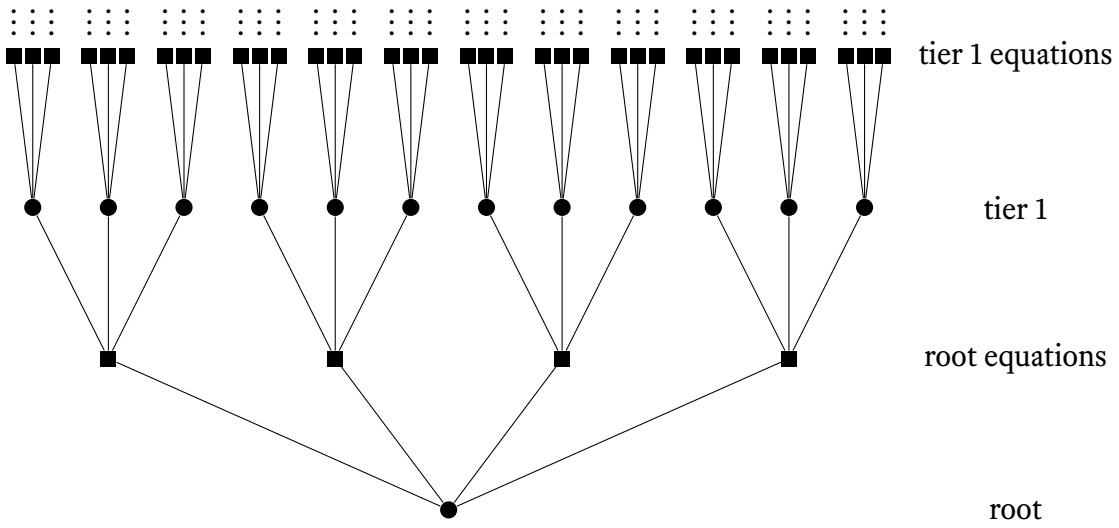
In the case of MDPC, decoding is usually achieved in less than ten iterations.

2.3.2 Belief propagation

In [Gal63], the author describes another algorithm which he calls "probabilistic decoding", it is now more commonly referred to as a belief propagation algorithm or sum-product algorithm.

The basic idea behind this algorithm is that we want to get a good estimation of the probabilities $P(c_j = 1 \mid y)$ and $P(c_j = 0 \mid y)$ where y is the received vector (a noisy codeword) and c is the sent codeword. The estimation of these probabilities is made iteratively.

First we compute the conditional probabilities for the each equation to be satisfied or not knowing the components of y affected by this equation. Then we compute the conditional probabilities for the components of y knowing the probabilities of the equations they are involved in. We then iterate the process. This can be seen as a probability tree.



Here we have represented by a circle the positions of the vector to decode and by a square the parity check equations. This tree corresponds to a parity check matrix where every row and every column have a fixed weight of four.

Given this tree where the root is j, we will call $T_j^{(I)}$ the subtree going up to the I-th tier.

We will assume that the values $c_1, \ldots, c_n$ are all independent and that the tree has no loop.

In our case, we know $P(c_j = 0 \mid T_j^{(0)})$ and $P(c_j = 1 \mid T_j^{(0)})$ from the parameters of the cryptosystem, they are respectively equal to $\frac{n-t}{n}$ and $\frac{t}{n}$.

To compute $P(c_j = 0 \mid T_j^{(I+1)})$ and $P(c_j = 1 \mid T_j^{(I+1)})$ when $I \geq 0$, we proceed iteratively and each iteration consists in two steps that we will have to repeat for all the positions j.

First step.

$$\begin{cases} P\left(s_i \middle| c_j = 0, T_j^{(I+1)}\right) \\ \qquad = \sum_{\{c_{j'} | j' \in h_i \setminus \{j\}\}} P\left(s_i | c_j = 0, \{c_{j'} \mid j' \in h_i \setminus \{j\}\}\right) \prod_{j' \in h_i \setminus \{j\}} P\left(c_{j'} \middle| T_{j'}^{(I)}\right) \\ P\left(s_i \middle| c_j = 1, T_j^{(I+1)}\right) \\ \qquad = \sum_{\{c_{j'} | j' \in h_i \setminus \{j\}\}} P\left(s_i | c_j = 1, \{c_{j'} \mid j' \in h_i \setminus \{j\}\}\right) \prod_{j' \in h_i \setminus \{j\}} P\left(c_{j'} \middle| T_{j'}^{(I)}\right) \end{cases}$$

Second step.

$$\begin{cases} P\left(c_j = 0 \middle| T_j^{(I+1)}\right) = K \cdot P\left(c_j = 0\right) \prod\limits_{i' \in h_j^\top \setminus \{i\}} P\left(s_{i'} \middle| c_j = 0, T_{j'}^{(I+1)}\right) \\ P\left(c_j = 1 \middle| T_j^{(I+1)}\right) = K \cdot P\left(c_j = 1\right) \prod\limits_{i' \in h_j^\top \setminus \{i\}} P\left(s_{i'} \middle| c_j = 1, T_{j'}^{(I+1)}\right) \end{cases}$$

It is not necessary to compute the factor K explicitely, we can use the fact that

$$P\left(c_j = 1 \middle| T_j^{(I+1)}\right) + P\left(c_j = 0 \middle| T_j^{(I+1)}\right) = 1\,.$$

The algorithm will first compute the probabilities concerning the equations which will then be used to compute probabilites concerning the positions. This is called a *message passing* algorithm.

Details on how to these computations can be done in practice are given in [Mac99].

Since we are computing probabilities, this algorithm requires the use of floating-point or fixed-point values. It usually needs many iterations to achieve decoding.

2.4 A key recovery attack using decoding failures

In [GJS16], the authors describe a way to recover the private key in the QC-MDPC scheme.

They call *spectrum* the set of all the existing distances between two ones in a vector. The decoding algorithm will have a slightly smaller probability to fail when the error vector's spectrum and the private key's spectrum have a nonempty intersection. Thus if we provide enough generated messages to a decoder which signals decoding failures, we are able to recover the private key's spectrum by looking at the distances in the error vectors triggering the least failures.

Then they provide a reconstruction algorithm to recover the private key from the spectrum.

Authors compare two cases: the CPA case where the attacker can build the error vector and the CCA case where the error vector is truly random. They claim that full recovery of the key can be done in 240 million calls to the decoder for the CPA case and 356 million for the CCA case using the 80-bit security parameters.

In this attack, the decoding algorithm used by the authors has the same fixed threshold for all the iterations. In [MTSB13], better approaches were suggested knowing that using a fixed threshold does not give the best error-decoding capability.

— 3 —

Objectives

The main goal of my internship at Inria Paris was improving the bitflipping algorithm used with QC-MDPC codes.

The need for an improvement in the decoding algorithm comes from the fact that it currently fails with a small probability.

This probability can actually be measured if we run many simulations. If we want to use this scheme it is important to limit those failures. It is also important to have a better understanding of the way it works if we want to implement it efficiently.

Also, in response to the attack using the decoding failures, an obvious countermeasure would be diminishing the failures until the attack is not practical.

During my internship I first tried to get a better understanding of the different decoding algorithms for MDPC and LDPC codes by searching and understanding the work already done on them. I tried to introduce some ideas coming from the soft decoding algorithms into the bitflipping algorithm. I then implemented them and ran simulations to see how much of an improvement could be expected from each technique.

Finally I tried to have a more theoretical point of view in order to understand where the improvements come from.

— 4 —

State of the art

If we look at variants of the bitflipping algorithm for LDPC codes, it is often proposed to flip all the positions reaching the maximum counter value. For LDPC codes, the counters range over a small interval (typically $[0; 3]$) but for MDPC codes that interval is bigger. Such a choice would thus require many iterations since not many positions would be flipped per iteration.

4.1 Algorithm originally proposed

In the original paper [MTSB13], the proposed algorithm suggests subtracting a small value Δ (usually $\Delta = 5$) to the maximum counter value and taking that value as the threshold.

In [Cho16] the choice of thresholds was made by a fixed table giving a good threshold value on average for each iteration step.

Both techniques are not always satisfying. In figure 4.1a, the distributions of the counters are shown in different color depending on the fact that they correspond to erroneous positions ($\mathcal{H}_1$) or not ($\mathcal{H}_0$). A good choice for the threshold is the one corresponding to the red dashed area: it maximises the difference between the number of errors removed and the number of errors added.

If we choose the algorithm from [MTSB13], with a non negligible probability, the maximum counter value will not give the best threshold. If that maximum is too high the choice for the threshold will be high as well. We might do an iteration in which we just flip a few errors hence not being as efficient as we could be. On the other hand if the maximum is too low we will be under the good threshold and it will flip many goood positions thus adding many errors. If we add too many errors we might get a decoding failure.

A first improvement would be using what was proposed in [Cho16]: choose the best threshold for each iteration based on an average value determined with a lot of simulations.

In figure 4.1a we have the average distribution for the counter values. In figure 4.1b we have the average distribution for the counter values when then syndrome weight is higher than its average value (in this case $|s| = 2,100$). Compared to the other distributions we can see that everything is shifted to the right. Thus in this case the good threshold based on the average distribution (28) will be too low and will probably lead to a failure.

The next section presents the works from [Cha17], in which the influence of the syndrome weight on those distributions was determined. It allowed a great improvement in the decoding failure rate of the algorithm.

4.2 Improvements using the syndrome weight

Chaulet showed that the distribution of the counters can be well approximated by random variables following binomial distributions. The probabilities involved have been conditioned by the syndrome weight to give a more precise approximation and it has been used to choose better threshold values.

We will use the following values to get a better estimation of the distributions for a specific iteration.

Definition 4.1. We call E_l the number of equations affected by exactly l errors. It is the number of rows for which the intersection with the vector e has a cardinal of exactly l.

$$E_l = |\{i \in \{1, \dots, p\} \mid |h_i \cap e| = l\}|$$

Definition 4.2. For a position j, we define σ_j as the *counter* value of j that is:

$$\sigma_j = |h_j^\top \cap s| = |\{i \in \{1, \dots, p\} | j \in h_i, |h_i \cap e| \text{ is odd}\}| \,.$$

Definition 4.3. For a position j, we write $\mathcal{H}_0$ the event $\{j \notin e\}$ and $\mathcal{H}_1$ the event $\{j \in e\}$.

