Improved Decoders for *p*-ary MDPC [⋆]

Isaac Canales-Martínez[1], Qian Guo[1], and Thomas Johansson[2]

[1] University of Bergen, Box 7803, N-5020 Bergen, Norway
{isaac.canales,qian.guo}@uib.no
[2] Lund University, Box 117, 221 00 Lund, Sweden
{thomas.johansson}@eit.lth.se

Abstract. The p-ary MDPC scheme is a new NTRU-type lattice-based post-quantum primitive that employs iterative decoding technique. In this paper, we present several novel decoding algorithms that reduce the decryption failure rate (DFR), applying fundamental ideas from bit-flipping algorithms for binary MDPC/LDPC codes. Such decryption improvements are crucial for proposing a practical lattice-based scheme with very compact key.

Keywords: MDPC · NTRU · p-ary MDPC · lattice-based cryptography · iterative decoding.

1 Introduction

Searching for secure and efficient post-quantum encryption primitives is one of the central problems in recent cryptographic research. Standardisation efforts (e.g., the NIST Post-Quantum Standardisation project [1]) have been made and among the submitted schemes, the variants of NTRU [10] and MDPC-McEliece [12] have shown great efficiency and proven their competence in the realms of lattice-based and code-based cryptography, respectively.

Based upon these two important primitives, the p-ary MDPC scheme was first proposed at ISIT 2016 [7], which can be viewed as an extension of the MDPC-McEliece into the Euclidean metric, or as an NTRU-type lattice-based scheme with iterative decoding. The scheme was further investigated by Deneuville et al. in [4] to have an efficient Ouroboros-type lattice-based key-exchange protocol.

The goal of using iterative decoding technique in lattice-based cryptography is to reduce the alphabet size while good error performance is maintained. This technique can enhance both efficiency and security. On one hand, it allows to propose a smaller public key size; on the other hand, lattice reduction algorithms like sieving and enumeration can be less efficient when the alphabet is small – these attack algorithms are considered as the main threat nowadays.

[⋆] Supported in part by the Norwegian Research Council (Grant No. 247742/070), the Swedish Research Council (Grant No. 2015-04528) and the Swedish Foundation for Strategic Research (SSF) project RIT17-0005.

In this paper, we present improved bit-flipping decoding algorithms for the p-ary MDPC. We focus on the average-case performance and reduce their decryption failure rate (DFR) by changing the thresholds employed in each decoding iteration. The improvements are demonstrated by extensive simulation.

The remaining parts of the paper are organised as follows. We present the background on the p-ary MDPC primitive in Section 2. This is followed by the main section presenting the different versions of the newly proposed decoders. We conclude this paper in Section 4.

2 Preliminary

We introduce background on the p-ary MDPC scheme, consisting of basic knowledge on coding theory and a particular instantiation of the p-ary MDPC scheme. Vectors and matrices will be denoted by bold characters throughout this paper.

2.1 Coding Basics

Let p be a prime integer and let $\mathcal{C}$ be a subspace of $\mathbb{F}_p^n$ with dimension k and codimension $r = n - k$. Then $\mathcal{C}$ is called an $[n, k]$ *linear code* of *length* n and *dimension* k. A vector $\mathbf{c} \in \mathcal{C}$ is called *codeword*. From now on, we will refer to a linear code simply as a code. The *size* of a code is the number of codewords, i.e. $|\mathcal{C}| = p^k$. Let $\mathbf{G}$ be a $k \times n$ matrix whose rows are the vectors of a basis of $\mathcal{C}$, then we call $\mathbf{G}$ a *generator matrix* and it defines the code as $\mathcal{C} = \{\mathbf{uG} \mid \mathbf{u} \in \mathbb{F}_p^k\}$. Likewise, $\mathcal{C}$ can be defined by an $r \times n$ matrix $\mathbf{H}$, called *parity-check matrix*, as $\mathcal{C} = \{\mathbf{v} \in \mathbb{F}_p^n \mid \mathbf{H}\mathbf{v}^T = \mathbf{0}\}$, i.e. $\mathcal{C}$ is the kernel of $\mathbf{H}$. The *syndrome* of a vector $\mathbf{v} \in \mathbb{F}_p^n$ is defined as $\mathbf{H}\mathbf{v}^T$. It follows that for $\mathbf{c} \in \mathcal{C}$, $\mathbf{H}\mathbf{c}^T = \mathbf{0}$ and for $\mathbf{c} \notin \mathcal{C}$, $\mathbf{H}\mathbf{c}^T \neq \mathbf{0}$. When $p = 2$, it is customary to call the code binary; otherwise we call it a p-ary code.

