Hamming Quasi-Cyclic (HQC)

Second round version

Updated version 04/21/2020

HQC is an IND-CCA2 KEM running for standardization to NIST's competition in the category "post-quantum public key encryption scheme". Two different sets of parameters HQC and HQC-RMRS are given for the three categories 1, 3 and 5. The main features of the HQC submission are:

- IND-CCA2 KEM
- Small public key size
- Precise DFR analysis
- Efficient implementations based on classical decoding algorithms

Principal Submitters (by alphabetical order):

- Carlos Aguilar Melchor (Supaero)
- Nicolas Aragon (Univ. of Limoges/ partially funded by DGA)
- Slim Bettaieb (Worldline)
- Loïc Bidoux (Worldline)
- Olivier Blazy (Univ. of Limoges)
- Jurjen Bos (Worldline)
- Jean-Christophe Deneuville (ENAC)
- Philippe Gaborit (Univ. of Limoges)
- Edoardo Persichetti (Florida Atlantic Univ.)
- Jean-Marc Robert (Univ. of Toulon)
- Pascal Véron (Univ. of Toulon)
- Gilles Zémor (Univ. of Bordeaux)

Inventors: Same as submitters

Developers: Same as submitters

Owners: Same as submitters

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
jean-christophe.deneuville@enac.fr
+33-631-142-705
ENAC Toulouse
7 avenue Edouard Belin
31 400 Toulouse
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.

1 History of updates on HQC

1.1 Updates for April the 21th 2020

We provide in this update two main theoretical improvements which do not change the scheme and updates on our implementations.

- **(Improvement 1)** We provide in Section 2.4 a more precise analysis of the modelization of the error distribution. This new analysis permits to lower the DFR of our parameters and permits to decrease the size of our public keys by 3% (new parameters are given in Table 2 of Section 2.8.1). The size for 128 security bits is now (3,024 Bytes).

- **(Improvement 2)** We introduce in Section 2.6 a new decoding algorithm based on the concatenation of Reed-Muller and Reed-Solomon codes. This new algorithm does not change the general scheme nor its security and permits to decrease the size of the public key by 17% for 128 security bit (now of size 2,607 Bytes), a new set of parameters, HQC-RMRS, is given in Section 2.8.2 for 128, 192 and 256 bits of security.

- For parameters, we now only consider DFR corresponding to the security level and remove three parameters compared to the round 2 submission. We now only have one set of parameters for each level of security (both for HQC and the HQC-RMRS decoding variation).

- Our implementations gained in efficiency and are now implemented in a constant-time way whenever relevant and they should not leak any sensitive information with respect to timing attacks. We provide new optimized implementations in C and AVX2 for the two sets of parameters HQC and HQC-RMRS in Section 3.1 and 3.2. Moreover our implementations no longer rely on third party libraries.

- We highlight in Section 2.8.3 how it could be possible to further decrease by 10% the size of the public keys with a security reduction to a slight variation of the 3-QCSD problem.

- We welcome Jean-Marc Robert and Pascal Véron from the University of Toulon (France) as new members of our team.

- For 128 bits of security, we obtain the following sizes (in bytes) and performances (in kilocycles):

	Public key size	Ciphertext size	KeyGen	Encaps	Decaps	DFR
HQC	3,024	6,017	142	231	372	$< 2^{-128}$
HQC-RMRS	2,607	5,191	126	213	325	$< 2^{-128}$

1.2 Modifications between Round 1 and Round 2

- Jurjen Bos (from Worldline) joined the HQC team.
- Problems with parity: As previously announced few months ago, the 2 and 3-DQCSD problems with parity distributions have been introduced to counter distinguisher from parity.
- Minor scheme modification : due to the specific use of tensor product codes (BCH and repetition), the length of the code is not required to be a prime. Specifically, the tensor product code has length $n_1 n_2$ with n_1 (resp. n_2) the length of the BCH (resp. repetition) code. In order to avoid algebraic attacks using polynomial factorization, we chose primitive primes n immediately greater than $n_1 n_2$. This results in extra bits, that are truncated where useless. The proof has been modified accordingly.
- The reference implementation now relies on NTL.
- We added an optimized implementation written in C that uses AVX2 instructions and takes advantages of the low Hamming weight of the vectors in HQC.
- We added a constant time implementation of the decoding of BCH codes.
- Parameters providing a Decryption Failure Rate (DFR) higher than 2^{-128} have been discarded.

Contents

2 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].

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 [25], and allows to get an hybrid encryption scheme with strong security guarantees (IND-CCA2),
- 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 detailed and precise upper bound for the decryption failure probability analysis.

Organization of the Specifications. This section is organized as follows: we provide the required background in Sec. 2.1, we make some recalls on encryption and security in Sec. 2.2 then present our proposal in Sec. 2.3. An analysis of the decryption failure rate is proposed in Sec. 2.4. Details about codes being used are provided in Sec. 2.5, together with a specific analysis for these codes. Finally, concrete sets of parameters are provided in Sec. 2.8.

2.1 Preliminaries

2.1.1 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^{th} column is the vector corresponding to $\mathbf{h}X^i$. This is captured by the following definition.

Definition 2.1.1 (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. 2.3 the description of the scheme, and also to [26] for a complete survey on code-based cryptography.

Definition 2.1.2 (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 2.1.3 (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 2.1.4 (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 2.1.5 (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 2.1.6 (Minimum Distance)**.** *Let* $\mathcal{C}$ *be an* $[n, k]$ *linear code over* $\mathcal{R}$ *and let* ω *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 [19] for shortening keys. This results in Quasi-Cyclic Codes, as defined below.

Definition 2.1.7 (Quasi-Cyclic Codes [37])**.** *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 2.1.8 (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 2.1. *The definition of systematic quasi-cyclic codes of index* s *can of course be generalized to all rates* ℓ/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.*

2.1.2 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 2.1.9 (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 2.1.10 (Computational 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 [6]. 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 2.1.11 (Decisional SD Problem)**.** *On input* $(\mathbf{H}, \mathbf{y}^\top) \in \mathbb{F}_2^{(n-k)\times n} \times \mathbb{F}_2^{(n-k)}$*, the* Decisional 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. 2.1.7 and 2.1.11. 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 [20]) and more recently in [37]. Quasi-cyclic codes of index 3 are also considered in [37].

Definition 2.1.12 (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. 2.1.8) 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 2.1.13 ((Computational) 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* Computational 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 [42] 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 decisional 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 [22, 31, 32][1]. The authors are grateful to Ray Perlner for pointing out the existence of such a distinguisher.

Definition 2.1.14 (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 2.1.15 (Decisional 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* Decisional 2-Quasi-Cyclic 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-DQCSD problem with parity. Following Def. 2.1.8, the s-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 2.1.16 (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 DQCSD problem rather than a variable weight version as suggested in [32] for efficiency issues.

Definition 2.1.17 (Decisional 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* Decisional 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. 2.1.15 and 2.1.17). 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 instantiations (BCH codes tensored with a repetition code or concatenated Reed-Muller and Reed-Solomon codes), 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 2.1.18 (Decoding with ℓ 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 ℓ *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 ℓ erasures, and let $(\mathbf{c}, \mathbf{G})$ be an instance of the decoding problem with no erasure. One starts by removing the last ℓ 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. 2.1.15 and 2.1.17 above. (In Sec. 5, $n = \text{next\_primitive\_prime}\,(n_1 n_2) > n_1 n_2$).

2.2 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 λ is the security parameter, generates the global parameters param of the scheme;
- $\mathsf{KeyGen}(\mathsf{param})$ outputs a pair of keys, a (public) encryption key pk and a (private) decryption key sk;
- $\mathsf{Encrypt}(\mathsf{pk}, \mathbf{m}, \theta)$ outputs a ciphertext $\mathbf{c}$, on the message $\mathbf{m}$, under the encryption key pk, with the randomness θ. 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 λ, every $\mathsf{param} \leftarrow \mathsf{Setup}(1^\lambda)$, every pair of keys $(\mathsf{pk}, \mathsf{sk})$ generated by 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 θ.

IND-CPA [23]: 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 [25] 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 [17, 18, 38, 14] but cannot be applied to our scheme due to potential decryption errors.

In [25] 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 γ-spreadness factor of [25] 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. Let us denote by $\mathsf{Encrypt}(\mathsf{pk}, \mathbf{m}, \theta)$ an encryption function that relies on θ to generate random values. The idea of [25] 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 [25] 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 [25, 41, 27, 10, 28] 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 [27] for instance), we chose to keep the presentation of HQC using [25] in order to avoid moving target for NIST evaluation. Any other conversion can be implemented simply.

2.3 Presentation of the scheme

In this section, we describe our proposal: HQC. We begin with the PKE version, then describe the transformation of [25] to obtain a KEM-DEM that achieves IND-CCA2. Parameter sets can be found in Sec. 2.8.

2.3.1 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 δ 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 2.5.1 for more details) or concatenated Reed-Muller and Reed-Solomon codes. The BCH codes are defined implicitly using their generator polynomials. In fact, as explained in Section 2.5.2, 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 ℓ 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. 2.4.

2.3.2 KEM/DEM version (HQC.KEM)

Let $\mathcal{E}$ be an instance of the HQC.PKE cryptosystem as described above. Let $\mathcal{G}$, $\mathcal{H}$, and $\mathcal{K}$ be hash functions, the KEM-DEM version of the HQC cryptosystem is described in Figure 3.

- $\mathsf{Setup}(1^\lambda)$: 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 [25], the HQC.KEM is IND-CCA2. More details regarding the tightness of the reduction are provided at the end of Sec. 2.8.

Security concerns and implementation details. Notice that while NIST only recommends SHA512 as a hash function, the transformation of [25] 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 SHA3-512 for $\mathcal{G}$ and SHA512 for $\mathcal{H}$.

2.3.3 A hybrid encryption scheme (HQC.HE)

NIST announced that they will be using generic transformations to convert any IND-CCA2 KEM into an IND-CCA2 PKE although 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.