We can see σ_j as a random variable for which the distribution differs whether we are under hypothesis $\mathcal{H}_0$ or $\mathcal{H}_1$. Experimentally the following assumption can be made.

Proposition 4.4. *Under hypothesis $\mathcal{H}_0$:*

$$\sigma_j \sim \mathcal{B}(d, p_0) \,.$$

Under hypothesis $\mathcal{H}_1$:

$$\sigma_j \sim \mathcal{B}(d, p_1) \,.$$

Where B indicates the binomial distribution. The value of a counter can be seen as repeating d times the independent identically distributed Bernoulli trials consisting in determining if $|\{h_i \cap e\}|$ is odd for all $i \in h_j^\top$.

The probabilities p_0 and p_1 can be calculated on average.

Proposition 4.5. *In proposition 4.4, we can take:*

$$p_0 = P(|\{h_i \cap e\}| \textit{is odd} | \mathcal{H}_0) = \sum_{\substack{l=0 \\ l \textit{ odd}}}^{t} \frac{\binom{w-1}{l}\binom{n-w}{t-l}}{\binom{n-1}{t}} ;$$

$$p_1 = P(|\{h_i \cap e\}| \textit{is odd} | \mathcal{H}_1) = \sum_{\substack{l=0 \\ l \textit{ even}}}^{t} \frac{\binom{w-1}{l}\binom{n-w}{t-1-l}}{\binom{n-1}{t-1}} .$$

This proposition supposes that the error vector is randomly taken out of the vectors of length n and weight t.

The next proposition allows a better knowledge of the values E_l for all l, which we will use to give better estimates for the probabilities p_0 and p_1 on a specific instance.

Proposition 4.6. *We have the following equations involving E_l:*

$$\sum_{l=0}^{w} E_l = p \, ; \tag{4.1}$$

$$\sum_{\substack{l=0 \\ l \textit{ odd}}}^{w} E_l = |s| \, ; \tag{4.2}$$

$$\sum_{j=1}^{n} \sigma_j = w|s| \, ; \tag{4.3}$$

$$\sum_{j \in e} \sigma_j = \sum_{\substack{l=0 \\ l \textit{ odd}}}^{w} l E_l \, ; \tag{4.4}$$

$$\sum_{j \in e} \sigma_j = |s| + \sum_{\substack{l=0 \\ l \textit{ odd}}}^{w} (l-1) E_l \, ; \tag{4.5}$$

$$\sum_{j \notin e} \sigma_j = (w-1)|s| - \sum_{\substack{l=0 \\ l \textit{ odd}}}^{w} (l-1) E_l \, . \tag{4.6}$$

$$\tag{4.7}$$

Proof. For any row $i \in \{1, \ldots, p\}$, $|h_i \cap e|$ is unique. This implies (4.1).

For any $i \in \{1, \ldots, p\}$, $s_i = 1$ if and only if $|h_i \cap e|$ is odd, implying (4.2).

We can prove (4.3) by using the definition of σ_j. We will use the fact that for any i and j, we have $i \in h_j^\top \iff j \in h_i$. Equivalently, using indicator functions, for any

$i, j, \mathbf{1}_{h_i}(j) = \mathbf{1}_{h_j^\top}(i)$.

$$\begin{aligned}\sum_{j=1}^{n} \sigma_j &= \sum_{j=1}^{n} |h_j^\top \cap s| = \sum_{j=1}^{n}\sum_{i=1}^{p} \mathbf{1}_{h_j^\top \cap s}(i) \\ &= \sum_{j=1}^{n}\sum_{i=1}^{p} \mathbf{1}_{h_j^\top}(i)\mathbf{1}_s(i) = \sum_{j=1}^{n}\sum_{i=1}^{p} \mathbf{1}_{h_i}(j)\mathbf{1}_s(i) \\ &= \sum_{i=1}^{p} \mathbf{1}_s(i) \left(\sum_{j=1}^{n} \mathbf{1}_{h_i}(j) \right) .\end{aligned}$$

We can then use the fact that each row of H has a fixed weight.

Similarly we can prove (4.4):

$$\begin{aligned}\sum_{j\in e} \sigma_j &= \sum_{j\in e} |h_j^\top \cap s| = \sum_{j=1}^{n}\sum_{i=1}^{p} \mathbf{1}_{h_j^\top \cap s}(i)\mathbf{1}_e(j) \\ &= \sum_{j=1}^{n}\sum_{i=1}^{p} \mathbf{1}_{h_j^\top}(i)\mathbf{1}_s(i)\mathbf{1}_e(j) = \sum_{j=1}^{n}\sum_{i=1}^{p} \mathbf{1}_{h_i \cap e}(j)\mathbf{1}_s(i) \\ &= \sum_{\substack{i=1 \\ i \text{ odd}}}^{p}\sum_{j=1}^{n} \mathbf{1}_{h_i \cap e}(j) = \sum_{\substack{i=1 \\ i \text{ odd}}}^{p} lE_l \, .\end{aligned}$$

The remaining formulas are clearly implied by the others. □

During the decoding, for each iteration, we need to recompute the syndrome to check if the decoding is done. Its weight can be used to condition the probabilities involved in the distributions of the counters. To do so, we use the results of proposition 4.6.

Proposition 4.7. *In proposition 4.4, we can take:*

$$p_0 = P(|\{h_i \cap e\}| \textit{is odd} | \mathcal{H}_0, |s|) = \frac{1}{d(n-t)} \left((w-1)|s| - \sum_{\substack{l=0 \\ l \textit{ odd}}}^{w} (l-1)E_l \right) ;$$

$$p_1 = P(|\{h_i \cap e\}| \textit{is odd} | \mathcal{H}_1, |s|) = \frac{1}{dt} \left(|s| + \sum_{\substack{l=0 \\ l \textit{ odd}}}^{w} (l-1)E_l \right) .$$

In the bitflipping algorithm, the goal is to flip as many erroneous position and as little good positions given a counter value. To express this in terms of probabilities, we want to find a threshold T such that

$$P(j \in e | \sigma_j \geq T) > P(j \notin e | \sigma_j \geq T) \, .$$

Using the Bayes' theorem this becomes

$$P(\sigma_j \geq T | j \in e)P(j \in e) > P(\sigma_j \geq T | j \notin e)P(j \notin e)\,.$$

And if we use the fact that we have binomial distributions this becomes

$$\left(\frac{p_1}{p_0}\frac{1-p_0}{1-p_1}\right)^T > \frac{n-t}{t}\left(\frac{1-p_0}{1-p_1}\right)^d\,.$$

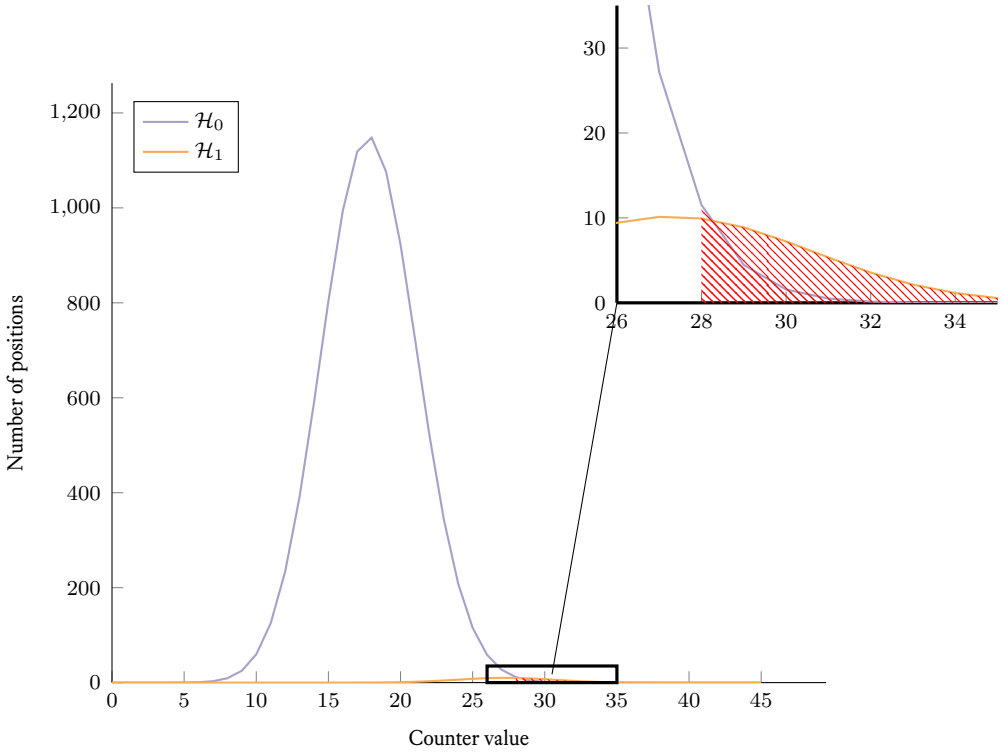
A good threshold can therefore easily be found as long as we can estimate t, p_0 and p_1.

The first value can be estimated by using the fact that the number of errors is correlated to the syndrome weight.