A *Low Density Parity Check* code (LDPC) is a linear code with a sparse parity-check matrix. A *Moderate Density Parity Check* code (MDPC) is a linear code with a still sparse but denser parity-check matrix.

A code is *quasi-cyclic* if there exists an integer s such that for any codeword $\mathbf{c}$, a cyclic shift by s positions results in another codeword $\mathbf{c}'$. Let $\mathcal{C}$ be an $[n, k]$ quasi-cyclic code with $n = sk$. Then the generator and parity-check matrices of $\mathcal{C}$ can be constructed by circulant blocks of size $k \times k$. Notice that only one vector determines the whole block (namely the first row), hence only one vector per block is needed to completely determine the generator and parity-check matrices. Moreover, due to the isomorphism between the algebra of these circular matrices and that of polynomials in $\mathbb{F}_p[X]/\langle X^k - 1\rangle$, the matrix/vector operations can be implemented and treated as operations in this polynomial ring.

2.2 p-ary MDPC

The general construction of a p-ary MDPC scheme is firstly presented in [7]. In this paper we employ a concrete instantiation [8] similar to the Ouroboros-E key-exchange protocol proposed in [4]. We choose to utilise a parity-check matrix of two blocks of size $k \times k$.

We can allow other parameter sets based on secure parameter suggestions from NTRU-based or Ring-LWE-based cryptosystems, e.g., choosing a non-prime alphabet size (denoted by q instead for the general setting), or using the ring $\mathcal{R} = \mathbb{Z}_q[X]/\langle X^k + 1\rangle$ rather than the quasi-cyclic structure. Safe parameters include q a prime, k a power of 2, and $\mathcal{R} = \mathbb{Z}_q[X]/\langle X^k + 1\rangle$; or q a power of 2, k a prime, and $\mathcal{R} = \mathbb{Z}_q[X]/\langle X^k - 1\rangle$.

Let d be a positive integer, and define $\mathcal{I}_d = \{-d, \dots, 0, \dots, d\}$, $\mathbf{h}_0^{(d)} = \lfloor \frac{q}{2d+1} \rceil (1, 0, \dots, 0)$ and $\mathbf{h}_1^{(d)} = \lfloor \frac{q}{(2d+1)^2} \rceil (1, 0, \dots, 0)$.

Key generation:

- Given a parameter d, compute $\mathbf{h}_0 = \mathbf{h}_0^{(d)} + \hat{\mathbf{h}}_0$ and $\mathbf{h}_1 = \mathbf{h}_1^{(d)} + \hat{\mathbf{h}}_1$, where $\hat{\mathbf{h}}_0 \xleftarrow{\$} \mathcal{I}_d^k$ and $\hat{\mathbf{h}}_1 \xleftarrow{\$} \mathcal{I}_d^k$. For simplicity, we assume the ring $\mathcal{R} = \mathbb{Z}_q[X]/\langle X^k - 1\rangle$ is employed and require the sum of the coefficients of $\hat{\mathbf{h}}_0$ ($\hat{\mathbf{h}}_1$) to be 0.
- Compute $\mathbf{H} = (\mathbf{H}_0|\mathbf{H}_1)$, where $\mathbf{H}_0$ and $\mathbf{H}_1$ are matrices obtained by performing $k-1$ cyclic shifts of $\mathbf{h}_0$ and $\mathbf{h}_1$, respectively. If $\mathbf{H}_0$ is singular, go to the previous step and regenerate $\mathbf{h}_0$ and $\mathbf{h}_1$.
- Compute $\mathbf{G} = (\mathbf{I}|\mathbf{H}_0^{-1}\mathbf{H}_1)$.
- Return the private key-public key pair $(\mathbf{H}, \mathbf{G})$.

Encryption:

- Let $\mathbf{m} \in \mathbb{F}_q^k$ be the plaintext and $\mathbf{e} \xleftarrow{\$} \mathcal{I}_d^n$.
- Return the ciphertext $\mathbf{c} = \mathbf{mG} + \mathbf{e}$.

Decryption:

- Get $\mathbf{mG}$ by using a decoder to remove the noise $\mathbf{e}$ from $\mathbf{c}$.
- Extract $\mathbf{m}$ from the first n entries of $\mathbf{mG}$ (because $\mathbf{G}$ is in systematic form.)
- Return the plaintext $\mathbf{m}$.