2.4 Analysis of the error vector distribution for Hamming distance

In this section we provide a more precise analysis of the error distribution approximation compared to the Round 2 submission. This analysis is taken from [3]. We first compute exactly the probability distribution of each fixed coordinate e'_k of the error vector

$$\mathbf{e}' = \mathbf{x} \cdot \mathbf{r}_2 - \mathbf{r}_1 \cdot \mathbf{y} + \mathbf{e} = (e'_0, \dots e'_{n-1}).$$

We obtain that every coordinate e'_k is Bernoulli distributed with parameter $p^* = P[e'_k = 1]$ given by Proposition 2.4.2.

To compute decoding error probabilities, we will then need the probability distribution of the weight of the error vector $\mathbf{e}'$ restricted to given sets of coordinates that correspond to codeword supports. We will make the simplifying assumption that the coordinates e'_k of $\mathbf{e}'$ are independent variables, which will let us work with the binomial distribution of parameter p^* for the weight distributions of $\mathbf{e}'$. In other words we modelize the error vector as a binary symmetric channel with parameters p^*. This working assumption is justified by remarking that, in the high weight regime relevant to us, since the component vectors $\mathbf{x}, \mathbf{y}, \mathbf{e}$ have fixed weights, the probability that a given coordinate e'_k takes the value 1 conditioned on abnormally many others equalling 1 can realistically only be $\leq p^*$. We support this modeling of the otherwise intractable weight distribution of $\mathbf{e}'$ by extensive simulations: these back up our assumption that our computations of decoding error probabilities and DFRs can only be upper bounds on their real values.

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

Proposition 2.4.1. *Let $\mathbf{x} = (x_0, \dots x_{n-1})$ be a random vector chosen uniformly among all binary vectors of weight w and let $\mathbf{r} = (r_0, \dots, r_{n-1})$ be a random vector chosen uniformly among all vectors of weight w_r and independently of $\mathbf{x}$. Then, denoting $\mathbf{z} = \mathbf{x} \cdot \mathbf{r}$, we have that for every $k \in \{0, \dots n-1\}$, the k-th coordinate z_k of $\mathbf{z}$ is Bernoulli distributed with parameter $\tilde{p} = P(z_k = 1)$ equal to:*

$$\tilde{p} = \frac{1}{\binom{n}{w}\binom{n}{w_{\mathbf{r}}}} \sum_{\substack{1 \leqslant \ell \leqslant \min(w, w_{\mathbf{r}}) \\ \ell \text{ odd}}} C_\ell$$

where $C_\ell = \binom{n}{\ell}\binom{n-\ell}{w-\ell}\binom{n-w}{w_{\mathbf{r}}-\ell}$.

Proof. The total number of ordered pairs $(\mathbf{x}, \mathbf{r})$ is $\binom{n}{w}\binom{n}{w_{\mathbf{r}}}$. Among those, we need to count how many are such that $z_k = 1$. We note that

$$z_k = \sum_{\substack{i+j=k \bmod n \\ 0 \le i,j \le n-1}} x_i r_j.$$

We need therefore to count the number of couples $(\mathbf{x}, \mathbf{r})$ such that we have $x_i r_{k-i} = 1$ an odd number of times when i ranges over $\{0, \dots, n-1\}$ (and $k-i$ is understood modulo n). Let us count the number C_ℓ of couples $(\mathbf{x}, \mathbf{r})$ such that $x_i r_{k-i} = 1$ exactly ℓ times. For $\ell > \min(w, w_{\mathbf{r}})$ we clearly have $C_\ell = 0$. For $\ell \le \min(w, w_{\mathbf{r}})$ we have $\binom{n}{\ell}$ choices for the set of coordinates i such that $x_i = r_{k-i} = 1$, then $\binom{n-\ell}{w}$ remaining choices for the set of coordinates i such that $x_i = 1$ and $r_{k-i} = 0$, and finally $\binom{n-w}{w_{\mathbf{r}}-\ell}$ remaining choices for the set of coordinates i such that $x_i = 0$ and $r_{k-i} = 1$. Hence $C_\ell = \binom{n}{\ell}\binom{n-\ell}{w-\ell}\binom{n-w}{w_{\mathbf{r}}-\ell}$. The formula for $\tilde{p}$ follows. □

Let $\mathbf{x}, \mathbf{y}$ (resp. $\mathbf{r}_1, \mathbf{r}_2$) be independent random vectors chosen uniformly among all binary vectors of weight w (resp. $w_{\mathbf{r}}$).

By independence of $(\mathbf{x}, \mathbf{r}_2)$ with $(\mathbf{y}, \mathbf{r}_1)$, the k-th coordinates of $\mathbf{x} \cdot \mathbf{r}_2$ and of $\mathbf{r}_1 \cdot \mathbf{y}$ are independent, and they are Bernoulli distributed with parameter $\tilde{p}$ by Proposition 2.4.1. Therefore their modulo 2 sum $\mathbf{t} = \mathbf{x} \cdot \mathbf{r}_2 - \mathbf{r}_1 \cdot \mathbf{y}$ is Bernoulli distributed with

$$\begin{cases} \Pr[t_k = 1] = 2\tilde{p}(1-\tilde{p}), \\ \Pr[t_k = 0] = (1-\tilde{p})^2 + \tilde{p}^2. \end{cases} \tag{14}$$

Finally, by adding modulo 2 coordinatewise the two independent vectors $\mathbf{e}$ and $\mathbf{t}$, we obtain the distribution of the coordinates of the error vector $\mathbf{e}' = \mathbf{x} \cdot \mathbf{r}_2 - \mathbf{r}_1 \cdot \mathbf{y} + \mathbf{e}$ given by the following proposition:

Proposition 2.4.2. *Let $\mathbf{x}, \mathbf{y}, \mathbf{r}_1, \mathbf{r}_2, \mathbf{e}$ be independent random vectors with uniform distributions among vectors of fixed weight w for $\mathbf{x}, \mathbf{y}$, among vectors of weight $w_{\mathbf{r}}$ for $\mathbf{r}_1, \mathbf{r}_2$, and among vectors of weight $w_{\mathbf{e}}$ for $\mathbf{e}$. Let $\mathbf{e}' = \mathbf{x} \cdot \mathbf{r}_2 - \mathbf{r}_1 \cdot \mathbf{y} + \mathbf{e} = (e'_0, \dots, e'_{n-1})$. Then for any $k = 0 \dots n-1$, the coordinate e'_k has distribution:*

$$\begin{cases} \Pr[e'_k = 1] = 2\tilde{p}(1-\tilde{p})(1-\frac{w_{\mathbf{e}}}{n}) + ((1-\tilde{p})^2 + \tilde{p}^2)\frac{w_{\mathbf{e}}}{n}, \\ \Pr[e'_k = 0] = ((1-\tilde{p})^2 + \tilde{p}^2)(1-\frac{w_{\mathbf{e}}}{n}) + 2\tilde{p}(1-\tilde{p})\frac{w_{\mathbf{e}}}{n}. \end{cases} \tag{15}$$

Proposition 2.4.2 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 α, 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. (15): $p^\star = 2\tilde{p}(1-\tilde{p})(1-\frac{w_{\mathbf{e}}}{n}) + ((1-\tilde{p})^2 + \tilde{p}^2)\frac{w_{\mathbf{e}}}{n}$. This approximation will give us, for $0 \le d \le \min(2 \times w \times w_{\mathbf{r}} + w_{\mathbf{e}}, n)$,

$$\Pr[\omega(\mathbf{e}') = d] = \binom{n}{d}(p^\star)^d(1-p^\star)^{(n-d)}. \tag{16}$$

Supporting elements for our modelization: we give in Fig. 4 and 5 simulations of the distribution of the weight of the error vector together with the distribution of the associated binomial law of parameters $p^\star$. These simulations show that error vectors are more likely to have a weight close to the mean than predicted by the binomial distribution, and that on the contrary the error is less likely to be of large weight than if it were binomially distributed. This is for instance illustrated on parameters sets I and II corresponding to real parameters used for 128 bits security. For cryptographic purposes we are mainly interested by very small DFR and large weight occurrences which are more likely to induce decoding errors. These tables show that the probability of obtaining a large weight is close but smaller for the error weight distribution of e' rather than for the binomial approximation. This supports our modelization and the fact that computing the decoding failure probability with this binomial approximation permits to obtain an upper bound on the real DFR. This will be confirmed in the next sections by simulations with real weight parameters (but smaller lengths).

Examples of simulations. We consider two examples of parameters: Parameter sets I and II which correspond to cryptographic parameters and for which we simulate the error distribution versus the binomial approximation together with the probability of obtaining large error weights. In order to match definition 2.1.18 we computed vectors of length n and then truncated the last $l = n - n_1 n_2$ bits before measuring the Hamming weight of the vectors.

Parameter set	w	$w_{\mathbf{e}} = w_{\mathbf{r}}$	n	$n_1 n_2$	$p^\star$
I	67	77	23869	23746	0.2918
II	67	77	20533	20480	0.3196

Simulations for Parameter set I

Simulation results are shown figure 4. We computed the weights such that 0.1%, 0.01% and 0.001% of the vectors are of weight greater than this value, to study how often extreme weight values occur. Results are presented table 1.

