# Hamming Quasi-Cyclic (HQC)

Second round version

*HQC is an IND-CCA2 KEM running for standardization to NIST's competition in the category "post-quantum public key encryption scheme". Different sets of parameters are proposed for security strength categories 1, 3, and 5.*

**Principal Submitters (by alphabetical order):**

- Carlos Aguilar Melchor
- Nicolas Aragon
- Slim Bettaieb
- Loïc Bidoux
- Olivier Blazy
- Jean-Christophe Deneuville
- Philippe Gaborit
- Edoardo Persichetti
- Gilles Zémor

**Inventors:** Same as submitters

**Developers:** Same as submitters, and Jurjen Bos (Worldline)

**Owners:** Same as submitters

This work was partially funded by French DGA.

**Main contact**

Philippe Gaborit
philippe.gaborit@unilim.fr
+33-626-907-245
University of Limoges
123 avenue Albert Thomas
87 060 Limoges Cedex
France

**Backup point of contact**

Jean-Christophe Deneuville
jch.deneuville@gmail.com
+33-631-142-705
INSA-CVL Bourges &
University of Limoges
4 rue Jean le Bail
87 000 Limoges
France

**Signatures**

Digital copies of the signed statements were provided to NIST in the original submission on Nov. 30, 2017. The paper versions have been provided to NIST at the First PQC

Standardization Conference on Apr. 13, 2018.

# Contents

# 1 Specifications

In this section, we introduce HQC, an efficient encryption scheme based on coding theory. HQC stands for Hamming Quasi-Cyclic. This proposal has been published in IEEE Transactions on Information Theory [1]. Many notations, definitions and properties are very similar to [13]. We nevertheless include them in this proposal for completeness.

HQC is a code-based public key cryptosystem with several desirable properties:

- It is proved IND-CPA assuming the hardness of (a decisional version of) the Syndrome Decoding on structured codes. By construction, HQC perfectly fits the recent KEM-DEM transformation of [23], and allows to get an hybrid encryption scheme with strong security guarantees (IND-CCA2) and good efficiency,
- In contrast with most code-based cryptosystems, the assumption that the family of codes being used is indistinguishable among random codes is no longer required, and
- It features a decryption failure probability analysis.

**Organization of the Specifications.** This section is organized as follows: we provide the required background in Sec. 1.1, we make some recalls on encryption and security in Sec. 1.2 then present our proposal in Sec. 1.3. An analysis of the decryption failure rate is proposed in Sec. 1.4. Details about codes being used are provided in Sec. 1.5, together with a specific analysis for these codes. Finally, concrete sets of parameters are provided in Sec. 1.7.

## 1.1 Preliminaries

### 1.1.1 General definitions

Throughout this document, $\mathbb{Z}$ denotes the ring of integers and $\mathbb{F}_2$ the binary finite field. Additionally, we denote by $\omega(\cdot)$ the Hamming weight of a vector *i.e.* the number of its non-zero coordinates, and by $\mathcal{S}_w^n(\mathbb{F}_2)$ the set of words in $\mathbb{F}_2^n$ of weight $w$. Formally:

$$\mathcal{S}_w^n(\mathbb{F}_2) = \{\mathbf{v} \in \mathbb{F}_2^n, \text{ such that } \omega(\mathbf{v}) = w\}.$$

$\mathcal{V}$ denotes a vector space of dimension $n$ over $\mathbb{F}_2$ for some positive $n \in \mathbb{Z}$. Elements of $\mathcal{V}$ can be interchangeably considered as row vectors or polynomials in $\mathcal{R} = \mathbb{F}_2[X]/(X^n - 1)$. Vectors/Polynomials (resp. matrices) will be represented by lower-case (resp. upper-case) bold letters. A prime integer $n$ is said primitive if the polynomial $X^n - 1/(X - 1)$ is irreducible in $\mathcal{R}$.

For $\mathbf{u}, \mathbf{v} \in \mathcal{V}$, we define their product similarly as in $\mathcal{R}$, *i.e.* $\mathbf{uv} = \mathbf{w} \in \mathcal{V}$ with

$$w_k = \sum_{i+j \equiv k \mod n} u_i v_j, \text{ for } k \in \{0, 1, \ldots, n-1\}. \tag{1}$$

Our new protocol takes great advantage of the cyclic structure of matrices. In the same fashion as [1], $\mathbf{rot}(\mathbf{h})$ for $\mathbf{h} \in \mathcal{V}$ denotes the circulant matrix whose $i^{\text{th}}$ column is the vector corresponding to $\mathbf{h}X^i$. This is captured by the following definition.

**Definition 1.1.1** (Circulant Matrix)**.** *Let* $\mathbf{v} = (v_0, \ldots, v_{n-1}) \in \mathbb{F}_2^n$*. The* circulant matrix *induced by* $\mathbf{v}$ *is defined and denoted as follows:*

$$\mathbf{rot}(\mathbf{v}) = \begin{pmatrix} v_0 & v_{n-1} & \ldots & v_1 \\ v_1 & v_0 & \ldots & v_2 \\ \vdots & \vdots & \ddots & \vdots \\ v_{n-1} & v_{n-2} & \ldots & v_0 \end{pmatrix} \in \mathbb{F}_2^{n \times n} \tag{2}$$

As a consequence, it is easy to see that the product of any two elements $\mathbf{u}, \mathbf{v} \in \mathcal{R}$ can be expressed as a usual vector-matrix (or matrix-vector) product using the $\mathbf{rot}(\cdot)$ operator as

$$\mathbf{u} \cdot \mathbf{v} = \mathbf{u} \times \mathbf{rot}(\mathbf{v})^\top = \left(\mathbf{rot}(\mathbf{u}) \times \mathbf{v}^\top\right)^\top = \mathbf{v} \times \mathbf{rot}(\mathbf{u})^\top = \mathbf{v} \cdot \mathbf{u}. \tag{3}$$

**Coding Theory.** We now recall some basic definitions and properties about coding theory that will be useful to our construction. We mainly focus on general definitions, and refer the reader to Sec. 1.3 the description of the scheme, and also to [24] for a complete survey on code-based cryptography.

**Definition 1.1.2** (Linear Code)**.** *A* Linear Code $\mathcal{C}$ *of length* $n$ *and dimension* $k$ *(denoted* $[n, k]$*) is a subspace of* $\mathcal{R}$ *of dimension* $k$*. Elements of* $\mathcal{C}$ *are referred to as codewords.*

**Definition 1.1.3** (Generator Matrix)**.** *We say that* $\mathbf{G} \in \mathbb{F}_2^{k \times n}$ *is a* Generator Matrix *for the* $[n, k]$ *code* $\mathcal{C}$ *if*

$$\mathcal{C} = \left\{\mathbf{mG}, \text{ for } \mathbf{m} \in \mathbb{F}_2^k\right\}. \tag{4}$$

**Definition 1.1.4** (Parity-Check Matrix)**.** *Given an* $[n, k]$ *code* $\mathcal{C}$*, we say that* $\mathbf{H} \in \mathbb{F}_2^{(n-k) \times n}$ *is a* Parity-Check Matrix *for* $\mathcal{C}$ *if* $\mathbf{H}$ *is a generator matrix of the dual code* $\mathcal{C}^\perp$*, or more formally, if*

$$\mathcal{C} = \left\{\mathbf{v} \in \mathbb{F}_2^n \text{ such that } \mathbf{Hv}^\top = \mathbf{0}\right\}, \text{ or equivalently } \mathcal{C}^\perp = \left\{\mathbf{uH}, \text{ for } \mathbf{u} \in \mathbb{F}_2^{n-k}\right\}. \tag{5}$$

**Definition 1.1.5** (Syndrome)**.** *Let* $\mathbf{H} \in \mathbb{F}_2^{(n-k) \times n}$ *be a parity-check matrix of some* $[n, k]$ *code* $\mathcal{C}$*, and* $\mathbf{v} \in \mathbb{F}_2^n$ *be a word. Then the* syndrome *of* $\mathbf{v}$ *is* $\mathbf{Hv}^\top$*, and we have* $\mathbf{v} \in \mathcal{C} \Leftrightarrow \mathbf{Hv}^\top = \mathbf{0}$*.*

**Definition 1.1.6** (Minimum Distance)**.** *Let* $\mathcal{C}$ *be an* $[n, k]$ *linear code over* $\mathcal{R}$ *and let* $\omega$ *be a norm on* $\mathcal{R}$*. The* Minimum Distance *of* $\mathcal{C}$ *is*

$$d = \min_{\mathbf{u}, \mathbf{v} \in \mathcal{C}, \mathbf{u} \neq \mathbf{v}} \omega(\mathbf{u} - \mathbf{v}). \tag{6}$$

A code with minimum distance $d$ is capable of decoding arbitrary patterns of up to $\delta = \lfloor \frac{d-1}{2} \rfloor$ errors. Code parameters are denoted $[n, k, d]$.

Code-based cryptography usually suffers from huge keys. In order to keep our cryptosystem efficient, we will use the strategy of Gaborit [18] for shortening keys. This results in Quasi-Cyclic Codes, as defined below.

**Definition 1.1.7** (Quasi-Cyclic Codes [34])**.** *View a vector* $\mathbf{c} = (\mathbf{c}_0, \ldots, \mathbf{c}_{s-1})$ *of* $\mathbb{F}_2^{sn}$ *as* $s$ *successive blocks (*$n$*-tuples). An* $[sn, k, d]$ *linear code* $\mathcal{C}$ *is* Quasi-Cyclic (QC) *of index* $s$ *if, for any* $\mathbf{c} = (\mathbf{c}_0, \ldots, \mathbf{c}_{s-1}) \in \mathcal{C}$*, the vector obtained after applying a simultaneous circular shift to every block* $\mathbf{c}_0, \ldots, \mathbf{c}_{s-1}$ *is also a codeword.*

*More formally, by considering each block* $\mathbf{c}_i$ *as a polynomial in* $\mathcal{R} = \mathbb{F}_2[X]/(X^n - 1)$*, the code* $\mathcal{C}$ *is QC of index* $s$ *if for any* $\mathbf{c} = (\mathbf{c}_0, \ldots, \mathbf{c}_{s-1}) \in \mathcal{C}$ *it holds that* $(X \cdot \mathbf{c}_0, \ldots, X \cdot \mathbf{c}_{s-1}) \in \mathcal{C}$*.*

**Definition 1.1.8** (Systematic Quasi-Cyclic Codes)**.** *A* systematic *Quasi-Cyclic* $[sn, n]$ *code of index* $s$ *and rate* $1/s$ *is a quasi-cyclic code with an* $(s-1)n \times sn$ *parity-check matrix of the form:*

$$\mathbf{H} = \begin{bmatrix} \mathbf{I}_n & 0 & \cdots & 0 & \mathbf{A}_0 \\ 0 & \mathbf{I}_n & & & \mathbf{A}_1 \\ & & \ddots & & \vdots \\ 0 & & \cdots & \mathbf{I}_n & \mathbf{A}_{s-2} \end{bmatrix} \tag{7}$$

*where* $\mathbf{A}_0, \ldots, \mathbf{A}_{s-2}$ *are circulant* $n \times n$ *matrices.*

**Remark 1.1.** *The definition of systematic quasi-cyclic codes of index* $s$ *can of course be generalized to all rates* $\ell/s$*,* $\ell = 1 \ldots s-1$*, but we shall only use systematic QC-codes of rates* $1/2$ *and* $1/3$ *and wish to lighten notation with the above definition. In the sequel, referring to a systematic QC-code will imply by default that it is of rate* $1/s$*. Note that arbitrary QC-codes are not necessarily equivalent to a systematic QC-code.*

### 1.1.2 Difficult problems for cryptography

In this section we describe difficult problems which can be used for cryptography and discuss their complexity.

All problems are variants of the *decoding problem*, which consists of looking for the closest codeword to a given vector: when dealing with linear codes, it is readily seen that the decoding problem stays the same when one is given the *syndrome* of the received vector rather than the received vector. We therefore speak of *Syndrome Decoding* (SD).

**Definition 1.1.9** (SD Distribution)**.** *For positive integers* $n$*,* $k$*, and* $w$*, the* $\mathsf{SD}(n, k, w)$ Distribution *chooses* $\mathbf{H} \xleftarrow{\$} \mathbb{F}_2^{(n-k)\times n}$ *and* $\mathbf{x} \xleftarrow{\$} \mathbb{F}_2^n$ *such that* $\omega(\mathbf{x}) = w$*, and outputs* $(\mathbf{H}, \sigma(\mathbf{x}) = \mathbf{H}\mathbf{x}^\top)$*.*

**Definition 1.1.10** (Search SD Problem)**.** *On input* $(\mathbf{H}, \mathbf{y}^\top) \in \mathbb{F}_2^{(n-k)\times n} \times \mathbb{F}_2^{(n-k)}$ *from the SD distribution, the* Syndrome Decoding Problem $\mathsf{SD}(n, k, w)$ *asks to find* $\mathbf{x} \in \mathbb{F}_2^n$ *such that* $\mathbf{H}\mathbf{x}^\top = \mathbf{y}^\top$ *and* $\omega(\mathbf{x}) = w$*.*

For the Hamming distance the SD problem has been proven NP-complete [4]. This problem can also be seen as the Learning Parity with Noise (LPN) problem with a fixed number of samples [2]. For cryptography we also need a decision version of the problem, which is given in the following definition.

**Definition 1.1.11** (Decision SD Problem)**.** *On input* $(\mathbf{H}, \mathbf{y}^\top) \in \mathbb{F}_2^{(n-k)\times n} \times \mathbb{F}_2^{(n-k)}$*, the* Decision SD Problem $\mathsf{DSD}(n, k, w)$ *asks to decide with non-negligible advantage whether* $(\mathbf{H}, \mathbf{y}^\top)$ *came from the* $\mathsf{SD}(n, k, w)$ *distribution or the uniform distribution over* $\mathbb{F}_2^{(n-k)\times n} \times \mathbb{F}_2^{(n-k)}$*.*

As mentioned above, this problem is the problem of decoding random linear codes from random errors. The random errors are often taken as independent Bernoulli variables acting independently on vector coordinates, rather than uniformly chosen from the set of errors of a given weight, but this hardly makes any difference and one model rather than the other is a question of convenience. The DSD problem has been shown to be polynomially equivalent to its search version in [2].

Finally, as our cryptosystem will use QC-codes, we explicitly define the problem on which our cryptosystem will rely. The following definitions describe the DSD problem in the QC configuration, and are just a combination of Def. 1.1.7 and 1.1.11. Quasi-Cyclic codes are very useful in cryptography since their compact description allows to decrease considerably the size of the keys. In particular the case $s = 2$ corresponds to double circulant codes with generator matrices of the form $(\mathbf{I}_n \ \mathbf{A})$ for $\mathbf{A}$ a circulant matrix. Such double circulant codes have been used for almost 10 years in cryptography (cf [19]) and more recently in [34]. Quasi-cyclic codes of index 3 are also considered in [34].

**Definition 1.1.12** ($s$-QCSD Distribution)**.** *For positive integers* $n$*,* $w$ *and* $s$*, the* $s$-$\mathsf{QCSD}(n, w)$ Distribution *chooses uniformly at random a parity-check matrix* $\mathbf{H} \xleftarrow{\$} \mathbb{F}_2^{(sn-n)\times sn}$ *of a systematic QC code* $\mathcal{C}$ *of index* $s$ *and rate* $1/s$ *(see Def. 1.1.8) together with a vector* $\mathbf{x} = (\mathbf{x}_0, \ldots, \mathbf{x}_{s-1}) \xleftarrow{\$} \mathbb{F}_2^{sn}$ *such that* $\omega(\mathbf{x}_i) = w$*,* $i = 0..s-1$*, and outputs* $(\mathbf{H}, \mathbf{H}\mathbf{x}^\top)$*.*

**Definition 1.1.13** ((Search) $s$-QCSD Problem)**.** *For positive integers* $n$*,* $w$*,* $s$*, a random parity check matrix* $\mathbf{H}$ *of a systematic QC code* $\mathcal{C}$ *of index* $s$ *and* $\mathbf{y} \xleftarrow{\$} \mathbb{F}_2^{sn-n}$*, the* Search $s$-Quasi-Cyclic SD Problem $s$-$\mathsf{QCSD}(n, w)$ *asks to find* $\mathbf{x} = (\mathbf{x}_0, \ldots, \mathbf{x}_{s-1}) \in \mathbb{F}_2^{sn}$ *such that* $\omega(\mathbf{x}_i) = w$*,* $i = 0..s-1$*, and* $\mathbf{y} = \mathbf{x}\mathbf{H}^\top$*.*

It would be somewhat more natural to choose the parity-check matrix $\mathbf{H}$ to be made up of independent uniformly random circulant submatrices, rather than with the special form required by (7). We choose this distribution so as to make the security reduction to follow less technical. It is readily seen that, for fixed $s$, when choosing quasi-cyclic codes with this more general distribution, one obtains with non-negligible probability, a quasi-cyclic code that admits a parity-check matrix of the form (7). Therefore requiring quasi-cyclic codes to be systematic does not hurt the generality of the decoding problem for quasi-cyclic codes. A similar remark holds for the slightly special form of weight distribution of the vector $\mathbf{x}$.