3 Reducing DFR by Varying Thresholds

In this section we show a decoding algorithm for the p-ary MDPC scheme in Section 2.2. We also improve the decoding performance using varying thresholds for different iterations, as the bit-flipping decoders in [6, 12].

3.1 The State-of-the-Art Bit-Flipping Decoder for p-ary MDPC

The decoder that we present here, in Algorithm 1, is close to the noisy p-ary bit-flipping decoder for Ouroboros-E [4], with adjustments for the p-ary MDPC. We set $d = 1$ for simplicity.

The decoder takes as inputs the ciphertext $\mathbf{c}$, the parity-check matrix $\mathbf{H}$ and a parameter *iter* specifying the number of iterations. The algorithm outputs the

Algorithm 1 p-ary bit flipping

```
Input: The ciphertext $\mathbf{c}$, the private key $\mathbf{H}$, and the number of iterations $iter$
Output: The error $\mathbf{e}$ if success, $\perp$ otherwise
 1: $\mathbf{e} \leftarrow \mathbf{0} \in \mathbb{Z}_q^n$
 2: Compute the syndrome $\mathbf{s} = \mathbf{H} \cdot \mathbf{c}^T$
 3: $\mathbf{p} \leftarrow \mathbf{s}$
 4: for $i \leftarrow 1$ to $iter$ do
 5:     $\mathbf{e}' \leftarrow$ DECIDE($\mathbf{p}$)
 6:     $\mathbf{e} \leftarrow \mathbf{e} + \mathbf{e}'$
 7:     TRANSFORM($\mathbf{e}$)
 8:     $\mathbf{p} \leftarrow \mathbf{s} - \mathbf{H} \cdot \mathbf{e}^T$
 9:     if $p_j = 0$ for all $j \in \{1, \ldots, k\}$ then
10:         return $\mathbf{e}$
11:     end if
12: end for
13: return $\perp$
```

error vector $\mathbf{e}$ if decoding was successful, or $\perp$ (decoding error) otherwise. The main idea is to recover the error $\mathbf{e}$ by iteratively updating the value of each entry e_i according to a decision rule. When the correct value of $\mathbf{e}$ is found, we have that $\mathbf{s} - \mathbf{H} \cdot \mathbf{e}^T = 0$, where $\mathbf{s} = \mathbf{H} \cdot \mathbf{c}^T$. If the computed error $\mathbf{e}$ is such that $\mathbf{s} - \mathbf{H} \cdot \mathbf{e}^T \neq 0$ after $iter$ iterations, decoding was unsuccessful.

Algorithm 2 details the updating decision rule for recovering $\mathbf{e}$. Recall that each parity check (row) of $\mathbf{H}$ has two significant entries, namely, the $\mathbf{h}_0^{(1)}$ and $\mathbf{h}_1^{(1)}$ parts of $\mathbf{h}_0$ and $\mathbf{h}_1$, respectively. Each of these entries is sampled from $\{-1, 0, 1\}$, and thus we have 9 different signal points to consider, i.e., $\{-1, 0, 1\} \times \{-1, 0, 1\}$. We spread these 9 points throughout $[0, q] \pmod{q}$ as shown in figure 1a. In order to update $\mathbf{e}$ for the $\mathbf{h}_0^{(1)}$ part, the set $[0, q] \pmod{q}$ is divided into 3 intervals. These intervals are determined by $\frac{q}{6}$, $\frac{q}{2}$ and $\frac{5q}{6}$, and each one has 3 signal points and an associated update value, as shown in Figure 1b. Notice that any of these intervals has "length" $\frac{q}{3}$ and three signal points which represent the possible update values for the $\mathbf{h}_1^{(1)}$ part. Updating $\mathbf{e}$ for the $\mathbf{h}_1^{(1)}$ part is done in a similar way as for $\mathbf{h}_0^{(1)}$. We divide the set $[0, \frac{q}{3}] \pmod{\frac{q}{3}}$ into 3 intervals determined by $\frac{q}{18}$, $\frac{q}{6}$ and $\frac{5q}{18}$, each one having 1 signal point, as shown in Figure 1c.