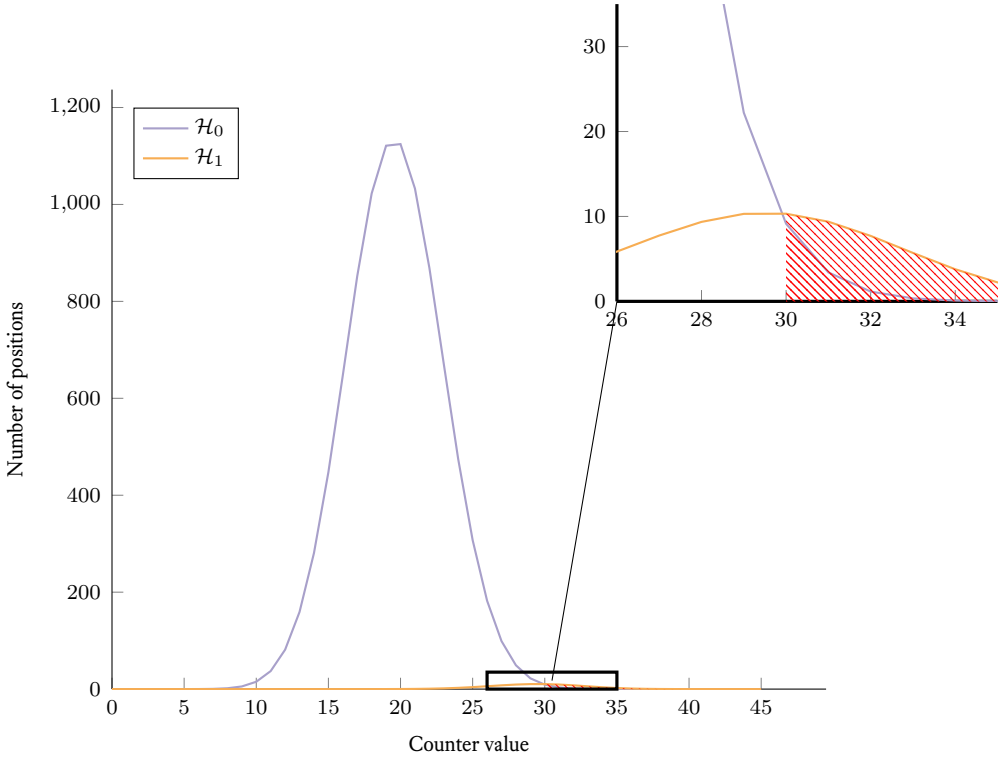
The other two involve the value t, the syndrome weight $|s|$ which we can compute and the value $\sum_{\substack{l=0 \\ l \text{ odd}}}^{w}(l-1)E_l$. The values of E_l becomes really small for $l \geq 4$ (see figure 4.2). So far, a choice has been made to use an average value in the computation since the values do not vary too much. The following formula can be used to determine the average values. They suppose that the error pattern e is randomly chosen among the vectors of length n and weight t. This is true for the first iteration but because of the way the algorithm works, it will not be true for the other iterations. This is discussed in section 5.2.1.

Proposition 4.8. *On the first iteration, for any $l \in \{0, \ldots, w\}$, we have:*

$$\mathbb{E}[E_l] = p\frac{\binom{w}{l}\binom{n-w}{t-l}}{\binom{n}{t}}\,.$$



(a) Distribution of counter values on average



(b) Distribution of counter values when $|s| = 2,100$

Figure 4.1 – Distribution of counter values separated whether the position is erroneous or not

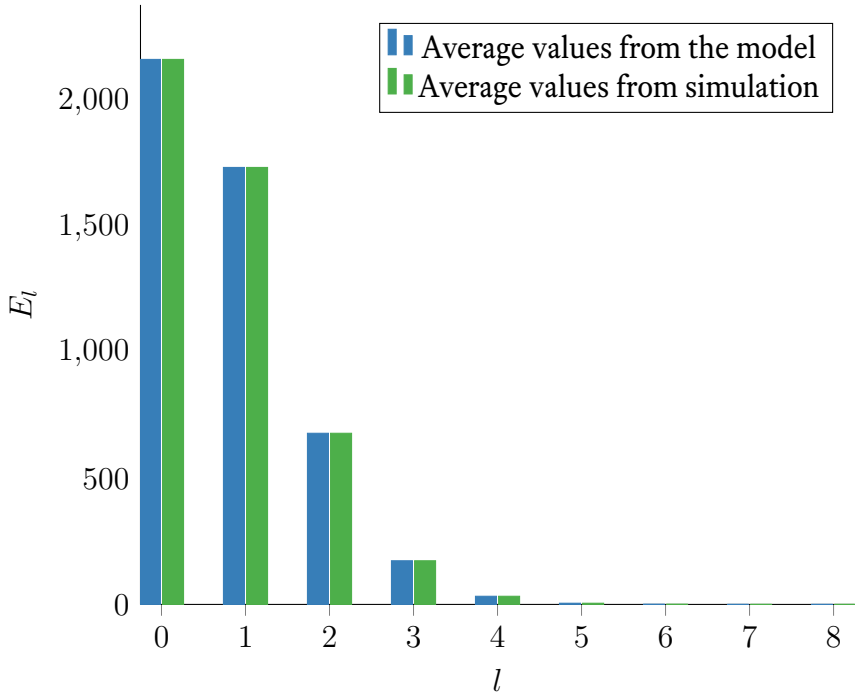


Figure 4.2 – Distribution of E_l

— 5 —
Decoding QC-MDPC codes

5.1 Empirical improvements

In this section I will present variants of the bitflipping algorithm that aim at reducing the decoding failure rate while keeping the simplicity of the algorithm. Results in terms of decoding failure rate are given in section 5.1.4 (page 23).

5.1.1 Dynamic decoding

In the bitflipping algorithm, the dominating operations are the computation of the counters and the computation of the syndrome. The computation of the syndrome is done at the beginning of the decoding process and then it is updated depending on the flipped positions. It is the multiplications of the sparse quasi-cyclic parity check matrix by a vector of length n.

During an iteration we need to compute the counters $\langle h_j^\top, s\rangle_{\mathbb{Z}}$ for all $j \in \{1, \ldots, n\}$ or equivalently $s^\top H$ in $\mathbb{Z}$. It is the multiplication of the sparse quasi-cyclic parity check matrix by a dense vector of length p. Here we try to limit the number of counter computations thus computing a smaller amount of dot product rather than the whole matrix-vector multiplication.

We do so by observing that any one in the syndrome corresponds to an odd number of errors among the w positions involved in the parity equation. So we know that for an unsatisfied equation, at least one of its w positions is an error. Knowing that, we can simplify the algorithm.

We select the w positions corresponding to an unsatisfied equation and then we only compute the corresponding counters. As with the classic bitflipping we will flip a position if its counters is over a certain threshold.

This algorithm has the advantage of being really fast and really simple when we only have a few errors to decode. However it greatly increases the decoding failure rate.

Two issues can increase the complexity of this algorithm. First we can have good positions with a high counter value, if we flip them we will increase the number of errors

Algorithm 2 The dynamic bitflipping algorithm

procedure DYNAMIC BITFLIPPING(y, H) ▷ $y = (y_1, \ldots, y_n) \in \{0,1\}^n$
  while $Hy^\intercal \neq 0$ **do** ▷ $H = (h_1, \ldots, h_n) \in \{0,1\}^{k \times n}$
    $s \leftarrow Hy^\intercal$
    for $i = 1, \ldots, p$ **do**
      if $s_i \neq 0$ **then**
        for $j \in h_i$ **do**
          if $\sigma_j = \langle s, h_j^\top \rangle_{\mathbb{Z}} \geq \text{threshold}$ **then**
            $y_j \leftarrow 1 - y_j$
  return y

(as with the classic bitflipping). Second we can have no counter over the threshold in a given unsatisfied equation. This means we would have computed w counters but not found any error.

5.1.2 Grey zones

We know the distribution of the counters corresponding to a good position and the distribution of the counters corresponding to an erroneous position (see figure 4.1). In the classic bitflipping algorithm, we use the fact that a high counter is more likely to correspond to an error so we flip that position. Yet some positions had a high counter but not high enough to be flipped, they still are more likely to correspond to an error. We will refer to these positions with a "high but not high enough" counter as "grey", and the set of grey positions as the "grey zone".

The main idea behind this improvement of the algorithm is to keep track of those grey positions to save some computation time. As with the dynamic decoding, we try to limit the number of counter computations. For a few grey iterations we will focus only on the grey zone and only compute the counters of the grey positions. Those lighter iterations will require only a few hundreds counter computations rather than thousands for a full iteration.

The basic principle is as follows:

— we compute all the counters,
— we flip the positions whose counter is higher than a specified threshold,
— we select the positions for which the counter was above a threshold (lower than the precedent one),
— we do regular iterations restricted to those selected positions until nothing gets changed or until we reach I_G iterations,
— repeat all these steps until the word is decoded.

In the simulations, empirical values were chosen to select only a few hundreds positions and I_G was set to 10.

From simulation on a high number of instances, we can see in section 5.1.4 that this

Algorithm 3 The grey zone bitflipping algorithm

procedure GREY BITFLIPPING(y, H) ▷ $y = (y_1, \ldots, y_n) \in \{0,1\}^n$
  $I \leftarrow 0$ ▷ $H = (h_1, \ldots, h_n) \in \{0,1\}^{k \times n}$
  $G \leftarrow \emptyset$
  while $Hy^\intercal \neq 0$ **do**
    $s \leftarrow Hy^\intercal$
    if I is a multiple of I_G **then**
      $G \leftarrow \emptyset$
      for $j \in \{0, \ldots, n\}$ **do**
        if $\sigma_j = \langle s, h_j^\top \rangle_{\mathbb{Z}} \geq \text{threshold}_G$ **then**
          $G \leftarrow G \cup \{j\}$
    for $j \in G$ **do**
      if $\sigma_j = \langle s, h_j^\top \rangle_{\mathbb{Z}} \geq \text{threshold}$ **then**
        $y_j \leftarrow 1 - y_j$
    $I \leftarrow I + 1$
  return y

algorithm does not only reduce the complexity but it also decreases the decoding failure rate.

Most of the choices in dimensioning the constants were done empirically. There are some ways to improve the algorithm by choosing better thresholds (the one deciding on which position to flip and the one deciding on which positions get selected in the grey zone). In section 5.2.3, we propose a way to make a better choice for the threshold used during the grey iterations.

5.1.3 Multi-bitflipping

With the grey zones, what we do in a way is adding some information on each position telling that it is a bit more likely to be erroneous and therefore we focus on those marked positions for a few iterations. We can see some similarities between this idea and what was done in [NV14] for LDPC codes. In this paper, the authors define variants of the bitflipping algorithm by giving each component of the vector to decode an additional bit of information about its "strength". These algorithms greatly improve the decoding failure rate over the original bitflipping algorithm, and in comparison to the belief propagation algorithms, they have a lower complexity but a higher decoding failure rate.

For each bit of the vector to decode, they add a bit which tells if the bit is "weak" or "strong". A weak bit will be easier to flip than a strong one.

The generic structure is described in algorithm 4. Here the additional information is given by the vector z containing $(b-1)$-bit unsigned integers. A higher value corresponds to a weaker position. First all the values are set as strong. Then this information as well as the counter will be used to choose the evolution of y and z. This function will be

explicited below.

