Hamming Quasi-Cyclic (HQC)

22/08/2025

Submitters (by joining date then alphabetical order):

- Philippe Gaborit (University of Limoges, FR)
- Carlos Aguilar-Melchor (SandboxAQ, USA)
- Nicolas Aragon (Naquidis Center, FR)
- Slim Bettaieb (Technology Innovation Institute, UAE)
- Loïc Bidoux (Technology Innovation Institute, UAE)
- Olivier Blazy (Ecole Polytechnique, FR)
- Jean-Christophe Deneuville (ENAC, University of Toulouse, FR)
- Edoardo Persichetti (Florida Atlantic University, USA)
- Gilles Zémor (IMB, University of Bordeaux, FR)
- Jurjen Bos (Worldline, NL)
- Arnaud Dion (ISAE-SUPAERO, University of Toulouse, FR)
- Jérôme Lacan (ISAE-SUPAERO, University of Toulouse, FR)
- Jean-Marc Robert (University of Toulon, FR)
- Pascal Véron (University of Toulon, FR)
- Paulo L. Barreto (University of Washington Tacoma, USA)
- Santosh Ghosh (Intel, USA)
- Shay Gueron (University of Haifa, Israel and Meta, USA)
- Tim Güneysu (Ruhr-Universität Bochum, DE and DFKI, DE)
- Rafael Misoczki (Meta, USA)
- Jan Richter-Brokmann (Ruhr-Universität Bochum, DE)
- Nicolas Sendrier (INRIA, FR)
- Jean-Pierre Tillich (INRIA, FR)
- Valentin Vasseur (Thales, FR)

Contact: team@pqc-hqc.org

Changelog

Hereafter, we list the main modifications made to HQC design. Modifications related to implementations are provided with the source code (available at https://pqc-hqc.org).

2025/08/22

- We have refactored this document to improve its readability notably adding detailed figures describing HQC-PKE and HQC-KEM and fixing several typographical errors.
- We have updated the Fujisaki-Okamoto used to construct HQC-KEM from HQC-PKE by fixing the rejection of the scheme [34], using the salted $\mathsf{SFO}^{\not\perp}_m$ transform instead of the $\mathsf{FO}^{\not\perp}$ one [18] and adding $(\mathsf{ek}_{\mathsf{KEM}}, \mathsf{salt})$ within K and θ derivation.
- We have updated HQC keypair format by adding $\mathsf{seed}_{\mathsf{KEM}}$ in the decapsulation key $\mathsf{dk}_{\mathsf{KEM}}$ so that the keypair can be easily checked if it is received from a third party and removing $\mathbf{x}$ from $\mathsf{dk}_{\mathsf{KEM}}$ as it is not used in HQC-KEM.Decaps. In addition, an alternative compressed format $\mathsf{dk}_{\mathsf{KEM}} = (\mathsf{seed}_{\mathsf{KEM}})$ have been included.
- We have updated fixed weight vectors sampling in HQC-PKE.Keygen by using a sampler that outputs uniformly distributed vectors but rely on rejection sampling ($\mathsf{SampleFixedWeightVect}_{\$}$) instead of a sampler that don't rely on rejection sampling but output a a slightly biased distribution (SampleFixedWeightVect).
- We have made several minor modifications to align HQC specifications with design choices made in FIPS-203 such as computing $(\mathsf{seed}_{\mathsf{PKE.dk}}, \mathsf{seed}_{\mathsf{PKE.ek}})$ using SHA3-512 instead of SHAKE256, absorbing all the bytes of $\mathsf{ek}_{\mathsf{KEM}}$ in the computation of (K, θ) and reducing the sizes of K and θ from 40 to 32 bytes.
- Paulo L. Barreto, Santosh Ghosh, Shay Gueron, Tim Güneysu, Rafael Misoczki, Jan Richter-Brokmann, Nicolas Sendrier, Jean-Pierre Tillich, Valentin Vasseur have joined the HQC team.

2024/10/30

- We have modified the order of variable sampling in both key generation and encryption as this results in performance gains in hardware implementation as suggested by [2].
- We have updated the countermeasure against multi-target attacks by including only the first 32 bytes of the public key instead of the entire public key.

2023/04/30

- We have updated the Fujisaki-Okamoto used to construct HQC-KEM from HQC-PKE by considering the $\mathsf{FO}^{\not\perp}$ transform with implicit rejection for the scheme.
- We have provided an analysis showing that sampling small weights vectors non-uniformly, yet close to uniform, has a negligible effect on HQC security following [36].

2022/10/01

- We have updated the computation of the randomness θ used in HQC-PKE.Encrypt to include a salt and $\mathsf{ek}_{\mathsf{KEM}}$ as a counter-measure to multi-ciphertext attacks [32].
- We have updated constant-weight words sampling using the technique from [36] as a counter-measure to the timing attack from [20].

2020/10/01

- We have removed the HQC variant using the BCH-Repetition decoder as the newly introduced RMRS decoder is strictly better.
- We have updated the sizes of the decoded messages for the concatenated RMRS code to the targeted security levels (*i.e.* 128 and 192 rather than 256) for level 1 and 3 which improves the decoding capacity of the RMRS code and improves our parameters.
- We have improved the theoretical lower bound for the Reed-Muller decoder which permits to lower our theoretical bound for the DFR and improve our parameters.
- Jérôme Lacan and Arnaud Dion have joined the HQC team.

2020/05/04

- We have provided a more precise analysis of the modelization of the error distribution which permits to lower the DFR of HQC and decrease the sizes of its public keys.
- We have introduced a new decoding algorithm based on Reed-Muller and Reed-Solomon codes which permits to decrease the sizes of public keys. A new set of parameters denoted HQC-RMRS has been provided.
- We have removed parameter sets providing a DFR lower than the security parameter thus we now only consider parameters with a DFR corresponding to the security level.
- Jean-Marc Robert and Pascal Véron have joined the HQC team.

2019/04/10

- We have introduced the DQCSD problems with parity as a counter-measure to distinguishers from parity. In addition, we are using tensor product code of length $n_1 n_2$ while working with vectors of size n with n the primitive prime immediately greater than $n_1 n_2$ to avoid algebraic attacks. The proof has been updated to take into account the DQCSD problem with parity along with the $\ell = n - n_1 n_2$ truncated bits.
- We have removed parameter sets providing a DFR higher than 2^{-128}.
- Jurjen Bos has joined the HQC team.

Contents

1 Introduction

HQC is a code-based IND-CCA2 secure Key Exchange Mechanism (KEM) scheme whose security is based on the Quasi-Cyclic Syndrome Decoding (QCSD) problem. In contrast with many code-based cryptosystems, the family of codes being used is not required to be indistinguishable among random codes. HQC is built from an IND-CPA secure Public Key Encryption (PKE) scheme that was first described in [1] along with the HHK transform [22]. In order to avoid any ambiguity, the PKE and KEM constructions will be denoted as HQC-PKE and HQC-KEM respectively.

Organization. We provide the required background in Section 2 and present the specifications of HQC in Section 3. Parameter sets are given in Section 4 while Sections 5 and 6 provide a performance analysis and a security analysis of the scheme respectively. Finally, advantages and limitations of HQC are discussed in Section 7.

2 Preliminaries

2.1 Notations

Let $\mathbb{Z}$ denote the ring of integers, $\mathbb{F}_2$ denote the binary finite field and $\mathcal{B}$ denote the set $\{0, \ldots, 255\}$ of unsigned 8-bits integers. Given a set S, let $s \leftarrow\!\!\$\, S$ denote that s is chosen uniformly at random from S. Let $\mathcal{V}$ denote 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 and polynomials (respectively, matrices) are represented by lower-case (respectively 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 as in $\mathcal{R}$ *i.e.* $\mathbf{w} = \mathbf{uv} \in \mathcal{V}$ with:

$$w_k = \sum_{i+j \ \equiv \ k \pmod{n}} u_i \cdot v_j \ \forall k \in \{0, 1, \ldots, n-1\}.$$

Given $\mathbf{v} \in \mathbb{F}_2^n$, let $\mathbf{rot}(\mathbf{v})$ denotes the circulant matrix induced by $\mathbf{v}$ namely:

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

One can 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:

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

Let denote $\omega(\cdot)$ the Hamming weight of a vector *i.e.* by the number of its nonzero coordinates. For a given positive integer ω, let $\mathcal{R}_\omega := \{\mathbf{v} \in \mathcal{R} \text{ such that } \omega(\mathbf{v}) = \omega\}$ denote the set of vectors having Hamming weight ω.

Let $\mathbf{v} = (v_0, \ldots, v_{n-1}) \in \{0,1\}^n$ with index 0 the least-significant bit. For $0 \leq n' \leq n$, we define $\mathsf{Truncate}(\mathbf{v}, n')$ as the function that discards the $n - n'$ most-significant bits of $\mathbf{v}$ and keep the n' least-significant ones.

2.2 Coding theory

In the following, we provide basic definitions and properties about coding theory and refer the reader to [23] for a complete survey.