	0.1%	0.01%	0.001%	0.0001%
Error vectors	7101	7134	7163	7190
Binomial approximation	7147	7191	7228	7267

Table 1: Simulated probabilities of large weights for Parameter I for the distributions of the error vector and the binomial approximation

Simulations for Parameter set II

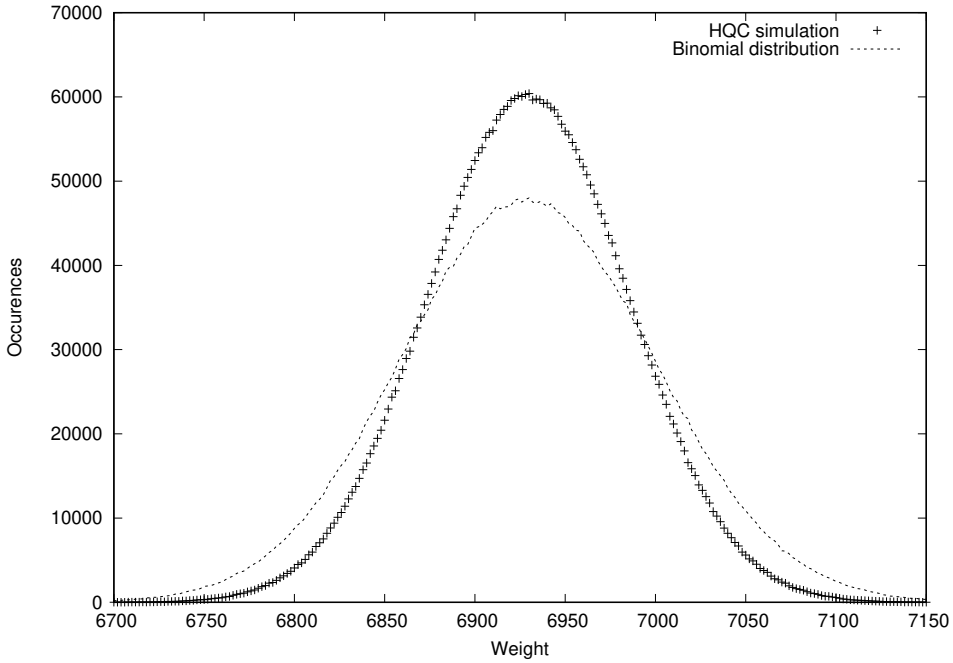

Figure 4: Comparison between error $\mathbf{e}'$ generated using parameter set I and its binomial approximation.

	0.1%	0.01%	0.001%	0.0001%
Error vectors	6715	6749	6779	6808
Binomial approximation	6753	6796	6834	6859

Table 2: Simulated probabilities of large weights for Parameter II for the distributions of the error vector and the binomial approximation

Simulation results are shown on figure 5. We perform the same analysis as for the parameter set I about extreme weight values. Results are presented table 2.

Comparison with the previous analysis of the initial Round 2 submission: the present analysis is better than the previous one, in practice in the case of decoding with BCH and repetition codes for security parameter 128 bits, the present analysis leads to a DFR in 2^{-154} when the previous one leaded to 2^{-128}, this is what enables us to reduce in the following sections the key size in the case of the BCH-repetition code decoder.

2.5 Decoding with tensor product of BCH and repetition codes

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

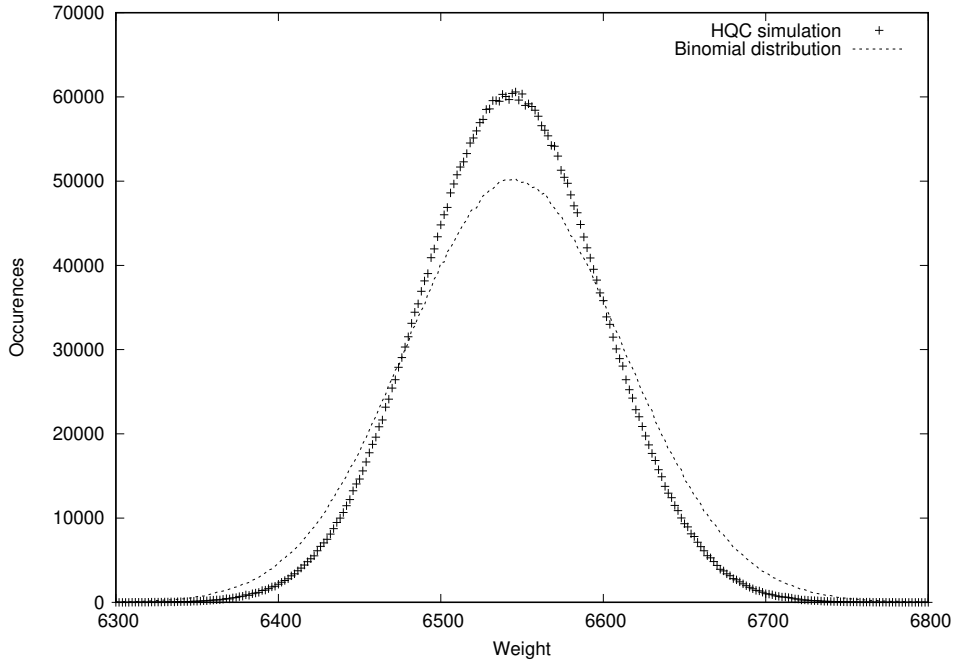

Figure 5: Comparison between error $\mathbf{e}'$ generated using parameter set II and its binomial approximation.

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 [11], 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.

2.5.1 Tensor product codes

Definition 2.5.1 (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

$$\mathcal{C}_1 \otimes \mathcal{C}_2 = \left\{ \mathbf{G}_2^\top \mathbf{X} \mathbf{G}_1 \text{ for } \mathbf{X} \in \mathbb{F}_2^{k_2 \times k_1} \right\} \tag{17}$$

Remark 2.2. *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. 2.4 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, δ_1) code of length n_1, dimension k_1, and correcting capability δ_1 (*i.e.* it can correct any pattern of δ_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[2] (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:

$$\mathbb{1}_{n_2}.\mathsf{Decode}(\tilde{\mathbf{m}}_{1,j}) = \begin{cases} 1 \text{ if } \sum_{i=0}^{n_2-1} \tilde{\mathbf{m}}_{1,j,i} \geq \lceil \frac{n_2+1}{2} \rceil, \\ 0 \text{ otherwise.} \end{cases} \tag{18}$$

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

2.5.2 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 [30]:

- Block length $n = 2^m - 1$
- Number of parity-check digits $n - k \leq m\delta$, with δ, 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 α be a primitive element in $\mathbb{F}_{2^m}$, the generator polynomial $g(x)$ of the BCH$[n, k, \delta]$ code is given by:

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

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

with $\phi_i(x)$ being the minimal polynomial of α^i (refer to [30] 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 [30]).

Code	n	k	δ
BCH-1	1023	513	57
BCH-2	1023	483	60
BCH-S1	766	256	57
BCH-S2	796	256	60

Table 3: 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 [30]). 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. $g_2(x) = 1+x+x^3+x^4+x^6+x^{11}+x^{13}+x^{15}+x^{16}+$
$x^{17}+x^{18}+x^{19}+x^{20}+x^{22}+x^{23}+x^{25}+x^{26}+x^{27}+x^{28}+x^{29}+x^{32}+x^{35}+x^{37}+x^{38}+x^{40}+x^{43}+$
$x^{44}+x^{45}+x^{48}+x^{49}+x^{50}+x^{54}+x^{56}+x^{58}+x^{60}+x^{61}+x^{64}+x^{66}+x^{69}+x^{73}+x^{74}+x^{75}+x^{76}+$
$x^{77}+x^{78}+x^{79}+x^{82}+x^{83}+x^{84}+x^{85}+x^{87}+x^{89}+x^{91}+x^{92}+x^{94}+x^{97}+x^{99}+x^{100}+x^{101}+$
$x^{105}+x^{106}+x^{107}+x^{108}+x^{112}+x^{114}+x^{115}+x^{116}+x^{117}+x^{119}+x^{122}+x^{123}+x^{124}+x^{125}+$
$x^{127}+x^{131}+x^{134}+x^{135}+x^{136}+x^{137}+x^{138}+x^{140}+x^{144}+x^{146}+x^{147}+x^{151}+x^{153}+x^{156}+$
$x^{160}+x^{161}+x^{163}+x^{165}+x^{167}+x^{168}+x^{170}+x^{171}+x^{174}+x^{175}+x^{176}+x^{179}+x^{181}+x^{184}+$
$x^{187}+x^{188}+x^{190}+x^{195}+x^{196}+x^{201}+x^{203}+x^{204}+x^{206}+x^{207}+x^{209}+x^{213}+x^{214}+x^{215}+$
$x^{217}+x^{219}+x^{220}+x^{223}+x^{224}+x^{226}+x^{227}+x^{231}+x^{233}+x^{234}+x^{238}+x^{245}+x^{250}+x^{251}+$
$x^{254}+x^{258}+x^{259}+x^{262}+x^{263}+x^{264}+x^{268}+x^{271}+x^{272}+x^{273}+x^{274}+x^{280}+x^{281}+x^{284}+$
$x^{286}+x^{287}+x^{288}+x^{289}+x^{290}+x^{291}+x^{293}+x^{294}+x^{295}+x^{298}+x^{299}+x^{302}+x^{303}+x^{304}+$
$x^{305}+x^{306}+x^{311}+x^{313}+x^{314}+x^{315}+x^{317}+x^{319}+x^{324}+x^{325}+x^{330}+x^{334}+x^{336}+x^{338}+$
$x^{340}+x^{341}+x^{345}+x^{347}+x^{348}+x^{352}+x^{355}+x^{358}+x^{359}+x^{362}+x^{364}+x^{367}+x^{368}+x^{369}+x^{370}+$
$x^{373}+x^{374}+x^{377}+x^{378}+x^{380}+x^{382}+x^{383}+x^{392}+x^{394}+x^{395}+x^{402}+x^{403}+x^{405}+x^{407}+x^{413}+$
$x^{414}+x^{415}+x^{417}+x^{419}+x^{420}+x^{422}+x^{423}+x^{424}+x^{425}+x^{427}+x^{428}+x^{429}+x^{431}+x^{432}+x^{434}+$
$x^{435}+x^{436}+x^{437}+x^{438}+x^{441}+x^{444}+x^{445}+x^{452}+x^{454}+x^{460}+x^{462}+x^{463}+x^{464}+x^{465}+x^{466}+$
$x^{469}+x^{470}+x^{471}+x^{476}+x^{478}+x^{479}+x^{480}+x^{481}+x^{484}+x^{487}+x^{488}+x^{489}+x^{490}+x^{491}+x^{502}+$
$x^{504}+x^{506}+x^{507}+x^{509}+x^{512}+x^{514}+x^{515}+x^{521}+x^{522}+x^{523}+x^{524}+x^{526}+x^{529}+x^{534}+x^{540}$.

2.5.3 Encoding shortened BCH codes

In the following we present the encoding of classical BCH codes which can also be used to encode 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 [30], 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 $c(x) = b(x) + x^{n-k}u(x)$.