Algorithm 4 The b-bitflipping algorithm

procedure b-BITFLIPPING(y, H) ▷ $y = (y_1, \dots, y_n) \in \{0, 2^b - 1\}^n$
  $z \leftarrow (0, \dots, 0)$ ▷ $H = (h_1, \dots, h_n) \in \{0, 1\}^{k \times n}$
  while $Hy^\intercal \neq 0$ **do**
    $s \leftarrow Hy^\intercal$
    for $j = 1, \dots, n$ **do**
      $\sigma_j \leftarrow \langle s, h_j^\top \rangle_{\mathbb{Z}}$
      $(y_j, z_j) \leftarrow f(y_j, z_j, \sigma_j)$
  return y

In the paper, they proposed a 2-bit algorithm which is really specific to LDPC codes with a column weight of 3 so the function $f\colon \{0,1\}^2 \times [0,3]$ can easily be described by a 4×4 table. In the paper, authors also proposed to try every possible function f and keep the ones performing the best. They reduced the sample size using graph isomorphisms. This is not something we can do since the graph reduction leaves too many possibilities to try given the high degree of each node for such graphs with MDPC codes.

I tried to adapt their algorithm to the case of MDPC codes. I made the following choices for the function f, which permits the use of an arbitrary precision b as long as we can find a good function g described below.

$$f(y, z, \sigma) = \begin{cases} (1 - y, (2^b - 1)[-](z + g(\sigma)) & \text{if } z + g(\sigma) \geq 2^{b-1} \\ (y, z[+]g(\sigma)) & \text{otherwise} \end{cases}.$$

Here the symbols $[-]$ and $[+]$ indicate saturating operations for integers of precision $(b-1)$.

$$x[+]y = \begin{cases} 0 & \text{if } x + y < 0 \\ 2^{b-1} - 1 & \text{if } x + y \geq 2^{b-1} - 1 \\ x + y & \text{otherwise} \end{cases} \qquad x[-]y = \begin{cases} 0 & \text{if } x - y < 0 \\ 2^{b-1} - 1 & \text{if } x - y \geq 2^{b-1} - 1 \\ x - y & \text{otherwise} \end{cases}$$

With this choice, a higher strength (lower values of z) will be harder to flip and a lower strength (higher values of z) will be easier.

We now have to find a good function g. I chose to rely on the function

$$\phi\colon \sigma_j \mapsto \log\left(\frac{P(j \in e \mid \sigma_j)}{P(j \notin e \mid \sigma_j)}\right).$$

We already know how to estimate the conditional probabilities (see section 4.2). And in fact the threshold used in the state of the art decoder is the smallest value for which this function is positive. This function is also affine which facilitates computations.

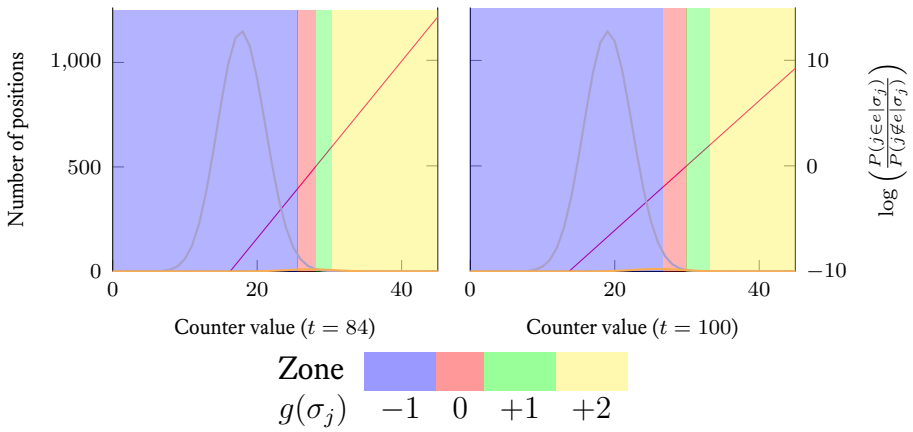


Figure 5.1

I chose the function g as a discretisation of ϕ. For example in figure 5.1, g is defined as:

$$g(\sigma_j) = \begin{cases} -1 & \text{if } \phi(\sigma_j) \in (-\infty, -2.4) \\ 0 & \text{if } \phi(\sigma_j) \in [-2.4, 0) \\ 1 & \text{if } \phi(\sigma_j) \in [0, 2.4) \\ 2 & \text{if } \phi(\sigma_j) \in [2.4, +\infty) \end{cases}$$

In this figure, two graphs are given: one corresponds to a number of errors of 84 the other one with 100 errors. Intuitively, the choice of the function g has been made so that when errors increases, the most uncertain zones — the ones around the threshold — are bigger since the slope of ϕ will decrease.

Using the same empirical choices, I also implemented a 3-bit variant whose results are given in figure 5.2. This algorithm is further discussed in section 5.2.3.

This algorithm can greatly decrease the decoding failure rate. However, it necessitates more memory. If we use b bits of precision, the memory necessary for the values of the n positions will have to be multiplied by b.

5.1.4 Results

To compare the results of the different algorithms, we use the decoding failure rate: the number of decoding failures among a sample of instances scaled to the size of the sample. If we use the state of the art for decoding, the decoding failure rate is already low enough to be hard to measure on simulations (it is about 10^{-10}). So, to compare those different algorithms, I have taken the 80-bit security parameters ($n = 9,602$, $p = 4,801$, $t = 84$) but I have increased the number of errors t. In figure 5.2, we can see the decoding failure rates in function of the number of errors with a logarithmic scale on the y-axis.

The limit curve for the multi-bitflipping algorithm is calculated by using 64-bit precision floating-point numbers and using the transition probabilities detailed in section 5.2.2

with parameters that we could not obtain in a normal decoding process since they require to know the error vector. Its only purpose is to give an idea of the best that can be achieved by keeping the same algorithm structure.

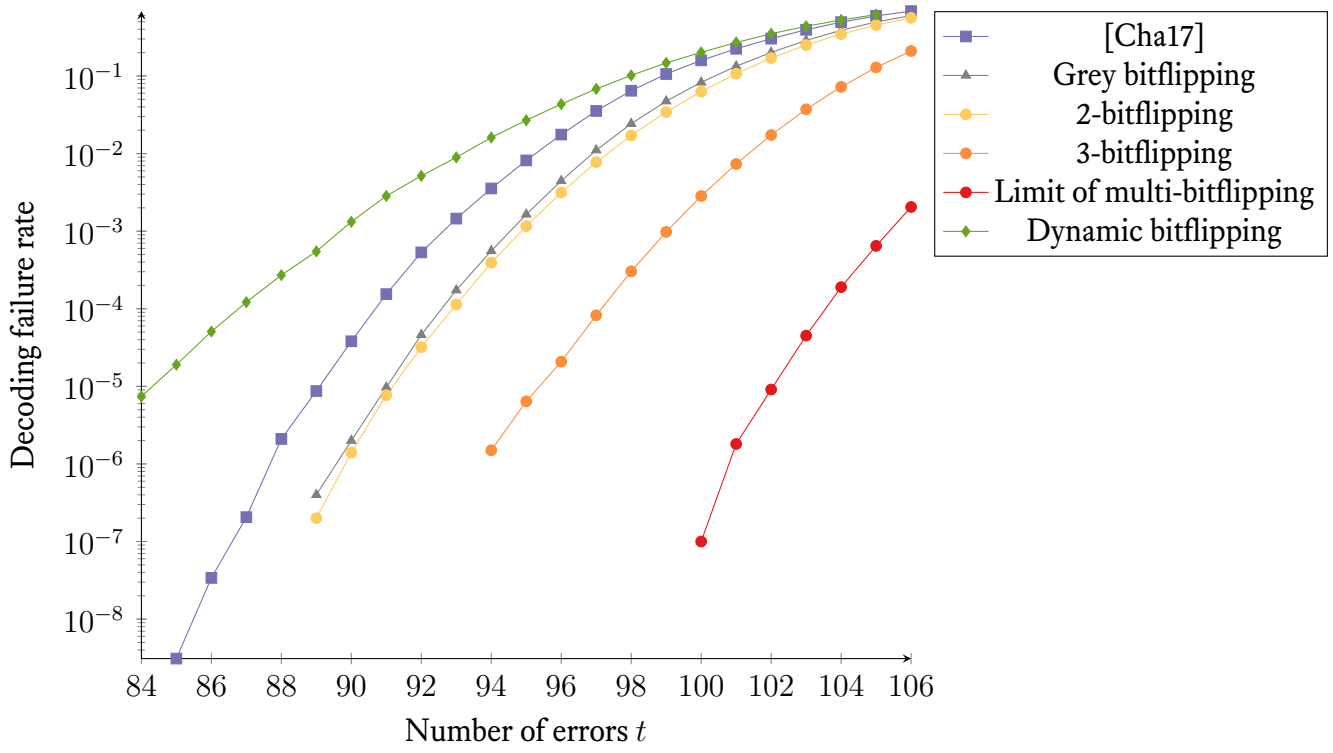


Figure 5.2 – Decoding failure rate of the different variants of the bitflipping algorithm for oversized error weights

5.2 Theoretical analysis

When dealing with probabilistic decoding algorithms, we often make assumptions of independence between the bits of the vector to decode or the bits of the syndrome. In this section we will make such assumptions, find models and compare them to average values concerning the evolution of different quantities between the two consecutive iterations.

5.2.1 Evolution of the syndrome during the decoding

In proposition 4.8, we gave a way to estimate the average values of E_l for any $l \in \{0, \dots, w\}$ on the first iteration. This estimation does not work anymore in the next iterations (see figure 5.3).

In this section, we will look at the changes between the first and the second iteration in terms of the the number of erroneous bits affecting each equation.

Recall that when decoding at the beginning of the first iteration, we have a syndrome s and we want to find e such that $s = He^\top$ knowing H. After a first iteration, some positions will be flipped which will have the effect of correcting as well as mistakenly adding some errors. We denote by f the set of positions which were flipped and thus, on the second iteration, we will have to decode

$$s' = He'^\top = He^\top \oplus Hf^\top$$

with $e' = e \oplus f$.

Since f represents the positions that were flipped between the first and the second iteration, in the algorithm described before, we have

$$f = \{j \mid \sigma_j \geq T\}$$

where T is the chosen threshold.