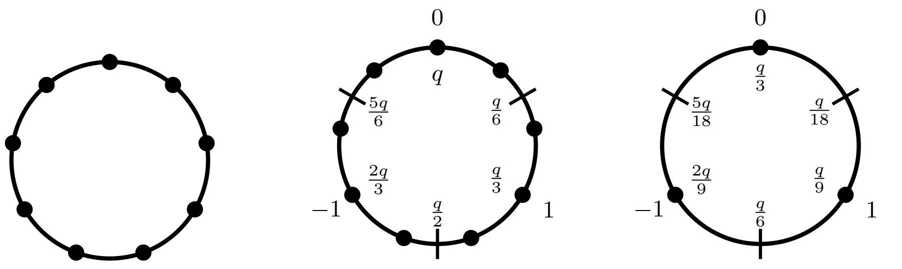
The vector $\mathbf{e}$ is updated in line 6 of Algorithm 1. Notice that the addition of $\mathbf{e}$ and $\mathbf{e}'$ may lead to coefficients equivalent to 2 or -2. We thus employ the function TRANSFORM to ensure the resulting vector is valid, i.e., from $\mathcal{I}_1^n$. In [4], the function TRANSFORM sets a coefficient to be -1 if it is 2 and 1 if it is -2.

3.2 The Essential Idea

The decision rule in Algorithm 2 keeps the same interval bounds for all iterations in the main loop of Algorithm 1. We propose to update these bounds. For this, we introduce a threshold $thr^{(i)}$ that determines the new interval bounds in iteration i, $i = 1, \ldots, iter$. Figure 2 depicts how the intervals might change throughout the

Algorithm 2 DECIDE($\mathbf{p}$)

```
Input: Vector $\mathbf{p}$
Output: Vector $\mathbf{e}'$ resulting from applying the decision rule to $\mathbf{p}$
 1: $\mathbf{e}' \leftarrow \mathbf{0}$
 2: for $j \leftarrow 1$ to $k$ do
 3:     if $p_j \in [[\lceil \frac{q}{6} \rceil, \lfloor \frac{q}{2} \rfloor]]$ then
 4:         $e'_j \leftarrow 1$
 5:     else if $p_j \in [[\lceil \frac{q}{2} \rceil, \lfloor \frac{5q}{6} \rfloor]]$ then
 6:         $e'_j \leftarrow -1$
 7:     end if
 8:     if $(p_j \bmod \lfloor \frac{q}{3} \rceil) \in [[\lceil \frac{q}{18} \rceil, \lfloor \frac{q}{6} \rfloor]]$ then
 9:         $e'_{k+j} \leftarrow 1$
10:     else if $(p_j \bmod \lfloor \frac{q}{3} \rceil) \in [[\lceil \frac{q}{6} \rceil, \lfloor \frac{5q}{18} \rfloor]]$ then
11:         $e'_{k+j} \leftarrow -1$
12:     end if
13: end for
14: return $\mathbf{e}'$
```



(a) Signal points in $[0, q]$ (mod q).

(b) Intervals of $[0, q]$ (mod q) with their signal points and associated values for $\mathbf{h}_0^{(1)}$.

(c) Intervals of $[0, \frac{q}{3}]$ (mod $\frac{q}{3}$) with their signal points and associated values for $\mathbf{h}_1^{(1)}$.

Fig. 1: Graphical representation of the decoding decision rule. When $d = 1$, there are 9 signal points to consider.

main loop. When an entry p_j does not lie within any of the intervals for 1 and -1, we are not able to decide the value for $\mathbf{e}'$ at the corresponding position, and we set it to zero. Algorithm 3 shows the new decision rule using the proposed thresholds. In Sections 3.3 and 3.4 we propose new methods for computing $thr^{(i)}$.

Note that this new proposal is a generalisation of the previous decoder: when $thr^{(i)} = 0$, it is equivalent to using the interval thresholds as in Algorithm 2. Moreover, Algorithm 3 can be further generalised: at iteration i, instead of choosing $thr^{(i)}$ only, it is possible to choose different thresholds for the upper and lower bounds of each of the intervals.

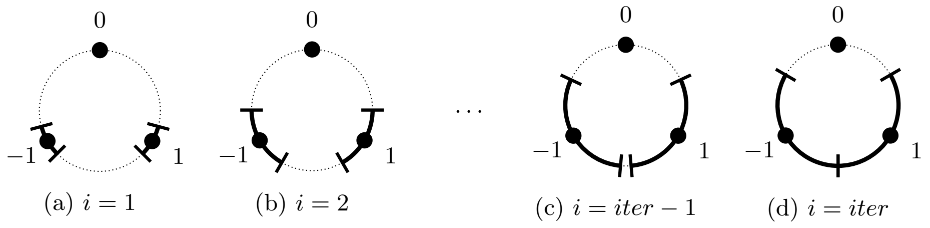


Fig. 2: Intervals used for iterations $i = 1, \ldots, iter$ in the new decoding procedure.