Notice that for the AVX implementation we classically encode the message m using the systematic form of the generator matrix G since each column can be represented in a 256-bit integer.

2.5.4 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 [30], 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 α 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 the 2δ syndromes using the received polynomial. In our implementation, we compute these syndromes by the transpose of the additive

Fast Fourier Transform (FFT) as suggested in [8]. Notice that for the AVX implementation, the syndromes are computed in a classical way. We evaluate $v(\alpha^i)$ for each value of i. Since each S_i is a 16-bit integer, 16 syndromes can be stored in a 256-bit value. This trick combined to the use of a precomputed table of the value α^{ij} allows a faster computation.

2. The second step is the computation of the error-location polynomial $\sigma(x)$ from the 2δ syndromes computed in the first step. Here we use Berlekamp's simplified algorithm [29].

3. The third step is to find the error-location numbers by calculating the roots of the polynomial $\sigma(x)$ and returning their inverses. We implement this step with an additive Fast Fourier Transform algorithm from [21].

4. The fourth step is the correction of errors in the received polynomial.

Remark 2.3. *As mentioned before, in our implementation, we deal with shortened BCH codes. We notice that we will be using the same decoding procedure described above.*

2.5.5 Decryption Failure Probability

With a tensor product code $\mathcal{C} = \mathrm{BCH}(n_1, k_1, \delta) \otimes \mathbb{1}_{n_2}$ 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.

We first focus on the probability that an error occurs while decoding the repetition code. As shown in Sec. 2.4, 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. (15)). As mentioned above, $\mathbb{1}_{n_2}$ can decode up to $\delta_2 = \lfloor \frac{n_2 - 1}{2} \rfloor$ errors. Therefore, the probability that $\mathbb{1}_{n_2}$ does not decode correctly is given by for odd n_2 by:

$$p_i = \sum_{i=n_2/2}^{n_2} \binom{n_2}{i} p^{\star i} (1 - p^\star)^{n_2 - j} \tag{19}$$

for even n_2, when the error has weight $\frac{n_2}{2}$, the probability of not decoding correctly is $1/2$, we fix for instance that in that case we decode the word as 0. In which case the probability of not decoding correctly becomes:

$$p_i = \frac{1}{2} \binom{n_2}{\frac{n_2}{2}} p^{\star \frac{n_2}{2}} (1 - p^\star)^{\frac{n_2}{2}} + \sum_{i=\frac{n_2}{2}+1}^{n_2} \binom{n_2}{i} p^{\star i} (1 - p^\star)^{n_2 - j} \tag{20}$$

Notice that in practice (except for simulations) we only consider odd n_2 in our parameters.

We now focus on the BCH (n_1, k_1, δ_1) code, and recall that it can correct any pattern of δ_1 errors. Now the probability p_e that the BCH(n_1, k_1, δ_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 the following theorem:

Theorem 2.4. *The probability p_e that the BCH (n_1, k_1, δ_1) code does not decode correctly is given by:*

$$p_e = \sum_{l=\delta_1+1}^{n_1} \binom{n_1}{l} p_i^l (1-p_i)^{n_1-l} \tag{21}$$

Eq. (21) gives a theoretical approximation of the decryption failure rate. The parameters presented in Tab. 7 were obtained using this formula. Experimental evidences supporting the validity of the assumptions made on the binomial law to obtain this upper bound on the DFR are provided in Fig. 6.

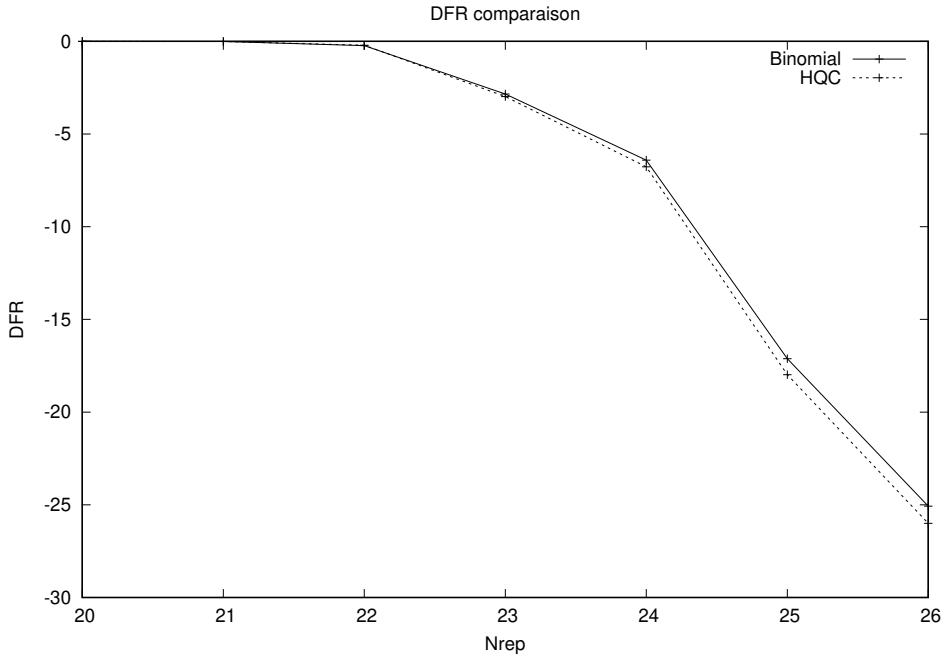

Figure 6: Comparison of the Decryption Failure Rate of tensor product codes against approximation by a binary symmetric channel and against HQC error vectors. Parameters simulated are derived from HQC parameters for 128 security bits: $w = 67, w_r = w_e = 77$, a $[766, 256]$ BCH code which corrects 57 errors and Repetition codes of lengths Nrep.

2.6 Decoding with concatenated Reed-Muller and Reed-Solomon codes

In this section taken from [3] we propose to consider a new decoding algorithm based on Reed-Muller and Reed-Solomon concatenated codes, this different decoding algorithm does not modify the general scheme which is valid for any decoding algorithm nor the general security reduction of the scheme. This novel approach is presented by members of our team in [3]. Overall this new decoding algorithm permits to obtain a gain in the size of the public key of order 15% to 17%.

2.6.1 Definitions

Definition 2.6.1 (Concatenated codes)**.** *A concatenated code consists of an external code* $[n_e, k_e, d_e]$ *over* $\mathbb{F}_q$ *and an internal code* $[n_i, k_i, d_i]$ *over* $\mathbb{F}_2$*, with* $q = 2^{k_i}$*. We use a bijection between elements of* $\mathbb{F}_q$ *and the words of the internal code, this way we obtain a transformation:*

$$\mathbb{F}_q^{n_e} \to \mathbb{F}_2^N$$

where $N = n_e n_i$*. The external code is thus transformed into a binary code of parameters* $[N = n_e n_i, K = k_e k_i, D \geqslant d_e d_i]$.

For the external code, we chose a Reed-Solomon code of dimension 32 over $\mathbb{F}_{256}$ and, for the internal code, we chose the Reed-Muller code $[128, 8, 64]$ that we are going to duplicate between 2 and 6 times (i.e duplicating each bit to obtain codes of parameters $[256, 8, 128], [512, 8, 256], [768, 8, 384]$).

We perform maximum likelihood decoding on the internal code. Doing that we obtain a vector of $\mathbb{F}_q^{n_e}$ that we then decode using an algebraic decoder for the Reed-Solomon code.

2.6.2 Reed-Solomon codes

Let p be a prime number and q is any power of p. Following [30], a Reed-Solomon code with symbols in $\mathbb{F}_{q^p}$ has the following parameters:

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

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

$$g(x) = (x + \alpha)(x + \alpha^2) \cdots (x + \alpha^{2\delta})$$

Depending on HQC-RMRS parameters, we construct shortened Reed-Solomon (RS-S1, RS-S2 and RS-S3) codes such that $k = 256$ from the two following RS codes RS-1, RS-2 and RS-3 (codes from [30]).

Code	n	k	δ
RS-1	255	207	24
RS-2	255	211	22
RS-3	255	209	23
RS-S1	80	32	24
RS-S2	76	32	22
RS-S3	78	32	23

Table 4: Original and shortened Reed-Solomon codes.

The shortened codes are obtained by subtracting 175 from the parameters n and k of the code RS-1 and subtracting 179 from the parameters n and k of the code RS-2 and by subtracting 177 from the parameters n and k of the code RS-3. Notice that shortening the Reed-Solomon code does not affect the correcting capacity, thus we have the following shortened Reed-Solomon codes :

- RS-S1$[80 = 255 - 175,\ 32 = 207 - 175,\ 24]$
- RS-S2$[76 = 255 - 179,\ 32 = 211 - 179,\ 22]$
- RS-S3$[78 = 255 - 177,\ 32 = 209 - 177,\ 23]$

In our case, we will be working in $\mathbb{F}_{2^m}$ with $m = 8$. To do so, we use the primitive polynomial $1 + x^2 + x^3 + x^4 + x^8$ of degree 8 to build this field (polynomial from [30]). We denote by $g_1(x)$, $g_2(x)$ and $g_3(x)$ the generator polynomials of RS-S1, RS-S2 and RS-S3 respectively, which are equal to the generator polynomials of Reed-Solomon codes RS-1, RS-2 and RS-3 respectively. We precomputed the generator polynomials $g_1(x)$, $g_2(x)$ and $g_3(x)$ of the code RS-S1, RS-S2 and RS-S3 and we included them in the file `parameters.h`. One can use the functions provided in the file `reed_solomon.h` to reconstruct the generator polynomials for those codes.

Generator polynomial of RS-1. $g_1(x) = 228 + 231x + 214x^2 + 81x^3 + 113x^4 + 204x^5 + 19x^6 + 169x^7 + 10x^8 + 244x^9 + 117x^{10} + 219x^{11} + 130x^{12} + 12x^{13} + 160x^{14} + 151x^{15} + 195x^{16} + 170x^{17} + 150x^{18} + 151x^{19} + 251x^{20} + 218x^{21} + 245x^{22} + 166x^{23} + 149x^{24} + 183x^{25} + 109x^{26} + 176x^{27} + 148x^{28} + 218x^{29} + 21x^{30} + 161x^{31} + 240x^{32} + 25x^{33} + 15x^{34} + 71x^{35} + 62x^{36} + 5x^{37} + 17x^{38} + 32x^{39} + 157x^{40} + 194x^{41} + 73x^{42} + 195x^{43} + 218x^{44} + 14x^{45} + 12x^{46} + 122x^{47} + x^{48}$.