Definition 2.2.1 (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 2.2.2 (Generator Matrix)**.** *Given an* $[n, k]$ *code* $\mathcal{C}$*, a matrix* $\mathbf{G} \in \mathbb{F}_2^{k\times n}$ *is a* generator matrix *for* $\mathcal{C}$ *if:*

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

Definition 2.2.3 (Parity-Check Matrix)**.** *Given an* $[n, k]$ *code* $\mathcal{C}$*, a matrix* $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}$ *is a* parity-check matrix *for* $\mathcal{C}$ *if* $\mathbf{H}$ *is a generator matrix of the dual code* $\mathcal{C}^\perp$*:*

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

Definition 2.2.4 (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. The* syndrome *of* $\mathbf{v}$ *is* $\mathbf{Hv}^\top$*, and one has* $\mathbf{v} \in \mathcal{C} \Leftrightarrow \mathbf{Hv}^\top = \mathbf{0}$*.*

Definition 2.2.5 (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:*

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

A code of length n and dimension k with minimum distance d is capable of decoding arbitrary patterns of up to $\Delta = \lfloor \frac{d-1}{2} \rfloor$ errors and is denoted as an $[n, k, d]$ code.

HQC relies on Quasi-Cyclic codes in order to shorten its keys as suggested in [15].

Definition 2.2.6 (Quasi-Cyclic Codes [31])**.** *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 2.2.7 (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:*

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

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

The definition of systematic quasi-cyclic codes of index s can 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 $1/3$. Hereafter, 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.

2.3 Security assumptions

In this section we describe difficult problems which are relevant for HQC 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 2.3.1 ($\mathcal{SD}$ Distribution)**.** *Let n, k and ω be positive integers. The Syndrome Decoding Distribution $\mathcal{SD}(n,k,\omega)$ samples $\mathbf{H} \leftarrow\$ \mathbb{F}_2^{(n-k)\times n}$ and $\mathbf{x} \leftarrow\$ \mathbb{F}_2^n$ such that $\omega(\mathbf{x}) = \omega$, computes $\mathbf{y}^\top = \mathbf{H}\mathbf{x}^\top$ and outputs $(\mathbf{H}, \mathbf{y})$.*

Definition 2.3.2 (Computational SD Problem)**.** *Let n, k and ω be positive integers. Given $(\mathbf{H},\mathbf{y}) \in \mathbb{F}_2^{(n-k)\times n} \times \mathbb{F}_2^{n-k}$ from the $\mathcal{SD}(n,k,\omega)$ distribution, the Syndrome Decoding Problem $\mathsf{SD}(n,k,\omega)$ asks to find $\mathbf{x} \in \mathbb{F}_2^n$ such that $\mathbf{y}^\top = \mathbf{H}\mathbf{x}^\top$ and $\omega(\mathbf{x}) = \omega$.*

Definition 2.3.3 (DSD Problem)**.** *Let n, k and ω be positive integers. Given $(\mathbf{H}, \mathbf{y}) \in \mathbb{F}_2^{(n-k)\times n} \times \mathbb{F}_2^{n-k}$, the Decisional Syndrome Decoding Problem $\mathsf{DSD}(n,k,\omega)$ asks to decide with non-negligible advantage whether $(\mathbf{H}, \mathbf{y})$ came from the $\mathcal{SD}(n,k,\omega)$ distribution or the uniform distribution over $\mathbb{F}_2^{(n-k)\times n} \times \mathbb{F}_2^{n-k}$.*

For the Hamming distance, the SD problem has been proven NP-complete [7]. This problem can also be seen as the Learning Parity with Noise (LPN) problem with a fixed number of samples [3]. The DSD problem has been shown to be polynomially equivalent to its search version in [3]. 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.

As our cryptosystem use QC-codes, the following definitions describe the DSD problems in the QC setting. 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 more than 15 years in cryptography [16].

Definition 2.3.4 (s-$\mathcal{QCSD}$ Distribution)**.** *Let n, s and ω be positive integers. The s-Quasi-Cyclic Syndrome Decoding Distribution s-$\mathcal{QCSD}(n,\omega)$ samples a parity-check matrix $\mathbf{H} \leftarrow\$ \mathbb{F}_2^{(sn-n)\times sn}$ of a systematic Quasi Cyclic code $\mathcal{C}$ of index s and rate $1/s$ and a vector $\mathbf{x} = (\mathbf{x}_0, \ldots, \mathbf{x}_{s-1}) \leftarrow\$ \mathbb{F}_2^{sn}$ such that $\omega(\mathbf{x}_i) = \omega\ \forall i \in [0, s-1]$, computes $\mathbf{y}^\top = \mathbf{H}\mathbf{x}^\top$ and outputs $(\mathbf{H}, \mathbf{y})$.*

Definition 2.3.5 (s-DQCSD Problem)**.** *Let n, s and ω be positive integers. Given $(\mathbf{H}, \mathbf{y}) \in \mathbb{F}_2^{(sn-n)\times sn} \times \mathbb{F}_2^{sn-n}$, the Decisional s-Quasi-Cyclic Syndrome Decoding Problem s-$\mathsf{DQCSD}(n,\omega)$ asks to decide with non-negligible advantage whether $(\mathbf{H}, \mathbf{y})$ came from the s-$\mathcal{QCSD}(n,\omega)$ distribution or the uniform distribution over $\mathbb{F}_2^{(sn-n)\times sn} \times \mathbb{F}_2^{sn-n}$.*

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 Definition (2.2.7). We choose this distribution to make the security reduction that follows less technical. It is readily seen that, for fixed s, when choosing QC codes with this more general distribution, one obtains with non-negligible probability, a QC code that admits a parity-check matrix of the form given in Definition (2.2.7). Therefore, requiring QC codes to be systematic does not hurt the generality of the decoding problem for QC codes. A similar remark holds for the slightly special form of weight distribution of the vector $\boldsymbol{x}$.

Although there is no general complexity result for QC codes, decoding these codes is considered hard by the community. There exist general attacks that use the cyclic structure of the code [35] but these attacks have only a small impact (sublinear in code length) on the 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.

In order to avoid trivial distinguishers, an additional condition on the parity of the syndrome is added to the problem. For $b_1 \in \{0,1\}$, we define the finite set $\mathbb{F}_{2,b_1}^n = \{\mathbf{h} \in \mathbb{F}_2^n \text{ s.t. } \mathbf{h}(1) = b_1 \mod 2\}$ *i.e.* binary vectors of length n and parity b_1. Similarly for matrices, we define the finite sets:

$$\mathbb{F}_{2,b_1}^{n\times 2n} = \left\{\mathbf{H} = (\mathbf{I}_n \ \mathbf{rot}\,(\mathbf{h})) \in \mathbb{F}_2^{n\times 2n} \text{ s.t. } \mathbf{h} \in \mathbb{F}_{2,b_1}^n\right\}, \text{ and}$$

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

This is pure technicality and has almost no effect on the parameters of our proposal as it results in a security loss of at most 1 bit. Meanwhile, this permits to discard attacks such as [19, 25, 26][1].

Definition 2.3.6 (2-$\mathcal{QCSD}$-$\mathcal{P}$ Distribution)**.** *Let n, ω, b_1 be positive integers and let $b_2 = \omega + b_1 \times \omega \mod 2$. The 2-Quasi-Cyclic Syndrome Decoding with Parity Distribution 2-$\mathcal{QCSD}$-$\mathcal{P}(n,\omega,b_1,b_2)$ samples $\mathbf{H} \in \mathbb{F}_{2,b_1}^{n\times 2n}$ and $\mathbf{x} = (\mathbf{x}_1,\mathbf{x}_2) \leftarrow\!\!\$\ \mathbb{F}_2^{2n}$ such that $\omega(\mathbf{x}_1) = \omega(\mathbf{x}_2) = \omega$, compute $\mathbf{y}^\top = \mathbf{H}\mathbf{x}^\top$ and outputs $(\mathbf{H},\mathbf{y}) \in \mathbb{F}_{2,b_1}^{n\times 2n} \times \mathbb{F}_{2,b_2}^n$.*

Definition 2.3.7 (2-DQCSD-P Problem)**.** *Let n, ω, b_1 be positive integers and let $b_2 = \omega + b_1 \times \omega \mod 2$. Given $(\mathbf{H},\mathbf{y}) \in \mathbb{F}_{2,b_1}^{n\times 2n} \times \mathbb{F}_{2,b_2}^n$, the Decisional 2-Quasi-Cyclic Syndrome Decoding with Parity Problem 2-DQCSD-P(n,ω,b_1,b_2) asks to decide with non-negligible advantage whether $(\mathbf{H},\mathbf{y})$ came from the 2-$\mathcal{QCSD}$-$\mathcal{P}(n,\omega,b_1,b_2)$ distribution or the uniform distribution over $\mathbb{F}_{2,b_1}^{n\times 2n} \times \mathbb{F}_{2,b_2}^n$.*

[1] The authors chose to use a parity version of the 2-DQCSD problem rather than a variable weight version as suggested in [26] for efficiency considerations.

In order to thwart structural attacks, one needs to work with a code of primitive prime length n, so that $X^n - 1$ has only two irreducible factors mod 2. However for the considered parameters and codes (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 which is 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 3-DQCSD problem that we define hereafter.