Algorithm 3 DecideThr($\mathbf{p}$)

```
Input: Vector $\mathbf{p}$
Output: Vector $\mathbf{e}'$ resulting from applying the decision rule to $\mathbf{p}$
 1: $\mathbf{e}' \leftarrow \mathbf{0}$
 2: Compute $thr^{(i)}$
 3: for $j \leftarrow 1$ to $k$ do
 4:     if $p_j \in [\lceil \frac{q}{6} \rceil + thr^{(i)}, \lfloor \frac{q}{2} \rfloor - thr^{(i)}]$ then
 5:         $e'_j \leftarrow 1$
 6:     else if $p_j \in [\lceil \frac{q}{2} \rceil + thr^{(i)}, \lfloor \frac{5q}{6} \rfloor - thr^{(i)}]$ then
 7:         $e'_j \leftarrow -1$
 8:     end if
 9:     if $(p_j \bmod \lfloor \frac{q}{3} \rceil) \in [\lceil \frac{q}{18} \rceil + thr^{(i)}, \lfloor \frac{q}{6} \rfloor - thr^{(i)}]$ then
10:         $e'_{k+j} \leftarrow 1$
11:     else if $(p_j \bmod \lfloor \frac{q}{3} \rceil) \in [\lceil \frac{q}{6} \rceil + thr^{(i)}, \lfloor \frac{5q}{18} \rfloor - thr^{(i)}]$ then
12:         $e'_{k+j} \leftarrow -1$
13:     end if
14: end for
15: return $\mathbf{e}'$
```

3.3 Gallager-B type Decoders

Here we estimate the error probability and derive a series of theoretical values for the decoding thresholds. The analogue in the Hamming metric is the tree-based analysis in [6].

First, we assume that the noise random variable in each iteration is Gaussian with mean 0 (due to the intuition from the central limit theorem) since it is a sum of many small noise variable with mean 0. For a fast estimation, we ignore the independence issue that may occur in the decoding process. We track the change of the average error variance in each iteration.

Every entry p_j in $\mathbf{p}$ is a sum of the contribution $e_j \lfloor \frac{q}{3} \rceil + e_{k+j} \lfloor \frac{q}{9} \rceil$ and a second part called noise, denoted N_j, where $N_j = (\hat{\mathbf{h}}_0^{[j]}, \hat{\mathbf{h}}_1^{[j]})\mathbf{e}^T$ and $\hat{\mathbf{h}}_i^{[j]}$ means $\hat{\mathbf{h}}_i$ cyclically shifted j steps. The noise variance σ_0^2 is initially $\frac{4n}{9} = \frac{8k}{9}$. We now only record the probability that a signal point (e_j, e_{k+j}) is wrongly decoded to its neighbour in the torus, e.g., (0,0) to (0,-1) or (0,1); (1,1) to (-1,-1) or (1,0).

Let us denote

$$\pi_{i+1,+} = \mathsf{erfc}\left(\frac{\frac{q}{18} + thr_{i+1}}{\sqrt{2}\sigma_i}\right),$$

and

$$\pi_{i+1,-} = \mathsf{erfc}\left(\frac{\frac{q}{18} - thr_{i+1}}{\sqrt{2}\sigma_i}\right).$$

Let $\mathbf{e}$ be the original error vector and $\hat{\mathbf{e}}^{(i)}$ be the guessed error vector in the i-th iteration. We have $\hat{\mathbf{e}}^{(0)} = \mathbf{0}$ and we get

$$\mathbf{p}^{(i)} = \mathbf{s} - \mathbf{H}\hat{\mathbf{e}}^{(i)} = \mathbf{H}(\mathbf{e} - \hat{\mathbf{e}}^{(i)}),$$

which is the input to the DecideThr() procedure in the i-th iteration.