Generator polynomial of RS-2. $g_2(x) = 36 + 248x + 168x^2 + 57x^3 + 215x^4 + 30x^5 + 166x^6 + 88x^7 + 196x^8 + 173x^9 + 61x^{10} + 214x^{11} + 100x^{12} + 134x^{13} + 168x^{14} + 41x^{15} + 57x^{16} +$

$216x^{17} + 254x^{18} + 252x^{19} + 68x^{20} + 194x^{21} + 32x^{22} + 234x^{23} + 205x^{24} + 39x^{25} + 99x^{26} + 6x^{27} + 66x^{28} + 127x^{29} + 53x^{30} + 221x^{31} + 1x^{32} + 17x^{33} + 7x^{34} + 219x^{35} + 161x^{36} + 30x^{37} + 173x^{38} + 30x^{39} + 51x^{40} + 95x^{41} + 58x^{42} + 65x^{43} + x^{44}$.

Generator polynomial of RS-3. $g_3(x) = 111 + 204x + 161x^2 + 232x^3 + 250x^4 + 107x^5 + 58x^6 + 200x^7 + 161x^8 + 225x^9 + 89x^{10} + 57x^{11} + 143x^{12} + 12x^{13} + 106x^{14} + 116x^{15} + 218x^{16} + 68x^{17} + 19x^{18} + 127x^{19} + 236x^{20} + 48x^{21} + 3x^{22} + 210x^{23} + 243x^{24} + 174x^{25} + 203x^{26} + 57x^{27} + 78x^{28} + 15x^{29} + 185x^{30} + 168x^{31} + 204x^{32} + 194x^{33} + 152x^{34} + 17x^{35} + 189x^{36} + 200x^{37} + 197x^{38} + 162x^{39} + 244x^{40} + 8x^{41} + 248x^{42} + 163x^{43} + 217x^{44} + 31x^{45} + x^{46}$.

2.6.3 Encoding shortened Reed-Solomon codes

The encoding of shortened of Reed-Solomon codes is performed in the same way as the encoding of BCH codes using an LFSR (see Section 2.5.3).

2.6.4 Decoding shortened Reed-Solomon codes

The decoding of classical Reed-Solomon codes can be used to decode shortened Reed-Solomon codes. For sake of simplicity, we will detail the process of decoding classical Reed-Solomon codes. Following [30], consider the Reed-Solomon 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 $e_i \neq 0$. 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 α 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.* $e(x) = e_{j_1}x^{j_1} + e_{j_2}x^{j_2} + \cdots + e_{j_t}x^{j_t}$. We obtain the following set of equations, where $\alpha^{j_1}, \alpha^{j_2}, \cdots, \alpha^{j_t}$ are unknown:

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

The goal of a Reed-Solomon 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:

$$\begin{cases} S_1 &= e_{j_1}\beta_1 + e_{j_2}\beta_2 + \cdots + e_{j_t}\beta_t \\ S_2 &= e_{j_1}\beta_1^2 + e_{j_2}\beta_2^2 + \cdots + e_{j_t}\beta_t^2 \\ S_3 &= e_{j_1}\beta_1^3 + e_{j_2}\beta_2^3 + \cdots + e_{j_t}\beta_t^3 \\ & \vdots \\ S_{2\delta} &= e_{j_1}\beta_1^{2\delta} + e_{j_2}\beta_2^{2\delta} + \cdots + e_{j_t}\beta_t^{2\delta} \end{cases}$$

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. After retrieving the coefficients of $\sigma(x)$, we can compute the error values. Let

$$Z(x) = 1 + (S_1 + \sigma_1)x + (S_2 + \sigma_1 S_1 + \sigma_2)x^2 + \cdots + (S_t + \sigma_1 S_{t-1} + \sigma_2 S_{t-2} + \cdots + \sigma_t)x^t$$

The error value at location β_l is given by [5]

$$e_{j_l} = \frac{Z(\beta_l^{-1})}{\prod_{\substack{i=1 \\ i \neq l}}^{t} (1 + \beta_i \beta_l^{-1})}$$

The decoding is completed by computing $r(x) - e(x)$.
We can summarize the decoding procedure by the following steps:

1. The first step is the computation of the 2δ syndromes using the received polynomial. The syndromes are computed in a classical way by evaluating $r(\alpha^i)$ for each value of i.

2. The second step is the computation of the error-location polynomial $\sigma(x)$ from the 2δ syndromes computed in the first step. Here we use Berlekamp's algorithm [30].

3. The third step is to find the error-location numbers by calculating the roots of the polynomial $\sigma(x)$ and returning their inverses. We implement this step with an additive Fast Fourier Transform algorithm from [21].

4. The fourth step is the computation of the polynomial $Z(x)$.

5. The fifth step is the computation of the error values.

6. The sixth step is the correction of errors in the received polynomial.

2.6.5 Duplicated Reed-Muller codes

For any positive integers m and r with $0 \leq r \leq m$, there exists a binary r^{th} order Reed-Muller code denoted by $RM(r, m)$ with the following parameters:

- Code length $n = 2^m$
- Dimension $k = \sum_{i=0}^{r} \binom{m}{i}$
- Minimum distance $d_{min} = 2^{m-r}$

HQC-RMRS uses duplicated Reed-Muller codes. In particular, we are using first-order Reed-Muller denoted $RM(1, 7)$ which is the binary code $[128, 8, 64]$.

Decoding the internal Reed-Muller code:

The Reed-Muller code of order 1 can be decoded using a fast Hadamard transform (see chapter 14 of MacWilliams and Sloane for example). The algorithm needs to be slightly adapted when decoding duplicated codes. For example, if the Reed-Muller is duplicated three times, we create the function $F : \mathbb{F}_2^7 \to 3, 1, -1, -3^7$ where we started with transforming each block of three bits $x_1x_2x_3$ of the received vector in

$$(-1)^{x_1} + (-1)^{x_2} + (-1)^{x_3}$$

We then apply the Hadamard transform to the function F. We take the maximum value in $\hat{F}$ and $x \in \mathbb{F}_2^7$ that maximizes the value of $|\hat{F}|$. If $\hat{F}(x)$ is positive, then the closest codeword is xG where G is the generator matrix of the Hadamard code (without the all-one-vector). If $\hat{F}(x)$ is negative, then we need to add the all-one-vector to it.

2.6.6 Encoding Duplicated Reed-Muller codes

Following [34], the encoding is done in classical way by using a matrix vector multiplication. The codeword is then duplicated depending on the used parameter (see Table 5).

Scheme	Reed-Muller Code	Multiplicity	Duplicated Reed-Muller Code
HQC-RMRS-128	[128, 8, 64]	2	[256, 8, 128]
HQC-RMRS-192	[128, 8, 64]	4	[512, 8, 256]
HQC-RMRS-256	[128, 8, 64]	6	[768, 8, 384]

Table 5: Duplicated Reed-Muller codes.

2.6.7 Decoding Duplicated Reed-Muller codes

Following [34] (Chapter 14), the decoding of duplicated Reed-Muller codes is done in three steps:

1. The first step is the computation of the function F described in Section 2.6.5. We apply F on the received codeword. We give details about how this process is done for HQC-RMRS-128 where the multiplicity is equal to 2. Let v a duplicated Reed-Muller codeword, it can be seen as $v = (a_1 b_1, \cdots, a_{n_2} b_{n_2})$ where each a_i, b_i has 128 bits size ($a_i = (a_{i_0}, \cdots, a_{i_{128}})$ and $b_i = (b_{i_0}, \cdots, b_{i_{128}})$). The transformation F is applied to each element in v as follows $((-1)^{a_{i_0}} + (-1)^{b_{i_0}}, \cdots, (-1)^{a_{i_{128}}} + (-1)^{b_{i_{128}}})$. The cases when multiplicity is equal to 4 (in HQC-RMRS-192) and 6 (in HQC-RMRS-256) follow a similar process.

2. The second step is the computation of Hadamard transform which is the first phase of the Green machine.

3. The third step is the computation of the location of the highest value on the output of the previous step. This is the second phase of the Green machine. When the peak is positive we add all-one-vector and if there are two identical peaks, the peak with smallest value in the lowest 7 bits it taken.

2.6.8 Decryption failure rate analysis

In this section we analyze the DFR of the concatenated codes. As for the previous decoding algorithm, we use the binomial law approximation p^* of the error vector of Section 2.4.

For concatenated codes the situation is slightly different. Indeed for tensor codes, it is possible to obtain exact decoding probability formulas for the decoding of the repetition codes and the BCH codes. For concatenated codes, it is only possible for the Reed-Solomon codes as for Reed-Muller codes we consider a maximum-likelihood decoding for which there is no exact formula. We provide in the following proposition a lower bound on the decoding probability in that case.

Proposition 2.6.1. *Decryption Failure Rate of the internal code*

Let p be the transition probability of the binary symmetric channel. Then the DFR of a Reed-Muller code of dimension 8 and minimal distance d_i can be upper bounded by:

$$p_i = 255 \sum_{j=d_i/2}^{d_i} \binom{d_i}{j} p^j (1-p)^{d_i - j}$$

Proof. For any linear code C of length n, when transmitting a codeword $\mathbf{c}$, the probability that the channel makes the received word $\mathbf{y}$ at least as close to a word $\mathbf{c}' = \mathbf{c} + \mathbf{x}$ as $\mathbf{c}$ is:

$$\sum_{j \geqslant |\mathbf{x}|/2} \binom{n}{j} p^j (1-p)^{n-j}$$

The probability of a decryption failure can thus be upper bounded by:

$$\sum_{\mathbf{x}\in C, \mathbf{x}\neq 0} \sum_{j\geqslant |\mathbf{x}|/2} \binom{n}{j} p^j (1-p)^{n-j}$$

The number of codewords of a binary code of dimension 8 is 256, hence the result.