**Assumption 1.** *Although there is no general complexity result for quasi-cyclic codes, decoding these codes is considered hard by the community. There exist general attacks which uses the cyclic structure of the code [38] but these attacks have only a very limited impact on the practical complexity of the problem. The conclusion is that in practice, the best attacks are the same as those for non-circulant codes up to a small factor.*

The problem also has a decision version. In order to avoid trivial distinguishers, an additional condition on the parity of the syndrome needs to be appended. For $b \in \{0,1\}$, we define the finite set $\mathbb{F}_{2,b}^n = \{\mathbf{h} \in \mathbb{F}_2^n \text{ s.t. } \mathbf{h}(1) = b \mod 2\}$, *i.e.* binary vectors of length $n$ and parity $b$. Similarly for matrices, we define the finite sets

$$\mathbb{F}_{2,b}^{n\times 2n} = \left\{\mathbf{H} = (\mathbf{I}_n \ \mathbf{rot}(\mathbf{h})) \in \mathbb{F}_2^{n\times 2n} \text{ s.t. } \mathbf{h} \in \mathbb{F}_{2,b}^n\right\}, \text{ and}$$

$$\mathbb{F}_{2,b_1,b_2}^{2n\times 3n} = \left\{\mathbf{H} = \begin{pmatrix} \mathbf{I}_n & \mathbf{0} & \mathbf{rot}(\mathbf{h}_1) \\ \mathbf{0} & \mathbf{I}_n & \mathbf{rot}(\mathbf{h}_2) \end{pmatrix} \in \mathbb{F}_2^{2n\times 3n} \text{ s.t. } \mathbf{h}_1 \in \mathbb{F}_{2,b_1}^n \text{ and } \mathbf{h}_2 \in \mathbb{F}_{2,b_2}^n\right\}.$$

This is pure technicality and does not affect the parameters of our proposal. Meanwhile, this trick permits to discard attacks such as [20, 29, 30][1]. The authors are grateful to Ray Perlner for pointing out the existence of such a distinguisher.

**Definition 1.1.14** (2-QCSD Distribution (with parity))**.** *For positive integers $n$, $w$ and $b$, the* 2-$\mathsf{QCSD}(n,w,b)$ Distribution with parity *chooses uniformly at random a parity-check matrix $\mathbf{H} \in \mathbb{F}_{2,b}^{n\times 2n}$ together with a vector $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2) \xleftarrow{\$} \mathbb{F}_2^{2n}$ such that $\omega(\mathbf{x}_1) = \omega(\mathbf{x}_2) = w$, and outputs $(\mathbf{H}, \mathbf{H}\mathbf{x}^\top)$.*

**Definition 1.1.15** (Decision 2-QCSD Problem (with parity))**.** *Let $\mathbf{h} \in \mathbb{F}_{2,b}^n$, $\mathbf{H} = (\mathbf{I}_n \ \mathbf{rot}(\mathbf{h}))$, and $b' = w + b \times w \mod 2$. For $\mathbf{y} \in \mathbb{F}_{2,b'}^n$, the* Decision 2-Quasi-Cyclic $\mathsf{SD}$ Problem with parity 2-$\mathsf{DQCSD}(n,w,b)$ *asks to decide with non-negligible advantage whether $(\mathbf{H}, \mathbf{y})$ came from the 2-$\mathsf{QCSD}(n,w,b)$ distribution with parity or the uniform distribution over $\mathbb{F}_{2,b}^{n\times 2n} \times \mathbb{F}_{2,b'}^n$.*

In order to fully explicit the problems upon which HQC relies, we also define the 3-$\mathsf{DQCSD}$ problem with parity. Following Def. 1.1.8, the $s$-$\mathsf{DQCSD}$ problem with parity can be easily generalized to higher $s \geq 3$, but we avoid such a description for the sake of clarity.

**Definition 1.1.16** (3-QCSD Distribution (with parity))**.** *For positive integers $n$, $w$, $b_1$ and $b_2$, the* 3-$\mathsf{QCSD}(n,w,b_1,b_2)$ Distribution with parity *chooses uniformly at random a parity-check matrix $\mathbf{H} \in \mathbb{F}_{2,b_1,b_2}^{2n\times 3n}$ together with a vector $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3) \xleftarrow{\$} \mathbb{F}_2^{3n}$ such that $\omega(\mathbf{x}_1) = \omega(\mathbf{x}_2) = \omega(\mathbf{x}_3) = w$, and outputs $(\mathbf{H}, \mathbf{H}\mathbf{x}^\top)$.*

[1] The authors chose to use a parity version of the $\mathsf{DQCSD}$ problem rather than a variable weight version as suggested in [30] for efficiency issues.

**Definition 1.1.17** (Decision 3-QCSD Problem (with parity))**.** *Let* $\mathbf{h}_1 \in \mathbb{F}_{2,b_1}^n, \mathbf{h}_2 \in \mathbb{F}_{2,b_2}^n$, $\mathbf{H} = \begin{pmatrix} \mathbf{I}_n & \mathbf{0} & \mathbf{rot}(\mathbf{h}_1) \\ \mathbf{0} & \mathbf{I}_n & \mathbf{rot}(\mathbf{h}_2) \end{pmatrix}$, $b_1' = w + b_1 \times w \mod 2$ *and* $b_2' = w + b_2 \times w \mod 2$. *For* $(\mathbf{y}_1, \mathbf{y}_2) \in \mathbb{F}_{2,b_1'}^n \times \mathbb{F}_{2,b_2'}^n$, *the* Decision 3-Quasi-Cyclic SD Problem with parity 3-DQCSD$(n, w, b_1, b_2)$ *asks to decide with non-negligible advantage whether* $(\mathbf{H}, (\mathbf{y}_1, \mathbf{y}_2))$ *came from the* 3-*QCSD*$(n, w, b_1, b_2)$ *distribution with parity or the uniform distribution over* $\mathbb{F}_{2,b_1,b_2}^{2n \times 3n} \times \left(\mathbb{F}_{2,b_1'}^n \times \mathbb{F}_{2,b_2'}^n\right)$.

As for the ring-LPN problem, there is no known reduction from the search version of $s$-QCSD problem to its decision version. The proof of [2] cannot be directly adapted in the quasi-cyclic case, however the best known attacks on the decision version of the $s$-QCSD problem remain the direct attacks on the search version.

The IND-CPA security of HQC essentially relies on the hardness of the 2 and 3-DQCSD problems described above (Def. 1.1.15 and 1.1.17). However, in order to thwart structural attacks, we need to work with a code of primitive prime length $n$, so that $X^n - 1$ has only two irreducible factors mod $q$. But for parameters and codes considered in the proposed instantiation (BCH codes tensored with a repetition code), the encoding of a message $\mathbf{m}$ has size $n_1 n_2$, which is obviously not prime. Therefore we use as ambient length $n = \text{next\_primitive\_prime}\,(n_1 n_2)$, the first primitive prime greater than $n_1 n_2$, and truncate the last $\ell = n - n_1 n_2$ bits wherever needed. This results in a slightly modified version of the DQCSD problem, that we will argue to be at least as hard as the original ones. We first define this truncated version in its primal version.

**Definition 1.1.18** (Decoding with $\ell$ erasures)**.** *Let* $\mathcal{C}[n, k]$ *be a QC-code generated by* $\mathbf{G}$ *and* $\mathbf{c} = \mathbf{mG} + \mathbf{e}$ *for some random* $\mathbf{e} \xleftarrow{\$} \mathcal{S}_w^n(\mathbb{F}_2)$. *Consider the matrix* $\mathbf{G}' \in \mathbb{F}_2^{k \times n'}$ *(resp. vector* $\mathbf{e}' \in \mathbb{F}_2^{n'}$*) obtained by removing the last* $\ell = n - n' \geq 1$ *columns from* $\mathbf{G}$ *(resp.* $\mathbf{e}$*).*

*The Decoding with* $\ell$ *erasures problem asks to recover* $\mathbf{m} \in \mathbb{F}_2^k$ *from* $\mathbf{c}' = \mathbf{mG}' + \mathbf{e}' \in \mathbb{F}_2^{n'}$ *and* $\mathbf{G}' \in \mathbb{F}_2^{k \times n'}$.

Conceptually speaking, the above problem asks to recover the encoded message, given less information. It then becomes obvious that Decoding with erasures is harder than with full knowledge of the encoding. Assume that $\mathcal{A}$ can solve the decoding problem with $\ell$ erasures, and let $(\mathbf{c}, \mathbf{G})$ be an instance of the decoding problem with no erasure. One starts by removing the last $\ell$ columns from $\mathbf{c}$ and $\mathbf{G}$, then uses $\mathcal{A}$ to recover $\mathbf{m} \in \mathbb{F}_2^k$. Since the dimension is unchanged in both problems, $\mathbf{m}$ is also solution to the decoding problem with no erasure, which confirms the hardness statement.

As the decoding problem and the syndrome decoding problem are equivalent, the argument previously exposed applies. Therefore the corresponding 2 and 3-DQCSD problems with $\ell = n - n_1 n_2$ erasures obtained to avoid structural attacks are at least as hard as those defined in Def. 1.1.15 and 1.1.17 above. (In Sec. 4, $n = \text{next\_primitive\_prime}\,(n_1 n_2) > n_1 n_2$).

## 1.2 Encryption and security

**Encryption Scheme.** An encryption scheme is a tuple of four polynomial time algorithms $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$:

- $\mathsf{Setup}(1^\lambda)$, where $\lambda$ is the security parameter, generates the global parameters $\mathsf{param}$ of the scheme;
- $\mathsf{KeyGen}(\mathsf{param})$ outputs a pair of keys, a (public) encryption key $\mathsf{pk}$ and a (private) decryption key $\mathsf{sk}$;
- $\mathsf{Encrypt}(\mathsf{pk}, \mathbf{m}, \theta)$ outputs a ciphertext $\mathbf{c}$, on the message $\mathbf{m}$, under the encryption key $\mathsf{pk}$, with the randomness $\theta$. We also use $\mathsf{Encrypt}(\mathsf{pk}, \mathbf{m})$ for the sake of clarity;
- $\mathsf{Decrypt}(\mathsf{sk}, \mathbf{c})$ outputs the plaintext $\mathbf{m}$, encrypted in the ciphertext $\mathbf{c}$ or $\perp$.

Such an encryption scheme has to satisfy both *Correctness* and *Indistinguishability under Chosen Plaintext Attack* (IND-CPA) security properties.

**Correctness**: For every $\lambda$, every $\mathsf{param} \leftarrow \mathsf{Setup}(1^\lambda)$, every pair of keys $(\mathsf{pk}, \mathsf{sk})$ generated by $\mathsf{KeyGen}$, every message $\mathbf{m}$, we should have $\Pr[\mathsf{Decrypt}(\mathsf{sk}, \mathsf{Encrypt}(\mathsf{pk}, \mathbf{m}, \theta)) = \mathbf{m}] = 1 - \mathsf{negl}(\lambda)$ for $\mathsf{negl}(\cdot)$ a negligible function, where the probability is taken over varying randomness $\theta$.

**IND-CPA** [21]: This notion formalized by the game depicted in Fig. 1, states that an adversary should not be able to efficiently guess which plaintext has been encrypted even if he knows it is one among two plaintexts of his choice.

$\mathbf{Exp}^{\mathsf{ind}-b}_{\mathcal{E},\mathcal{A}}(\lambda)$
1. $\mathsf{param} \leftarrow \mathsf{Setup}(1^\lambda)$
2. $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{param})$
3. $(\mathbf{m}_0, \mathbf{m}_1) \leftarrow \mathcal{A}(\texttt{FIND} : \mathsf{pk})$
4. $\mathbf{c}^* \leftarrow \mathsf{Encrypt}(\mathsf{pk}, \mathbf{m}_b, \theta)$
5. $b' \leftarrow \mathcal{A}(\texttt{GUESS} : \mathbf{c}^*)$
6. `RETURN` $b'$

Figure 1: Game for the IND-CPA security of an asymmetric encryption scheme.

In the following, we denote by $|\mathcal{A}|$ the running time of an adversary $\mathcal{A}$. The global advantage for polynomial time adversaries running in time less than $t$ is:

$$\mathsf{Adv}^{\mathsf{ind}}_{\mathcal{E}}(\lambda, t) = \max_{|\mathcal{A}| \leq t} \mathsf{Adv}^{\mathsf{ind}}_{\mathcal{E},\mathcal{A}}(\lambda), \tag{8}$$

where $\mathsf{Adv}^{\mathsf{ind}}_{\mathcal{E},\mathcal{A}}(\lambda)$ is the advantage the adversary $\mathcal{A}$ has in winning game $\mathbf{Exp}^{\mathsf{ind}-b}_{\mathcal{E},\mathcal{A}}(\lambda)$:

$$\mathsf{Adv}^{\mathsf{ind}}_{\mathcal{E},\mathcal{A}}(\lambda) = \left|\Pr[\mathbf{Exp}^{\mathsf{ind}-1}_{\mathcal{E},\mathcal{A}}(\lambda) = 1] - \Pr[\mathbf{Exp}^{\mathsf{ind}-0}_{\mathcal{E},\mathcal{A}}(\lambda) = 1]\right|. \tag{9}$$

**IND-CPA, IND-CCA2 and Hybrid Encryption.** Note that the standard (highest) security requirement for a public key cryptosystem is *indistinguishability against adaptive chosen-ciphertext attacks* (IND-CCA2), and not just IND-CPA. The main difference is that for IND-CCA2, indistinguishability must hold even if the attacker is given a *decryption oracle* first when running the `FIND` algorithm and also when running the `GUESS` algorithm (but cannot query the oracle on the challenge ciphertext $\boldsymbol{c}^*$). We do not present the associated formal game and definition as an existing (and inexpensive) transformation can be used [23] for our scheme to pass from IND-CPA to IND-CCA2. Various generic techniques transforming an IND-CPA scheme into an IND-CCA2 scheme are known [16, 17, 35, 12] but cannot be applied to our scheme due to potential decryption errors.

In [23] Hofheinz et al. present a generic transformation that takes into account decryption errors and can be applied directly to our scheme. Roughly, their construction provides a way to convert a guarantee against passive adversaries into indistinguishability against active ones by turning a public key cryptosystem into a KEM-DEM. The tightness (the quality factor) of the reduction depends on the ciphertext distribution. Regarding our scheme, random words only have a negligible (in the security parameter) probability of being valid ciphertexts. In other words, the $\gamma$-spreadness factor of [23] is small enough so that there is no loss between the IND-CPA security of our public key cryptosystem and the IND-CCA2 security of the KEM-DEM version presented in Fig. 3.

The security reduction is tight in the random oracle model and does not require any supplemental property from our scheme as we have the IND-CPA property (instead of just a weaker property called *One-Wayness*). Let us denote by $\mathsf{Encrypt}(\mathsf{pk}, \mathbf{m}, \theta)$ the encryption function defined in Fig. 2 that uses randomness $\theta$ to generate uniformly random values $\mathbf{r}_1$, $\mathbf{r}_2$, and $\mathbf{e}$. The idea of [23] transformation is to de-randomize the encryption function $\mathsf{Encrypt}(\mathsf{pk}, \mathbf{m}, \theta)$ by using a hash function $\mathcal{G}$ and do a deterministic encryption of $\mathbf{m}$ by calling $c = \mathsf{Encrypt}(\mathsf{pk}, \mathbf{m}, \mathcal{G}(\mathbf{m}))$. The ciphertext is sent together with a hash $K = \mathcal{H}(\mathbf{c}, \mathbf{m})$ that ties the ciphertext to the plaintext. The receiver then decrypts $\mathbf{c}$ into $\mathbf{m}$, checks the hash value, and uses again the deterministic encryption to check that $\mathbf{c}$ is indeed *the* ciphertext associated to $\mathbf{m}$.

As the reduction is tight we do not need to change our parameters when we pass from IND-CPA to IND-CCA2. From a computational point of view, the overhead for the sender is two hash calls and for the receiver it is two hash calls and an encrypt call. From a communication point of view the overhead is the bitsize of a hash (or two if the reduction must hold in the Quantum Random Oracle Model, see [23] for more details).

Note that there is currently a lot of research activity around generic transformations from IND-CPA (or OW-CPA) PKE to IND-CCA2 KEM [23, 37, 25, 7, 26] with very few feedback. While it is possible to use state-of-the-art conversions to make HQC IND-CCA2 secure in the QROM with limited computational and bandwidth overhead (using the $FO^{\not\perp}$ transform in [25] for instance), we chose to keep the presentation of HQC using [23] in order to avoid moving target for NIST evaluation. Any other conversion can be implemented simply.