Definition 2.3.8 (3-$\mathcal{QCSD}$-$\mathcal{PT}$ Distribution)**.** *Let n, ω, b_1, b_2, ℓ be positive integers and let $b_3 = \omega + b_1 \times \omega \mod 2$. The 3-Quasi-Cyclic Syndrome Decoding with Parity and Truncation Distribution 3-$\mathcal{QCSD}$-$\mathcal{PT}$$(n, \omega, b_1, b_2, b_3, \ell)$ samples $\mathbf{H} \leftarrow\!\!\$\ \mathbb{F}_{2,b_1,b_2}^{2n \times 3n}$ and $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3) \leftarrow\!\!\$\ \mathbb{F}_2^{3n}$ such that $\omega(\mathbf{x}_1) = \omega(\mathbf{x}_2) = \omega(\mathbf{x}_3) = \omega$, computes $\mathbf{y}^\top = \mathbf{H}\mathbf{x}^\top$ where $\mathbf{y} = (\mathbf{y}_1, \mathbf{y}_2)$ and outputs $(\mathbf{H}, (\mathbf{y}_1, \mathsf{Truncate}(\mathbf{y}_2, \ell))) \in \mathbb{F}_{2,b_1,b_2}^{2n \times 3n} \times (\mathbb{F}_{2,b_3}^n \times \mathbb{F}_2^{n-\ell})$.*

Definition 2.3.9 (3-QCSD-PT Problem)**.** *Let n, ω, b_1, b_2, ℓ be positive integers and let $b_3 = \omega + b_1 \times \omega \mod 2$. Given $(\mathbf{H}, (\mathbf{y}_1, \mathbf{y}_2)) \in \mathbb{F}_{2,b_1,b_2}^{2n \times 3n} \times (\mathbb{F}_{2,b_3}^n \times \mathbb{F}_2^{n-\ell})$, the Decisional 3-Quasi-Cyclic Syndrome Decoding with Parity and Truncation Problem 3-DQCSD-PT$(n, \omega, b_1, b_2, b_3, \ell)$ asks to decide with non-negligible advantage whether $(\mathbf{H}, (\mathbf{y}_1, \mathbf{y}_2))$ came from the 3-$\mathcal{QCSD}$-$\mathcal{PT}$ $(n, \omega, b_1, b_2, b_3, \ell)$ distribution or the uniform distribution over $\mathbb{F}_{2,b_1,b_2}^{2n \times 3n} \times (\mathbb{F}_{2,b_3}^n \times \mathbb{F}_2^{n-\ell})$.*

Regarding the security of the 3-DQCSD-PT problem with parity and truncation, whenever the number of truncated positions is very small compared to the block length n, the impact on the security is negligible with respect to the 3-DQCSD problem since the best attack is the ISD attack. Moreover since the truncation breaks the quasi-cyclicity, it also weaknesses the advantage of quasi-cyclicity for the attacker.

2.4 Security definitions

In this section, we introduce the security definitions that are used for the security analysis of HQC-PKE and HQC-KEM provided in Section 6.

Definition 2.4.1. *Let* $\mathsf{PKE} = (\mathsf{Keygen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ *be a public-key encryption scheme. The* PKE.IND-CPA *game is defined as in Figure 1 and the* IND-CPA *advantage of an adversary* $\mathcal{A} = (\mathcal{A}_{\mathsf{CHOOSE}}, \mathcal{A}_{\mathsf{GUESS}})$ *against a* PKE *is defined as:*

$$\mathsf{Adv}_{\mathsf{PKE}}^{\mathsf{IND\text{-}CPA}}(\mathcal{A}) = \left| \Pr[\mathsf{PKE.IND\text{-}CPA}(\mathcal{A}) = 1] - \frac{1}{2} \right|.$$

$\mathsf{Exp}\ \mathsf{PKE.IND\text{-}CPA}(\mathcal{A})$

1. $(\mathsf{ek}_{\mathsf{PKE}}, \mathsf{dk}_{\mathsf{PKE}}) \leftarrow \mathsf{Keygen}()$
2. $(\mathbf{m}_0, \mathbf{m}_1) \leftarrow \mathcal{A}_{\mathsf{CHOOSE}}(\mathsf{ek}_{\mathsf{PKE}})$
3. $b \leftarrow\!\!\$\ \{0,1\}$
4. $\theta \leftarrow\!\!\$\ \mathcal{B}^{|\theta|}, \mathbf{c}_{\mathsf{PKE}} \leftarrow \mathsf{Encrypt}(\mathsf{ek}_{\mathsf{PKE}}, \mathbf{m}_b, \theta)$
5. $b' \leftarrow \mathcal{A}_{\mathsf{GUESS}}(\mathsf{ek}_{\mathsf{PKE}}, \mathbf{c}_{\mathsf{PKE}})$
6. **return** $(b = b')$

Figure 1: Experiment for the IND-CPA security of a PKE

Definition 2.4.2. *Let* $\mathsf{KEM} = (\mathsf{Keygen}, \mathsf{Encaps}, \mathsf{Decaps})$ *be a key encapsulation mechanism. The* KEM.IND-CCA2 *game is defined as in Figure 2 and the* IND-CCA2 *advantage of an adversary* $\mathcal{A} = (\mathcal{A}_{\mathsf{CHOOSE}}, \mathcal{A}_{\mathsf{GUESS}})$ *against a* KEM *is defined as:*

$$\mathsf{Adv}^{\mathsf{IND\text{-}CCA2}}_{\mathsf{KEM}}(\mathcal{A}) = \left|\Pr[\mathsf{KEM.IND\text{-}CCA2}(\mathcal{A}) \Rightarrow 1] - \frac{1}{2}\right|.$$

$\mathsf{Exp}\ \mathsf{KEM.IND\text{-}CCA2}(\mathcal{A})$

1. $(\mathsf{ek}_{\mathsf{KEM}}, \mathsf{dk}_{\mathsf{KEM}}) \leftarrow \mathsf{Keygen}()$
2. $b \leftarrow\!\!\$\ \{0,1\}$
3. $(K_0, \mathbf{c}_{\mathsf{KEM}}) \leftarrow \mathsf{Encaps}(\mathsf{ek}_{\mathsf{KEM}})$
4. $K_1 \leftarrow\!\!\$\ \mathcal{B}^{|K|}$
5. $b' \leftarrow \mathcal{A}^{\textsc{Decaps}(\cdot)}_{\mathsf{GUESS}}(K_b, \mathbf{c}_{\mathsf{KEM}})$
6. **return** $(b = b')$

Figure 2: Experiment for the IND-CCA2 security of a KEM

3 Specifications

3.1 XOF and Hash functions

Representation of objects. Seeds and salt are represented as byte strings while elements of $\mathbb{F}_2^k$, $\mathbb{F}_2^{n_1 n_2}$ and $\mathbb{F}_2^n$ are represented as binary arrays.

XOF. The XOF is instantiated using SHAKE256 along with appropriate domain separation as described in Table 1.

Hash functions. Hash functions G and I are instantiated using SHA3-512 while hash function H and J are instantiated using SHA3-256. Each hash function uses its own domain separator as described in Table 1.

Function	Instantiation
ctx ← XOF.Init(seed)	ctx ← SHAKE256.Init() in ← seed || XOF_DOMAIN_SEPARATOR SHAKE256.Absorb(ctx, in, |in|)
(ctx, out) ← XOF.GetBytes(ctx, |out|)	SHAKE256.Squeeze(ctx, out, |out|)
G(str)	SHA3-512(str || G_DOMAIN_SEPARATOR)
I(str)	SHA3-512(str || I_DOMAIN_SEPARATOR)
H(str)	SHA3-256(str || H_DOMAIN_SEPARATOR)
J(str)	SHA3-256(str || J_DOMAIN_SEPARATOR)

Table 1: Instantiation of XOF and Hash functions

3.2 Vector sampling

Sampling random vectors. Random vectors are sampled from $\mathbb{F}_2^k$ and $\mathbb{F}_2^n$ either uniformly or using SampleVect. For SampleVect, the random bits are derived from a seed using XOF.GetBytes. Unused bits within the vector representation are set to zero.

Sampling random fixed weight vectors. Sampling vectors from $\mathbb{F}_2^n$ of given Hamming weight is done using the $\mathsf{SampleFixedWeightVect}_{\$}$ and SampleFixedWeightVect functions. $\mathsf{SampleFixedWeightVect}_{\$}$ outputs uniformly distributed fixed weight vectors but rely on rejection sampling while SampleFixedWeightVect don't rely on rejection sampling but the

distribution of the sampled vectors is slightly biased with respect to the uniform distribution. However, it is shown in Section 6.2.3 that this bias does not significantly affect the security of the scheme. The SampleFixedWeightVect function follows Algorithm 5 from [36]. Let $\mathsf{rand}(n, \mathsf{ctx})$ be a function that produces a random integer uniformly distributed in $\{0, 1, \cdots, n-1\}$ from uniformly distributed integers in $\{0, 1, \cdots, 2^B - 1\}$ produced by $\mathsf{randBits}(B, \mathsf{ctx})$ for some integer B and XOF context ctx. The SampleFixedWeightVect routine starts by generating the vector support as described in GenerateRandomSupport, then convert it to an n-dimensional array.