□

Remark 2.5. *The previous formula permits to obtain a lower bound on the decoding probability; when the error rate gets smaller the bound becomes closer to the real value of the decoding probability. For cryptographic parameters the approximation is less precise, which means that the DFR obtained will be conservative compared to what happens in practice. We performed simulations to compare the real decryption failure rate with the theoretical one from proposition 2.6.1 for* $[256, 8, 128]$*,* $[512, 8, 256]$ *and* $[768, 8, 384]$ *duplicated Reed-Muller codes using* $p^\star$ *values from actual parameters. Simulation results are presented table 6.*

Security level	$p^\star$	Reed-Muller code	DFR from 2.6.1	Observed DFR
128	0.3196	$[256, 8, 128]$	-7.84	-8.72
192	0.3535	$[512, 8, 256]$	-11.81	-12.22
256	0.3728	$[768, 8, 384]$	-13.90	-14.18

Table 6: Comparison between the observed Decryption Failure Rate and the formula from proposition 2.6.1. Results are presented as $log_2(DFR)$.

From the previous lower bound p_i on the probability decoding of the Reed-Muller codes we deduce the decryption failure rate for these codes:

Theorem 2.6. *Decryption Failure Rate of the concatenated code*

Using a Reed-Solomon code $[n_e, k_e, d_e]_{\mathbb{F}_{256}}$ *as the external code, the DFR of the concatenated code can be upper bounded by:*

$$\sum_{l=\delta_e+1}^{n_e} \binom{n_e}{l} p_i^l (1-p_i)^{n_e-l}$$

Where $d_e = 2\delta_e + 1$ *and* p_i *is defined as in proposition 2.6.1.*

2.6.9 Simulation results

In Fig. 7, we tested the Decryption Failure rate of the concatenated codes against both symmetric binary channels and HQC vectors. For Reed-Muller codes, rather than considering the lower bound approximation we effectively decoded the Reed-Muller, which means than in practice the lower bound that we use for our theoretical DFR, is greater than what is obtained in the simulations. In practice simulations of Fig. 7 support our theoretical model.

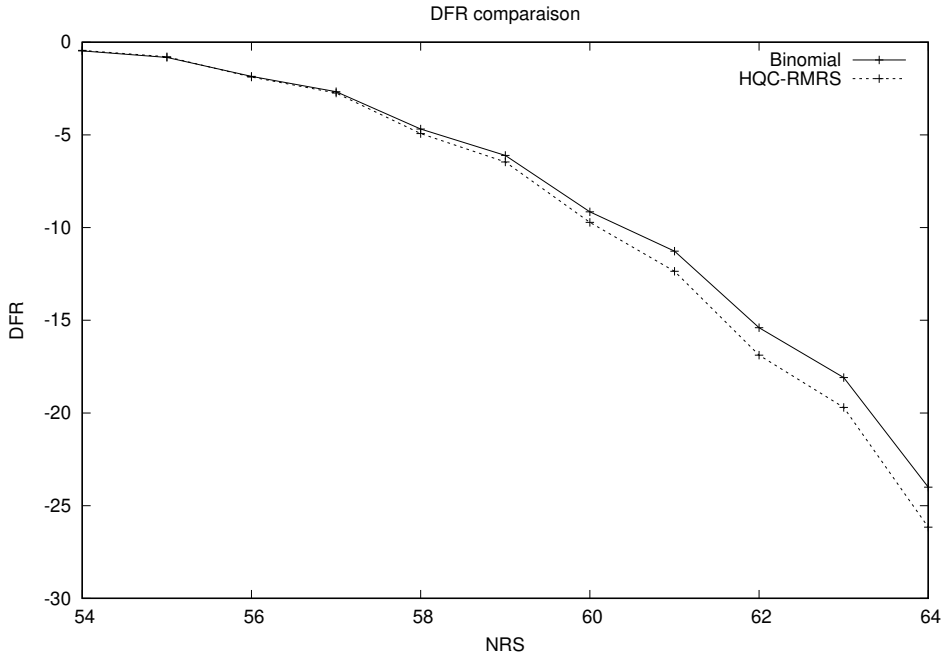

Figure 7: Comparison of the Decryption Failure Rate of concatenated codes against approximation by a binary symmetric channel and against HQC error vectors. Parameters simulated are derived from those of HQC-RMRS for 128 security bits: $w = 67, w_r = w_e = 77$, a $[256, 8, 128]$ duplicated Reed-Muller code for internal code and a $[NRS, 32]$ Reed-Solomon code for external code.

2.7 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 seed-expander 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 $2^{32} - 1$.

2.7.1 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.

2.7.2 Randomness and vector generation

Random bytes are generated using the NIST provided `randombytes` or `seedexpander` functions. The `randombytes` function is used to generate **seed1**, **seed2** as well as **m**. The `seedexpander` function is used to generate θ (using **m** as seed) as well as **x**, **y** (using **seed1** as seed), **h** (using **seed2** as seed) and $\mathbf{r}_1$, $\mathbf{r}_2$, **e** (using θ as seed). Each multiplication done in the whole protocol is computed using some tables with randomized access. For key generation, the randomized access is done using the `seedexpander` with **seed1** as seed. For encryption process, randomized access is done using the `seedexpander` function with θ as seed. For decryption process, the `randombytes` function is used to generate **perm_seed** and the randomized access is done using the `seedexpander` with **perm_seed** 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.

2.8 Parameters

In this section, we specify which codes are used for HQC and give concrete sets of parameters. As mentioned in the previous sections, we consider two type of codes: tensor product codes (Def. 2.5.1) $\mathcal{C} = \mathrm{BCH}(n_1, k, \delta) \otimes \mathbb{1}_{n_2}$ and concatenated Reed-Muller and Reed-Solomon codes. These two types of codes will provide two sets of parameters: HQC parameters which corresponds to tensor codes of the initial Round 2 submission and the HQC-RMRS set of parameters corresponding to the concatenated codes decoding variation.

We propose several sets of parameters, targeting different levels of security with DFR related to these security level. 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 [12, 9, 16, 4] and for quantum attacks, the work of [7]. We consider $w = \mathcal{O}(\sqrt{n})$ and follow the complexity described in [13] (see Sec. 6 for more details).

2.8.1 Tensor product codes

When we use a tensor product code (Def. 2.5.1) $\mathcal{C} = \mathrm{BCH}(n_1, k, \delta) \otimes \mathbb{1}_{n_2}$. 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. 2.3, 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 $\mathbf{c} = (\mathbf{u} = \mathbf{r}_1 + \mathbf{h} \cdot \mathbf{r}_2, \mathbf{v} = \mathbf{mG} + \mathbf{s} \cdot \mathbf{r}_2 + \mathbf{e})$.

In Tab. 7, 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 $\mathcal{C}$. δ 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.

Instance	n_1	n_2	n	k	δ	w	$w_{\mathbf{r}} = w_{\mathbf{e}}$	security	p_{fail}
hqc-128	766	31	23,869	256	57	67	77	128	$< 2^{-128}$
hqc-192	766	59	45,197	256	57	101	117	192	$< 2^{-192}$
hqc-256	796	87	69,259	256	60	133	153	256	$< 2^{-256}$

Table 7: 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. 2.5.1). The considered BCH codes are initially of length 1023, then shortened to support 256 bits dimension (see Tab. 3 and Sec. 2.5.2). The resulting public key, secret key and ciphertext sizes, are given in Tab. 8. 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	3,024	40	6,017	64
hqc-192	5,690	40	11,364	64
hqc-256	8,698	40	17,379	64

Table 8: Sizes in bytes for HQC (see section 2.7).

2.8.2 Concatenated codes

When we use a Concatenated code (Def. 2.6.1). A message $\mathbf{m} \in \mathbb{F}_2^k$ is encoded into $\mathbf{m}_1 \in \mathbb{F}_{2^8}^{n_1}$ with the Reed-Solomon 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 the duplicated Reed-Muller code. In the latter step, the encoding is done in two phases. First, we use the $RM(1,7)$ to encode $\mathbf{m}_{1,i}$ and we obtain $\bar{\mathbf{m}}_{1,i} \in \mathbb{F}_2^{128}$. Then, $\bar{\mathbf{m}}_{1,i}$ is duplicated depending on the multiplicity of the Reed-Muller code (see Tab. 5).

To match the description of our cryptosystem in Sec. 2.3, 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 $\mathbf{c} = (\mathbf{u} = \mathbf{r}_1 + \mathbf{h} \cdot \mathbf{r}_2, \mathbf{v} = \mathbf{mG} + \mathbf{s} \cdot \mathbf{r}_2 + \mathbf{e})$.

In Tab. 7, n_1 denotes the length of the Reed-Solomon code, n_2 the length of the Reed-Muller code so that the length of the concatenated 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). w

Instance	n_1	n_2	n	w	$w_{\mathbf{r}} = w_{\mathbf{e}}$	security	p_{fail}
hqc-RMRS-128	80	256	20,533	67	77	128	$< 2^{-128}$
hqc-RMRS-192	76	512	38,923	101	117	192	$< 2^{-192}$
hqc-RMRS-256	78	768	59,957	133	153	256	$< 2^{-256}$

Table 9: Parameter sets for HQC-RMRS. The concatenated code used is consists of a $[n_2, 8, n_2/2]$ Reed-Muller code as the internal code, and a $[n_1, 32, n_1 - k + 1]$ Reed-Solomon code as the external code. The resulting public key, secret key and ciphertext sizes, are given in Tab. 10. 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-RMRS-128	2,607	40	5,191	64
hqc-RMRS-192	4,906	40	9,794	64
hqc-RMRS-256	7,535	40	15,047	64

Table 10: Sizes in bytes for HQC-RMRS (see section 2.7).

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.

2.8.3 Further potential improvements on the size of public keys

The idea behind HQC is to provide the higher standards for the security reduction. First we recall that the secret key is a pair (x, y) where $\omega(x) = \omega(y) = w$ and that the encryption process involves a tuple (r_1, e, r_2) such that $\omega(r_1) = \omega(r_2) = \omega(e) = w_r$. We will see in Section 5 that the security reduction reduces to decoding a quasi-cyclic $[3n, n]$ code, for a small weight vector (r_1, e, r_2) of length $3n$ and weight $3w_r$, together with decoding a $[2n, n]$ code and a small weight vector of weight $2w$, the regular 3- and 2-QCSD standard problems. The first problem is linked to the decryption of the message from the ciphertext and the second one to the secret key recovery from the public key.