## 1.3 Presentation of the scheme

In this section, we describe our proposal: HQC. We begin with the PKE version, then describe the transformation of [23] to obtain a KEM-DEM that achieves IND-CCA2. Parameter sets can be found in Sec. 1.7.

### 1.3.1 Public key encryption version (HQC.PKE)

**Presentation of the scheme.** HQC uses two types of codes: a decodable $[n, k]$ code $\mathcal{C}$, generated by $\mathbf{G} \in \mathbb{F}_2^{k\times n}$ and which can correct at least $\delta$ errors via an efficient algorithm $\mathcal{C}.\mathsf{Decode}(\cdot)$; and a random double-circulant $[2n, n]$ code, of parity-check matrix $(\mathbf{1}, \mathbf{h})$. The four polynomial-time algorithms constituting our scheme are depicted in Fig. 2.

- $\mathsf{Setup}(1^\lambda)$: generates and outputs the global parameters $\mathsf{param} = (n, k, \delta, w, w_{\mathbf{r}}, w_{\mathbf{e}})$.
- $\mathsf{KeyGen}(\mathsf{param})$: samples $\mathbf{h} \xleftarrow{\$} \mathcal{R}$, the generator matrix $\mathbf{G} \in \mathbb{F}_2^{k\times n}$ of $\mathcal{C}$, $\mathsf{sk} = (\mathbf{x}, \mathbf{y}) \xleftarrow{\$} \mathcal{R}^2$ such that $\omega(\mathbf{x}) = \omega(\mathbf{y}) = w$, sets $\mathsf{pk} = (\mathbf{h}, \mathbf{s} = \mathbf{x} + \mathbf{h} \cdot \mathbf{y})$, and returns $(\mathsf{pk}, \mathsf{sk})$.
- $\mathsf{Encrypt}(\mathsf{pk}, \mathbf{m})$: generates $\mathbf{e} \xleftarrow{\$} \mathcal{R}$, $\mathbf{r} = (\mathbf{r}_1, \mathbf{r}_2) \xleftarrow{\$} \mathcal{R}^2$ such that $\omega(\mathbf{e}) = w_{\mathbf{e}}$ and $\omega(\mathbf{r}_1) = \omega(\mathbf{r}_2) = w_{\mathbf{r}}$, sets $\mathbf{u} = \mathbf{r}_1 + \mathbf{h} \cdot \mathbf{r}_2$ and $\mathbf{v} = \mathbf{mG} + \mathbf{s} \cdot \mathbf{r}_2 + \mathbf{e}$, returns $\mathbf{c} = (\mathbf{u}, \mathbf{v})$.
- $\mathsf{Decrypt}(\mathsf{sk}, \mathbf{c})$: returns $\mathcal{C}.\mathsf{Decode}(\mathbf{v} - \mathbf{u} \cdot \mathbf{y})$.

Figure 2: Description of our proposal HQC.PKE.

Notice that the generator matrix $\mathbf{G}$ of the code $\mathcal{C}$ is publicly known, so the security of the scheme and the ability to decrypt do not rely on the knowledge of the error correcting code $\mathcal{C}$ being used.

Also notice that the code $\mathcal{C}$ is instantiated using tensor codes: BCH and repetition codes (see Section 1.5.1 for more details). The BCH codes are defined implicitly using their generator polynomials. In fact, as explained in Section 1.5.2, by exploiting the algebraic properties of BCH codes, one can use the generator polynomial to compute code words without computing the generator matrix explicitly. In order to keep the description of the scheme simple, the matrix form of the codes is used. Furthermore, we will have $\mathbf{G} \in \mathbb{F}_2^{n_1 n_2}$ and $\mathbf{h} \in \mathbb{F}_2^n$, with $n$ the smallest primitive prime greater than $n_1 n_2$. All computations are made in the ambient space $\mathbb{F}_2^n$ and the remaining $\ell = n - n_1 n_2$ bits are truncated where useless.

In particular, the ciphertext will be $\left(\mathbf{u}, \bar{\mathbf{v}}^{(\ell)}\right)$, where $\bar{\mathbf{v}}^{(\ell)}$ denotes the $\ell$ first coordinates (bits) of $\mathbf{v}$. For sake of readability, we keep the notation $\mathbf{v}$ even for the truncated vector, and explicitly mention the length of the vectors.

**Correctness.** The correctness of our encryption scheme clearly relies on the decoding capability of the code $\mathcal{C}$. Specifically, assuming $\mathcal{C}.\mathsf{Decode}$ correctly decodes $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$, we

have:

$$\mathsf{Decrypt}\left(\mathsf{sk}, \mathsf{Encrypt}\left(\mathsf{pk}, \mathbf{m}\right)\right) = \mathbf{m}. \tag{10}$$

And $\mathcal{C}.\mathsf{Decode}$ correctly decodes $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$ whenever

$$\omega\left(\mathbf{s} \cdot \mathbf{r}_2 - \mathbf{u} \cdot \mathbf{y} + \mathbf{e}\right) \leq \delta \tag{11}$$

$$\omega\left((\mathbf{x} + \mathbf{h} \cdot \mathbf{y}) \cdot \mathbf{r}_2 - (\mathbf{r}_1 + \mathbf{h} \cdot \mathbf{r}_2) \cdot \mathbf{y} + \mathbf{e}\right) \leq \delta \tag{12}$$

$$\omega\left(\mathbf{x} \cdot \mathbf{r}_2 - \mathbf{r}_1 \cdot \mathbf{y} + \mathbf{e}\right) \leq \delta \tag{13}$$

In order to provide an upper bound on the decryption failure probability, an analysis of the distribution of the error vector $\mathbf{e}' = \mathbf{x} \cdot \mathbf{r}_2 - \mathbf{r}_1 \cdot \mathbf{y} + \mathbf{e}$ is provided in Sec. 1.4.

### 1.3.2 KEM/DEM version (HQC.KEM)

Let $\mathcal{E}$ be an instance of the HQC cryptosystem as described above. Let $\mathcal{G}$, $\mathcal{H}$, and $\mathcal{K}$ be hash functions, typically SHA512 as advised by NIST[2]. The KEM-DEM version of the HQC cryptosystem is defined as follows:

- $\mathsf{Setup}(1^\lambda)$: as before, except that $k$ will be the length of the symmetric key being exchanged, typically $k = 256$.
- $\mathsf{KeyGen}(\mathsf{param})$: exactly as before.
- $\mathsf{Encapsulate}(\mathsf{pk})$: generate $\mathbf{m} \xleftarrow{\$} \mathbb{F}_2^k$ (this will serve as a seed to derive the shared key). Derive the randomness $\theta \leftarrow \mathcal{G}(\mathbf{m})$. Generate the ciphertext $c \leftarrow (\mathbf{u}, \mathbf{v}) = \mathcal{E}.\mathsf{Encrypt}(\mathsf{pk}, \mathbf{m}, \theta)$, and derive the symmetric key $K \leftarrow \mathcal{K}(\mathbf{m}, \mathbf{c})$. Let $\mathbf{d} \leftarrow \mathcal{H}(\mathbf{m})$, and send $(\mathbf{c}, \mathbf{d})$.
- $\mathsf{Decapsulate}(\mathsf{sk}, \mathbf{c}, \mathbf{d})$: Decrypt $\mathbf{m}' \leftarrow \mathcal{E}.\mathsf{Decrypt}(\mathsf{sk}, \mathbf{c})$, compute $\theta' \leftarrow \mathcal{G}(\mathbf{m}')$, and (re-)encrypt $\mathbf{m}'$ to get $\mathbf{c}' \leftarrow \mathcal{E}.\mathsf{Encrypt}(\mathsf{pk}, \mathbf{m}', \theta')$. If $\mathbf{c} \neq \mathbf{c}'$ or $\mathbf{d} \neq \mathcal{H}(\mathbf{m}')$ then abort. Otherwise, derive the shared key $K \leftarrow \mathcal{K}(\mathbf{m}, \mathbf{c})$.

Figure 3: Description of our proposal HQC.KEM.

According to [23], the KEM-DEM version of HQC is IND-CCA2. More details regarding the tightness of the reduction are provided at the end of Sec. 1.7.

**Security concerns and implementation details.** Notice that while NIST only recommends SHA512 as a hash function (or TupleHash256 for hardware efficiency purposes), the transformation of [23] would be dangerous – at least in our setting – if one sets $\mathcal{G} = \mathcal{H}$. Indeed, publishing the randomness $\theta = \mathcal{G}(\mathbf{m}) = \mathcal{H}(\mathbf{m}) = \mathbf{d}$ used to generate $\mathbf{r}_1$, $\mathbf{r}_2$, and $\mathbf{e}$ would allow an eavesdropper to retrieve $\mathbf{m}$ from $\mathbf{mG} + \mathbf{sr}_2 + \mathbf{e}$ and hence, the seed for the shared secret key.

We therefore suggest to use a pseudo-random function for $\mathcal{G}$, such as an AES-based seed expander, and SHA512 for $\mathcal{H}$.

[2]See Dustin Moody's mail entitled "new FAQ question" on PQC-forum (20/07/2017 – 12:58 CET)

### 1.3.3 A hybrid encryption scheme (HQC.HE)

While NIST claimed that they will be using generic transformations to convert any IND-CCA2 KEM into an IND-CCA2 PKE, no detail on these conversions have been provided. We therefore refer to HQC.HE to designate the PKE scheme resulting from applying a generic conversion to HQC.KEM.

## 1.4 Analysis of the error vector distribution for Hamming distance

The aim of this section is to determine the probability that the condition in Eq. (13) holds. In order to do so, we study the error distribution of the error vector $\mathbf{e}' = \mathbf{x} \cdot \mathbf{r}_2 - \mathbf{r}_1 \cdot \mathbf{y} + \mathbf{e}$.

The vectors $\mathbf{x}, \mathbf{y}, \mathbf{r}_1, \mathbf{r}_2, \mathbf{e}$ have been taken uniformly random and independently chosen among vectors of weight $w$, $w_{\mathbf{r}}$ or $w_{\mathbf{e}}$. This distribution can be closely approximated by a binomial distribution $\mathcal{B}$, where a vector consists of $n$ Bernoulli variables of parameter $p = w/n$ (or $p_{\mathbf{r}} = w_{\mathbf{r}}/n$ and $p_{\mathbf{e}} = w_{\mathbf{e}}/n$ respectively). In other words, $\mathcal{S}_w^n(\mathbb{F}_2)$ is close to $\mathcal{B}(n, w/n)$, similarly for $w_{\mathbf{r}}$ and $w_{\mathbf{e}}$. To simplify the analysis we shall assume this model rather than the constant weight uniform model. Both models are very close, and our cryptographic protocols work just as well in both settings.

We first evaluate the distributions of the products $\mathbf{x} \cdot \mathbf{r}_2$ and $\mathbf{r}_1 \cdot \mathbf{y}$.

**Proposition 1.4.1.** *Let $\mathbf{x} = (X_0, \ldots, X_{n-1})$ (resp. $\mathbf{r} = (R_0, \ldots, R_{n-1})$) be a random vector where the $X_i$ (resp. $R_i$) are independent Bernoulli variables of parameter $p$ (resp. $p_{\mathbf{r}}$), $P(X_i = 1) = p$ and $P(R_i = 1) = p_{\mathbf{r}}$. Assuming $\mathbf{x}$ and $\mathbf{r}$ are independent, and denoting $\mathbf{z} = \mathbf{x} \cdot \mathbf{r} = (Z_0, \ldots, Z_{n-1})$ as defined in Eq. (1), we have:*

$$\begin{cases} \Pr[Z_k = 1] = \frac{1}{2} - \frac{1}{2}(1 - 2pp_{\mathbf{r}})^n, \\ \Pr[Z_k = 0] = \frac{1}{2} + \frac{1}{2}(1 - 2pp_{\mathbf{r}})^n. \end{cases} \tag{14}$$

*Proof.* We have

$$Z_k = \sum_{i+j=k \bmod n} X_i R_j \mod 2. \tag{15}$$

Every term $X_i R_j$ is the product of two independent Bernoulli variables of parameter respectively $p$ and $p_{\mathbf{r}}$, and is therefore a Bernoulli variable of parameter $p \times p_{\mathbf{r}}$. The variable $Z_k$ is the sum modulo 2 of $n$ such products, which are all independent since every variable $X_i$ (for $0 \le i \le n-1$) is involved exactly once in (15), and similarly every variable $R_j$ is involved once in (15). Therefore $Z_k$ is the sum modulo 2 of $n$ independent Bernoulli variables of parameter $p \times p_{\mathbf{r}}$, and we have

$$\Pr[Z_k = 1] = \sum_{0 \le i \le n, i \text{ odd}} \binom{n}{i} (pp_{\mathbf{r}})^i (1 - pp_{\mathbf{r}})^{n-i}$$

which, using the equations:

$$\sum_{\substack{0 \le i \le n, \\ i \text{ odd}}} \binom{n}{i} a^i b^{n-i} = \frac{(a+b)^n - (a-b)^n}{2}, \text{ and } \sum_{\substack{0 \le i \le n, \\ i \text{ even}}} \binom{n}{i} a^i b^{n-i} = \frac{(a+b)^n + (a-b)^n}{2} \tag{16}$$

with $a = pp_{\mathbf{r}}$ and $b = 1 - pp_{\mathbf{r}}$, simplifies into the claimed result. □

Let us denote by $\tilde{p} = \tilde{p}(n, w) = \Pr[Z_k = 1]$ from Eq. (14). Let $\mathbf{x}, \mathbf{y}$ (resp. $\mathbf{r}_1, \mathbf{r}_2$) be independent random vectors whose coordinates are independently Bernoulli distributed with parameter $p$ (resp. $p_{\mathbf{r}}$). Then the $k$-th coordinates of $\mathbf{x} \cdot \mathbf{r}_2$ and of $\mathbf{r}_1 \cdot \mathbf{y}$ are independent and Bernoulli distributed with parameter $\tilde{p}$. Therefore their modulo 2 sum $\mathbf{t} = \mathbf{x} \cdot \mathbf{r}_2 - \mathbf{r}_1 \cdot \mathbf{y}$ is Bernoulli distributed with

$$\begin{cases} \Pr[t_k = 1] = 2\tilde{p}(1 - \tilde{p}), \\ \Pr[t_k = 0] = (1 - \tilde{p})^2 + \tilde{p}^2. \end{cases} \tag{17}$$

Finally, by adding the term $\mathbf{e}$ to $\mathbf{t}$, we obtain the distribution of the coordinates of the error vector $\mathbf{e}' = \mathbf{x} \cdot \mathbf{r}_2 - \mathbf{r}_1 \cdot \mathbf{y} + \mathbf{e}$. Since the coordinates of $\mathbf{e}$ are Bernoulli of parameter $p_{\mathbf{e}}$ and those of $\mathbf{t}$ are Bernoulli distributed as (17) and independent from $\mathbf{e}$, we obtain the following proposition.

**Proposition 1.4.2.** *Let* $\mathbf{x}, \mathbf{y} \sim \mathcal{B}\left(n, \frac{w}{n}\right)$, $\mathbf{r}_1, \mathbf{r}_2 \sim \mathcal{B}\left(n, \frac{w_{\mathbf{r}}}{n}\right)$ *and* $\mathbf{e} \sim \mathcal{B}\left(n, \frac{w_{\mathbf{e}}}{n}\right)$, *and let* $\mathbf{e}' = \mathbf{x} \cdot \mathbf{r}_2 - \mathbf{r}_1 \cdot \mathbf{y} + \mathbf{e}$. *Then*

$$\begin{cases} \Pr[e'_k = 1] = 2\tilde{p}(1 - \tilde{p})(1 - \frac{w_{\mathbf{e}}}{n}) + ((1 - \tilde{p})^2 + \tilde{p}^2)\,\frac{w_{\mathbf{e}}}{n}, \\ \Pr[e'_k = 0] = ((1 - \tilde{p})^2 + \tilde{p}^2)\,(1 - \frac{w_{\mathbf{e}}}{n}) + 2\tilde{p}(1 - \tilde{p})\frac{w_{\mathbf{e}}}{n}. \end{cases} \tag{18}$$

Proposition 1.4.2 gives us the probability that a coordinate of the error vector $\mathbf{e}'$ is 1. In our simulations, which occur in the regime $w = \alpha\sqrt{n}$ with constant $\alpha$, we make the simplifying assumption that the coordinates of $\mathbf{e}'$ are independent, meaning that the weight of $\mathbf{e}'$ follows a binomial distribution of parameter $p^\star$, where $p^\star$ is defined as in Eq. (18): $p^\star = 2\tilde{p}(1 - \tilde{p})(1 - \frac{w_{\mathbf{e}}}{n}) + ((1 - \tilde{p})^2 + \tilde{p}^2)\,\frac{w_{\mathbf{e}}}{n}$. This approximation will give us, for $0 \leq d \leq \min(2 \times w \times w_{\mathbf{r}} + w_{\mathbf{e}}, n)$,

$$\Pr[\omega(\mathbf{e}') = d] = \binom{n}{d} (p^\star)^d (1 - p^\star)^{(n-d)}. \tag{19}$$

In practice, the results obtained by simulation on the decryption failure are very coherent with this assumption.