GenerateRandomSupport$(\mathsf{ctx}, \mathcal{R}_\omega)$	rand(n, ctx)
1. **for** $i = \omega - 1$ **to** 0 **do**	1. $x \leftarrow \mathsf{randBits}(B, \mathsf{ctx})$
2. $\quad l \leftarrow i + \mathsf{rand}(n - i, \mathsf{ctx})$	2. **return** $x \bmod n$
3. $\quad \mathrm{pos}[i] \leftarrow (l \in \{\mathrm{pos}[j], i < j < \omega\})\ ?\ i : l$	
4. **return** $\mathrm{pos}[0], \cdots, \mathrm{pos}[\omega - 1]$	

Sampling order in HQC-PKE.Encrypt. During encryption, the randomness values are sampled in the following sequence: $\mathbf{r}_2$, then $\mathbf{e}$, and finally $\mathbf{r}_1$ which allows some speed-up for hardware implementations as suggested in [2].

3.3 Vector multiplication

Multiplications over $\mathbb{F}_2[X]/(X^n - 1)$ are performed without taking account the sparsity the considered polynomials in order to avoid potential leakage of information. Multiplications are computed using a combination of Toom-Cook and Karatsuba algorithms.

Toom-Cook multiplication over $\mathbb{F}_2[X]$. One wants to multiply two arbitrary polynomials over $\mathbb{F}_2[X]$ of degree at most $N-1$, using the Toom-Cook algorithm. Several approaches have been extensively detailed in the literature. Let A and B be two binary polynomials of degree at most $N-1$. These polynomials are packed into a table of 64 bit words, whose size is $\lceil N/64 \rceil$. Let $t = 3n$ with n a value ensuring $t \geqslant \lceil N/64 \rceil$. Now, A and B are considered as polynomials of degree at most $64 \cdot t - 1$. A and B are split into three parts. One wants now to evaluate the result $C = A \cdot B$ with

$$A = a_0 + a_1 \cdot X^{64n} + a_2 \cdot X^{2 \cdot 64n} \in \mathbb{F}_2[X],$$
$$B = b_0 + b_1 \cdot X^{64n} + b_2 \cdot X^{2 \cdot 64n} \in \mathbb{F}_2[X],$$

(of maximum degree $64t - 1$, and a_i, b_i of maximum degree $64n - 1$) and,

$$C = c_0 + c_1 \cdot X^{64n} + c_2 \cdot X^{2 \cdot 64n} + c_3 \cdot X^{3 \cdot 64n} + c_4 \cdot X^{4 \cdot 64n} \in \mathbb{F}_2[X]$$

of maximum degree $6 \cdot 64n - 2$.

The "word-aligned" version evaluates the polynomial for the values 0, 1, $x = X^w$, $x+1 = X^w + 1$, ∞, w being the word size, typically 64 in modern processors. Furthermore, on Intel processors, one can set $w = 256$ to take advantage of the vectorized instruction set `AVX-AVX2` at the cost of a slight size reduction. After the evaluation phase, one performs an interpolation to get the result coefficients.

For the evaluation phase, one has:

$$\begin{array}{lcl} C(0) & = & a_0 \cdot b_0 \\ C(1) & = & (a_0 + a_1 + a_2) \cdot (b_0 + b_1 + b_2) \\ C(x) & = & (a_0 + a_1 \cdot x + a_2 \cdot x^2) \cdot (b_0 + b_1 \cdot x + b_2 \cdot x^2) \\ C(x+1) & = & (a_0 + a_1 \cdot (x+1) + a_2 \cdot (x^2+1)) \cdot (b_0 + b_1 \cdot (x+1) + b_2 \cdot (x^2+1)) \\ C(\infty) & = & a_2 \cdot b_2 \end{array}$$

The implementation of this phase is straightforward, providing that the multiplications $a_i \cdot b_i$ is either another Toom-Cook or Karatsuba multiplication. One may notice that the multiplications by x or x^2 are virtually free word shifts.

Finally, the interpolation phase gives :

$$\begin{array}{lcl} c_0 & = & C(0) \\ c_1 & = & (x^2+x+1)/(x^2+x) \cdot C(0) + C(1) + C(x)/x + C(x+1)/(x+1) + (x^2+x) \cdot C(\infty) \\ c_2 & = & C(1)/(x^2+x) + C(x)/(x+1) + C(x+1)/x + (x^2+x+1) \cdot C(\infty) \\ c_3 & = & C(0)/(x^2+x) + C(1)/(x^2+x) + C(x)/(x^2+x) + C(x+1)/(x^2+x) \\ c_4 & = & C(\infty) \end{array}$$

Karatsuba algorithm. Let A and B be two binary polynomials of degree at most $N-1$. These polynomials are packed into a table of 64 bit words, whose size is $\lceil N/64 \rceil$. Let $t = 2^r$ with r the minimum value ensuring $t \geqslant \lceil N/64 \rceil$. Now, A and B are considered as polynomials of degree at most $64 \cdot t - 1$. The corresponding multiplication algorithm is described in Karatsuba. In this algorithm, the polynomials A and B are split into two parts, however, variants with other splits can be extrapolated. In particular, we used a 3-part split (`3-Karatsuba`) as the Toom-Cook elementary multiplication for HQC-128 and HQC-192, and a 5-part split (`5-Karatsuba`) as the Toom-Cook elementary multiplication for HQC-256. The multiplication line 2 (denoted `Mult64`) can be performed using a single processor instruction (`pclmul` for carry-less multiplier).

Application to HQC multiplications. The HQC parameters lead to the construction of the multiplications over $\mathbb{F}_2[X]$ depicted in Table 2.

Karatsuba(A, B, t)

Input: A and B on $t = 2^r$ computer words.
Output: $R = A \times B$.

1. **if** $t = 1$
2. **return** `Mult64`(A, B)
3. **else**
   ▷ Split in two halves of word size $t/2$
4. $A = A_0 + x^{64t/2} A_1$
5. $B = B_0 + x^{64t/2} B_1$

   ▷ Recursive multiplication
6. $R_0 \leftarrow$ Karatsuba$(A_0, B_0, t/2)$
7. $R_1 \leftarrow$ Karatsuba$(A_1, B_1, t/2)$
8. $R_2 \leftarrow$ Karatsuba$(A_0 + A_1, B_0 + B_1, t/2)$

   ▷ Reconstruction
9. $R \leftarrow R_0 + (R_0 + R_1 + R_2)X^{64t/2} + R_1 X^{64t}$
10. **return** R
11. **endif**

Instance	HQC-1	HQC-3	HQC-5
HQC Size (bits)	17 669	35 851	57 637
Main multiplication	`Toom3-Karatsuba3`	`Toom3-Karatsuba3`	`Toom-Cook3`
Size (bits)	18 048	36 480	59 904
Elementary multiplication	`3-Karatsuba`	`3-Karatsuba`	`5-Karatsuba`
Size (bits)	6144	12 288	20 480

Table 2: Implementation of HQC multiplications over $\mathbb{F}_2[X]$

3.4 Concatenated Reed-Muller and Reed-Solomon codes

HQC relies on a code $\mathcal{C}$ and associated $\mathcal{C}.\mathsf{Encode}$ and $\mathcal{C}.\mathsf{Decode}$ algorithms that are instantiated using concatenated Reed-Muller and Reed-Solomon codes as described in [4].

3.4.1 Concatenated codes

Definition 3.4.1 (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 to 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 use a Reed-Solomon code of dimension 32 over $\mathbb{F}_{256}$. For the internal code, we use the Reed-Muller code $[128, 8, 64]$ that we duplicate 3 or 5 times (*i.e.* duplicating each bit to obtain codes of parameters $[384, 8, 192]$ and $[640, 8, 320]$). We perform maximum likelihood decoding on the internal code. Doing that, we obtain a vector of $\mathbb{F}_q^{n_e}$ that is then decoded using an algebraic decoder for the Reed-Solomon code.

3.4.2 Reed-Solomon codes

Let p be a prime number and q is any power of p. Following [24], a Reed-Solomon code $\mathrm{RS}[n, k, d_{min}]$ with symbols in $\mathbb{F}_q$ 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 $d_{min} = 2\delta + 1$.

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

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

Shortened Reed-Solomon codes used in HQC. Depending on HQC parameters, we construct shortened Reed-Solomon (RS-S1, RS-S2 and RS-S3) codes such that k is equal to 16, 24 or 32 from the following RS codes RS-1, RS-2 and RS-3 from [24]. The shortened codes are obtained by subtracting 209 from the parameters n and k of the code RS-1, subtracting 199 from the parameters n and k of the code RS-2 and by subtracting 165 from the parameters n and k of the code RS-3. As a result, we obtain the following shortened Reed-Solomon codes:

- RS-S1$[46 = 255 - 209,\ 16 = 225 - 209,\ 31]$;
- RS-S2$[56 = 255 - 199,\ 24 = 223 - 199,\ 33]$;
- RS-S3$[90 = 255 - 165,\ 32 = 197 - 165,\ 49]$.

One should note that shortening the Reed-Solomon code does not affect its error correcting capacity. Table 3 provides the code parameters for both the original and resulting shortened Reed-Solomon codes.