We remind that for each row $i \in \{1, \ldots, p\}$, the corresponding syndrome bit s_i is the parity of the number of bits in e and in h_i:

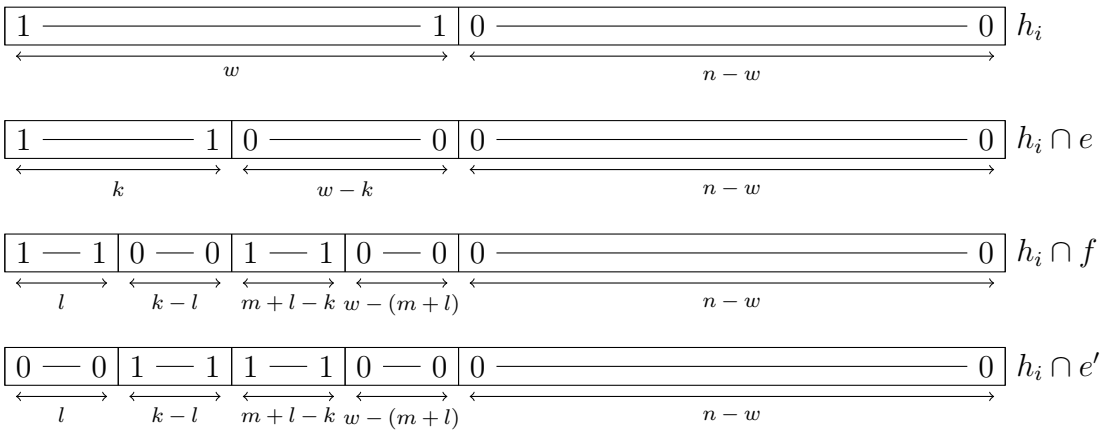
$$s_i = |h_i \cap e| \mod 2\,.$$

We would like to determine the transition probabilities from $|h_i \cap e|$ to $|h_i \cap e'|$.

The bitflipping algorithm works by counting the number of unsatisfied equations (nonzero syndrome bits) involving a certain position, hence the nonzero syndrome bits will have a different (higher most likely) probability to be changed than the zero syndrome bits.

We therefore have to condition probabilities with the fact that the parity equation considered is satisfied or not.

To explain what is happening to the syndrome bits between two iterations, we can reorder the columns of the parity check matrix to obtain the situation in the following diagram. We know that the rows of H all have a fixed weight w by construction. We consider a specific row h_i. We suppose that the cardinality of the intersection between h_i and the vector e is k. Among those k bits, we suppose that l will be flipped and among the $(w-k)$ zero bits $m+l-k$ will be flipped thus giving a total of m nonzero bits in the end.



Proposition 5.1. *We make the hypothesis that for the k nonzero bits in $h_i \cap e$ the fact that they will be flipped (thus decreasing the value of $|h_i \cap e|$) or not are independent identically distributed Bernoulli trials.*

Therefore

$$\begin{cases} |h_i \cap e \cap f| \sim B(k, \pi_{11}) & \textit{when } h_i \cap e \textit{ is odd}; \\ |h_i \cap e \cap f| \sim B(k, \pi_{10}) & \textit{when } h_i \cap e \textit{ is even}. \end{cases}$$

The same hypothesis can be made for the $w-k$ other bits in h_i, that is to say the nonzero bits in $h_i \cap \bar{e}$ (the bits increasing the value of $|h_i \cap e|$):

$$\begin{cases} |h_i \cap \bar{e} \cap f| \sim B(w-k, \pi_{01}) & \textit{when } h_i \cap e \textit{ is odd}; \\ |h_i \cap \bar{e} \cap f| \sim B(w-k, \pi_{00}) & \textit{when } h_i \cap e \textit{ is even}. \end{cases}$$

The probabilities involved can be linked to other quantities of the algorithm.

$$\pi_{11} \triangleq P\left(j \in f | j \in h_i \cap e, |h_i \cap e| \textit{ odd}\right) = \frac{\sum_{j \in e \cap f} \sigma_j}{\sum_{l \textit{ odd}} l E_l}$$

$$\pi_{01} \triangleq P\left(j \in f | j \in h_i \cap e, |h_i \cap e| \textit{ even}\right) = \frac{\sum_{j \in e \cap f} d - \sigma_j}{\sum_{l \textit{ even}} l E_l}$$

$$\pi_{10} \triangleq P\left(j \in f | j \in h_i \cap \bar{e}, |h_i \cap e| \textit{ odd}\right) = \frac{\sum_{j \in \bar{e} \cap f} \sigma_j}{\sum_{l \textit{ odd}} (w-l) E_l}$$

$$\pi_{00} \triangleq P\left(j \in f | j \in h_i \cap \bar{e}, |h_i \cap e| \textit{ even}\right) = \frac{\sum_{j \in \bar{e} \cap f} d - \sigma_j}{\sum_{l \textit{ even}} (w-l) E_l}$$

Proof. We assumed that we have independent identically distributed Bernoulli trials, so we have binomial distributions when we sum them. The probabilities are then determ-

ined by using the expected values:

$$\pi_{11} = \frac{\sum_{\substack{i \\ |h_i \cap e| \text{ odd}}} |h_i \cap e \cap f|}{\sum_{\substack{i \\ |h_i \cap e| \text{ odd}}} |h_i \cap e|}\,; \qquad \pi_{10} = \frac{\sum_{\substack{i \\ |h_i \cap \bar{e}| \text{ odd}}} |h_i \cap \bar{e} \cap f|}{\sum_{\substack{i \\ |h_i \cap \bar{e}| \text{ odd}}} |h_i \cap \bar{e}|}\,;$$
$$\pi_{01} = \frac{\sum_{\substack{i \\ |h_i \cap e| \text{ even}}} |h_i \cap e \cap f|}{\sum_{\substack{i \\ |h_i \cap e| \text{ even}}} |h_i \cap e|}\,; \qquad \pi_{00} = \frac{\sum_{\substack{i \\ |h_i \cap \bar{e}| \text{ even}}} |h_i \cap \bar{e} \cap f|}{\sum_{\substack{i \\ |h_i \cap \bar{e}| \text{ even}}} |h_i \cap \bar{e}|}\,.$$

We first compute the denominators. By definition of E_l, we have

$$\sum_{\substack{i \\ |h_i \cap e| \text{ odd}}} |h_i \cap e| = \sum_{l \text{ odd}} l E_l\,.$$

We also have

$$\sum_{\substack{i \\ |h_i \cap \bar{e}| \text{ odd}}} |h_i \cap \bar{e}| = \sum_{\substack{i \\ |h_i \cap e| \text{ odd}}} |h_i| - |h_i \cap e| = \sum_{l \text{ odd}} E_l \cdot (w - l)\,.$$

Obviously we have the corresponding formulas for even values.
We now look at the numerators.

$$\begin{aligned}\sum_{\substack{i \\ |h_i \cap e| \text{ odd}}} |h_i \cap e \cap f| &= \sum_{\substack{i \\ |h_i \cap e| \text{ odd}}} \sum_j \mathbf{1}_{h_i \cap e \cap f}(j) = \sum_{\substack{i \\ |h_i \cap e| \text{ odd}}} \sum_j \mathbf{1}_{h_i}(j) \mathbf{1}_{e \cap f}(j) \\ &= \sum_{j \in e \cap f} \sum_{\substack{i \\ |h_i \cap e| \text{ odd}}} \mathbf{1}_{h_j^\top}(i) = \sum_{j \in e \cap f} \sum_{\substack{i \\ s_i = 1}} \mathbf{1}_{h_j^\top}(i) \\ &= \sum_{j \in e \cap f} |h_j^\top \cap s| = \sum_{j \in e \cap f} \sigma_j\end{aligned}$$

And for even values we have

$$\begin{aligned}\sum_{\substack{i \\ |h_i \cap e| \text{ even}}} |h_i \cap e \cap f| &= \sum_{j \in e \cap f} \sum_{\substack{i \\ s_i = 0}} \mathbf{1}_{h_j^\top}(i) = \sum_{j \in e \cap f} |h_j^\top \cap \bar{s}| \\ &= \sum_{j \in e \cap f} |h_j^\top| - |h_j^\top \cap s| = \sum_{j \in e \cap f} d - \sigma_j\,.\end{aligned}$$

The other formulas follow immediately. □

We now know the probability that, for any k, $|h_i \cap e| = k$ gets a decrease of l and an increase of $m+l-k$ thus we find the transition probabilities for a counter to go from the value k to m by summing for l ranging from 0 to k. Once more, we use the independence hypotheses we made:

$$P\left(|h_i \cap e'| = m \mid |h_i \cap e| = k \text{ odd}\right) = \sum_{l=0}^{k} \binom{k}{l} \pi_{11}^{l} (1-\pi_{11})^{k-l} \binom{w-k}{m+l-k} \pi_{10}^{m+l-k} (1-\pi_{10})^{w-(m+l)};$$

$$P\left(|h_i \cap e'| = m \mid |h_i \cap e| = k \text{ even}\right) = \sum_{l=0}^{k} \binom{k}{l} \pi_{01}^{l} (1-\pi_{01})^{k-l} \binom{w-k}{m+l-k} \pi_{00}^{m+l-k} (1-\pi_{00})^{w-(m+l)}.$$

Since we already know the distribution of the values E_l for the first iteration (proposition 4.8), we can now determine the distribution for the second iteration.

In figure 5.3 we compare the average values of E_l at the second iteration with the values given by the formula in proposition 4.8 and the ones using the transition probabilities we just determined. The probabilities π_{ij} for $i, j \in \{0, 1\}$ were determined by running many simulations. Since we generated the error patterns in the simulations, we know the positions in $e \cap f$ and $\bar{e} \cap f$, the counters for those positions, and the values of E_l for all l.

The formula of proposition 4.8 is a little bit off since it does not take into account that the algorithm tends to decrease the syndrome weight and the error pattern is not random anymore. The bitflipping algorithm tends to decrease the E_l values for l odd and increase the E_l values for l even.

With the newly determined model, we can see that compared to the average values, we tend to slightly underestimate the distributions.

5.2.2 Evolution of the counters

The counters can be studied in a similar way to the syndrome bits.

We use the same notation, we denote by f the set of positions which were flipped and thus, on the second iteration, we will have to decode

$$s' = He'^{\top} = He^{\top} \oplus Hf^{\top} = s \oplus \Delta$$

with $e' = e \oplus f$ and $\Delta = \bigoplus_{j \in f} h_j^{\top}$.