## 1.5 Decoding codes with low rates and good decoding properties

The previous section allowed us to determine the distribution of the error vector $\mathbf{e}$ in the configuration where a simple linear code is used. Now the decryption part corresponds to decoding the error described in the previous section. Any decodable code can be used at this point, depending on the considered application: clearly small dimension codes will allow better decoding, but at the cost of a lower encryption rate. The particular case that we consider corresponds typically to the case of key exchange or authentication, where only a small amount of data needs to be encrypted (typically 80, 128 or 256 bits, a symmetric

secret key size). We therefore need codes with low rates which are able to correct many errors. Again, a tradeoff is necessary between efficiently decodable codes but with a high Decoding Failure Rate (DFR) and less efficiently decodable codes but with a smaller DFR.

An example of such a family of codes with good decoding properties, meaning a simple decoding algorithm which can be analyzed, is given by Tensor Product Codes, which are used for biometry [8], where the same type of issue appears. More specifically, we will consider a special simple case of Tensor Product Codes (BCH codes and repetition codes), for which a precise analysis of the decryption failure can be obtained in the Hamming distance case.

### 1.5.1 Tensor product codes

**Definition 1.5.1** (Tensor Product Code)**.** *Let $\mathcal{C}_1$ (resp. $\mathcal{C}_2$) be a $[n_1, k_1, d_1]$ (resp. $[n_2, k_2, d_2]$) linear code over $\mathbb{F}_2$. The* Tensor Product Code *of $\mathcal{C}_1$ and $\mathcal{C}_2$ denoted $\mathcal{C}_1 \otimes \mathcal{C}_2$ is defined as the set of all $n_2 \times n_1$ matrices whose rows are codewords of $\mathcal{C}_1$ and whose columns are codewords of $\mathcal{C}_2$.*

*More formally, if $\mathcal{C}_1$ (resp. $\mathcal{C}_2$) is generated by $\mathbf{G}_1$ (resp. $\mathbf{G}_2$), then*

$$\mathcal{C}_1 \otimes \mathcal{C}_2 = \left\{ \mathbf{G}_2^\top \mathbf{X} \mathbf{G}_1 \text{ for } \mathbf{X} \in \mathbb{F}_2^{k_2 \times k_1} \right\} \tag{20}$$

**Remark 1.2.** *Using the notation of the above definition, the tensor product of two linear codes is a $[n_1 n_2, k_1 k_2, d_1 d_2]$ linear code.*

**Specifying the tensor product code.** Even if tensor product codes seem well-suited for our purpose, an analysis similar to the one in Sec. 1.4 becomes much more complicated. Therefore, in order to provide strong guarantees on the decryption failure probability for our cryptosystem, we chose to restrict ourselves to a tensor product code $\mathcal{C} = \mathcal{C}_1 \otimes \mathcal{C}_2$, where $\mathcal{C}_1$ is a BCH$(n_1, k_1, \delta_1)$ code of length $n_1$, dimension $k_1$, and correcting capability $\delta_1$ (*i.e.* it can correct any pattern of $\delta_1$ errors), and $\mathcal{C}_2$ is the repetition code of length $n_2$ and dimension 1, denoted $\mathbb{1}_{n_2}$. (Notice that $\mathbb{1}_{n_2}$ can decode up to $\delta_2 = \lfloor \frac{n_2 - 1}{2} \rfloor$.) Subsequently, the analysis becomes possible and remains accurate but the negative counterpart is that there probably are some other tensor product codes achieving better efficiency (or smaller key sizes).

In HQC, a message $\mathbf{m} \in \mathbb{F}_2^{k_1}$ is first encoded into $\mathbf{m}_1 \in \mathbb{F}_2^{n_1}$ with a BCH$(n_1, k_1 = k, \delta_1)$ code, then each coordinate $\mathbf{m}_{1,i}$ of $\mathbf{m}_1$ is re-encoded into $\tilde{\mathbf{m}}_{1,i} \in \mathbb{F}_2^{n_2}$ with a repetition code $\mathbb{1}_{n_2}$. We denote $n_1 n_2$ the length of the tensor product code[3] (its dimension is $k = k_1 \times 1$), and by $\tilde{\mathbf{m}}$ the resulting encoded vector, *i.e.* $\tilde{\mathbf{m}} = (\tilde{\mathbf{m}}_{1,1}, \ldots, \tilde{\mathbf{m}}_{1,n_1}) \in \mathbb{F}_2^{n_1 n_2}$.

The efficient algorithm used for the repetition code is the majority decoding. Formally:

$$\mathbb{1}_{n_2}.\mathsf{Decode}(\tilde{\mathbf{m}}_{1,j}) = \begin{cases} 1 \text{ if } \sum_{i=0}^{n_2 - 1} \tilde{\mathbf{m}}_{1,j,i} \geq \lceil \frac{n_2 + 1}{2} \rceil, \\ 0 \text{ otherwise.} \end{cases} \tag{21}$$

The decoding of BCH codes is discussed in the next section.

[3] In practice, the length is the smallest primitive prime $n$ greater than $n_1 n_2$ to avoid algebraic attacks.

### 1.5.2 BCH codes

For any positive integers $m \geq 3$ and $t \leq 2^{m-1}$, there exists a binary BCH code with the following parameters [28]:

- Block length $n = 2^m - 1$
- Number of parity-check digits $n - k \leq m\delta$, with $\delta$, the correcting capacity of the code and $k$ the number of information bits
- Minimum distance $d_{min} \geq 2\delta + 1$

We denote this code by BCH$[n, k, \delta]$. Let $\alpha$ be a primitive element in $\mathbb{F}_{2^m}$, the generator polynomial $g(x)$ of the BCH$[n, k, \delta]$ code is given by:

$$g(x) = \text{LCM}\left\{\phi_1(x), \phi_2(x), \cdots, \phi_{2\delta}(x)\right\}$$

with $\phi_i(x)$ being the minimal polynomial of $\alpha^i$ (refer to [28] for more details on generator polynomial).

Depending on HQC parameters, we construct shortened BCH (BCH-S1 and S2) codes such that $k = 256$ from the two following BCH codes BCH-1 and BCH-2 (codes from [28]).

| Code | n | k | $\delta$ |
|---|---|---|---|
| BCH-1 | 1023 | 513 | 57 |
| BCH-2 | 1023 | 483 | 60 |
| BCH-S1 | 766 | 256 | 57 |
| BCH-S2 | 796 | 256 | 60 |

Table 1: Original and shortened BCH codes.

The shortened codes are obtained by subtracting 257 from the parameters $n$ and $k$ of the code BCH-1 and subtracting 227 from the parameters $n$ and $k$ of the code BCH-2. Notice that shortening the BCH code does not affect the correcting capacity, thus we have the following shortened BCH codes :

- BCH-S1$[766 = 1023 - 257,\ 256 = 513 - 257,\ 57]$
- BCH-S2$[796 = 1023 - 227,\ 256 = 483 - 227,\ 60]$

In our case, we will be working in $\mathbb{F}_{2^m}$ with $m = 10$. To do so, we use the primitive polynomial $1 + X^3 + X^{10}$ of degree 10 to build this field (polynomial from [28]). We denote by $g_1(x)$ and $g_2(x)$ the generator polynomials of BCH-S1 and BCH-S2 respectively, which are equal to the generator polynomials of BCH codes BCH-1 and BCH-2 respectively. We precomputed the generator polynomials $g_1(x)$ and $g_2(x)$ of the code BCH-S1 and BCH-S2 and we included their hexadecimal formats in the file `parameters.h`. One can use the functions provided in the file `bch.h` to reconstruct the generator polynomials for both codes.

**Generator polynomial of BCH-1.** $g_1(x) = 1 + x + x^6 + x^9 + x^{10} + x^{12} + x^{13} + x^{15} + x^{17} +$
$x^{18} + x^{21} + x^{24} + x^{25} + x^{26} + x^{27} + x^{28} + x^{29} + x^{32} + x^{33} + x^{35} + x^{36} + x^{37} + x^{38} + x^{40} + x^{41} + x^{42} +$
$x^{43} + x^{45} + x^{49} + x^{52} + x^{53} + x^{54} + x^{56} + x^{57} + x^{59} + x^{61} + x^{62} + x^{63} + x^{65} + x^{67} + x^{70} + x^{75} + x^{76} +$
$x^{77} + x^{78} + x^{80} + x^{81} + x^{82} + x^{83} + x^{84} + x^{89} + x^{92} + x^{95} + x^{96} + x^{97} + x^{101} + x^{102} + x^{105} + x^{107} +$
$x^{111} + x^{116} + x^{120} + x^{122} + x^{123} + x^{128} + x^{129} + x^{132} + x^{133} + x^{135} + x^{137} + x^{139} + x^{141} + x^{142} +$
$x^{143} + x^{144} + x^{146} + x^{149} + x^{150} + x^{152} + x^{154} + x^{155} + x^{158} + x^{159} + x^{161} + x^{162} + x^{163} + x^{164} +$
$x^{165} + x^{167} + x^{169} + x^{170} + x^{171} + x^{173} + x^{177} + x^{178} + x^{180} + x^{181} + x^{182} + x^{183} + x^{185} + x^{186} + x^{187} +$
$x^{188} + x^{193} + x^{200} + x^{206} + x^{207} + x^{208} + x^{209} + x^{212} + x^{213} + x^{215} + x^{220} + x^{223} + x^{226} + x^{228} + x^{231} +$
$x^{232} + x^{234} + x^{236} + x^{237} + x^{238} + x^{239} + x^{240} + x^{242} + x^{243} + x^{245} + x^{247} + x^{248} + x^{249} + x^{250} + x^{251} +$
$x^{252} + x^{256} + x^{258} + x^{259} + x^{260} + x^{261} + x^{262} + x^{263} + x^{265} + x^{267} + x^{269} + x^{270} + x^{274} + x^{275} + x^{278} +$
$x^{279} + x^{281} + x^{282} + x^{283} + x^{284} + x^{285} + x^{286} + x^{287} + x^{291} + x^{292} + x^{293} + x^{294} + x^{296} + x^{300} + x^{303} +$
$x^{304} + x^{306} + x^{307} + x^{310} + x^{312} + x^{313} + x^{315} + x^{317} + x^{319} + x^{320} + x^{326} + x^{327} + x^{328} + x^{329} + x^{330} +$
$x^{331} + x^{332} + x^{333} + x^{334} + x^{335} + x^{337} + x^{338} + x^{340} + x^{342} + x^{343} + x^{344} + x^{345} + x^{346} + x^{347} + x^{348} +$
$x^{350} + x^{351} + x^{353} + x^{354} + x^{358} + x^{361} + x^{362} + x^{363} + x^{364} + x^{365} + x^{367} + x^{369} + x^{374} + x^{376} + x^{377} +$
$x^{378} + x^{379} + x^{381} + x^{387} + x^{390} + x^{392} + x^{393} + x^{394} + x^{395} + x^{396} + x^{403} + x^{404} + x^{405} + x^{406} + x^{407} +$
$x^{409} + x^{412} + x^{415} + x^{416} + x^{418} + x^{427} + x^{428} + x^{432} + x^{433} + x^{434} + x^{437} + x^{438} + x^{442} + x^{443} + x^{445} +$
$x^{448} + x^{452} + x^{454} + x^{456} + x^{460} + x^{461} + x^{462} + x^{463} + x^{464} + x^{468} + x^{471} + x^{472} + x^{474} + x^{475} + x^{478} +$
$x^{480} + x^{481} + x^{483} + x^{484} + x^{485} + x^{490} + x^{491} + x^{493} + x^{494} + x^{495} + x^{497} + x^{502} + x^{506} + x^{509} + x^{510}.$

**Generator polynomial of BCH-2.** $g_2(x) = 1 + x + x^3 + x^4 + x^6 + x^{11} + x^{13} + x^{15} + x^{16} +$
$x^{17} + x^{18} + x^{19} + x^{20} + x^{22} + x^{23} + x^{25} + x^{26} + x^{27} + x^{28} + x^{29} + x^{32} + x^{35} + x^{37} + x^{38} + x^{40} + x^{43} +$
$x^{44} + x^{45} + x^{48} + x^{49} + x^{50} + x^{54} + x^{56} + x^{58} + x^{60} + x^{61} + x^{64} + x^{66} + x^{69} + x^{73} + x^{74} + x^{75} + x^{76} +$
$x^{77} + x^{78} + x^{79} + x^{82} + x^{83} + x^{84} + x^{85} + x^{87} + x^{89} + x^{91} + x^{92} + x^{94} + x^{97} + x^{99} + x^{100} + x^{101} +$
$x^{105} + x^{106} + x^{107} + x^{108} + x^{112} + x^{114} + x^{115} + x^{116} + x^{117} + x^{119} + x^{122} + x^{123} + x^{124} + x^{125} +$
$x^{127} + x^{131} + x^{134} + x^{135} + x^{136} + x^{137} + x^{138} + x^{140} + x^{144} + x^{146} + x^{147} + x^{151} + x^{153} + x^{156} +$
$x^{160} + x^{161} + x^{163} + x^{165} + x^{167} + x^{168} + x^{170} + x^{171} + x^{174} + x^{175} + x^{176} + x^{179} + x^{181} + x^{184} +$
$x^{187} + x^{188} + x^{190} + x^{195} + x^{196} + x^{201} + x^{203} + x^{204} + x^{206} + x^{207} + x^{209} + x^{213} + x^{214} + x^{215} +$
$x^{217} + x^{219} + x^{220} + x^{223} + x^{224} + x^{226} + x^{227} + x^{231} + x^{233} + x^{234} + x^{238} + x^{245} + x^{250} + x^{251} +$
$x^{254} + x^{258} + x^{259} + x^{262} + x^{263} + x^{264} + x^{268} + x^{271} + x^{272} + x^{273} + x^{274} + x^{280} + x^{281} + x^{284} +$
$x^{286} + x^{287} + x^{288} + x^{289} + x^{290} + x^{291} + x^{293} + x^{294} + x^{295} + x^{298} + x^{299} + x^{302} + x^{303} + x^{304} +$
$x^{305} + x^{306} + x^{311} + x^{313} + x^{314} + x^{315} + x^{317} + x^{319} + x^{324} + x^{325} + x^{330} + x^{334} + x^{336} + x^{338} +$
$x^{340} + x^{341} + x^{345} + x^{347} + x^{348} + x^{352} + x^{355} + x^{358} + x^{359} + x^{362} + x^{364} + x^{367} + x^{368} + x^{369} + x^{370} +$
$x^{373} + x^{374} + x^{377} + x^{378} + x^{380} + x^{382} + x^{383} + x^{392} + x^{394} + x^{395} + x^{402} + x^{403} + x^{405} + x^{407} + x^{413} +$
$x^{414} + x^{415} + x^{417} + x^{419} + x^{420} + x^{422} + x^{423} + x^{424} + x^{425} + x^{427} + x^{428} + x^{429} + x^{431} + x^{432} + x^{434} +$
$x^{435} + x^{436} + x^{437} + x^{438} + x^{441} + x^{444} + x^{445} + x^{452} + x^{454} + x^{460} + x^{462} + x^{463} + x^{464} + x^{465} + x^{466} +$
$x^{469} + x^{470} + x^{471} + x^{476} + x^{478} + x^{479} + x^{480} + x^{481} + x^{484} + x^{487} + x^{488} + x^{489} + x^{490} + x^{491} + x^{502} +$
$x^{504} + x^{506} + x^{507} + x^{509} + x^{512} + x^{514} + x^{515} + x^{521} + x^{522} + x^{523} + x^{524} + x^{526} + x^{529} + x^{534} + x^{540}.$

### 1.5.3 Encoding shortened BCH codes

In the following we present the decoding of classical BCH codes which can also be used to decode shortened BCH codes. We denote by $u(x) = u_0 + \cdots + u_{k-1}x^{k-1}$ the polynomial

corresponding to the message $u = (u_0, \cdots, u_{k-1})$ to be encoded and $g(x)$ the generator polynomial. We use the systematic form of encoding where the rightmost k elements of the code word polynomial are the message bits and the leftmost $n-k$ bits are the parity-check bits. Following [28], the code word is given by $c(x) = b(x) + x^{n-k}u(x)$, where $b(x)$ is the reminder of the division of the polynomial $x^{n-k}u(x)$ by $g(x)$. In consequence, the encoding in systematic form consists of three steps :

1. Multiply the message $u(x)$ by $x^{n-k}$.
2. Compute the remainder $b(x)$ by dividing $x^{n-k}u(x)$ by the generator polynomial $g(x)$.
3. Combine $b(x)$ and $x^{n-k}u(x)$ to obtain the code polynomial $c(x) = b(x) + x^{n-k}u(x)$.