Code	n	k	δ
RS-1	255	225	15
RS-2	255	223	16
RS-3	255	197	29
RS-S1	46	16	15
RS-S2	56	24	16
RS-S3	90	32	29

Table 3: Original and shortened Reed-Solomon codes.

Generator polynomials. For HQC, we are working in $\mathbb{F}_{2^m}$ with $m = 8$. To do so, we use the primitive polynomial $1 + \alpha^2 + \alpha^3 + \alpha^4 + \alpha^8$ of degree 8 to build this field [24]. 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 identical to the generator polynomials of Reed-Solomon codes RS-1, RS-2 and RS-3 respectively. The generator polynomials $g_1(x)$, $g_2(x)$ and $g_3(x)$ of the codes RS-S1, RS-S2 and RS-S3 have been precomputed and are given hereafter. One can use the functions provided in the file `reed_solomon.h` to reconstruct these generator polynomials.

Generator polynomial of RS-S1. $g_1(x) = 89 + 69x + 153x^2 + 116x^3 + 176x^4 + 117x^5 + 111x^6 + 75x^7 + 73x^8 + 233x^9 + 242x^{10} + 233x^{11} + 65x^{12} + 210x^{13} + 21x^{14} + 139x^{15} + 103x^{16} + 173x^{17} + 67x^{18} + 118x^{19} + 105x^{20} + 210x^{21} + 174x^{22} + 110x^{23} + 74x^{24} + 69x^{25} + 228x^{26} + 82x^{27} + 255x^{28} + 181x^{29} + x^{30}$.

Generator polynomial of RS-S2. $g_2(x) = 45 + 216x + 239x^2 + 24x^3 + 253x^4 + 104x^5 + 27x^6 + 40x^7 + 107x^8 + 50x^9 + 163x^{10} + 210x^{11} + 227x^{12} + 134x^{13} + 224x^{14} + 158x^{15} + 119x^{16} + 13x^{17} + 158x^{18} + x^{19} + 238x^{20} + 164x^{21} + 82x^{22} + 43x^{23} + 15x^{24} + 232x^{25} + 246x^{26} + 142x^{27} + 50x^{28} + 189x^{29} + 29x^{30} + 232x^{31} + x^{32}$.

Generator polynomial of RS-S3. $g_3(x) = 49 + 167x + 49x^2 + 39x^3 + 200x^4 + 121x^5 + 124x^6 + 91x^7 + 240x^8 + 63x^9 + 148x^{10} + 71x^{11} + 150x^{12} + 123x^{13} + 87x^{14} + 101x^{15} + 32x^{16} + 215x^{17} +$

$159x^{18} + 71x^{19} + 201x^{20} + 115x^{21} + 97x^{22} + 210x^{23} + 186x^{24} + 183x^{25} + 141x^{26} + 217x^{27} + 123x^{28} + 12x^{29} + 31x^{30} + 243x^{31} + 180x^{32} + 219x^{33} + 152x^{34} + 239x^{35} + 99x^{36} + 141x^{37} + 4x^{38} + 246x^{39} + 191x^{40} + 144x^{41} + 8x^{42} + 232x^{43} + 47x^{44} + 27x^{45} + 141x^{46} + 178x^{47} + 130x^{48} + 64x^{49} + 124x^{50} + 47x^{51} + 39x^{52} + 188x^{53} + 216x^{54} + 48x^{55} + 199x^{56} + 187x^{57} + x^{58}$.

Encoding shortened Reed-Solomon codes. In the following we present the encoding of Reed-Solomon codes which can also be used to encode shortened Reed-Solomon 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 codeword polynomial are the message bits and the leftmost $n - k$ bits are the parity-check bits. Following [24], 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 $c(x) = b(x) + x^{n-k}u(x)$.

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 [24], consider the Reed-Solomon code defined by $[n, k, d_{min}]$, with $n = 2^m - 1$ ($m \geq 0$ of positive integer) and suppose that a codeword $v(x) = v_0 + v_1x + \cdots + v_{n-1}x^{n-1}$ is transmitted. We denote $r(x) = r_0 + r_1x + \cdots + r_{n-1}x^{n-1}$ the received word, potentially altered by some errors.

We denote the error polynomial $e(x) = e_0 + e_1x + \cdots + e_{n-1}x^{n-1}$, meaning that there is an error in position j whenever $e_j \neq 0$. 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 $r(\alpha^i) = e(\alpha^i)$, 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:

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

One 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 $\sigma(x)$, one 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 [6] by:

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

The decoding is completed by computing $r(x) - e(x)$.

One can summarize the decoding procedure by the following steps:

1. Compute 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. Compute the error-location polynomial $\sigma(x)$ from the 2δ syndromes computed in the first step. Here we use Berlekamp's algorithm [24].

3. 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 [17].

4. Compute the polynomial $Z(x)$.

5. Compute the error values.

6. Correct the errors in the received polynomial.

3.4.3 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 $\text{RM}(r, m)$ 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 uses duplicated Reed-Muller codes. In particular, we are using first-order Reed-Muller denoted $\text{RM}(1, 7)$ 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 [28] 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 $F : \mathbb{F}_2^3 \to \{3, 1, -1, -3\}$ 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 $x \in \mathbb{F}_2^3$ 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.

Encoding Duplicated Reed-Muller codes. Following [28], the encoding is done in classical way by using a matrix vector multiplication. The codeword is then duplicated depending on the used parameter as described in Table 4.

Instance	Reed-Muller Code	Multiplicity	Duplicated Reed-Muller Code
HQC-1	[128, 8, 64]	3	[384, 8, 192]
HQC-3	[128, 8, 64]	5	[640, 8, 320]
HQC-5	[128, 8, 64]	5	[640, 8, 320]

Table 4: Duplicated Reed-Muller codes.

Decoding Duplicated Reed-Muller codes. Following [28], the decoding of duplicated Reed-Muller codes is done in three steps:

1. Compute the function F described and apply it on the received codeword. We give details about how this process is done where the multiplicity is equal to 2,

as an example. Let v be 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 3 or 5 follow a similar process.

2. Compute the Hadamard transform which is the first phase of the Green machine.

3. Compute 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 the all-one-vector and if there are two identical peaks, the peak with smallest value in the lowest 7 bits it taken.

3.5 HQC-PKE

HQC uses a decodable $[n_1 n_2, k]$ code $\mathcal{C}$ which can correct at least Δ errors as well as a random double-circulant $[2n, n]$ code with parity-check matrix $(\mathbf{I}_n \ \mathbf{rot}(\mathbf{h}))$. The code $\mathcal{C}$ is instantiated using concatenated Reed-Muller and Reed-Solomon codes which are described in Section 3.4. The codewords of $\mathcal{C}$ are in $\mathbb{F}_2^{n_1 n_2}$ while other vectors are in $\mathbb{F}_2^n$ where n is the smallest primitive prime greater than $n_1 n_2$. All computations are made in $\mathbb{F}_2^n$ and the remaining $\ell = n - n_1 n_2$ bits are truncated whenever required.

Correctness. The correctness of HQC relies on the decoding capability of the code $\mathcal{C}$. Specifically, $\mathcal{C}$ correctly decodes $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$ whenever:

$$\begin{aligned}
&\omega(\mathbf{s} \cdot \mathbf{r}_2 - \mathbf{u} \cdot \mathbf{y} + \mathbf{e}) \leq \Delta \\
&\omega((\mathbf{x} + \mathbf{h} \cdot \mathbf{y}) \cdot \mathbf{r}_2 - (\mathbf{r}_1 + \mathbf{h} \cdot \mathbf{r}_2) \cdot \mathbf{y} + \mathbf{e}) \leq \Delta \\
&\omega(\mathbf{x} \cdot \mathbf{r}_2 - \mathbf{r}_1 \cdot \mathbf{y} + \mathbf{e}) \leq \Delta
\end{aligned}$$

An analysis of the distribution of the error vector $\mathbf{e}' = \mathbf{x} \cdot \mathbf{r}_2 - \mathbf{r}_1 \cdot \mathbf{y} + \mathbf{e}$ and the resulting decoding failure rate of the scheme is provided in Section 6.1.

Keypair format. The decapsulation key $\mathsf{dk}_{\mathsf{KEM}}$ can be stored using the default format $\mathsf{dk}_{\mathsf{KEM}} = (\mathsf{ek}_{\mathsf{KEM}}, \mathsf{dk}_{\mathsf{PKE}}, \sigma, \mathsf{seed}_{\mathsf{KEM}})$ or the compressed format $\mathsf{dk}_{\mathsf{KEM}} = (\mathsf{seed}_{\mathsf{KEM}})$. In both cases, the keypair $(\mathsf{ek}_{\mathsf{KEM}}, \mathsf{dk}_{\mathsf{KEM}})$ can be checked using $\mathsf{seed}_{\mathsf{KEM}}$ to derive $(\mathsf{seed}_{\mathsf{PKE}}, \sigma)$ which in turn can be used to derive $(\mathsf{ek}_{\mathsf{PKE}}, \mathsf{dk}_{\mathsf{PKE}})$ where $\mathsf{ek}_{\mathsf{KEM}} = \mathsf{ek}_{\mathsf{PKE}}$.