Recall that for a position j, the value of its counter is:

$$\sigma_j = |h_j^{\top} \cap s|.$$

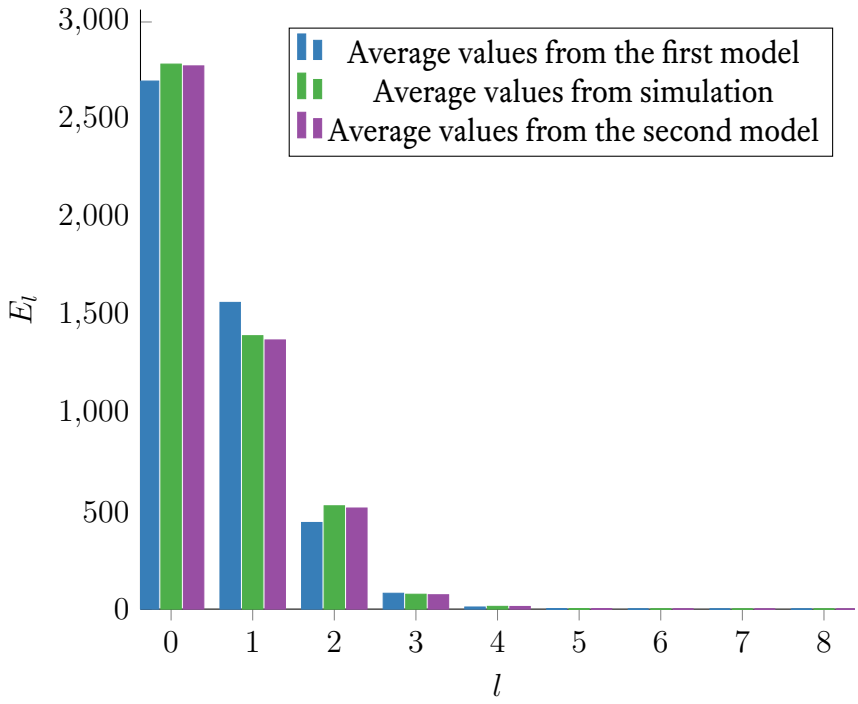


Figure 5.3 – Distribution of E_l on the second iteration ($\pi_{00} = 0.00054$, $\pi_{01} = 0.31088$, $\pi_{10} = 0.00152$, $\pi_{11} = 0.42507$)

So, for the second iteration, it will be

$$\begin{aligned}
\sigma'_j &= |h_j^\top \cap s'| \\
&= |h_j^\top \cap (s \oplus \Delta) \\
&= |(h_j^\top \cap s) \oplus (h_j^\top \cap \Delta)| \\
&= |(h_j^\top \cap s) \oplus (h_j^\top \cap \Delta \cap s) \oplus (h_j^\top \cap \Delta \cap \bar{s})| \\
&= |h_j^\top \cap s| - |h_j^\top \cap \Delta \cap s| + |h_j^\top \cap \Delta \cap \bar{s}| \,.
\end{aligned}$$

We now define $\Delta^+ \triangleq \Delta \cap \bar{s}$ and $\Delta^- \triangleq \Delta \cap s$ since they are the part of Δ responsible respectively for an increase and a decrease in the counter values. We also write

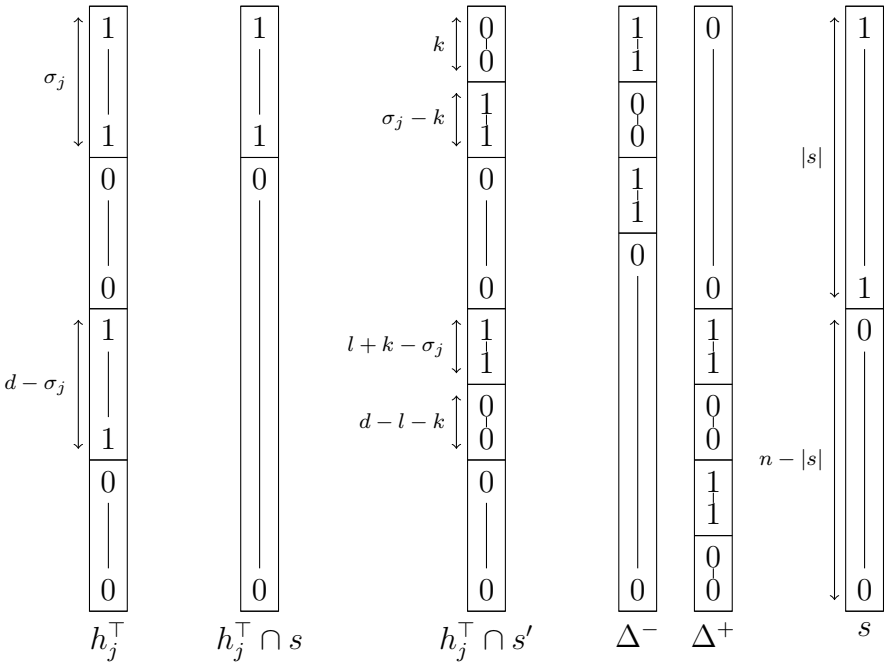
$$\sigma_j^+ \triangleq |h_j^\top \cap \Delta^+| \qquad \text{and} \qquad \sigma_j^- \triangleq |h_j^\top \cap \Delta^-| \,.$$

We now have $\sigma'_j = \sigma_j - \sigma_j^- + \sigma_j^+$.

In the previous section, we reordered the columns of the matrix H to get a better understanding of the effect of the algorithm on the error pattern after the flips. We do the same thing with the rows.

We suppose that a position j has a counter value σ_j. We suppose that among the σ_j bits intersecting with the syndrome, k will be flipped and thus decreasing the counter

value. We also suppose that among the syndrome bits which were previously zero but are then flipped, $l + k - \sigma_j$ are intersecting with $h_j^\top$ and thus increasing the counter value. In the end the counter value for position j will be $\sigma_j - k + l + k - \sigma_j = l$.



Four things can happen between two iterations:
— an erroneous position has been flipped;
— a good position has been flipped;
— an erroneous position has not been flipped;
— a good position has not been flipped.

We therefore choose to condition all the probabilities according to these four situations *i.e.* for a position j whether $j \in e$ or $j \notin e$ and $j \in f$ or $j \notin f$.

Proposition 5.2. *We make the hypothesis for all the σ_j nonzero bits in $h_j^\top \cap s$ the fact that they will be flipped or not are independent identically distributed Bernoulli trials. Therefore*

$$\begin{cases} \sigma_j^- \sim B(\sigma_j, p_{11}^-) & \textit{when } j \in e \cap f; \\ \sigma_j^- \sim B(\sigma_j, p_{10}^-) & \textit{when } j \in e \cap \bar{f}; \\ \sigma_j^- \sim B(\sigma_j, p_{01}^-) & \textit{when } j \in \bar{e} \cap f; \\ \sigma_j^- \sim B(\sigma_j, p_{00}^-) & \textit{when } j \in \bar{e} \cap \bar{f}. \end{cases}$$

With the following probabilities.

$$\begin{cases} p_{11}^{+} \triangleq P(j \in \Delta | j \in h_j^\top \cap \bar{s}, j \in e \cap f) = \dfrac{\sum_{j \in e \cap f} \sigma_j^+}{\sum_{j \in e \cap f} d - \sigma_j} \\ p_{10}^{+} \triangleq P(j \in \Delta | j \in h_j^\top \cap \bar{s}, j \in e \cap \bar{f}) = \dfrac{\sum_{j \in e \cap \bar{f}} \sigma_j^+}{\sum_{j \in e \cap \bar{f}} d - \sigma_j} \\ p_{01}^{+} \triangleq P(j \in \Delta | j \in h_j^\top \cap \bar{s}, j \in \bar{e} \cap f) = \dfrac{\sum_{j \in \bar{e} \cap f} \sigma_j^+}{\sum_{j \in \bar{e} \cap f} d - \sigma_j} \\ p_{00}^{+} \triangleq P(j \in \Delta | j \in h_j^\top \cap \bar{s}, j \in \bar{e} \cap \bar{f}) = \dfrac{\sum_{j \in \bar{e} \cap \bar{f}} \sigma_j^+}{\sum_{j \in \bar{e} \cap \bar{f}} d - \sigma_j} \end{cases}$$

The same hypothesis can be made for the $d - \sigma_j$ other bits in $h_j^\top \cap \bar{s}$:

$$\begin{cases} \sigma_j^+ \sim \mathcal{B}(d - \sigma_j, p_{11}^+) & \textit{when } j \in e \cap f; \\ \sigma_j^+ \sim \mathcal{B}(d - \sigma_j, p_{10}^+) & \textit{when } j \in e \cap \bar{f}; \\ \sigma_j^+ \sim \mathcal{B}(d - \sigma_j, p_{01}^+) & \textit{when } j \in \bar{e} \cap f; \\ \sigma_j^+ \sim \mathcal{B}(d - \sigma_j, p_{00}^+) & \textit{when } j \in \bar{e} \cap \bar{f}. \end{cases}$$

With the following probabilities.

$$\begin{cases} p_{11}^{-} \triangleq P(j \in \Delta | j \in h_j^\top \cap s, j \in e \cap f) = \dfrac{\sum_{j \in e \cap f} \sigma_j^-}{\sum_{j \in e \cap f} \sigma_j} \\ p_{10}^{-} \triangleq P(j \in \Delta | j \in h_j^\top \cap s, j \in e \cap \bar{f}) = \dfrac{\sum_{j \in e \cap \bar{f}} \sigma_j^-}{\sum_{j \in e \cap \bar{f}} \sigma_j} \\ p_{01}^{-} \triangleq P(j \in \Delta | j \in h_j^\top \cap s, j \in \bar{e} \cap f) = \dfrac{\sum_{j \in \bar{e} \cap f} \sigma_j^-}{\sum_{j \in \bar{e} \cap f} \sigma_j} \\ p_{00}^{-} \triangleq P(j \in \Delta | j \in h_j^\top \cap s, j \in \bar{e} \cap \bar{f}) = \dfrac{\sum_{j \in \bar{e} \cap \bar{f}} \sigma_j^-}{\sum_{j \in \bar{e} \cap \bar{f}} \sigma_j} \end{cases}$$

Proof. As in the previous section, the assumption that we sum independent identically distributed Bernoulli trials gives binomial distributions.