### 1.5.4 Decoding shortened BCH codes

As for encoding, the decoding of classical BCH codes can be used to decode shortened BCH codes. For sake of simplicity, we will detail the process of decoding classical BCH codes. Following [28], consider the BCH code defined by $[n, k, \delta]$, with $n = 2^m - 1$ ($m \geq 0$ of positive integer) and suppose that a codeword $v(x) = v_0 + v_1 x + \cdots + v_{n-1}x^{n-1}$ is transmitted. We denote $r(x) = r_0 + r_1 x + \cdots + r_{n-1}x^{n-1}$ the received word, potentially altered by some errors.

We denote the error polynomial $e(x) = e_0 + e_1 x + \cdots + e_{n-1}x^{n-1}$, meaning that there is an error in position $i$ whenever $e_i = 1$. Hence, $r(x) = v(x) + e(x)$.

We define the set of syndromes $S_1, S_2, \cdots, S_{2\delta}$ as $S_i = r(\alpha^i)$, with $\alpha$ being a primitive element in $\mathbb{F}_{2^m}$. We have that $r(\alpha^i) = e(\alpha^i)$, since $v(\alpha^i) = 0$ ($v$ is a codeword). Suppose that $e(x)$ has $t$ errors at locations $j_1, \cdots, j_t$, *i.e.* $e(x) = x^{j_1} + x^{j_2} + \cdots + x^{j_t}$. We obtain the following set of equations, where $\alpha^{j_1}, \alpha^{j_2}, \cdots, \alpha^{j_t}$ are unknown:

$$\begin{cases} S_1 &= \alpha^{j_1} + \alpha^{j_2} + \cdots + \alpha^{j_t} \\ S_2 &= (\alpha^{j_1})^2 + (\alpha^{j_2})^2 + \cdots + (\alpha^{j_t})^2 \\ S_3 &= (\alpha^{j_1})^3 + (\alpha^{j_2})^3 + \cdots + (\alpha^{j_t})^3 \\ &\vdots \\ S_{2\delta} &= (\alpha^{j_1})^{2\delta} + (\alpha^{j_2})^{2\delta} + \cdots + (\alpha^{j_t})^{2\delta} \end{cases}$$

The goal of a BCH decoding algorithm is to solve this system of equations. We define the error location numbers by $\beta_i = \alpha^{j_i}$, which indicate the location of the errors. The equations above, can be expressed as follows:

$$\begin{cases} S_1 &= \beta_1 + \beta_2 + \cdots + \beta_t \\ S_2 &= \beta_1^2 + \beta_2^2 + \cdots + \beta_t^2 \\ S_3 &= \beta_1^3 + \beta_2^3 + \cdots + \beta_t^3 \\ &\vdots \\ S_{2\delta} &= \beta_1^{2\delta} + \beta_2^{2\delta} + \cdots + \beta_t^{2\delta} \end{cases}$$

we define the error location polynomial as:

$$\begin{aligned}\sigma(x) &= (1+\beta_1 x)(1+\beta_2 x)\cdots(1+\beta_t x)\\ &= 1+\sigma_1 x+\sigma_2 x^2+\cdots+\sigma_t x^t\end{aligned}$$

We can see that the roots of $\sigma(x)$ are $\beta_1^{-1}, \beta_2^{-1}, \cdots, \beta_t^{-1}$ which are the inverses of the error location numbers. By inverting those roots we can construct the error polynomial $e(x)$.

We can summarize the decoding procedure of a BCH$[n, k, \delta]$ code by the following steps:

1. The first step is the computation of $2\delta$ syndromes using the received polynomial
2. The second step is the computation of the error-location polynomial $\sigma(x)$ from the $2\delta$ syndromes computed in the first step (in our implementation we will use the Simplified Berlekamp's Algorithm [27])
3. The third step is to find the error-location numbers by calculating the roots of the polynomial $\sigma(x)$ and returning their inverse (in our implementation we will be using the Chien search algorithm [11])
4. The fourth step is the correction of errors in the received polynomial

**Remark 1.3.** *As mentioned before, in our implementation, we deal with shortened BCH code. We notice that we will be using the same decoding procedure described above.*

**Step 1. Syndrome computations.** The following function computes the syndromes.

```
void syndrome_gen(syndrome_set* synd_set, gf_tables* tables, uint8_t* v); // bch.h
```

The syndromes are computed by evaluating the received polynomial stored in the vector `v` at the $2\times$ `PARAM_DELTA` consecutive roots of the generator polynomial $\alpha^i$ for $i = 1, 2, \cdots, 2 * \texttt{PARAM\_DELTA}$. Let us denote by $r(x)$ the polynomial in the vector `v`, thus the syndromes are

$$r(\alpha), r(\alpha^2), \cdots, r(\alpha^{2\times \texttt{PARAM\_DELTA}})$$

and they are stored as $\mathbb{F}_{2^m}$ elements in the structure `synd_set` which is the output of the function.

**Step 2. Computing the Error-Location Polynomial.** The following function computes the error location polynomial $\sigma(x)$ as defined above and store it in the vector `sigma`.

```
void get_error_location_poly(sigma_poly* sigma, gf_tables* tables, syndrome_set*
    synd_set);// bch.h
```

This function implements the simplified Berlekamp's algorithm for finding the error location polynomial for binary BCH codes given by Joiner and Komo in [27].

**Step 3. Finding the Error-Location Numbers.** The following function computes the roots of the error location polynomial and finds their inverses which are the error location numbers.

```
void chien_search(uint16_t* error_pos, uint16_t* size, gf_tables* tables,
    sigma_poly* sigma);// bch.h
```

To find the roots of the polynomial $\sigma(x)$ stored in the structure `sigma`, we have to evaluate $\sigma(x)$ in all the elements of the Galois Field: let $\alpha$ be the generator of the field then we have to check for $j = 1, 2, ...$ if $\sigma(\alpha^j) = 0$. Then if $\alpha^k$ is a root we store $\alpha^{-k}$ in the output array of the function. The Chien procedure permits to compute $\sigma(\alpha^{k+1})$ from $\sigma(\alpha^k)$, in fact:

- Suppose that $\sigma$ is of degree $t$. If we have evaluated $\alpha^k$, we obtain

$$\sigma(\alpha^k) = 1 + \sigma_1\alpha^k + \sigma_2\alpha^{2k} + \cdots + \sigma_t\alpha^{tk}$$

- Then, we can obtain $\sigma(\alpha^{k+1})$ in $\mathcal{O}(t)$ operations. In fact the $i$-th term in $\sigma(\alpha^{k+1})$ can be obtained from the $i$-th term of $\sigma(\alpha^k)$ by multiplying that term by $\alpha^i$.

  Suppose that we are using a BCH$[n, k, \delta]$ code (one of the shortened BCH codes described above). Then, the inverses of the roots of the elements $\alpha^i$ with $i \in \{1, \cdots, 2^{10} - 1 - n\}$ will not be valid error positions. In fact the location number obtained will be greater than $n$. For that it is useless to evaluate the error location polynomial $\sigma(x)$ in the element $\alpha^i$ for $i \in \{1, \cdots, 2^{10} - 1 - n\}$. Therefore, in our implementation we start the evaluation at $\alpha^i$ with $i = 2^{10} - n$.

**Step 4. Error correction.** To correct the errors in the received polynomial, we have to build the error polynomial $e(x)$ using the error location numbers obtained by the Chien search algorithm. We then add the error polynomial to the received polynomial. The following function builds $e(x)$ and stores the result in the vector `e`.

```
void error_poly_gen(uint8_t* e, uint16_t* error_pos, uint16_t size);// bch.h
```

### 1.5.5 Decryption Failure Probability

With a tensor product code $\mathcal{C} = \text{BCH}(n_1, k_1, \delta) \otimes \mathbb{1}_{n_2}$ as defined above, a decryption failure occurs whenever the decoding algorithm of the BCH code does not succeed in correcting errors that would have arisen after wrong decodings by the repetition code. Therefore, the analysis of the decryption failure probability is again split into three steps: evaluating the probability that the repetition code does not decode correctly, the conditional probability of a wrong decoding for the BCH code given an error weight and finally, the decryption failure probability using the law of total probability.

**Step 1.** We now focus on the probability that an error occurs while decoding the repetition code. As shown in Sec. 1.4, the probability for a coordinate of $\mathbf{e}' = \mathbf{x} \cdot \mathbf{r}_2 - \mathbf{r}_1 \cdot \mathbf{y} + \mathbf{e}$ to be 1 is $p^\star$ (see Eq. (18)). As mentioned above, $\mathbb{1}_{n_2}$ can decode up to $\delta_2 = \lfloor \frac{n_2-1}{2} \rfloor$ errors. Therefore, assuming that the error vector $\mathbf{e}'$ has weight $\gamma$ (which occurs with the probability given in Eq. (19)), the probability of getting a decoding error on a single block of the repetition code $\mathbb{1}_{n_2}$ is hence given by:

$$\bar{p}_\gamma = \bar{p}_\gamma(n_1, n_2) = \sum_{i=\lfloor \frac{n_2-1}{2} \rfloor + 1}^{n_2} \binom{n_2}{i} \left(\frac{\gamma}{n_1 n_2}\right)^i \left(1 - \frac{\gamma}{n_1 n_2}\right)^{n_2 - i}. \tag{22}$$

**Step 2.** We now focus on the BCH$(n_1, k_1, \delta_1)$ code, and recall that it can correct any pattern of $\delta_1$ errors. Now the probability $\mathcal{P}$ that the BCH$(n_1, k_1, \delta_1)$ code fails to decode correctly the encoded message $\mathbf{m}_1$ back to $\mathbf{m}$ is given by the probability that an error occurred on at least $\delta_1 + 1$ blocks of the repetition code. Therefore, we have

$$\mathcal{P} = \mathcal{P}(\delta_1, n_1, n_2, \gamma) = \sum_{i=\delta_1+1}^{n_1} \binom{n_1}{i} (\bar{p}_\gamma)^i (1 - \bar{p}_\gamma)^{n_1 - i}. \tag{23}$$

**Step 3.** Finally, using the law of total probability, we have that the decryption failure probability is given by the sum *over all the possible weights* of the probability that the error has this specific weight times the probability of a decoding error for this weight. This is captured in the following theorem, whose proof is a straightforward consequence of the formulae of Sec. 1.4 and 1.5.1.

**Theorem 1.4.** *Let* $\mathcal{C} = \text{BCH}(n_1, k_1, \delta) \otimes \mathbb{1}_{n_2}$, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}$, *and* $\mathbf{m} \xleftarrow{\$} \mathbb{F}_2^{k_1}$, *then with the notations above, the decryption failure probability is*

$$\begin{aligned} p_{\text{fail}} &= \Pr\left[\mathsf{Decrypt}\left(\mathsf{sk}, \mathsf{Encrypt}\left(\mathsf{pk}, \mathbf{m}\right)\right) \neq \mathbf{m}\right] && (24) \\ &= \sum_{\gamma=0}^{\min(2\times w \times w_{\mathbf{r}} + w_{\mathbf{e}}, n_1 n_2)} \Pr[\omega(\mathbf{e}') = \gamma] \mathcal{P}(\delta_1, n_1, n_2, \gamma) && (25) \end{aligned}$$

Eq. (25) gives a theoretical approximation of the decryption failure rate. The parameters presented in Tab. 2 were obtained using this formula. Experimental evidences supporting the validity of the assumptions made to obtained this formula are provided in Fig. 4.

## 1.6 Representation of objects

**Vectors.** Elements of $\mathbb{F}_2^n$, $\mathbb{F}_2^{n_1 n_2}$ and $\mathbb{F}_2^k$ are represented as binary arrays.

**Seeds.** The considered seedexpander has been provided by the NIST. It is initialized with a byte string of length 40 of which 32 are used as the `seed` and 8 are used as the `diversifier`. In addition, it is initialized with `max_length` equal to $2^{32} - 1$.

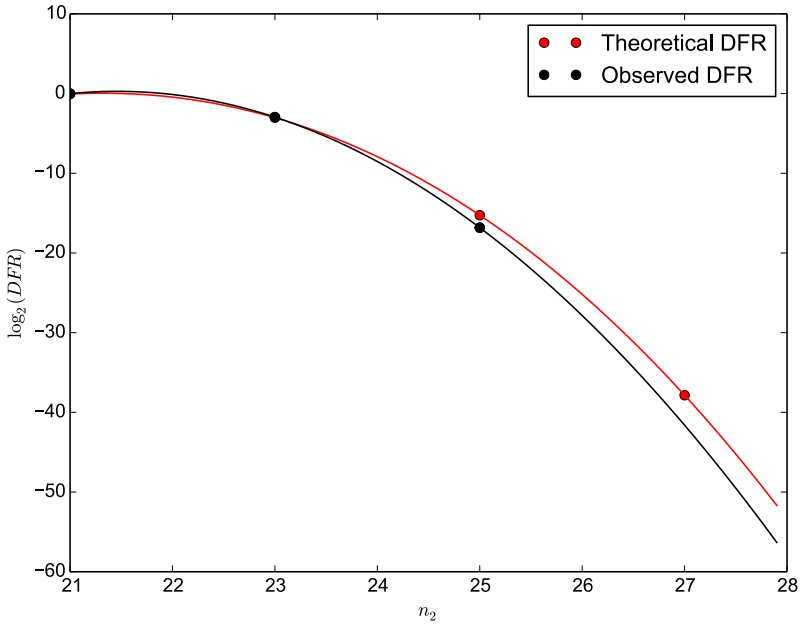


Figure 4: Logarithm of theoretical and observed decryption failure rates (DFR). The red curve corresponding to theoretical DFR was obtained using Eq. (25) while the black curve corresponding to experimental DFR was obtained by running $10^5$ encryption/decryption over $10^3$ codes with $n_1 = 766$, $k_1 = 256$, $\delta_1 = 57$, $w = 67$, $w_{\mathbf{r}} = 77$. The parameters have been selected to make the theoretical DFR sufficiently high to compare it to experiments. Finally, the curves have been interpolated to the second order on the logarithm of the probability.

### 1.6.1 Keys and ciphertext representation

The secret key $\mathsf{sk} = (\mathbf{x}, \mathbf{y})$ is represented as $\mathsf{sk} = (\mathbf{seed1})$ where **seed1** is used to generate $\mathbf{x}$ and $\mathbf{y}$. The public key $\mathsf{pk} = (\mathbf{h}, \mathbf{s})$ is represented as $\mathsf{pk} = (\mathbf{seed2}, \mathbf{s})$ where **seed2** is used to generate $\mathbf{h}$. The ciphertext $\mathbf{c}$ is represented as $(\mathbf{u}, \mathbf{v}, \mathbf{d})$ where $\mathbf{d}$ is generated using SHA512. The secret key has size 40 bytes, the public key has size $40 + \lceil n/8 \rceil$ bytes and the ciphertext has size $\lceil n/8 \rceil + \lceil n_1 n_2/8 \rceil + 64$ bytes.

### 1.6.2 Randomness and vector generation

Random bytes are generated using the NIST provided `randombytes` or `seedexpander` functions. The `randombytes` function is used to generate **seed1** and **seed2** as well as $\mathbf{m}$. The `seedexpander` function is used to generate $\theta$ (using $\mathbf{m}$ as seed) as well as $\mathbf{x}$, $\mathbf{y}$ (using **seed1** as seed), $\mathbf{h}$ (using **seed2** as seed) and $\mathbf{r}_1$, $\mathbf{r}_2$, $\mathbf{e}$ (using $\theta$ as seed).

Random vectors are sampled uniformly from $\mathbb{F}_2^k$, $\mathbb{F}_2^n$ or from $\mathbb{F}_2^n$ with a given Hamming weight. Sampling from $\mathbb{F}_2^k$ and $\mathbb{F}_2^n$ is performed by filling the mathematical representation

of the vector with random bits. Sampling a vector from $\mathbb{F}_2^n$ of a given weight starts by generating uniformly at random the support using a rejection sampling process. Next, the sampled support is converted to an $n$-dimensional array.