HQC-PKE.Keygen($\mathsf{seed}_{\mathsf{PKE}}$)

Input: Seed $\mathsf{seed}_{\mathsf{PKE}}$.

Output: Encryption key $\mathsf{ek}_{\mathsf{PKE}}$, decryption key $\mathsf{dk}_{\mathsf{PKE}}$.

▷ Compute $\mathsf{dk}_{\mathsf{PKE}}$ and $\mathsf{ek}_{\mathsf{PKE}}$ seeds

1. $(\mathsf{seed}_{\mathsf{PKE.dk}}, \mathsf{seed}_{\mathsf{PKE.ek}}) \leftarrow \mathsf{I}(\mathsf{seed}_{\mathsf{PKE}})$

▷ Compute decryption key $\mathsf{dk}_{\mathsf{PKE}}$

2. $\mathsf{ctx}_{\mathsf{PKE.dk}} \leftarrow \mathsf{XOF.Init}(\mathsf{seed}_{\mathsf{PKE.dk}})$
3. $(\mathsf{ctx}_{\mathsf{PKE.dk}}, \mathbf{y}) \leftarrow \mathsf{SampleFixedWeightVect}_{\$}(\mathsf{ctx}_{\mathsf{PKE.dk}}, \mathcal{R}_{\omega})$
4. $(\mathsf{ctx}_{\mathsf{PKE.dk}}, \mathbf{x}) \leftarrow \mathsf{SampleFixedWeightVect}_{\$}(\mathsf{ctx}_{\mathsf{PKE.dk}}, \mathcal{R}_{\omega})$
5. $\mathsf{dk}_{\mathsf{PKE}} \leftarrow \mathsf{seed}_{\mathsf{PKE.dk}}$

▷ Compute encryption key $\mathsf{ek}_{\mathsf{PKE}}$

6. $\mathsf{ctx}_{\mathsf{PKE.ek}} \leftarrow \mathsf{XOF.Init}(\mathsf{seed}_{\mathsf{PKE.ek}})$
7. $(\mathsf{ctx}_{\mathsf{PKE.ek}}, \mathbf{h}) \leftarrow \mathsf{SampleVect}(\mathsf{ctx}_{\mathsf{PKE.ek}}, \mathcal{R})$
8. $\mathbf{s} \leftarrow \mathbf{x} + \mathbf{h} \cdot \mathbf{y}$
9. $\mathsf{ek}_{\mathsf{PKE}} \leftarrow (\mathsf{seed}_{\mathsf{PKE.ek}}, \mathbf{s})$

10. **return** $(\mathsf{ek}_{\mathsf{PKE}}, \mathsf{dk}_{\mathsf{PKE}})$

HQC-PKE.Encrypt($\mathsf{ek}_{\mathsf{PKE}}, \mathbf{m}, \theta$)

Input: Encryption key $\mathsf{ek}_{\mathsf{PKE}}$, message $\mathbf{m}$, randomness θ.

Output: Ciphertext $\mathbf{c}_{\mathsf{PKE}}$.

▷ Parse encryption key $\mathsf{ek}_{\mathsf{PKE}}$

1. $\mathsf{seed}_{\mathsf{PKE.ek}} \leftarrow \mathsf{ek}_{\mathsf{PKE}}[0 : |\mathsf{seed}|]$
2. $\mathsf{ctx}_{\mathsf{PKE.ek}} \leftarrow \mathsf{XOF.Init}(\mathsf{seed}_{\mathsf{PKE.ek}})$
3. $(\mathsf{ctx}_{\mathsf{PKE.ek}}, \mathbf{h}) \leftarrow \mathsf{SampleVect}(\mathsf{ctx}_{\mathsf{PKE.ek}}, \mathcal{R})$
4. $\mathbf{s} \leftarrow \mathsf{ek}_{\mathsf{PKE}}[|\mathsf{seed}| : |\mathsf{seed}| + |\mathbf{s}|]$

▷ Compute ciphertext $\mathbf{c}_{\mathsf{PKE}}$

5. $\mathsf{ctx}_\theta \leftarrow \mathsf{XOF.Init}(\theta)$
6. $(\mathsf{ctx}_\theta, \mathbf{r}_2) \leftarrow \mathsf{SampleFixedWeightVect}(\mathsf{ctx}_\theta, \mathcal{R}_{\omega_r})$
7. $(\mathsf{ctx}_\theta, \mathbf{e}) \leftarrow \mathsf{SampleFixedWeightVect}(\mathsf{ctx}_\theta, \mathcal{R}_{\omega_e})$
8. $(\mathsf{ctx}_\theta, \mathbf{r}_1) \leftarrow \mathsf{SampleFixedWeightVect}(\mathsf{ctx}_\theta, \mathcal{R}_{\omega_r})$
9. $\mathbf{u} \leftarrow \mathbf{r}_1 + \mathbf{h} \cdot \mathbf{r}_2$
10. $\mathbf{v} \leftarrow \mathcal{C}.\mathsf{Encode}(\mathbf{m}) + \mathsf{Truncate}(\mathbf{s} \cdot \mathbf{r}_2 + \mathbf{e}, \ell)$
11. $\mathbf{c}_{\mathsf{PKE}} \leftarrow (\mathbf{u}, \mathbf{v})$

12. **return** $\mathbf{c}_{\mathsf{PKE}}$

HQC-PKE.Decrypt($\mathsf{dk}_{\mathsf{PKE}}, \mathbf{c}_{\mathsf{PKE}}$)

Input: Decryption key $\mathsf{dk}_{\mathsf{PKE}}$, ciphertext $\mathbf{c}_{\mathsf{PKE}}$.

Output: Message $\mathbf{m}$.

▷ Parse decryption key $\mathsf{dk}_{\mathsf{PKE}}$

1. $\mathsf{seed}_{\mathsf{PKE.dk}} \leftarrow \mathsf{dk}_{\mathsf{PKE}}[0 : |\mathsf{seed}|]$
2. $\mathsf{ctx}_{\mathsf{PKE.dk}} \leftarrow \mathsf{XOF.Init}(\mathsf{seed}_{\mathsf{PKE.dk}})$
3. $(\mathsf{ctx}_{\mathsf{PKE.dk}}, \mathbf{y}) \leftarrow \mathsf{SampleFixedWeightVect}_{\$}(\mathsf{ctx}_{\mathsf{PKE.dk}}, \mathcal{R}_\omega)$

▷ Parse ciphertext $\mathbf{c}_{\mathsf{PKE}}$

4. $\mathbf{u} \leftarrow \mathbf{c}_{\mathsf{PKE}}[0 : |\mathbf{u}|]$
5. $\mathbf{v} \leftarrow \mathbf{c}_{\mathsf{PKE}}[|\mathbf{u}| : |\mathbf{u}| + |\mathbf{v}|]$

▷ Compute plaintext $\mathbf{m}$

6. $\mathbf{m} \leftarrow \mathcal{C}.\mathsf{Decode}(\mathbf{v} - \mathsf{Truncate}(\mathbf{u} \cdot \mathbf{y}, \ell))$

7. **return** $\mathbf{m}$

3.6 HQC-KEM

HQC-KEM is derived from HQC-PKE using the Fujisaki-Okamoto transformation [14] instantiated following the HHK framework with implicit rejection [22].

HQC-KEM.Keygen()

Input: None.

Output: Encapsulation key $\mathsf{ek}_{\mathsf{KEM}}$, decapsulation key $\mathsf{dk}_{\mathsf{KEM}}$.

▷ Sample $\mathsf{seed}_{\mathsf{KEM}}$

1. $\mathsf{seed}_{\mathsf{KEM}} \xleftarrow{\$} \mathcal{B}^{|\mathsf{seed}|}$

▷ Compute $\mathsf{seed}_{\mathsf{PKE}}$ and randomness σ

2. $\mathsf{ctx}_{\mathsf{KEM}} \leftarrow \mathsf{XOF.Init}(\mathsf{seed}_{\mathsf{KEM}})$
3. $(\mathsf{ctx}_{\mathsf{KEM}}, \mathsf{seed}_{\mathsf{PKE}}) \leftarrow \mathsf{XOF.GetBytes}(\mathsf{ctx}_{\mathsf{KEM}}, |\mathsf{seed}|)$
4. $(\mathsf{ctx}_{\mathsf{KEM}}, \sigma) \leftarrow \mathsf{XOF.GetBytes}(\mathsf{ctx}_{\mathsf{KEM}}, |k|)$

▷ Compute HQC-PKE keypair

5. $(\mathsf{ek}_{\mathsf{PKE}}, \mathsf{dk}_{\mathsf{PKE}}) \leftarrow \mathsf{HQC\text{-}PKE.Keygen}(\mathsf{seed}_{\mathsf{PKE}})$

▷ Compute HQC-KEM keypair

6. $\mathsf{ek}_{\mathsf{KEM}} \leftarrow \mathsf{ek}_{\mathsf{PKE}}$
7. $\mathsf{dk}_{\mathsf{KEM}} \leftarrow (\mathsf{ek}_{\mathsf{KEM}}, \mathsf{dk}_{\mathsf{PKE}}, \sigma, \mathsf{seed}_{\mathsf{KEM}})$

8. **return** $(\mathsf{ek}_{\mathsf{KEM}}, \mathsf{dk}_{\mathsf{KEM}})$

HQC-KEM.Encaps($\mathsf{ek}_{\mathsf{KEM}}$)

Input: Encapsulation key $\mathsf{ek}_{\mathsf{KEM}}$.