In practice decoding a $[3n, n]$ codes for weight $3w$ is easier than decoding a $[2n, n]$ code for weight $2w$. This explains why for parameters we choose w_r greater than w. It could be possible to avoid this constraint on w_r by choosing $w_r = w$ and considering a greater error weight w_e of the error e.

For the actual parameters of HQC, one can see that we have chosen w_r roughly greater than $1.15w$, so that decoding a quasi-cyclic $[3n, n]$ code for a vector (r_1, e, r_2) of weight $3w_r$ be harder than decoding a $[2n, n]$ code for a vector of weight $2w$. Hence the weight of (r_1, e, r_2) is roughly equal to $\frac{17}{5}w$. Now, if we set $\omega(r_1) = \omega(r_2) = w$ and $\omega(e) = w_e$, to keep our assumption on the weight of (r_1, e, r_2), w_e must be chosen roughly equal to $\frac{7}{5}w$. Hence

we can choose $w_e \geqslant 2w$, and we suggest to take $w_e = 3w$ to provide a safety margin.

In this configuration the error weight of (r_1, e, r_2) becomes non-homogeneous, since it would have weight (say) $(w_r, 3w_r, w_r)$, and hence the problem becomes a slight variation on the 3-QCSD problem.

In fact increasing the weight of e makes the problem in practice as hard as decoding a $[2n, n]$ code for weight $2w_r$. Indeed remember that for this type of very small weight of order $\sqrt{n}$, there is only one type of attack (the original ISD algorithm by Prange), and it is easy to adapt it to show that decoding a vector of weight $(w_r, 3w_r, w_r)$ for length $3n$ becomes more costly than decoding a $[2n, n]$ code for weight $2w_r$.

In practice it means that we could consider $w_r = w$ and hence it could lead to lower the size of the public keys by approximately 10% for all parameters, for both HQC and HQC-RMRS, in that case the public key size for the 128 security bits parameter for HQC-RMRS could be as low as 2,250 Bytes but at the price of having a reduction to a slight variation on the 3-QCSD problem.

3 Performance Analysis

This section provide performance measures of our HQC.KEM implementations.

Benchmark platform. The benchmarks have been performed on a machine that 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 9.2.0) and use the `openssl` (version 1.1.1d) library as a provider for SHA2. For each parameter set, the results have been obtained by computing the mean from 1000 random instances. In order to minimize biases from background tasks running on the benchmark platform, each instances have been repeated 100 times and averaged.

Constant time. The provided implementations have been implemented in a constant time way and as such their running time should not leak any information with respect to sensible data. For instance, such sensible data include secret keys as well as the weight of the error to be decoded by the error-correcting codes used in the schemes. In particular, our implementations avoid any conditional branching.

3.1 HQC

3.1.1 Reference Implementation

The performances of our reference implementation on the aforementioned benchmark platform are described Tab. 11. The following optimization flags have been used during compilation: `-O3 -funroll-all-loops -flto -pedantic -Wall -Wextra`.

In the sequel, we provide some information about one of the most costly operation in HQC namely the multiplication in $\mathbb{F}_2[X]/(X^n - 1)$.

Multiplication over $\mathbb{F}_2[X]/(X^n-1)$ This operation is a sparse-dense polynomial multiplication over $\mathbb{F}_2[X]$. In this case, the schoolbook algorithm can be adapted and remains the most efficient, since the sparsity of one of the polynomial gives a lower complexity. One wants to multiply $A[X]$ and $B[X] \in \mathbb{F}_2[X]$ to get $C = A \cdot B$. The polynomial $B[X]$ being sparse, we represent it by a position vector vB of ω coordinates, with ω the Hamming weight of the sparse polynomial.

In this approach, one considers each monomial, *i.e.* each coordinate vB_i and the dense operand is first shifted of the corresponding degree. In order to speed-up the computation of the dense operand shifts, we first compute a table which contains all the shifts of the dense operand from 0 to $ts = \textbf{TABLE\_SIZE} - 1$. We chose the value $ts = 16$, in order to deal with word shifts in the sequel. The shift corresponding to the vB_i value is then $A[X] \cdot X^{vB_i \bmod ts}$. Then these shifts are added (XOR) starting from the corresponding place of the result (in order to add the complete shift $A[X] \cdot X^{vB_i} = A[X] \cdot X^{vB_i \bmod ts} X^{ts \cdot \lfloor vB_i/ts \rfloor}$) and finally, to get the final result.

In order to avoid information leakage, especially in the decapsulation mechanisms which might expose the secret key ($\mathbf{y}$ in the computation of $\mathsf{Decode}(\mathbf{v} - \mathbf{u} \cdot \mathbf{y})$), we implemented in the multiplication a randomized access to the positions (i.e. the values vB_i are accessed in a randomized order). Furthermore, the shifts modulo ts in the table are also stored according to a random permutation. Both random permutations are computed using the NIST `seedexpander` function, using a seed passed as a parameter in our vector multiplication function. In the decapsulation mechanism, this seed is itself provided by the NIST `randombytes` function.

Instance	KeyGen	Encaps	Decaps
hqc-128	161	415	629
hqc-192	318	687	973
hqc-256	511	1043	1457

Table 11: Performance in kilocycles of the reference implementation for different instances of HQC.

3.1.2 Optimized Implementation

An optimized implementation leveraging AVX2 instructions have been provided. Its performances on the aforementioned benchmark platform are described in Tab. 12. The following optimization flags have been used during compilation: `-O3 -mavx -mavx2 -funroll-all-loops -flto -pedantic -Wall -Wextra`. There are two main differences between the reference and the optimized implementation. Firstly, the former uses a LFSR to realize the BCH encoding and the latter uses the classical encoding with the generator matrix. Secondly, in the optimized implementation we added some optimizations to the BCH decoding algorithm. In the sequel we give some details on the optimizations done in this version.

Multiplication over $\mathbb{F}_2[X]/(X^n-1)$ The approach is the same as the one presented in the reference implementation, improved by taking into account the `AVX2` instruction set. In this approach, one considers each monomial, *i.e.* each coordinate vB_i and the dense operand is shifted of the corresponding degree, and all the shifts are added (XOR) to get the final result. As we use 256-bit registers, the operand size is $t = \lceil N/256 \rceil$ 256-bit words. In order to shift the dense operand, we use the `_mm256_slli_epi64(X,sh)` shift instruction, which shifts 64 bit blocks within the register of `sh` bits to the left. Then, we add to the result the corresponding most significant bits (carry) of the previous words within each block, obtained by the corresponding right shift (`_mm256_srli_epi64(X,64-sh)`). Depending on the compilation flag, this leads to a speed-up by a factor about 2 in comparison with the reference implementation.

BCH-code encoding function In this function, due to the message length (of 256 bits), we use a 256-bit register. The encoding matrix is a table containing `PARAM_N1` columns of one 256-bit word. We store only the `PARAM_N1-PARAM_K` columns, since we use the systematic form. The columns are XORed with the message to provide the encoded word (using `_mm_popcnt_u64` intrinsic). This XOR is performed using one operation on 256-bit words. The message itself is appended at the end to get the complete code word. Compared to the reference implementation, the whole encoding process (BCH+repetition) leads to a speedup factor of 8 to 15 depending on the HQC version (128, 192 or 256).

BCH-code decoding function For this optimized version we do not make use of the FFT transform. Syndromes of the polynomial $v(x)$ are computed by evaluating $v(\alpha^i)$ for i = 1, 2,... , `2*PARAM_DELTA`. Since these values are stored in 16-bit integers, 16 syndromes can be represented in one 256-bit word. Next, to speedup the computation the value α^{ij} have been precomputed and stored in a table whose size is `4*PARAM_DELTA*PARAM_N1`.

Instance	KeyGen	Encaps	Decaps
hqc-128	142	231	372
hqc-192	276	459	655
hqc-256	442	750	1032

Table 12: Performance in kilocycles of the optimized implementation using `AVX2` instructions for different instances of HQC.

3.2 HQC-RMRS

The reference implementation uses the same multiplication procedure described in section 3.1.1. Both optimized and additional implementations use the same multiplication algorithm described in section 3.1.2.

3.2.1 Reference Implementation

The performances of our reference implementation on the aforementioned benchmark platform are described Tab. 13. The following optimization flags have been used during compilation: `-O3 -funroll-all-loops -flto -pedantic -Wall -Wextra`

Instance	KeyGen	Encaps	Decaps
hqc-RMRS-128	143	245	506
hqc-RMRS-192	277	477	837
hqc-RMRS-256	455	800	1314

Table 13: Performance in kilocycles of the reference implementation for different instances of HQC-RMRS.

3.2.2 Optimized Implementation

An optimized implementation leveraging AVX2 instructions have been provided. Its performances on the aforementioned benchmark platform are described in Tab. 14. The following optimization flags have been used during compilation: `-O3 -mavx -mavx2 -funroll-all-loops -flto -pedantic -Wall -Wextra`. There are two main differences between the reference and the optimized implementation. Firstly, the multiplication of two polynomial is vectorized and uses the same algorithm described in section 3.1.2. Secondly, in the optimized implementation we added a vectorized version of the Reed-Muller decoding algorithm.

Instance	KeyGen	Encaps	Decaps
hqc-RMRS-128	126	211	372
hqc-RMRS-192	243	406	642
hqc-RMRS-256	395	670	1025

Table 14: Performance in kilocycles of the optimized implementation using `AVX2` instructions for different instances of HQC-RMRS.

3.2.3 Additional Implementation

We provide an additional optimized implementation of the scheme where the decoding of Reed-Muller codes is done using `AVX-512` instructions. Its performances on the aforementioned benchmark platform are described in Tab. 15. The following optimization flags have been used during compilation: `-O3 -mavx -mavx2 -mavx512bw -mavx512vl -funroll-all-loops -flto -pedantic -Wall -Wextra`.