Then the probabilities are determined by using the expected values. □

We can now determine the transition probabilities between σ_j and σ'_j for any j.

$$P(\sigma'_j = l | \sigma_j, j \in \bar{e} \cap \bar{f}) = \sum_{k=0}^{\sigma_j} \binom{\sigma_j}{k} (p_{00}^-)^k (1 - p_{00}^-)^{\sigma_j - k} \binom{d - \sigma_j}{l + k - \sigma_j} (p_{00}^+)^{l+k-\sigma_j} (1 - p_{00}^+)^{d-l-k}$$

$$P(\sigma'_j = l|\sigma_j, j \in \bar{e} \cap f) = \sum_{k=0}^{\sigma_j} \binom{\sigma_j}{k} (p_{01}^-)^k (1-p_{01}^-)^{\sigma_j - k} \binom{d-\sigma_j}{l+k-\sigma_j} (p_{01}^+)^{l+k-\sigma_j} (1-p_{01}^+)^{d-l-k}$$

$$P(\sigma'_j = l|\sigma_j, j \in e \cap \bar{f}) = \sum_{k=0}^{\sigma_j} \binom{\sigma_j}{k} (p_{10}^-)^k (1-p_{10}^-)^{\sigma_j - k} \binom{d-\sigma_j}{l+k-\sigma_j} (p_{10}^+)^{l+k-\sigma_j} (1-p_{10}^+)^{d-l-k}$$

$$P(\sigma'_j = l|\sigma_j, j \in e \cap f) = \sum_{k=0}^{\sigma_j} \binom{\sigma_j}{k} (p_{11}^-)^k (1-p_{11}^-)^{\sigma_j - k} \binom{d-\sigma_j}{l+k-\sigma_j} (p_{11}^+)^{l+k-\sigma_j} (1-p_{11}^+)^{d-l-k}$$

When decoding, we are interested in the posterior probabilities. Those can be determined using Bayes'theorem.

$$\begin{aligned} P(j \in e|\sigma_j, \sigma'_j, j \in f) &= P(\sigma'_j|\sigma_j, j \in e \cap f) \frac{P(j \in e|\sigma_j, j \in f)}{P(\sigma'_j|\sigma_j, j \in f)} \\ &= P(\sigma'_j|\sigma_j, j \in e \cap f) \frac{P(j \in e|\sigma_j)}{P(\sigma'_j|\sigma_j, j \in f)} \end{aligned}$$

The same formulas are still true if we change e for $\bar{e}$ or f for $\bar{f}$.

Note that in the bitflipping algorithm, we are interested in knowing when the probability of having an error is greater than the probability of not having an error. We can therefore take a look at the values taken by

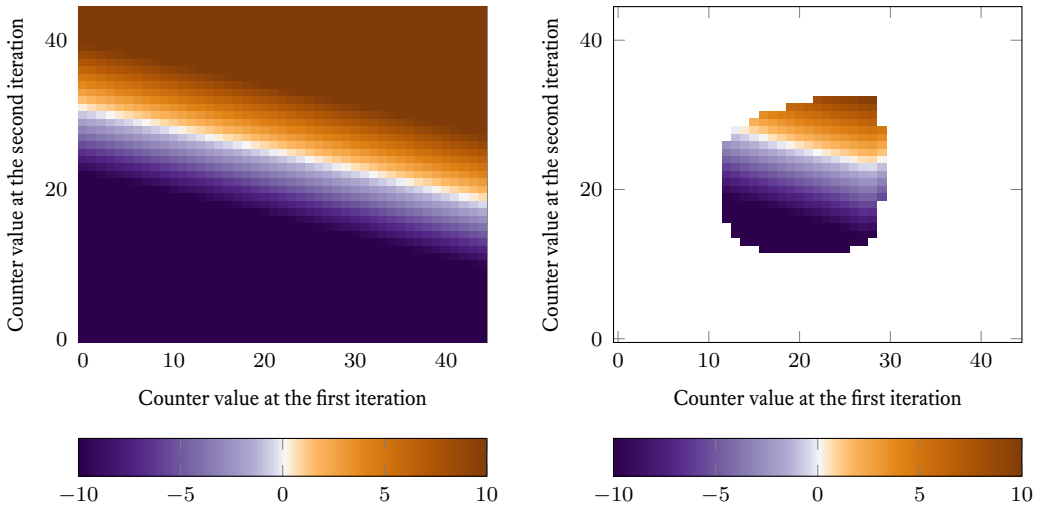
$$\frac{P(j \in e|\sigma_j, \sigma'_j, j \in f)}{P(j \notin e|\sigma_j, \sigma'_j, j \in f)} \quad \text{and} \quad \frac{P(j \in e|\sigma_j, \sigma'_j, j \notin f)}{P(j \notin e|\sigma_j, \sigma'_j, j \notin f)}.$$

Using the previous equations, these become:

$$\frac{P(\sigma'_j|\sigma_j, j \in e \cap f) P(j \in e|\sigma_j)}{P(\sigma'_j|\sigma_j, j \in \bar{e} \cap f) P(j \notin e|\sigma_j)} \quad \text{and} \quad \frac{P(\sigma'_j|\sigma_j, j \in e \cap \bar{f}) P(j \in e|\sigma_j)}{P(\sigma'_j|\sigma_j, j \in \bar{e} \cap \bar{f}) P(j \notin e|\sigma_j)}.$$

In figures 5.5 and 5.4 we compare those theoretical values (in logarithm) for the second iteration with the average values found by simulation. Note that because of the way the counters are distributed, we do not have empirical values for every couple (σ_j, σ'_j).

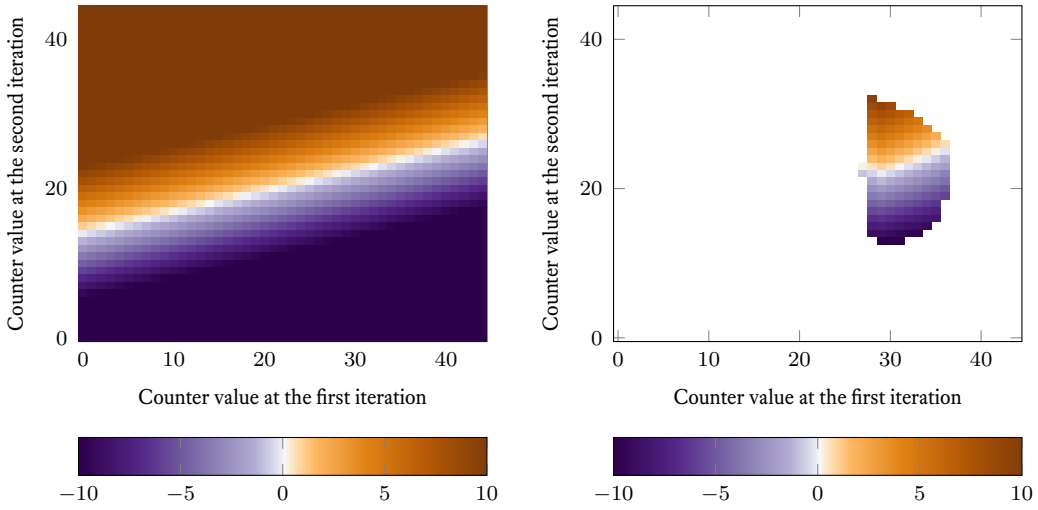
In the bitflipping algorithm, we flip a position when it is more likely to be an error. Here this happens when the ratio is greater than one or when the logarithm of the ratio is positive.



(a) Average values from the model ($p^+_{00} = 0.14084$, $p^-_{00} = 0.44592$, $p^+_{10} = 0.34323$, $p^-_{10} = 0.20684$)

(b) Average values from simulations

Figure 5.4 – Evolution of the counters between the first two iterations (on average) when the position was not flipped: $\log\left(\frac{P(j\in e|\sigma_j,\sigma'_j,j\notin f)}{P(j\notin e|\sigma_j,\sigma'_j,j\notin f)}\right)$



(a) Average values from the model ($p^+_{01} = 0.86216$, $p^-_{01} = 0.55589$, $p^+_{11} = 0.66590$, $p^-_{11} = 0.79448$)

(b) Average values from simulations

Figure 5.5 – Evolution of the counters between the first two iterations (on average) when the position was flipped: $\log\left(\frac{P(j\in e|\sigma_j,\sigma'_j,j\in f)}{P(j\notin e|\sigma_j,\sigma'_j,j\in f)}\right)$

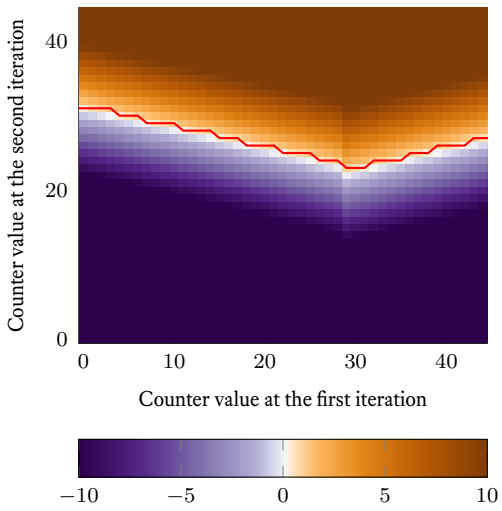


Figure 5.6 – Threshold curve in function of the previous counter value

5.2.3 Improving proposed variants

Grey zone In the proposed algorithm for the grey zone bitflipping, the same thresholds as the ones used in [Cha17] are used. But in this algorithm, only a small portion of the positions are selected, and because they are selected on their counter value at the first iteration, their counters at the second iteration will be distributed differently. For example in the figure 5.6 we can see the values of $\log\left(\frac{P(j\in e|\sigma_j,\sigma'_j,f)}{P(j\notin e|\sigma_j,\sigma'_j,f)}\right)$ after the first iteration. It appears that, rather than choosing the same threshold for all the positions, the curve in red is a better choice for the second iteration. This curve delimits the smallest counter values at the second iteration for which the logarithm is greater than zero *i.e.* such that $P(j \in e|\sigma_j, \sigma'_j, f) > P(j \notin e|\sigma_j, \sigma'_j, f)$.

We can see that for the second iteration, if we kept track of the previous counters, we can use the best threshold which can range from 23 to 31 with the chosen parameters. Keeping track of the previous counters might be expensive if we do it for all the positions. But in the grey bitflipping algorithm we already restrict ourselves to a few positions, so it might be better to use different thresholds for the grey iterations.