Output: Shared secret key K, ciphertext $\mathbf{c}_{\mathsf{KEM}}$.

▷ Sample message $\mathbf{m}$ and salt

1. $\mathbf{m} \leftarrow\!\!\$\ \mathcal{B}^{|k|}$
2. $\mathsf{salt} \leftarrow\!\!\$\ \mathcal{B}^{|\mathsf{salt}|}$

▷ Compute shared key K and ciphertext $\mathbf{c}_{\mathsf{KEM}}$

3. $(K, \theta) \leftarrow \mathsf{G}(\mathsf{H}(\mathsf{ek}_{\mathsf{KEM}}) \| \mathbf{m} \| \mathsf{salt})$
4. $\mathbf{c}_{\mathsf{PKE}} \leftarrow \mathsf{HQC\text{-}PKE.Encrypt}(\mathsf{ek}_{\mathsf{KEM}}, \mathbf{m}, \theta)$
5. $\mathbf{c}_{\mathsf{KEM}} \leftarrow (\mathbf{c}_{\mathsf{PKE}}, \mathsf{salt})$
6. **return** $(K, \mathbf{c}_{\mathsf{KEM}})$

HQC-KEM.Decaps($\mathsf{dk}_{\mathsf{KEM}}, \mathbf{c}_{\mathsf{KEM}}$)

Input: Decapsulation key $\mathsf{dk}_{\mathsf{KEM}}$, ciphertext $\mathbf{c}_{\mathsf{KEM}}$.

Output: Shared secret key K'.

▷ Parse decapsulation key $\mathsf{dk}_{\mathsf{KEM}}$

1. $\mathsf{ek}_{\mathsf{KEM}} \leftarrow \mathsf{dk}_{\mathsf{KEM}}[0 : |\mathsf{ek}_{\mathsf{KEM}}|]$
2. $\mathsf{dk}_{\mathsf{PKE}} \leftarrow \mathsf{dk}_{\mathsf{KEM}}[|\mathsf{ek}_{\mathsf{KEM}}| : |\mathsf{ek}_{\mathsf{KEM}}| + |\mathsf{dk}_{\mathsf{PKE}}|]$
3. $\sigma \leftarrow \mathsf{dk}_{\mathsf{KEM}}[|\mathsf{ek}_{\mathsf{KEM}}| + |\mathsf{dk}_{\mathsf{PKE}}| : |\mathsf{ek}_{\mathsf{KEM}}| + |\mathsf{dk}_{\mathsf{PKE}}| + |\sigma|]$

▷ Parse ciphertext $\mathbf{c}_{\mathsf{KEM}}$

4. $\mathbf{c}_{\mathsf{PKE}} \leftarrow \mathbf{c}_{\mathsf{KEM}}[0 : |\mathbf{c}_{\mathsf{PKE}}|]$
5. $\mathsf{salt} \leftarrow \mathbf{c}_{\mathsf{KEM}}[|\mathbf{c}_{\mathsf{PKE}}| : |\mathbf{c}_{\mathsf{PKE}}| + |\mathsf{salt}|]$

▷ Compute message $\mathbf{m}'$

6. $\mathbf{m}' \leftarrow \mathsf{HQC\text{-}PKE.Decrypt}(\mathsf{dk}_{\mathsf{PKE}}, \mathbf{c}_{\mathsf{PKE}})$

▷ Compute shared key K' and ciphertext $\mathbf{c}'_{\mathsf{KEM}}$

7. $(K', \theta') \leftarrow \mathsf{G}(\mathsf{H}(\mathsf{ek}_{\mathsf{KEM}}) \| \mathbf{m}' \| \mathsf{salt})$
8. $\mathbf{c}'_{\mathsf{PKE}} \leftarrow \mathsf{HQC\text{-}PKE.Encrypt}(\mathsf{ek}_{\mathsf{KEM}}, \mathbf{m}', \theta')$
9. $\mathbf{c}'_{\mathsf{KEM}} \leftarrow (\mathbf{c}'_{\mathsf{PKE}}, \mathsf{salt})$

▷ Compute rejection key $\bar{K}$

10. $\bar{K} \leftarrow \mathsf{J}(\mathsf{H}(\mathsf{ek}_{\mathsf{KEM}}) \| \sigma \| \mathbf{c}_{\mathsf{KEM}})$
11. **if** $\mathbf{m}' = \perp$ **or if** $\mathbf{c}'_{\mathsf{KEM}} \neq \mathbf{c}_{\mathsf{KEM}}$
12. $\quad K' \leftarrow \bar{K}$
13. **endif**

14. **return** K'

4 Parameters and Sizes

This section provides parameters sets for HQC and resulting ciphertext and key sizes. For each parameter set, the parameters are chosen so that the minimal workfactor of the best known attack exceeds the security parameter (see Section 6.3 for more details).

4.1 Parameter sets

The parameters sets of HQC are given in Table 5 where n_1 denote the length of the external Reed-Solomon code and n_2 denote the length of the internal Reed-Muller code so that the length of the concatenated code $\mathcal{C}$ is $n_1 n_2$, its dimension is k and its decoding failure rate (DFR) is adjusted for each security level. The parameter n denotes the length of the ambient space namely the smallest primitive prime greater than $n_1 n_2$. The parameters ω, ω_r and ω_e denote the weight of the vectors $(\mathbf{x}, \mathbf{y})$, $(\mathbf{r}_1, \mathbf{r}_2)$ and $\mathbf{e}$ respectively.

Instance	Security	n_1	n_2	n	k	ω	$\omega_r = \omega_e$	DFR
HQC-1	NIST-1	46	384	17 669	128	66	75	$< 2^{-128}$
HQC-3	NIST-3	56	640	35 851	192	100	114	$< 2^{-192}$
HQC-5	NIST-5	90	640	57 637	256	131	149	$< 2^{-256}$

Table 5: Parameter sets for HQC

4.2 Ciphertext and key sizes

The encapsulation key $\mathsf{ek}_{\mathsf{KEM}}$ has size $|\mathsf{ek}_{\mathsf{KEM}}| = |\mathsf{seed}| + \lceil n/8 \rceil$ while the decapsulation key $\mathsf{dk}_{\mathsf{KEM}}$ has size $|\mathsf{dk}_{\mathsf{KEM}}| = |\mathsf{ek}_{\mathsf{KEM}}| + |\mathsf{seed}| + \lceil k/8 \rceil + |\mathsf{seed}|$ or $|\mathsf{dk}_{\mathsf{KEM}}| = |\mathsf{seed}|$ if the compressed format is used. The ciphertext $\mathbf{c}_{\mathsf{KEM}}$ has size $|\mathbf{c}_{\mathsf{KEM}}| = \lceil n/8 \rceil + \lceil (n_1 n_2)/8 \rceil + |\mathsf{salt}|$ while the shared key K has size $|K|$. The seeds, salt and shared key have size $|\mathsf{seed}| = 32$ B, $|\mathsf{salt}| = 16$ B, $|K| = 32$ B respectively. The resulting sizes are given in bytes in Table 6.

Instance	Security	$\lvert\mathsf{ek}_{\mathsf{KEM}}\rvert$	$\lvert\mathsf{dk}_{\mathsf{KEM}}\rvert$		$\lvert\mathbf{c}_{\mathsf{KEM}}\rvert$	$\lvert K\rvert$
HQC-1	NIST-1	2 241	2 321	32	4 433	32
HQC-3	NIST-3	4 514	4 602	32	8 978	32
HQC-5	NIST-5	7 237	7 333	32	14 421	32

Table 6: HQC keypair $(\mathsf{ek}_{\mathsf{KEM}}, \mathsf{dk}_{\mathsf{KEM}})$, ciphertext $\mathbf{c}_{\mathsf{KEM}}$ and shared key K sizes (in Bytes)

5 Performance Analysis

This section provides performance measurements of our HQC.KEM implementations.

Benchmark platform. The benchmarks have been performed on a machine running Ubuntu 22.04.2 LTS, that has 32 GB of memory and an Intel® Core™ i7-11850H CPU @ 2.50GHz for which the Hyper-Threading and Turbo Boost features were disabled. The scheme have been compiled with `gcc` (version `11.4.0`). All benchmarks were conducted using the default key format, where the decapsulation key is represented as $\mathsf{dk}_{\mathsf{KEM}} = (\mathsf{ek}_{\mathsf{KEM}}, \mathsf{dk}_{\mathsf{PKE}}, \sigma, \mathsf{seed}_{\mathsf{KEM}})$. 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. Both the reference and the optimized AVX2 implementations have been implemented in constant time. We have thoroughly analyzed the code to ensure that only unused randomness (i.e., rejected based on public criteria) or otherwise non-sensitive data may be leaked.