Instance	KeyGen	Encaps	Decaps
hqc-RMRS-128	126	213	325
hqc-RMRS-192	242	406	561
hqc-RMRS-256	392	666	900

Table 15: Performance in kilocycles of the optimized implementation using `AVX-512` instructions for different instances of HQC-RMRS.

4 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 `KATs/Reference_Implementation/`, `KATs/Optimized_Implementation/` and `KATs/Additional_Implementation/`.

In addition, examples with intermediate values have also been provided in these folders.

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.

5 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 [25], and the tightness of the reduction provided by this transformation has been discussed at the end of Sec. 2.2.

Theorem 5.1. *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 5.1. 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. `RETURN` b'

The adversary has access to pk and $\mathbf{c}^*$. As he has access to pk and the $\mathsf{Encrypt}$ function, anything that is computed from pk and $\mathbf{c}^*$ can also be computed from just 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. 7), 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. 8. Therefore we can suppose the only input of the adversary is pk.

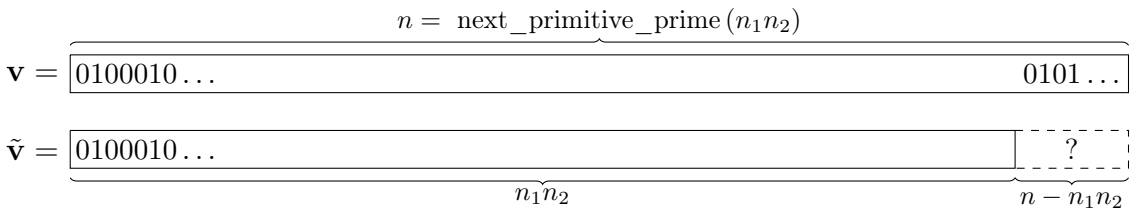

Figure 8: Truncation of vector $\mathbf{v}$ from $\mathbb{F}^n_2$ to $\tilde{\mathbf{v}} \in \mathbb{F}^{n_1 n_2}_2$.

Now suppose the adversary has an algorithm $\mathcal{D}_\lambda$, taking pk as input, that distinguishes with advantage ϵ Game $\boldsymbol{G}_1$ and Game $\boldsymbol{G}_2$, for some security parameter λ. 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 ϵ 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 pk as $(\mathbf{h}, \mathbf{y})$ and $(\mathbf{H}, \mathbf{y}^\top)$ from a 2-$\mathsf{QCSD}(n, w, b)$ distribution with parity, 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 $\mathbb{F}_{2,b}^{n \times 2n} \times \mathbb{F}_{2,b'}^n$, pk follows exactly the same distribution as in Game $\boldsymbol{G}_2$.

Thus we have:

$$\begin{aligned} \Pr\left[\mathcal{D}'_{\mathcal{E},\mathcal{D}_\lambda}((\mathbf{H}, \mathbf{y}^\top)) = \texttt{QCSD} | (\mathbf{H}, \mathbf{y}^\top) \leftarrow 2\text{-}\mathsf{QCSD}(n, w, b)\right] = \\ \Pr\left[\mathcal{D}_\lambda(\mathsf{pk}) = 1 | \mathsf{pk} \text{ from } \mathbf{Game}^0_{\mathcal{E},\mathcal{A}}(\lambda)\right], \text{ and} \end{aligned} \tag{22}$$

$$\begin{aligned} \Pr\left[\mathcal{D}'_{\mathcal{E},\mathcal{D}_\lambda}((\mathbf{H}, \mathbf{y}^\top)) = \texttt{UNIFORM} | (\mathbf{H}, \mathbf{y}^\top) \leftarrow 2\text{-}\mathsf{QCSD}(n, w, b)\right] = \\ \Pr\left[\mathcal{D}_\lambda(\mathsf{pk}) = 2 | \mathsf{pk} \text{ from } \mathbf{Game}^0_{\mathcal{E},\mathcal{A}}(\lambda)\right] \end{aligned} \tag{23}$$

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 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. $(\mathbf{m}_0, \mathbf{m}_1) \leftarrow \mathcal{A}(\texttt{FIND} : \mathsf{pk})$
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. $b' \leftarrow \mathcal{A}(\texttt{GUESS} : \mathbf{c}^*)$
6. $\texttt{RETURN}\ b'$

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

Note that an adversary is not able to obtain $\mathbf{c}^*$ from 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 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-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-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. $(\mathbf{m}_0, \mathbf{m}_1) \leftarrow \mathcal{A}(\texttt{FIND} : \mathsf{pk})$
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. $b' \leftarrow \mathcal{D}_\lambda(\texttt{GUESS} : \mathbf{c}^*)$
4. If $b' == 2$ output `QCSD`
5. If $b' == 3$ output `UNIFORM`

The distribution of 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-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. $(\mathbf{m}_0,\mathbf{m}_1) \leftarrow \mathcal{A}(\mathtt{FIND} : \mathsf{pk})$
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. $b' \leftarrow \mathcal{A}(\mathtt{GUESS} : \mathbf{c}^*)$
6. $\mathtt{RETURN}\ b'$

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:

$$\mathsf{Adv}^{\mathsf{ind}}_{\mathcal{E},\mathcal{A}}(\lambda) \leq 2\left(\mathsf{Adv}^{\text{2-DQCSD}}(\lambda) + \mathsf{Adv}^{\text{3-DQCSD}}(\lambda)\right). \tag{24}$$

□

6 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 [40] and improved later by Stern [43], then Dumer [15]. Recent works [35, 4, 36] 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 [13].

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 [42] 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 [24, 33, 42], 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. 2.8 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) [7]. Best known attacks include the works from [12, 9, 16, 35, 4, 36] and for quantum attacks, the work of [7]. 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 [13]. 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 [42], 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 [13].

Timing attack. It has been shown in [44, 39] that HQC is vulnerable to timing attack if the decoding of the BCH codes is implemented in a non constant time fashion. The attack exploits a correlation between the weight of the error to be decoded and the running time of the decoding algorithm of BCH codes. In the provided implementations the decoding of BCH codes is constant-time, thus preventing the aforementioned attack.

7 Advantages and Limitations

7.1 Advantages

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

- 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,
- small public key size
- close estimations of its decryption failure rate.
- efficient implementations based on classical decoding algorithms.

The fourth item allows to achieve a tight reduction for the IND-CCA2 security of the KEM-DEM version through the recent transformation of [25].

7.2 Limitations

We have proposed an instantiation of the HQC scheme using two possible decoding, first the original decoding algorithm with BCH codes tensored with repetition codes and second a new variation based on concatenated Reed-Muller and Reed-Solomon codes. Notice that considering different decoding algorithm does not change the general security reduction of the scheme. 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.

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. *7, 8*

[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. *10*, *12*

[3] Nicolas Aragon, Philippe Gaborit, and Gilles Zémor. A more efficient decoding algorithm for hqc based on concatenated codes. to appear in ArXiv. *17*, *28*

[4] 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. *36*, *48*

[5] Elwyn Berlekamp. *Algebraic coding theory*. World Scientific, 1968. *31*

[6] 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. *10*

[7] Daniel J Bernstein. Grover vs. mceliece. In *Post-Quantum Cryptography*, pages 73–80. Springer, 2010. https://cr.yp.to/codes/grovercode-20091123.pdf. *36*, *48*

[8] Daniel J Bernstein, Tung Chou, and Peter Schwabe. Mcbits: fast constant-time code-based cryptography. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 250–272. Springer, 2013. *26*

[9] 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. *36*, *48*

[10] Daniel J. Bernstein and Edoardo Persichetti. Towards KEM unification. Cryptology ePrint Archive, Report 2018/526, 2018. https://eprint.iacr.org/2018/526. *14*

[11] 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. *21*

[12] 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/. *36*, *48*

[13] 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. *36*, *48*

[14] 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. *14*

[15] 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. *48*

[16] 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. *36*, *48*

[17] 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. *14*

[18] 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. *14*

[19] 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. *9*

[20] 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. *10*

[21] Shuhong Gao and Todd Mateer. Additive fast fourier transforms over finite fields. *IEEE Transactions on Information Theory*, 56(12):6265–6272, 2010. *26*, *31*

[22] 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. *11*

[23] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984. *13*

[24] 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. *48*

[25] 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. *7*, *14*, *15*, *16*, *43*, *49*

[26] 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. *8*

[27] 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. *14*

[28] Haodong Jiang, Zhenfeng Zhang, and Zhi Ma. Tighter security proofs for generic key encapsulation mechanism in the quantum random oracle model. In Jintai Ding and Rainer Steinwandt, editors, *Post-Quantum Cryptography - 10th International Conference, PQCrypto 2019, Chongqing, China, May 8-10, 2019 Revised Selected Papers*, volume 11505 of *Lecture Notes in Computer Science*, pages 227–248. Springer, 2019. *14*

[29] Laurie L Joiner and John J Komo. Decoding binary bch codes. In *Southeastcon'95. Visualize the Future., Proceedings., IEEE*, pages 67–73. IEEE, 1995. *26*

[30] Shu Lin and Daniel J Costello. *Error control coding*, volume 2. Prentice Hall Englewood Cliffs, 2004. *22*, *23*, *24*, *25*, *28*, *29*, *30*, *31*

[31] 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. *11*

[32] 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. *11*

[33] 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. *48*

[34] Florence Jessie MacWilliams and Neil James Alexander Sloane. *The theory of error-correcting codes*, volume 16. Elsevier, 1977. *32*

[35] 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. *48*

[36] 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. *48*

[37] 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. *9*, *10*

[38] 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. *14*

[39] Thales Bandiera Paiva and Routo Terada. A timing attack on the hqc encryption scheme. In *International Conference on Selected Areas in Cryptography*, pages 551–573. Springer, 2019. *48*

[40] 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/. *48*

[41] 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. *14*

[42] 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. *11*, *48*

[43] 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. *48*

[44] Guillaume Wafo-Tapa, Slim Bettaieb, Loïc Bidoux, and Philippe Gaborit. A practicable timing attack against HQC and its countermeasure. *IACR Cryptology ePrint Archive*, 2019:909, 2019. *48*