We know that for $1 \leq j \leq k$, e_j and e_{k+j} are distributed uniformly in $\mathcal{I}_1 = \{-1, 0, 1\}$ before the first iteration. If $e_{k+j} = 0$, the coefficient e_j is almost certainly correctly decoded. If the signal point is $(0, 1)$ with probability $1/9$, then the probability to wrongly decode e_j to 1 is $0.5\pi_{1,+}$. If the signal point is $(1, -1)$ with probability $1/9$, then the probability to wrongly decode e_j to 0 is $0.5\pi_{1,-}$. If the signal point is $(1, 1)$ with probability $1/9$, then the probability to wrongly decode e_j to 0 is $0.5(\pi_{1,-} - \pi_{1,+})$ and to -1 is $0.5\pi_{1,+}$. We can compute the error variance by symmetry for the rest signal points. The noise variance introduced by the first part of the error vector can be estimated as $\frac{2}{3} \cdot \frac{2k}{9}(\pi_{1,-} + 2\pi_{1,+})$. Similarly, we estimate the noise variance introduced by the second part of the error vector as $\frac{2}{3}\frac{6k}{9}(\pi_{1,-} + 2\pi_{1,+})$. We can compute the noise variance σ_1^2 as $\sigma_1^2 = \frac{2}{3} \cdot \frac{8k}{9}(\pi_{1,-} + 2\pi_{1,+})$.

Since the error occurring in the first k positions is much easier to correct than the errors in the last k positions, we track only the noise variance from the last k positions of the error vector, from the second iteration. Let π_i denote the probability that the decision on the position $\hat{e}_{k+j}^{(i)}$ is correct. Thus, we have that

$$\pi_{i+1} = \pi_i(1 - \pi_{i+1,+}) + (1 - \pi_i)(1 - \pi_{i+1,-}),$$

and $\pi_0 = 1/3$.

Finally, we can iteratively estimate the value of the remaining noise in the $(i+1)$-th iteration, σ_{i+1}^2, as

$$\sigma_{i+1}^2 = \frac{2k}{3} \cdot \left(\pi_i\pi_{i+1,+} + (1 - \pi_i)(\pi_{i+1,-} + 1.5\pi_{i+1,+})\right), \qquad (1)$$

for $i \geq 1$. We can also alternatively introduce a new parameter[3] $c_\lambda > 1$ to have better performance heuristically, i.e., we compute σ_{i+1}^2 by

$$\sigma_{i+1}^2 = c_\lambda \cdot \frac{2k}{3} \cdot \left(\pi_i\pi_{i+1,+} + (1 - \pi_i)(\pi_{i+1,-} + 1.5\pi_{i+1,+})\right), \qquad (2)$$

for $i \geq 1$.

We choose the thresholds $thr^{(i)}$ such that the noise variance σ_i^2 is minimised, which can be solved numerically. We denote by $G1$ and $G2$ the decoders using thresholds computed from Equations (1) and (2), respectively.

[3] We have $c_\lambda > 1$ since the error contribution of the first k positions is omitted in Equation (1).

3.4 Heuristic Decoders

Here we present three heuristic decoders called $H1$, $H2$ and $H3$, respectively. The three decoders are similar to their Hamming counterpart from [11, 12] for MDPC-McEliece. Let $\mathcal{A}$ be the set of nine signal points, i.e.,

$$\mathcal{A} := \{0, \lfloor q/9 \rceil, \lfloor 2q/9 \rceil, \ldots, \lfloor 8q/9 \rceil\}.$$

In the i-th iteration of $H1$, we compute

$$thr^{(i)}_{\mathsf{max}} = \frac{q}{18} - \min_{\substack{j\in\{1,\ldots,k\},\\ a\in\mathcal{A}\setminus\{0\}}} |p^{(i)}_j - a|,$$

to make a parity-check equation with its updated syndrome $p^{(i)}_j$ closest to a non-zero signal point corrected (flipped). The threshold in the i-th iteration is

$$thr^{(i)} = \max(0, thr^{(i)}_{\mathsf{max}} - \delta), \tag{3}$$

where a positive constant δ determined by simulation is used to reduce the required number of iterations in the average case.

The decoder $H2$ is a variant of $H1$ and provides comparable (or even better) error performance in some simulations. In the i-th iteration, we compute

$$thr^{(i)}_{\mathsf{min}} = \min_{\substack{j\in\{1,\ldots,k\},\\ a\in\mathcal{A}}} |p^{(i)}_j - a|.$$

The threshold in the i-th iteration is

$$thr^{(i)} = thr^{(i)}_{\mathsf{min}} + \delta \tag{4}$$

for a positive constant δ determined by simulation.