## 1.7 Parameters

In this section, we specify which codes are used for HQC and give concrete sets of parameters. As mentioned in the previous section, we use a tensor product code (Def. 1.5.1) $\mathcal{C} = \mathrm{BCH}(n_1, k, \delta) \otimes \mathbb{1}_{n_2}$. A message $\mathbf{m} \in \mathbb{F}_2^k$ is encoded into $\mathbf{m}_1 \in \mathbb{F}_2^{n_1}$ with the BCH code, then each coordinate $\mathbf{m}_{1,i}$ of $\mathbf{m}_1$ is encoded into $\tilde{\mathbf{m}}_{1,i} \in \mathbb{F}_2^{n_2}$ with $\mathbb{1}_{n_2}$. To match the description of our cryptosystem in Sec. 1.3, we have $\mathbf{mG} = \tilde{\mathbf{m}} = (\tilde{\mathbf{m}}_{1,0}, \ldots, \tilde{\mathbf{m}}_{1,n_1-1}) \in \mathbb{F}_2^{n_1 n_2}$. To obtain the ciphertext, $\mathbf{r} = (\mathbf{r}_1, \mathbf{r}_2) \xleftarrow{\$} \mathcal{R}^2$ and $\mathbf{e} \xleftarrow{\$} \mathcal{R}$ are generated and the encryption of $\mathbf{m}$ is $\mathbf{c} = (\mathbf{u} = \mathbf{r}_1 + \mathbf{h} \cdot \mathbf{r}_2, \mathbf{v} = \mathbf{mG} + \mathbf{s} \cdot \mathbf{r}_2 + \mathbf{e})$.

We propose several sets of parameters, targeting different levels of security. According to NIST, it may be assumed that an attacker can only make $2^{64}$ queries to the decryption oracle. In this sense, we propose several decryption failure rates ranging from $2^{-64}$ to $2^{-\lambda}$ where $\lambda$ is the security parameter. The proposed sets of parameters cover security categories 1, 3, and 5 (for respectively 128, 192, and 256 bits of security). For each parameter set, the parameters are chosen so that the minimal workfactor of the best known attack exceeds the security parameter. For classical attacks, best known attacks include the works from [9, 6, 15, 3] and for quantum attacks, the work of [5]. We consider $w = \mathcal{O}(\sqrt{n})$ and follow the complexity described in [10] (see Sec. 5 for more details).

In Tab. 2, $n_1$ denotes the length of the BCH code, $n_2$ the length of the repetition code $\mathbb{1}$ so that the length of the tensor product code $\mathcal{C}$ is $n_1 n_2$ (the ambient space has length $n$, the smallest primitive prime greater than $n_1 n_2$ to avoid algebraic attacks). $k$ is the dimension of the BCH code and hence also the dimension of $\mathcal{C}$. $\delta$ is the decoding capability of the BCH code, *i.e.* the maximum number of errors that the BCH can decode. $w$ is the weight of the $n$-dimensional vectors $\mathbf{x}$, $\mathbf{y}$, $w_{\mathbf{r}}$ the weight of $\mathbf{r}_1$, and $\mathbf{r}_2$ and similarly $w_{\mathbf{e}} = \omega(\mathbf{e})$ for our cryptosystem.

**Computational costs of the system.** For encryption the main cost is a product of a cyclic matrix of size $n$ with a vector of weight $\mathcal{O}(\sqrt{n})$. Using the Fourier transform the asymptotical cost is in $\mathcal{O}(n \log(n))$ but for our range of parameters, taking into account the weight $\mathcal{O}(\sqrt{n})$ allows to obtain a cost in $\mathcal{O}(n^{\frac{3}{2}})$ which is better in practice that what is obtained with Fourier transform. For decryption, there is always the cost of a matrix times a small vector in $\mathcal{O}(n^{\frac{3}{2}})$, plus the cost of decoding. For our proposition the decoding consists in a repetition code of length $n_2$ and the decoding of BCH code of length $n_1$ ($766 \leq n_1 \leq 796$), the cost of the repetition code decoding is hence linear, when the cost of the BCH is quadratic in the length $n_1$ of the BCH code. Overall the main cost remains the computation of the matrix-vector product in $\mathcal{O}(n^{\frac{3}{2}})$.

| Instance | $n_1$ | $n_2$ | $n$ | $k$ | $\delta$ | $w$ | $w_{\mathbf{r}} = w_{\mathbf{e}}$ | security | $p_{\text{fail}}$ |
|---|---|---|---|---|---|---|---|---|---|
| hqc-128-1 | 796 | 31 | 24,677 | 256 | 60 | 67 | 77 | 128 | $< 2^{-128}$ |
| hqc-192-1 | 766 | 57 | 43,669 | 256 | 57 | 101 | 117 | 192 | $< 2^{-128}$ |
| hqc-192-2 | 766 | 61 | 46,747 | 256 | 57 | 101 | 117 | 192 | $< 2^{-192}$ |
| hqc-256-1 | 766 | 83 | 63,587 | 256 | 57 | 133 | 153 | 256 | $< 2^{-128}$ |
| hqc-256-2 | 796 | 85 | 67,699 | 256 | 60 | 133 | 153 | 256 | $< 2^{-192}$ |
| hqc-256-3 | 796 | 89 | 70,853 | 256 | 60 | 133 | 153 | 256 | $< 2^{-256}$ |

Table 2: Parameter sets for HQC. The tensor product code used is $\mathcal{C} = \text{BCH}(n_1, k_1, \delta_1) \otimes \mathbb{1}_{n_2}$ (see Sec. 1.5.1). The considered BCH codes are initially of length 1023, then shortened to support 256 bits dimension (see Tab. 1 and Sec. 1.5.2). For the resulting public key, secret key and ciphertext sizes, please see Tab. 3 below. One may use seeds to shorten keys thus obtaining sizes presented in Tab. 4. The aforementioned sizes are the ones used in our reference implementation except that we also concatenate the public key within the secret key in order to respect the NIST API.

| Instance | pk size | sk size | ct size | ss size |
|---|---|---|---|---|
| hqc-128-1 | 6,170 | 252 | 6,234 | 64 |
| hqc-192-1 | 10,918 | 404 | 10,981 | 64 |
| hqc-192-2 | 11,688 | 404 | 11,749 | 64 |
| hqc-256-1 | 15,898 | 532 | 15,960 | 64 |
| hqc-256-2 | 16,926 | 566 | 16,984 | 64 |
| hqc-256-3 | 17,714 | 566 | 17,777 | 64 |

Table 3: Resulting theoretical sizes in bytes for HQC. The public key pk is composed of ($\mathbf{h}$, $\mathbf{s}$) and has size $2n$ bits. The secret key sk is composed of ($\mathbf{x}$, $\mathbf{y}$) and has size $2w\lceil \log_2(n) \rceil$ bits. The ciphertext ct is composed of ($\mathbf{u}$, $\mathbf{v}$, $\mathbf{d}$) and has size $n + n_1 n_2 + 512$ bits. The shared secret ss is composed of $K$ and has size 512 bits (SHA512 output size).

# 2 Performance Analysis

## 2.1 Reference Implementation

In this section, we provide concrete performance measures of the reference implementation. For each parameter set, results have been obtained by running 100,000 random instances and computing their average execution time. The benchmarks have been performed on a machine running Archlinux. The latter has 16GB of memory and an Intel® Core™ i7-7820X CPU @ 3.6GHz for which the Hyper-Threading, Turbo Boost and SpeedStep features were disabled. The reference implementation is written in `C++` and have been compiled with `g++` (version 8.2.1) using the compilation flags `-O3 -pedantic -pthread`. The following third party libraries have been used: `openssl` (version 1.1.1b), `gmp` (version 6.1.2), `NTL`

| Instance | pk size | sk size | ct size | ss size |
|---|---|---|---|---|
| hqc-128-1 | 3,125 | 40 | 6,234 | 64 |
| hqc-192-1 | 5,499 | 40 | 10,981 | 64 |
| hqc-192-2 | 5,884 | 40 | 11,749 | 64 |
| hqc-256-1 | 7,989 | 40 | 15,960 | 64 |
| hqc-256-2 | 8,503 | 40 | 16,984 | 64 |
| hqc-256-3 | 8,897 | 40 | 17,777 | 64 |

Table 4: Resulting sizes in bytes for HQC using NIST seed expander initialized with 40 bytes long seeds. The public key pk is composed of (**seed1**, **s**) and has size $320 + n$ (in bits). The secret key sk is composed of (**seed2**) and has size 320 (in bits). The ciphertext ct is composed of (**u**, **v**, **d**) and has size $n + n_1 n_2 + 512$ (in bits). The shared secret ss is composed of $K$ and has size 512 bits (SHA512 output size).

(version 11.3.2) [39] and `GF2X` (version 1.2).

The performances of our reference implementation on the aforementioned benchmark platform are described in Tab. 5 (timings in ms) and Tab. 6 (millions of CPU cycles required).

| Instance | KeyGen | Encaps | Decaps |
|---|---|---|---|
| hqc-128-1 | 0.11 | 0.19 | 0.31 |
| hqc-192-1 | 0.19 | 0.33 | 0.51 |
| hqc-192-2 | 0.20 | 0.34 | 0,52 |
| hqc-256-1 | 0.27 | 0.47 | 0.69 |
| hqc-256-2 | 0.29 | 0.49 | 0.71 |
| hqc-256-3 | 0.30 | 0.51 | 0.75 |

Table 5: Timings (in ms) of the reference implementation for different instances of HQC.

| Instance | KeyGen | Encaps | Decaps |
|---|---|---|---|
| hqc-128-1 | 0.39 | 0.68 | 1.13 |
| hqc-192-1 | 0.68 | 1.13 | 1.78 |
| hqc-192-2 | 0.72 | 1.21 | 1.84 |
| hqc-256-1 | 0.98 | 1.65 | 2.45 |
| hqc-256-2 | 1.04 | 1.79 | 2.61 |
| hqc-256-3 | 1.08 | 1.84 | 2.67 |

Table 6: Millions of cycles for the reference implementation for different instances of HQC.

## 2.2 Optimized Implementation

In this section, we provide concrete performance measures of the optimized implementation. For each parameter set, results have been obtained by running 100,000 random instances and computing their average execution time. The benchmarks have been performed on a machine running Archlinux. The latter has 16GB of memory and an Intel® Core™ i7-7820X CPU @ 3.6GHz for which the Hyper-Threading, Turbo Boost and SpeedStep features were disabled. The scheme have been compiled with `gcc` (version 8.2.1) using the compilation flags `-O3 -std=c99 -mavx -mavx2 -pedantic -pthread`. The following third party library have been used: `openssl` (version 1.1.1b).

The optimized implementation is written in `C` and includes an optimized algorithm to perform the multiplication of two vectors by using `AVX2` instructions. The algorithm uses the matrix vector form of multiplication and takes advantage of the low Hamming weight of the vectors in the HQC scheme. The only difference between the reference and the optimized implementation is that the former uses `NTL` [39] to do the multiplication and the latter uses the aforementioned optimized algorithm.

The performances of our optimized implementation on the aforementioned benchmark platform are described in Tab. 7 (timings in ms) and Tab. 8 (millions of CPU cycles required).

| Instance | KeyGen | Encaps | Decaps |
|---|---|---|---|
| hqc-128-1 | 0.07 | 0.14 | 0.26 |
| hqc-192-1 | 0.13 | 0.26 | 0.41 |
| hqc-192-2 | 0.14 | 0.27 | 0,42 |
| hqc-256-1 | 0.20 | 0.40 | 0.60 |
| hqc-256-2 | 0.22 | 0.42 | 0.63 |
| hqc-256-3 | 0.23 | 0.45 | 0.66 |

Table 7: Timings (in ms) of the optimized implementation for different instances of HQC.

| Instance | KeyGen | Encaps | Decaps |
|---|---|---|---|
| hqc-128-1 | 0.25 | 0.52 | 0.94 |
| hqc-192-1 | 0.47 | 0.93 | 1.49 |
| hqc-192-2 | 0.50 | 0.98 | 1.51 |
| hqc-256-1 | 0.74 | 1.46 | 2.17 |
| hqc-256-2 | 0.78 | 1.53 | 2.27 |
| hqc-256-3 | 0.83 | 1.61 | 2.39 |

Table 8: Millions of cycles for the optimized implementation for different instances of HQC.

# 3 Known Answer Test Values

Known Answer Test (KAT) values have been generated using the script provided by the NIST. They are available in the folders `KAT/Reference_Implementation/` and `KAT/Optimized_Implementation/`.

In addition, we provide, for each parameter set, an example with *intermediate values* in the folder `KAT/Reference_Implementation/`.

Notice that one can generate the aforementioned test files using respectively the `kat` and `verbose` modes of our implementation. The procedure to follow in order to do so is detailed in the technical documentation.

# 4 Security

In this section we prove the security of our encryption scheme viewed as a PKE scheme (IND-CPA). The security of the KEM/DEM version is provided by the transformation described in [23], and the tightness of the reduction provided by this transformation has been discussed at the end of Sec. 1.2.

**Theorem 4.1.** *The scheme presented above is* IND-CPA *under the assumption that both the* 2*-DQCSD with parity and* 3*-DQCSD with parity and erasures are hard.*

*Proof of Theorem 4.1.* To prove the security of the scheme, we are going to build a sequence of games transitioning from an adversary receiving an encryption of message $\mathbf{m}_0$ to an adversary receiving an encryption of a message $\mathbf{m}_1$, and show that if the adversary manages to distinguish one from the other, then we can build a simulator breaking the DQCSD assumption with parity and $\ell \geq 1$ erasure(s), for QC codes of index 2 or 3 (codes with parameters $[2n, n]$ or $[3n, n]$), and running in approximately the same time.

**Game $G_1$:** This is the real game, which we can state algorithmically as follows:

$\mathbf{Game}^1_{\mathcal{E},\mathcal{A}}(\lambda)$
1. $\mathsf{param} \leftarrow \mathsf{Setup}(1^\lambda)$
2. $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{param})$ with $\mathsf{pk} = (\mathbf{h}, \mathbf{s} = \mathbf{x} + \mathbf{h} \cdot \mathbf{y})$ and $\mathsf{sk} = (\mathbf{x}, \mathbf{y})$
3. $(\mathbf{m}_0, \mathbf{m}_1) \leftarrow \mathcal{A}(\texttt{FIND} : \mathsf{pk})$
4. $\mathbf{c}^* \leftarrow \mathsf{Encrypt}(\mathsf{pk}, \mathbf{m}_0) = (\mathbf{u}, \mathbf{v}) \in \mathbb{F}_2^n \times \mathbb{F}_2^{n_1 n_2}$
5. $b' \leftarrow \mathcal{A}(\texttt{GUESS} : \mathbf{c}^*)$
6. `RETURN` $b'$

**Game $G_2$:** In this game we start by forgetting the decryption key $\mathsf{sk} = (\mathbf{x}, \mathbf{y})$, and taking $\mathbf{s}$ at random of same bit parity $b = w + \mathbf{h}(1) \times w \mod 2$ as $\mathbf{s}' = \mathbf{x} + \mathbf{h} \cdot \mathbf{y}$, and then proceed honestly:

$\mathbf{Game}^2_{\mathcal{E},\mathcal{A}}(\lambda)$
1. $\mathsf{param} \leftarrow \mathsf{Setup}(1^\lambda)$
2a. $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{param})$ with $\mathsf{pk} = (\mathbf{h},\, \mathbf{s}' = \mathbf{x} + \mathbf{h} \cdot \mathbf{y})$ and $\mathsf{sk} = (\mathbf{x}, \mathbf{y})$
2b. $\mathbf{s} \xleftarrow{\$} \mathbb{F}^n_{2,b}$, for $b = \mathbf{s}'(1) \mod 2$
2c. $(\mathsf{pk}, \mathsf{sk}) \leftarrow ((\mathbf{h},\, \mathbf{s}), \mathbf{0})$
3. $(\mathbf{m}_0, \mathbf{m}_1) \leftarrow \mathcal{A}(\texttt{FIND} : \mathsf{pk})$
4. $\mathbf{c}^* \leftarrow \mathsf{Encrypt}(\mathsf{pk}, \mathbf{m}_0) = (\mathbf{u}, \mathbf{v}) \in \mathbb{F}^n_2 \times \mathbb{F}^{n_1 n_2}_2$
5. $b' \leftarrow \mathcal{A}(\texttt{GUESS} : \mathbf{c}^*)$
6. $\texttt{RETURN}\ b'$

The adversary has access to $\mathsf{pk}$ and $\mathbf{c}^*$. As he has access to $\mathsf{pk}$ and the $\mathsf{Encrypt}$ function, anything that is computed from $\mathsf{pk}$ and $\mathbf{c}^*$ can also be computed from just $\mathsf{pk}$. Moreover, the distribution of $\mathbf{c}^*$ is independent of the game we are in. Indeed, assume that $\mathbf{m}_0$ and $\mathbf{m}_1$ have different bit parities. Without loss of generality, say even for $\mathbf{m}_0$ and odd for $\mathbf{m}_1$ and assume $\mathbf{h}$ has odd parity (a similar reasoning holds for $\mathbf{h}$ of even parity). As $w$, $w_{\mathbf{r}}$, and $w_{\mathbf{e}}$ are all odd (see Tab. 2), the adversary knows the parity of $\mathbf{m}_b\mathbf{G} \in \mathbb{F}^n_2$, $\mathbf{sr}_2 \in \mathbb{F}^n_2$, and $\mathbf{e} \in \mathbb{F}^n_2$. As the message is encrypted in $\mathbb{F}^{n_1 n_2}_2$, the last $\ell = n - n_1 n_2$ bits of the vector $\mathbf{v}$ are truncated, yielding a vector $\tilde{\mathbf{v}} \in \mathbb{F}^{n_1 n_2}_2$ of unknown parity. This is illustrated in Fig. 5. Therefore we can suppose the only input of the adversary is $\mathsf{pk}$.