A smaller threshold for the previously flipped position could help detecting early the bad flips (the good positions that were flipped). This could accelerate the algorithm by reducing the number of grey iterations needed and potentially increase the efficiency of the grey zone algorithm.

Multi-bitflipping In the multi-bitflipping algorithm, we estimate the probability of a position to be an error knowing all its past counter values. The model we now have for the transition probabilities of the counters can be used to replace the empirical choice made for the function f.

In a way, the multi-bitflipping is a discretisation of this algorithm 5 which computes the values of:

$$\log\left(\frac{P(j \in e | y_j, \sigma_j^{(0)}, \ldots, \sigma_j^{(I)})}{P(j \notin e | y_j, \sigma_j^{(0)}, \ldots, \sigma_j^{(I)})}\right)$$

where $\sigma_j^{(I)}$ is the value of the counter at the position j for the I-th iteration. In this algorithm we keep track of the values of the counters at the previous iteration σ'_j.

Algorithm 5 The multi-bitflipping algorithm

procedure MULTI-BITFLIPPING(y, H) ▷ $y = (y_1, \ldots, y_n) \in \{0, 2^b - 1\}^n$
  $z \leftarrow \left(\log\left(\frac{P(e_1=1|y_1)}{P(e_1=0|y_1)}\right), \ldots, \log\left(\frac{P(e_n=1|y_n)}{P(e_n=0|y_n)}\right)\right) \in \mathbb{R}^n$
  $\sigma \leftarrow (0, \ldots, 0)$ ▷ $H = (h_1, \ldots, h_n) \in \{0,1\}^{k \times n}$
  $\sigma' \leftarrow (0, \ldots, 0)$
  while $Hy^\intercal \neq 0$ **do**
    $s \leftarrow Hy^\intercal$
    for $j = 1, \ldots, n$ **do**
      $\sigma'_j = \langle s, h_j^\top \rangle_{\mathbb{Z}}$
      $(y_j, z_j) \leftarrow f(y_j, z_j, \sigma_j, \sigma'_j)$
      $\sigma_j = \sigma'_j$
  return y

The transition probabilities we determined can be used to determine those values. We can suppose that

$$P(\sigma_j^{(I+1)} | j \in e, f, y_j, \sigma_j^{(0)}, \ldots, \sigma_j^{(I)}) = P(\sigma_j^{(I+1)} | j \in e, f, \sigma_j^{(I)})$$

and

$$P(\sigma_j^{(I+1)} | j \notin e, f, y_j, \sigma_j^{(0)}, \ldots, \sigma_j^{(I)}) = P(\sigma_j^{(I+1)} | j \notin e, f, \sigma_j^{(I)})\,.$$

Therefore, the ratio we are looking for can be determined with the following formula using Bayes' theorem.

$$\frac{P(j \in e | y_j, \sigma_j^{(0)}, \ldots, \sigma_j^{(I+1)})}{P(j \notin e | y_j, \sigma_j^{(0)}, \ldots, \sigma_j^{(I+1)})} = \frac{P(\sigma_j^{(I+1)} | j \in e, f, \sigma_j^{(I)})}{P(\sigma_j^{(I+1)} | j \notin e, f, \sigma_j^{(I)})} \frac{P(j \in e | y_j, \sigma_j^{(0)}, \ldots, \sigma_j^{(I)})}{P(j \notin e | y_j, \sigma_j^{(0)}, \ldots, \sigma_j^{(I)})}$$

The ratio will be inverted if we flip the position j.
Hence we can therefore choose the function f as follows.

$$f(y, z, \sigma, \sigma') = \begin{cases} (1-y, -(z + \log\left(\frac{P(\sigma|j\in e,f,\sigma')}{P(\sigma|j\notin e,f,\sigma')}\right)) & \text{if } z + \log\left(\frac{P(\sigma|j\in e,f,\sigma')}{P(\sigma|j\notin e,f,\sigma')}\right) > 0 \\ (y, z + \log\left(\frac{P(\sigma|j\in e,f,\sigma')}{P(\sigma|j\notin e,f,\sigma')}\right)) & \text{otherwise} \end{cases}.$$

Rather than relying on empirical values to determine the function f in the multi-bitflipping algorithm, this algorithm may be used to systematically find a multi-bitflipping for a given precision by approximating it.

To get an idea of the limit in terms of decoding failure rate that we can expect with that kind of algorithm, I implemented it in heavily idealised conditions. When we generate an instance we know the error pattern and thus we can keep track of which position is an error and whether it was flipped or not. Therefore we can compute all the probabilities p_{ij}^{+} and p_{ij}^{-} for $i, j = 0, 1$ specifically for a given iteration of a given decoding instance. This can be used to compute the probabilities needed to define the function f.

The results were given in section 5.1.4.

— 6 —

Conclusion

In this document, we looked at a variant of the McEliece cryptosystem. We then focused on the decoding algorithm for which there are security concerns as well as natural questions arising when looking at an efficient implementation.

We saw three differents variants of the decoding algorithm:

— the dynamic decoding which is fast and lightweight but performs badly when the error weight is too high;

— the grey zone algorithm which achieves a reduced complexity and a decreased decoding failure rate;

— the multi-bitflipping which can greatly reduce the decoding failure rate but requires a higher complexity.

We then had a look at the evolution of certain quantities during the decoding process. We looked at the possibility of improving the different variants allowed by the knowledge of those evolutions.

Further work can be made to make an actual use of it during the decoding process and stop relying on quantities that can be only observed because we know the generated error pattern.

Since it successfully reduces the decoding failure rate and the complexity, it could be interesting to pursue the analysis of the grey zone algorithm. In particular, the importance of the size of the grey zone has not been studied. And given the fact that the selection of a position in the grey zone depends on its counter at the first iteration, better choices of threshold could be made for the grey iterations.

Finally a synthesis of those variants could be studied. We could use a multi-bit precision for the grey zone as a way to find a tradeoff between the reduced complexity of the grey zone algorithm and the decoding capability of the multi-bitflipping. The dynamic decoding could eventually be used when the number of errors has been sufficiently reduced.

Bibliography

[Aug+15] Daniel Augot et al. *Initial recommendations of long-term secure post-quantum systems*. Sept. 2015.

[BBC08] Marco Baldi, Marco Bodrato and Franco Chiaraluce. « A New Analysis of the McEliece Cryptosystem Based on QC-LDPC Codes ». In: *Security and Cryptography for Networks: 6th International Conference, SCN 2008, Amalfi, Italy, September 10-12, 2008. Proceedings*. Ed. by Rafail Ostrovsky, Roberto De Prisco and Ivan Visconti. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 246–262. ISBN: 978-3-540-85855-3. DOI: 10.1007/978-3-540-85855-3_17. URL: https://doi.org/10.1007/978-3-540-85855-3_17.

[BC07] M. Baldi and F. Chiaraluce. « Cryptanalysis of a new instance of McEliece cryptosystem based on QC-LDPC Codes ». In: *2007 IEEE International Symposium on Information Theory*. June 2007, pp. 2591–2595. DOI: 10.1109/ISIT.2007.4557609.

[BCGM07] M. Baldi et al. « Quasi-Cyclic Low-Density Parity-Check Codes in the McEliece Cryptosystem ». In: *2007 IEEE International Conference on Communications*. June 2007, pp. 951–956. DOI: 10.1109/ICC.2007.161.

[Cha17] Julia Chaulet. « Étude de cryptosystèmes à clé publique basés sur les codes MDPC quasi-cycliques ». PhD thesis. University Pierre et Marie Curie, 2017.

[Cho16] Tung Chou. « QcBits: Constant-Time Small-Key Code-Based Cryptography ». In: *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*. 2016, pp. 280–300. DOI: 10.1007/978-3-662-53140-2_14. URL: https://doi.org/10.1007/978-3-662-53140-2_14.

[Gal63] Robert G. Gallager. *Low-Density Parity-Check Codes*. Cambridge, MA, USA, 1963.

[GJS16] Qian Guo, Thomas Johansson and Paul Stankovski. « A Key Recovery Attack on MDPC with CCA Security Using Decoding Errors ». In: *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*. 2016, pp. 789–815. DOI: 10.1007/978-3-662-53887-6_29. URL: http://dx.doi.org/10.1007/978-3-662-53887-6_29.

[Mac99] D. J. C. MacKay. « Good error-correcting codes based on very sparse matrices ». In: *IEEE Transactions on Information Theory* 45.2 (Mar. 1999), pp. 399–431. ISSN: 0018-9448. DOI: 10.1109/18.748992.

[McE78] Robert J McEliece. « A public-key cryptosystem based on algebraic ». In: *Coding Thv* 4244 (1978), pp. 114–116.

[MTSB13] Rafael Misoczki et al. « MDPC-McEliece: New McEliece variants from Moderate Density Parity-Check codes ». In: *Proceedings of the 2013 IEEE International Symposium on Information Theory, Istanbul, Turkey, July 7-12, 2013*. 2013, pp. 2069–2073. DOI: 10.1109/ISIT.2013.6620590. URL: http://dx.doi.org/10.1109/ISIT.2013.6620590.

[NV14] D. V. Nguyen and B. 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. ISSN: 0090-6778. DOI: 10.1109/TCOMM.2014.021614.130884.

[OTD10] Ayoub Otmani, Jean-Pierre Tillich and Léonard Dallot. « Cryptanalysis of Two McEliece Cryptosystems Based on Quasi-Cyclic Codes ». In: *Mathematics in Computer Science* 3.2 (Apr. 2010), pp. 129–140. ISSN: 1661-8289. DOI: 10.1007/s11786-009-0015-8. URL: https://doi.org/10.1007/s11786-009-0015-8.

[Pra62] E. Prange. « The use of information sets in decoding cyclic codes ». In: *IRE Transactions on Information Theory* 8.5 (Sept. 1962), pp. 5–9. ISSN: 0096-1000. DOI: 10.1109/TIT.1962.1057777.

[PZ03] John Proos and Christof Zalka. « Shor's discrete logarithm quantum algorithm for elliptic curves ». In: *Quantum Information & Computation* 3.4 (2003), pp. 317–344. URL: http://portal.acm.org/citation.cfm?id=2011531.

[Sho97] Peter W. Shor. « Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer ». In: *SIAM J. Comput.* 26.5 (1997), pp. 1484–1509. DOI: 10.1137/S0097539795293172. URL: https://doi.org/10.1137/S0097539795293172.