The decoder $H3$ is also a variant of $H1$ and provides the best error performance in simulations. We choose $\delta = \delta_0$, for some value δ_0. In the i-th iteration, we compute $thr^{(i)}_{\mathsf{max}}$ and $thr^{(i)}$ as in $H1$. If decoding is unsuccessful, decrease the value of δ by 1 and restart the process. This is repeated until decoding is successful or $\delta = 0$ (decoding failure).

3.5 Experimental Results

Here we give the experimental results using the decoders presented in Sections 3.3 and 3.4. We compare the proposed decoders with the reference decoder [4]. Note that the values of k and q are only chosen for testing the error performance.

For $G1$ we computed the theoretical thresholds as in Section 3.3; the thresholds and variances obtained are shown in Table 1 when $c_\lambda = 1.00$. For $G2$ we computed the thresholds varying the value of c_λ from 1.0 to 2.0 by steps of 0.01. We heuristically chose to use for our experiments the values shown in Table 1, with $c_\lambda > 1.00$. For both, $G1$ and $G2$, let i be the smallest integer such that

k	q	c_λ	$i=1$		$i=2$		$i=3$		$i=4$		$i=5$		$i \geq 6$	
			$thr^{(i)}$	σ_i^2	$thr^{(i)}$	σ_i^2	$thr^{(i)}$	σ_i^2	$thr^{(i)}$	σ_i^2	$thr^{(i)}$	σ_i^2	$thr^{(i)}$	σ_i^2
491	345	1.00	8	288.32	7	123.64	4	31.30	2	0.12	0	0.0000	0	0.00
491	345	1.01	8	288.32	7	124.87	5	32.18	2	0.14	0	0.0000	0	0.00
491	345	1.24	8	288.32	7	153.31	6	54.80	3	2.71	0	0.0000	0	0.00
491	345	1.31	8	288.32	7	161.96	6	62.49	4	4.72	1	5.65×10^{-5}	0	0.00
491	360	1.00	8	269.43	7	101.18	4	15.16	1	3.70×10^{-5}	0	0.00	0	0.00
491	360	1.17	8	269.43	7	118.38	5	24.94	2	0.01	0	0.00	0	0.00
491	360	1.43	8	269.43	7	144.69	6	43.97	3	0.65	0	0.00	0	0.00
491	360	1.66	8	269.43	7	167.97	6	64.39	4	4.15	1	4.36×10^{-20}	0	0.00

Table 1: Computed theoretical thresholds for the Gallager-like decoders.

$thr^{(i)} = 0$. Then we have that $thr^{(j)} = 0$ for $j \geq i$. However, in the experiments we kept on using the last non-zero threshold up to iteration $\frac{iter}{2}$, i.e., $thr^{(j)} = thr^{(i-1)}$ for $j = i, \ldots, \frac{iter}{2}$ and $thr^{(j)} = 0$ for $j = \frac{iter}{2} + 1, \ldots, iter$. If we had not made this, both decoders would have been different to that in [4] in the first $i - 1$ iterations only.

For the heuristic decoders, we first executed some moderate size experiments (100 MDPC instances and 1 000 decoding executions per instance) to determine the best values of δ. We used these values to execute the experiments reported here.

Table 2 summarises the results obtained from the experiments. One experiment comprises the random generation of 10 000 MDPC instances and 1 000 decoding executions per instance. The entries in the table show the number of decoding errors among 10^7 decoding executions. For decoder $G2$, columns $c_{\lambda,2}$, $c_{\lambda,3}$ and $c_{\lambda,4}$ show the results when using the thresholds in the 2nd, 3rd and 4th rows of Table 1, respectively, for the different values of k and q. In general, our proposals have better performance than the reference decoder. The heuristic decoders present the best decoding failure rate. Decoder $H3$ presents a very low failure rate with the parameters in table 2. We also executed this decoder with $k = 491, q = 375$ obtaining 0 decoding errors for $iter = 20$.

We noted in the experiments that the execution time of the reference decoder was significantly higher than that of the decoders we propose. This difference in performance is due to the number of iterations required to finish the decoding procedure in the average case.

4 Conclusions and Future Works

In this paper, we have presented novel iterative decoders for the p-ary MDPC scheme by varying the thresholds used in each iteration. These thresholds are determined either by numerically optimising the error level in the next iteration, as was done by Gallager [6], or by applying heuristic methods. We have demonstrated improved decoding performance by simulation.