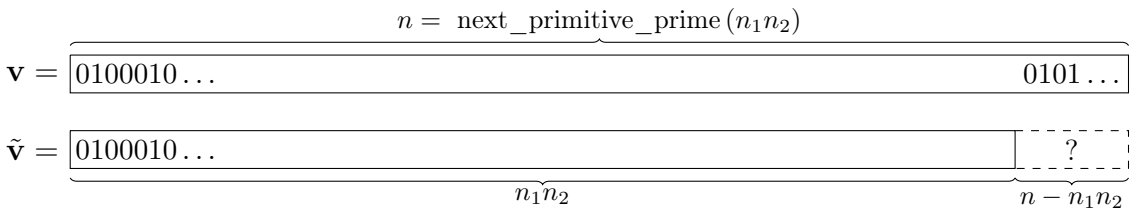


Figure 5: Truncation of vector $\mathbf{v}$ from $\mathbb{F}^n_2$ to $\tilde{\mathbf{v}} \in \mathbb{F}^{n_1 n_2}_2$.

Now suppose the adversary has an algorithm $\mathcal{D}_\lambda$, taking $\mathsf{pk}$ as input, that distinguishes with advantage $\epsilon$ Game $\boldsymbol{G}_1$ and Game $\boldsymbol{G}_2$, for some security parameter $\lambda$. Then he can also build an algorithm $\mathcal{D}'_{\mathcal{E},\mathcal{D}_\lambda}$ which solves the 2-$\mathsf{DQCSD}(n, w, b)$ problem with parity with the same advantage $\epsilon$ as the game distinguisher.

$\mathcal{D}'_{\mathcal{E},\mathcal{D}_\lambda}((\mathbf{H}, \mathbf{s}))$
1. Set $\mathsf{param} \leftarrow \mathsf{Setup}(1^\lambda)$
2. $\mathsf{pk} \leftarrow (\mathbf{h}, \mathbf{s})$
3. $b' \leftarrow \mathcal{D}_\lambda(\mathsf{pk})$
4. If $b' == 1$ output $\texttt{QCSD}$
5. If $b' == 2$ output $\texttt{UNIFORM}$

Note that if we define $\mathsf{pk}$ as $(\mathbf{h}, \mathbf{y})$ and $(\mathbf{H}, \mathbf{y}^\top)$ from a 2-$\mathsf{QCSD}(n, w, b)$ distribution with parity, $\mathsf{pk}$ follows exactly the same distribution as in Game $\boldsymbol{G}_1$. On the other

hand if $(\mathbf{H}, \mathbf{y}^\top)$ comes from a uniform distribution over $\mathbb{F}_{2,b}^{n\times 2n} \times \mathbb{F}_{2,b'}^{n}$, $\mathsf{pk}$ follows exactly the same distribution as in Game $\boldsymbol{G}_2$.

Thus we have:

$$\begin{aligned}\Pr\left[\mathcal{D}'_{\mathcal{E},\mathcal{D}_\lambda}((\mathbf{H}, \mathbf{y}^\top)) = \mathtt{QCSD} | (\mathbf{H}, \mathbf{y}^\top) \leftarrow 2\text{-}\mathsf{QCSD}(n, w, b)\right] = \\ \Pr\left[\mathcal{D}_\lambda(\mathsf{pk}) = 1 | \mathsf{pk} \text{ from } \mathbf{Game}^0_{\mathcal{E},\mathcal{A}}(\lambda)\right], \text{ and}\end{aligned} \tag{26}$$

$$\begin{aligned}\Pr\left[\mathcal{D}'_{\mathcal{E},\mathcal{D}_\lambda}((\mathbf{H}, \mathbf{y}^\top)) = \mathtt{UNIFORM} | (\mathbf{H}, \mathbf{y}^\top) \leftarrow 2\text{-}\mathsf{QCSD}(n, w, b)\right] = \\ \Pr\left[\mathcal{D}_\lambda(\mathsf{pk}) = 2 | \mathsf{pk} \text{ from } \mathbf{Game}^0_{\mathcal{E},\mathcal{A}}(\lambda)\right]\end{aligned} \tag{27}$$

And similarly when $(\mathbf{H}, \mathbf{y}^\top)$ is uniform the probabilities of $\mathcal{D}'_{\mathcal{E},\mathcal{D}_\lambda}$ outputs match those of $\mathcal{D}_\lambda$ when $\mathsf{pk}$ is from $\mathbf{Game}^2_{\mathcal{E},\mathcal{A}}(\lambda)$. The advantage of $\mathcal{D}'_{\mathcal{E},\mathcal{D}_\lambda}$ is therefore equal to the advantage of $\mathcal{D}_\lambda$.

**Game $\boldsymbol{G}_3$:** Now that we no longer know the decryption key, we can start generating random ciphertexts. So instead of picking correctly weighted $\mathbf{r}_1, \mathbf{r}_2, \mathbf{e}$, the simulator now picks random vectors in $\mathbb{F}_{2,w_\mathbf{r}}^n$ and $\mathbb{F}_{2,w_\mathbf{e}}^n$.

$\mathbf{Game}^3_{\mathcal{E},\mathcal{A}}(\lambda)$
1. $\mathsf{param} \leftarrow \mathsf{Setup}(1^\lambda)$
2a. $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{param})$ with $\mathsf{pk} = (\mathbf{h}, \mathbf{s}' = \mathbf{x} + \mathbf{h} \cdot \mathbf{y})$ and $\mathsf{sk} = (\mathbf{x}, \mathbf{y})$
2b. $\mathbf{s} \xleftarrow{\$} \mathbb{F}_{2,b}^n$, for $b = \mathbf{s}'(1) \mod 2$
2c. $(\mathsf{pk}, \mathsf{sk}) \leftarrow ((\mathbf{h}, \mathbf{s}), \mathbf{0})$
3. $(\mathbf{m}_0, \mathbf{m}_1) \leftarrow \mathcal{A}(\mathtt{FIND} : \mathsf{pk})$
4a. $\mathbf{e} \xleftarrow{\$} \mathbb{F}_{2,w_\mathbf{e}}^n, \mathbf{r} = (\mathbf{r}_1, \mathbf{r}_2) \xleftarrow{\$} \mathbb{F}_{2,w_\mathbf{r}}^n \times \mathbb{F}_{2,w_\mathbf{r}}^n$
4b. $\mathbf{u} \leftarrow \mathbf{r}_1 + \mathbf{h}\mathbf{r}_2$ and $\mathbf{v} \leftarrow \mathbf{m}_0\mathbf{G} + \mathbf{s} \cdot \mathbf{r}_2 + \mathbf{e}$
4c. $\mathbf{c}^* \leftarrow (\mathbf{u}, \mathbf{v})$, with $\mathbf{v}$ truncated in $\mathbb{F}_2^{n_1 n_2}$
5. $b' \leftarrow \mathcal{A}(\mathtt{GUESS} : \mathbf{c}^*)$
6. $\mathtt{RETURN}\ b'$

As we have

$$(\mathbf{u}, \mathbf{v} - \mathbf{m}_0\mathbf{G})^\top = \begin{pmatrix} \mathbf{I}_n & \mathbf{0} & \mathbf{rot}(\mathbf{h}) \\ \mathbf{0} & \mathbf{I}_n & \mathbf{rot}(\mathbf{s}) \end{pmatrix} \cdot (\mathbf{r}_1, \mathbf{e}, \mathbf{r}_2)^\top,$$

the difference between Game $\boldsymbol{G}_2$ and Game $\boldsymbol{G}_3$ is that in the former

$$\left(\begin{pmatrix} \mathbf{I}_n & \mathbf{0} & \mathbf{rot}(\mathbf{h}) \\ \mathbf{0} & \mathbf{I}_n & \mathbf{rot}(\mathbf{s}) \end{pmatrix}, (\mathbf{u}, \mathbf{v} - \mathbf{m}_0\mathbf{G})^\top\right)$$

follows the 3-$\mathsf{QCSD}$ distribution with parity, and in the latter it follows a uniform distribution (as $\mathbf{r}_1$ and $\mathbf{e}$ are uniformly distributed over $\mathbb{F}_{2,b}^n$ with $b$ odd) over $\mathbb{F}_{2,b_1,b_2}^{2n\times 3n} \times (\mathbb{F}_{2,b'_1}^n \times \mathbb{F}_{2,b'_2}^n)$.

Note that an adversary is not able to obtain $\mathbf{c}^*$ from $\mathsf{pk}$ anymore, as depending on which game we are $\mathbf{c}^*$ is generated differently. The input of a game distinguisher will therefore be $(\mathsf{pk}, \mathbf{c}^*)$. As it must interact with the challenger as usually we suppose it has two access modes `FIND` and `GUESS` to process first $\mathsf{pk}$ and later $\mathbf{c}^*$.

Suppose the adversary is able to distinguish Game $\boldsymbol{G}_2$ and Game $\boldsymbol{G}_3$, with a distinguisher $\mathcal{D}_\lambda$, which takes as input $(\mathsf{pk}, \mathbf{c}^*)$ and outputs a guess $b' \in \{2, 3\}$ of the game we are in.

Again, we can build a distinguisher $\mathcal{D}'_{\mathcal{E},\mathcal{D}_\lambda}$ that will break the 3-$\mathsf{DQCSD}(n, w, b_1, b_2)$ with parity and $\ell = n - n_1 n_2$ erasures assumption from $\mathsf{Setup}(1^\lambda)$ with the same advantage as the game distinguisher. In the 3-$\mathsf{DQCSD}(n, w, b_1, b_2)$ problem with parity, matrix $\mathbf{H}$ is assumed to be of the form

$$\begin{pmatrix} \mathbf{I}_n & \mathbf{0} & \mathbf{rot}(\mathbf{a}) \\ \mathbf{0} & \mathbf{I}_n & \mathbf{rot}(\mathbf{b}) \end{pmatrix}.$$

In order to use explicitly $\mathbf{a}$ and $\mathbf{b}$ we denote this matrix $\mathbf{H}_{\mathbf{a},\mathbf{b}}$ instead of just $\mathbf{H}$. We will also note $\mathbf{t} = (\mathbf{t_1}, \mathbf{t_2})$.

$\mathcal{D}'_{\mathcal{E},\mathcal{D}_\lambda}((\mathbf{H}_{\mathbf{a},\mathbf{b}}, (\mathbf{t_1}, \mathbf{t_2})^\top))$
1. $\mathsf{param} \leftarrow \mathsf{Setup}(1^\lambda)$
2a. $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{param})$ with $\mathsf{pk} = (\mathbf{h}, \mathbf{s} = \mathbf{x} + \mathbf{h} \cdot \mathbf{y})$ and $\mathsf{sk} = (\mathbf{x}, \mathbf{y})$
2b. $(\mathsf{pk}, \mathsf{sk}) \leftarrow ((\mathbf{a}, \mathbf{b}), \mathbf{0})$
3. $(\mathbf{m}_0, \mathbf{m}_1) \leftarrow \mathcal{A}(\mathtt{FIND} : \mathsf{pk})$
4. $\mathbf{u} \leftarrow \mathbf{t_1}$, $\mathbf{v} \leftarrow \mathbf{m}_0\mathbf{G} + \mathbf{t_2}$ and $\mathbf{c}^* \leftarrow (\mathbf{u}, \mathbf{v})$
5. $b' \leftarrow \mathcal{D}_\lambda(\mathtt{GUESS} : \mathbf{c}^*)$
4. If $b' == 2$ output `QCSD`
5. If $b' == 3$ output `UNIFORM`

The distribution of $\mathsf{pk}$ is unchanged with respect to the games. If $(\mathbf{H}_{\mathbf{a},\mathbf{b}}, (\mathbf{t}_1, \mathbf{t}_2)^\top)$ follows the 3-$\mathsf{QCSD}(n, w, b_1, b_2)$ distribution with parity, then

$$(\mathbf{t_1}, \mathbf{t_2})^\top = \begin{pmatrix} \mathbf{I}_n & \mathbf{0} & \mathbf{rot}(\mathbf{a}) \\ \mathbf{0} & \mathbf{I}_n & \mathbf{rot}(\mathbf{b}) \end{pmatrix} \cdot (\mathbf{z_1}, \mathbf{z_2}, \mathbf{z_3})^\top$$

with $\omega(\mathbf{z}_1) = \omega(\mathbf{z}_2) = \omega(\mathbf{z}_3) = w$. Thus, $\mathbf{c}^*$ follows the same distribution as in Game $\boldsymbol{G}_2$. If $(\mathbf{H}_{\mathbf{a},\mathbf{b}}, (\mathbf{t_1}, \mathbf{t_2})^\top)$ follows a uniform distribution with $\mathbf{a}$ of parity $b_1$ and $\mathbf{b}$ of parity $b_2$, then $\mathbf{c}^*$ follows the same distribution as in Game $\boldsymbol{G}_3$. We obtain therefore the same equalities for the output probabilities of $\mathcal{D}'_{\mathcal{E},\mathcal{D}_\lambda}$ and $\mathcal{D}_\lambda$ as with the previous games and therefore the advantages of both distinguishers are equal.

**Game $\boldsymbol{G}_4$:** We now encrypt the other plaintext. We chose $\mathbf{r}'_1, \mathbf{r}'_2, \mathbf{e}'$ uniformly at random in $\mathbb{F}^n_{2,w_\mathbf{r}}$ and $\mathbb{F}^n_{2,w_\mathbf{e}}$ and set $\mathbf{u} = \mathbf{r}'_1 + \mathbf{h}\mathbf{r}'_2$ and $\mathbf{v} = \mathbf{m}_1\mathbf{G} + \mathbf{s} \cdot \mathbf{r}'_2 + \mathbf{e}'$. This is the last game we describe explicitly since, even if it is a mirror of Game $\boldsymbol{G}_3$, it involves a new proof.

$\mathbf{Game}^4_{\mathcal{E},\mathcal{A}}(\lambda)$
1. $\mathsf{param} \leftarrow \mathsf{Setup}(1^\lambda)$
2a. $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{param})$ with $\mathsf{pk} = (\mathbf{h}, \mathbf{s}' = \mathbf{x} + \mathbf{h} \cdot \mathbf{y})$ and $\mathsf{sk} = (\mathbf{x}, \mathbf{y})$
2b. $\mathbf{s} \xleftarrow{\$} \mathbb{F}^n_{2,b}$, with $b = \mathbf{s}'(1) \mod 2$
2c. $(\mathsf{pk}, \mathsf{sk}) \leftarrow ((\mathbf{h}, \mathbf{s}), \mathbf{0})$
3. $(\mathbf{m}_0, \mathbf{m}_1) \leftarrow \mathcal{A}(\texttt{FIND} : \mathsf{pk})$
4a. $\mathbf{e}' \xleftarrow{\$} \mathbb{F}^n_{2,w_\mathbf{e}}, \mathbf{r}' = (\mathbf{r}'_1, \mathbf{r}'_2) \xleftarrow{\$} \mathbb{F}^n_{2,w_\mathbf{r}} \times \mathbb{F}^n_{2,w_\mathbf{r}}$
4b. $\mathbf{u} \leftarrow \mathbf{r}'_1 + \mathbf{h}\mathbf{r}'_2$ and $\mathbf{v} \leftarrow \mathbf{m}_1\mathbf{G} + \mathbf{s} \cdot \mathbf{r}'_2 + \mathbf{e}'$
4c. $\mathbf{c}^* \leftarrow (\mathbf{u}, \mathbf{v})$
5. $b' \leftarrow \mathcal{A}(\texttt{GUESS} : \mathbf{c}^*)$
6. $\texttt{RETURN}\ b'$

The outputs from Game $\boldsymbol{G}_3$ and Game $\boldsymbol{G}_4$ follow the exact same distribution, and therefore the two games are indistinguishable from an information-theoretic point of view. Indeed, for each tuple $(\mathbf{r}_1, \mathbf{r}_2, \mathbf{e})$ of Game $\boldsymbol{G}_3$, resulting in a given $(\mathbf{u}, \mathbf{v})$, there is a one to one mapping to a couple $(\mathbf{r}'_1, \mathbf{r}'_2, \mathbf{e}')$ resulting in Game $\boldsymbol{G}_4$ in the *same* $(\mathbf{u}, \mathbf{v})$, namely $\mathbf{r}'_1 = \mathbf{r}_1$, $\mathbf{r}'_2 = \mathbf{r}_2$ and $\mathbf{e}' = \mathbf{m}_0\mathbf{G} + \mathbf{m}_1\mathbf{G}$. This implies that choosing uniformly $(\mathbf{r}_1, \mathbf{r}_2, \mathbf{e})$ in Game $\boldsymbol{G}_3$ and choosing uniformly $(\mathbf{r}'_1, \mathbf{r}'_2, \mathbf{e}')$ in Game $\boldsymbol{G}_4$ leads to the same output distribution for $(\mathbf{u}, \mathbf{v})$.

**Game $\boldsymbol{G}_5$:** In this game, we now pick $\mathbf{r}'_1, \mathbf{r}'_2, \mathbf{e}'$ with the correct weight.

**Game $\boldsymbol{G}_6$:** We now conclude by switching the public key to an honestly generated one.

We do not explicit these last two games as Game $\boldsymbol{G}_4$ and Game $\boldsymbol{G}_5$ are the equivalents of Game $\boldsymbol{G}_3$ and Game $\boldsymbol{G}_2$ except that $\mathbf{m}_1$ is used instead of $\mathbf{m}_0$. A distinguisher between these two games breaks therefore the 3-DQCSD with parity and $\ell = n - n_1 n_2$ erasures assumption too. Similarly Game $\boldsymbol{G}_5$ and Game $\boldsymbol{G}_6$ are the equivalents of Game $\boldsymbol{G}_2$ and Game $\boldsymbol{G}_1$ and a distinguisher between these two games breaks the 2-DQCSD with parity assumption.

We managed to build a sequence of games allowing a simulator to transform a ciphertext of a message $\mathbf{m}_0$ to a ciphertext of a message $\mathbf{m}_1$. Hence, the advantage of an adversary against the IND-CPA experiment is bounded as:

$$\mathsf{Adv}^{\mathsf{ind}}_{\mathcal{E},\mathcal{A}}(\lambda) \leq 2\left(\mathsf{Adv}^{\text{2-DQCSD}}(\lambda) + \mathsf{Adv}^{\text{3-DQCSD}}(\lambda)\right). \tag{28}$$

□

# 5 Known Attacks

The practical complexity of the SD problem for the Hamming metric has been widely studied for more than 50 years. Most efficient attacks are based on Information Set Decoding, a

technique first introduced by Prange in 1962 [36] and improved later by Stern [40], then Dumer [14]. Recent works [32, 3, 33] suggest a complexity of order $2^{cw(1+\mathsf{negl}(1))}$, for some constant $c$. A particular work focusing on the regime $w = \mathsf{negl}(n)$ confirms this formula, with a close dependence between $c$ and the rate $k/n$ of the code being used [10].

**Specific structural attacks.** Quasi-cyclic codes have a special structure which may potentially open the door to specific structural attacks. A first generic attack is the DOOM attack [38] which because of cyclicity implies a gain of $\mathcal{O}(\sqrt{n})$ (when the gain is in $\mathcal{O}(n)$ for MDPC codes, since the code is generated by a small weight vector basis). It is also possible to consider attacks on the form of the polynomial generating the cyclic structure. Such attacks have been studied in [22, 31, 38], and are especially efficient when the polynomial $x^n - 1$ has many low degree factors. These attacks become inefficient as soon as $x^n - 1$ has only two irreducible factors of the form $(x - 1)$ and $x^{n-1} + x^{n-2} + ... + x + 1$, which is the case when $n$ is prime and $q$ generates the multiplicative group $(\mathbb{Z}/n\mathbb{Z})^*$. Such numbers are known up to very large values. We consider such primitive $n$ for our parameters.

**Parameters and tightness of the reduction.** We proposed different sets of parameters in Sec. 1.7 that provide 128 (category 1), 192 (category 3), and 256 (category 5) bits of classical (*i.e.* pre-quantum) security. The quantum-safe security is obtained by dividing the security bits by two (taking the square root of the complexity) [5]. For each security level, we provide different decryption failure rates to better adapt to the adversary computing power. Notice that even if the adversary has access to a quantum computer, this *does not* change the decryption failure rate.[4] Best known attacks include the works from [9, 6, 15, 32, 3, 33] and for quantum attacks, the work of [5]. In the setting $w = \mathcal{O}(\sqrt{n})$, best known attacks have a complexity in $2^{-t\ln(1-R)(1+o(1))}$ where $t = \mathcal{O}(w)$ and $R$ is the rate of the code [10]. In our configuration, we have $t = 2w$ and $R = 1/2$ for the reduction to the 2-DQCSD problem, and $t = 3w_{\mathbf{r}}$ and $R = 1/3$ for the 3-DQCSD problem. By taking into account the DOOM attack [38], and also the fact that we consider balanced vectors $(\mathbf{x}, \mathbf{y})$ and $(\mathbf{r}_1, \mathbf{e}, \mathbf{r}_2)$ for the attack (which costs only a very small factor, since random words have a good probability to be balanced on each block), we need to divide this complexity by approximately $\sqrt{n}$ (up to polylog factor). The term $o(1)$ is respectively $\log\left(\binom{n}{w}^2/\binom{2n}{2w}\right)$ and $\log\left(\binom{n}{w_{\mathbf{r}}}^3/\binom{3n}{3w_{\mathbf{r}}}\right)$ for the 2-DQCSD and 3-DQCSD problems. Overall our security reduction is tight corresponding to generic instances of the classical 2-DQCSD and 3-DQCSD problems according to the best attacks of [10].

# 6 Advantages and Limitations

## 6.1 Advantages

The main advantages of HQC over existing code-based cryptosystems are:

[4] We do not consider the very strong adversarial model where the adversary is given access to a *quantum* decryption oracle.

- its IND-CPA reduction to a well-understood problem on coding theory: the Quasi-Cyclic Syndrome Decoding problem,
- its immunity against attacks aiming at recovering the hidden structure of the code being used,
- close estimations of its decryption failure rate.

The last item allows to achieve a tight reduction for the IND-CCA2 security of the KEM-DEM version through the recent transformation of [23].

### 6.2 Limitations

We have proposed an instantiation of HQC using BCH codes tensored with repetition codes. As seen above, this construction presents the major advantage of making possible and easy to conduct a study of the error vector distribution, yielding a good estimation of the decryption failure rate. HQC might be more efficient with other families of codes, but another analysis would have to be done.

A first limitation to our cryptosystem (at least for the PKE version) is the low encryption rate. It is possible to encrypt 256 bits of plaintext as required by NIST, but increasing this rate also increases the parameters.

As a more general limitation and in contrast with lattices and the so-called Ring Learning With Errors problem, code-based cryptography does not benefit from search to decision reduction for structured codes.

## References


[1] Carlos Aguilar-Melchor, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, and Gilles Zémor. Efficient encryption from random quasi-cyclic codes. *IEEE Transactions on Information Theory*, 64(5):3927–3943, 2018. *4*, *5*

[2] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography with constant input locality. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 92–110. Springer, Heidelberg, August 2007. *7*, *9*

[3] Anja Becker, Antoine Joux, Alexander May, and Alexander Meurer. Decoding random binary linear codes in $2^{n/20}$: How $1 + 1 = 0$ improves information set decoding. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 520–536. Springer, Heidelberg, April 2012. *24*, *33*

[4] Elwyn R Berlekamp, Robert J McEliece, and Henk CA van Tilborg. On the inherent intractability of certain coding problems. *IEEE Transactions on Information Theory*, 24(3):384–386, 1978. http://authors.library.caltech.edu/5607/1/BERieeetit78.pdf. *7*

[5] Daniel J Bernstein. Grover vs. mceliece. In *Post-Quantum Cryptography*, pages 73–80. Springer, 2010. https://cr.yp.to/codes/grovercode-20091123.pdf. *24*, *33*

[6] Daniel J Bernstein, Tanja Lange, and Christiane Peters. Attacking and defending the mceliece cryptosystem. In *Post-Quantum Cryptography*, pages 31–46. Springer, 2008. https://cr.yp.to/codes/mceliece-20080807.pdf. *24*, *33*

[7] Daniel J. Bernstein and Edoardo Persichetti. Towards KEM unification. Cryptology ePrint Archive, Report 2018/526, 2018. https://eprint.iacr.org/2018/526. *11*

[8] Julien Bringer, Hervé Chabanne, Gérard Cohen, Bruno Kindarji, and Gilles Zémor. Optimal iris fuzzy sketches. In *Biometrics: Theory, Applications, and Systems, 2007. BTAS 2007. First IEEE International Conference on*, pages 1–6. IEEE, 2007. https://arxiv.org/abs/0705.3740. *16*

[9] Anne Canteaut and Florent Chabaud. A new algorithm for finding minimum weight words in a linear code: application to mceliece cryptosystem and to narrow-sense bch codes of length 511. *IEEE Transactions on Information Theory*, 44(1):367–378, 1998. http://ieeexplore.ieee.org/document/651067/. *24*, *33*

[10] Rodolfo Canto Torres and Nicolas Sendrier. Analysis of information set decoding for a sub-linear error weight. In Tsuyoshi Takagi, editor, *Post-Quantum Cryptography - 7th International Workshop, PQCrypto 2016, Fukuoka, Japan, February 24-26, 2016, Proceedings*, volume 9606 of *Lecture Notes in Computer Science*, pages 144–161. Springer, 2016. https://hal.inria.fr/hal-01244886. *24*, *33*

[11] Robert Chien. Cyclic decoding procedures for bose-chaudhuri-hocquenghem codes. *IEEE Transactions on information theory*, 10(4):357–363, 1964. *20*

[12] Jean-Sébastien Coron, Helena Handschuh, Marc Joye, Pascal Paillier, David Pointcheval, and Christophe Tymen. Gem: A generic chosen-ciphertext secure encryption method. In *Cryptographers' Track at the RSA Conference*, pages 263–276. Springer, 2002. http://www.di.ens.fr/~pointche/Documents/Papers/2002_rsa.pdf. *11*

[13] Jean-Christophe Deneuville, Philippe Gaborit, and Gilles Zémor. Ouroboros: A simple, secure and efficient key exchange protocol based on coding theory. In Tanja Lange and Tsuyoshi Takagi, editors, *Post-Quantum Cryptography - 8th International Workshop, PQCrypto 2017, Utrecht, The Netherlands, June 26-28, 2017, Proceedings*, volume 10346 of *Lecture Notes in Computer Science*, pages 18–34. Springer, 2017. http://www.unilim.fr/pages_perso/deneuville/files/ba43bf8d80cef2999dbf4308828213ec.pdf. *4*

[14] Ilya Dumer. On minimum distance decoding of linear codes. In *Proc. 5th Joint Soviet-Swedish Int. Workshop Inform. Theory*, pages 50–52, 1991.

https://www.researchgate.net/publication/296573348_On_minimum_distance_decoding_of_linear_codes. 33

[15] Matthieu Finiasz and Nicolas Sendrier. Security bounds for the design of code-based cryptosystems. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 88–105. Springer, Heidelberg, December 2009. 24, 33

[16] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In *Crypto*, volume 99, pages 537–554. Springer, 1999. https://link.springer.com/chapter/10.1007/3-540-48405-1_34. 11

[17] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. *Journal of cryptology*, pages 1–22, 2013. https://link.springer.com/article/10.1007/s00145-011-9114-1. 11

[18] Philippe Gaborit. Shorter keys for code based cryptography. In *Proceedings of the 2005 International Workshop on Coding and Cryptography (WCC 2005)*, pages 81–91, 2005. http://www.unilim.fr/pages_perso/philippe.gaborit/shortIC.ps. 6

[19] Philippe Gaborit and Marc Girault. Lightweight code-based identification and signature. In *2007 IEEE International Symposium on Information Theory*, pages 191–195. IEEE, 2007. https://www.unilim.fr/pages_perso/philippe.gaborit/isit_short_rev.pdf. 7

[20] Danilo Gligoroski. Pqc forum, official comment on bike submission. NIST PQC forum, December 2017. https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-1/official-comments/BIKE-official-comment.pdf. 8

[21] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984. 10

[22] Qian Guo, Thomas Johansson, and Carl Löndahl. A new algorithm for solving ring-lpn with a reducible polynomial. *IEEE Transactions on Information Theory*, 61(11):6204–6212, 2015. https://arxiv.org/abs/1409.0472. 33

[23] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the fujisaki-okamoto transformation. In *Theory of Cryptography Conference*, pages 341–371. Springer, 2017. 4, 11, 12, 13, 28, 34

[24] W Cary Huffman and Vera Pless. *Fundamentals of error-correcting codes*. Cambridge university press, 2010. https://www.amazon.fr/Fundamentals-Error-Correcting-Codes-Cary-Huffman/dp/0521131707. 5

[25] Haodong Jiang, Zhenfeng Zhang, Long Chen, Hong Wang, and Zhi Ma. IND-CCA-secure key encapsulation mechanism in the quantum random oracle model, revisited. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 96–125. Springer, Heidelberg, August 2018. *11*

[26] Haodong Jiang, Zhenfeng Zhang, and Zhi Ma. Tighter security proofs for generic key encapsulation mechanism in the quantum random oracle model. Cryptology ePrint Archive, Report 2019/134, 2019. https://eprint.iacr.org/2019/134. *11*

[27] Laurie L Joiner and John J Komo. Decoding binary bch codes. In *Southeastcon'95. Visualize the Future., Proceedings., IEEE*, pages 67–73. IEEE, 1995. *20*

[28] Shu Lin and Daniel J Costello. *Error control coding*, volume 2. Prentice Hall Englewood Cliffs, 2004. *17, 19*

[29] Zhen Liu and Yanbin Pan. Pqc forum, official comment on hqc submission. NIST PQC forum, January 2018. https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-1/official-comments/HQC-official-comment.pdf. *8*

[30] Zhen Liu, Yanbin Pan, and Tianyuan Xie. Breaking the hardness assumption and ind-cpa security of hqc submitted to nist pqc project. In *International Conference on Cryptology and Network Security*, pages 344–356. Springer, 2018. *8*

[31] Carl Löndahl, Thomas Johansson, Masoumeh Koochak Shooshtari, Mahmoud Ahmadian-Attari, and Mohammad Reza Aref. Squaring attacks on mceliece public-key cryptosystems using quasi-cyclic codes of even dimension. *Designs, Codes and Cryptography*, 80(2):359–377, 2016. https://link.springer.com/article/10.1007/s10623-015-0099-x. *33*

[32] Alexander May, Alexander Meurer, and Enrico Thomae. Decoding random linear codes in $\tilde{O}(2^{0.054n})$. In *Asiacrypt*, volume 7073, pages 107–124. Springer, 2011. https://link.springer.com/chapter/10.1007/978-3-642-25385-0_6. *33*

[33] Alexander May and Ilya Ozerov. On computing nearest neighbors with applications to decoding of binary linear codes. In *EUROCRYPT (1)*, pages 203–228, 2015. http://www.cits.rub.de/imperia/md/content/may/paper/codes.pdf. *33*

[34] Rafael Misoczki, Jean-Pierre Tillich, Nicolas Sendrier, and Paulo SLM Barreto. Mdpc-mceliece: New mceliece variants from moderate density parity-check codes. In *Information Theory Proceedings (ISIT), 2013 IEEE International Symposium on*, pages 2069–2073. IEEE, 2013. https://eprint.iacr.org/2012/409.pdf. *6, 7*

[35] Tatsuaki Okamoto and David Pointcheval. React: Rapid enhanced-security asymmetric cryptosystem transform. *Topics in Cryptology—CT-RSA 2001*, pages 159–174,

2001. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.150.5590&rep=rep1&type=pdf. *11*

[36] Eugene Prange. The use of information sets in decoding cyclic codes. *IRE Transactions on Information Theory*, 8(5):5–9, 1962. http://ieeexplore.ieee.org/document/1057777/. *33*

[37] Tsunekazu Saito, Keita Xagawa, and Takashi Yamakawa. Tightly-secure key-encapsulation mechanism in the quantum random oracle model. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 520–551. Springer, Heidelberg, April / May 2018. *11*

[38] Nicolas Sendrier. Decoding one out of many. In *International Workshop on Post-Quantum Cryptography*, pages 51–67. Springer, 2011. https://eprint.iacr.org/2011/367.pdf. *8*, *33*

[39] Victor Shoup. NTL: A library for doing number theory. 2001. http://www.shoup.net/ntl. *26*, *27*

[40] Jacques Stern. A method for finding codewords of small weight. In *International Colloquium on Coding Theory and Applications*, pages 106–113. Springer, 1988. https://link.springer.com/chapter/10.1007/BFb0019850. *33*