Two more interesting problems will be further investigated in an extended paper. Firstly, we intend to propose a better worst-case decoder, as was studied

k	q	$iter$	[4]	$G1$	$G2$			$H1$		$H2$		$H3$
					$c_{\lambda,2}$	$c_{\lambda,3}$	$c_{\lambda,4}$	$\delta = 16$	$\delta = 17$	$\delta = 2$	$\delta = 3$	$\delta_0 = \lfloor \frac{q}{18} \rceil$
491	345	100	271	11	14	17	27	4	5	2	6	0
491	345	75	405	19	24	17	22	10	13	5	11	0
491	345	50	992	60	53	49	98	29	23	27	28	0
491	345	25	14543	909	949	911	1309	346	367	402	402	12
491	360	100	5	1	2	3	1	1	1	0	0	0
491	360	75	4	2	2	3	2	4	2	1	1	0
491	360	50	17	10	11	2	11	5	8	1	5	0
491	360	25	267	163	131	122	188	44	32	32	48	2

Table 2: Decoding failure rate ($\times 10^{-7}$) of experiments for different parameters. Each experiment comprises the random generation of 10 000 MDPC instances and 1 000 decoding executions per instance.

in [3] for the binary MDPC. We will also test the heuristic independence assumption made in [7] for proposing parameters with an arbitrarily small decryption error probability. The latter can be crucial to resist the potential reaction attack [9, 5] that is already a threat to the MDPC/LDPC-based cryptosystems.

References

1. NIST Post-Quantum Cryptography Standardization. `https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization`, accessed: 2018-09-24
2. IEEE International Symposium on Information Theory, ISIT 2016, Barcelona, Spain, July 10-15, 2016. IEEE (2016), `http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=7532279`
3. Chaulet, J., Sendrier, N.: Worst case QC-MDPC decoder for mceliece cryptosystem. In: IEEE International Symposium on Information Theory, 2016, Barcelona, Spain [2], pp. 1366–1370, `https://doi.org/10.1109/ISIT.2016.7541522`
4. Deneuville, J., Gaborit, P., Guo, Q., Johansson, T.: Ouroboros-e: An efficient lattice-based key-exchange protocol. In: 2018 IEEE International Symposium on Information Theory, ISIT 2018, Vail, CO, USA, June 17-22, 2018. pp. 1450–1454. IEEE (2018), `https://doi.org/10.1109/ISIT.2018.8437940`
5. Fabsic, T., Hromada, V., Stankovski, P., Zajac, P., Guo, Q., Johansson, T.: A reaction attack on the QC-LDPC McEliece cryptosystem. In: Post-Quantum Cryptography - 8th International Workshop, PQCrypto 2017. pp. 51–68. Springer, Heidelberg, Germany, Utrecht, The Netherlands (Jun 26–28 2017)
6. Gallager, R.G.: Low-density parity-check codes. IRE Trans. Information Theory 8(1), 21–28 (1962), `https://doi.org/10.1109/TIT.1962.1057683`
7. Guo, Q., Johansson, T.: A p-ary MDPC scheme. In: IEEE International Symposium on Information Theory, ISIT 2016, Barcelona, Spain, July 10-15, 2016 [2], pp. 1356–1360, `https://doi.org/10.1109/ISIT.2016.7541520`
8. Guo, Q., Johansson, T.: The p-ary MDPC – an iterative trapdoor in lattice-based cryptogrpahy (In preparation)
9. Guo, Q., Johansson, T., Stankovski, P.: A key recovery attack on MDPC with CCA security using decoding errors. In: Advances in Cryptology – ASIACRYPT 2016, Part I. Lecture Notes in Computer Science, vol. 10031, pp. 789–815. Springer, Heidelberg, Germany, Hanoi, Vietnam (Dec 4–8, 2016)

10. Hoffstein, J., Pipher, J., Silverman, J.H.: NTRU: A ring-based public key cryptosystem. In: Buhler, J. (ed.) Algorithmic Number Theory, Third International Symposium, ANTS-III, Portland, Oregon, USA, June 21-25, 1998, Proceedings. Lecture Notes in Computer Science, vol. 1423, pp. 267–288. Springer (1998), `https://doi.org/10.1007/BFb0054868`
11. Huffman, W.C., Pless, V.: Fundamentals of error-correcting codes (2003)
12. Misoczki, R., Tillich, J., Sendrier, N., Barreto, P.S.L.M.: Mdpc-mceliece: New mceliece variants from moderate density parity-check codes. In: Proceedings of the 2013 IEEE International Symposium on Information Theory, Istanbul, Turkey. pp. 2069–2073. IEEE (2013), `https://doi.org/10.1109/ISIT.2013.6620590`