5.1 Reference implementation

The performances of our reference implementation on the aforementioned benchmark platform are given in Table 7. The reference implementation is written in `C` and have been compiled with `gcc` (version `11.4.0`) using the flags `-O3 -std=c99 -funroll-all-loops -flto -pedantic -Wall -Wextra`.

Instance	KeyGen	Encaps	Decaps
HQC-1	4 557	9 116	13 918
HQC-3	13 783	27 571	41 669
HQC-5	33 123	66 261	100 213

Table 7: Performances (in kiloCycles) of the reference implementation of HQC

5.2 Optimized implementation

The performances of our optimized implementation on the aforementioned benchmark platform are given in Table 8. Our optimized implementation leverages AVX2 instructions and have been compiled using `gcc` (version 8.2.1) using the flags `-O3 -std=c99 -funroll-all-loops -mavx -mavx2 -mpclmul -pedantic -Wall -Wextra`.

Instance	KeyGen	Encaps	Decaps
HQC-1	76	150	353
HQC-3	181	355	732
HQC-5	363	720	1 435

Table 8: Performance (in kiloCycles) of the optimized implementation of HQC

5.3 Known Answer Test values

Known Answer Test (KAT) values have been generated using the script provided by the NIST. They are available in the folders `KATs/Reference_Implementation/ and KATs/Optimized_Implementation/.` In addition, examples with intermediate values have also been provided in these folders. One should note 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.

6 Security Analysis

6.1 Decoding Failure Rate analysis

We analyze the distribution of the error vector $\mathbf{e}' = \mathbf{x} \cdot \mathbf{r}_2 - \mathbf{r}_1 \cdot \mathbf{y} + \mathbf{e}$ in Section 6.1.1 and the Decoding Failure Rate (DFR) of the internal Reed-Muller code in Section 6.1.2. The resulting DFR of the scheme is studied in Section 6.1.3.

6.1.1 Analysis of the error vector distribution

We provide a precise analysis of the error distribution approximation following [4]. 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 6.1.2.

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

Proposition 6.1.1. *Let $\mathbf{x} = (x_0, \dots x_{n-1})$ be a random vector chosen uniformly among all binary vectors of weight ω and let $\mathbf{r} = (r_0, \dots, r_{n-1})$ be a random vector chosen uniformly among all vectors of weight ω_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:*

$$\tilde{p} = \frac{1}{\binom{n}{\omega}\binom{n}{\omega_r}} \sum_{\substack{1 \leqslant \ell \leqslant \min(\omega, \omega_r) \\ \ell \text{ odd}}} C_\ell$$

where $C_\ell = \binom{n}{\ell}\binom{n-\ell}{\omega-\ell}\binom{n-\omega}{\omega_r-\ell}$.

Proof. The total number of ordered pairs $(\mathbf{x}, \mathbf{r})$ is $\binom{n}{\omega}\binom{n}{\omega_r}$. 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 $\ell \le \min(\omega, \omega_r)$ we have $\binom{n}{\ell}$ choices for the set of coordinates i such that $x_i = r_{k-i} = 1$, then $\binom{n-\ell}{\omega-\ell}$ remaining choices for the set of coordinates i such that $x_i = 1$ and $r_{k-i} = 0$, and finally $\binom{n-\omega}{\omega_r-\ell}$ remaining choices for the set of coordinates i such that $x_i = 0$ and $r_{k-i} = 1$. Hence $C_\ell = \binom{n}{\ell}\binom{n-\ell}{\omega-\ell}\binom{n-\omega}{\omega_r-\ell}$. The formula for $\tilde{p}$ follows. □

Let $\mathbf{x}, \mathbf{y}$ (respectively $\mathbf{r}_1, \mathbf{r}_2$) be independent random vectors chosen uniformly among all binary vectors of weight ω (respectively ω_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 6.1.1. Therefore their modulo 2 sum $\mathbf{t} = \mathbf{x} \cdot \mathbf{r}_2 - \mathbf{r}_1 \cdot \mathbf{y}$ is Bernoulli distributed with:

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

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

Proposition 6.1.2. *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 ω_r for $\mathbf{r}_1, \mathbf{r}_2$, and among vectors of weight ω_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:*

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

Proposition 6.1.2 gives us the probability that a coordinate of the error vector $\mathbf{e}'$ is 1. In our simulations, which occur in the regime $\omega = \alpha\sqrt{n}$ 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. (2): $p^\star = 2\tilde{p}(1-\tilde{p})(1-\frac{\omega_e}{n}) + ((1-\tilde{p})^2 + \tilde{p}^2)\,\frac{\omega_e}{n}$. This approximation will give us, for $0 \le d \le \min(2 \times \omega \times \omega_r + \omega_e, n)$,

$$\Pr[\omega(\mathbf{e}') = d] = \binom{n}{d}(p^\star)^d(1-p^\star)^{(n-d)}. \tag{3}$$

Simulation results. We provide 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 illustrated on the parameter set corresponding to HQC-1. 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 hereafter by simulations with real weight parameters (but smaller lengths).

We consider a parameter set that corresponds to cryptographic parameters and for which we simulate the error distribution versus the binomial approximation together with the probability of obtaining large error weights. 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	ω	$\omega_e = \omega_r$	n	$n_1 n_2$	$p^\star$
HQC-1	66	75	17 669	17 664	0.3398

Table 9: Probability p^* for HQC-1 parameter set

Simulation results are shown in Figure 3. 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 10.

	0.1%	0.01%	0.001%	0.0001%
Error vectors	6 169	6 203	6 232	6 257
Binomial approximation	6 197	6 237	6 272	6 301

Table 10: Simulated probabilities of large weight vectors for HQC-1 for the error vector distribution and the binomial approximation

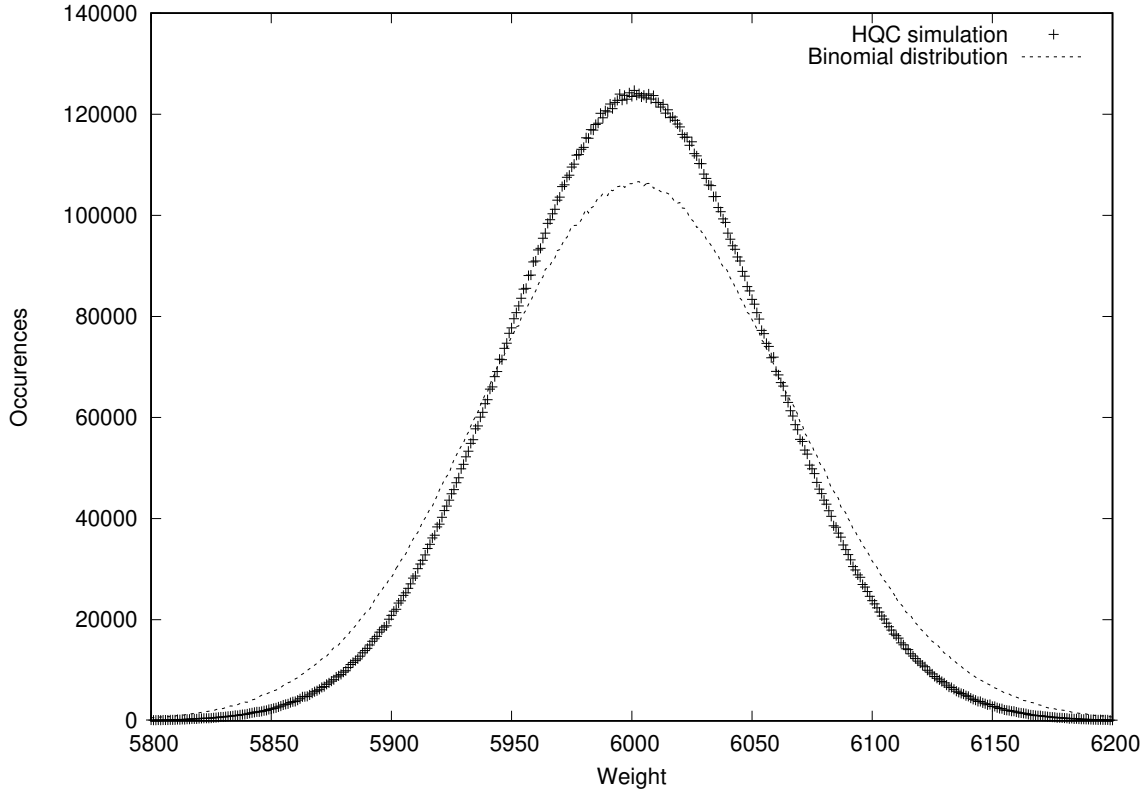

Figure 3: Comparison between error $\mathbf{e}'$ generated using HQC-1 parameter set and its binomial approximation

6.1.2 Upper bound for the DFR of the internal Reed-Muller code

It is only possible to obtain an exact decoding probability formula 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 6.1.3 (Simple Upper Bound for the DFR of the internal code)**.** *Let p be the transition probability of the binary symmetric channel. Then the DFR of a duplicated 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 $\mathbf{c}$ (for

$\mathbf{x}$ a non-zero word of C and $\omega(\mathbf{x})$ the weight of $\mathbf{x}$) is:

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

By the union bound applied on the different non-zero codewords $\mathbf{x}$ of C, we obtain that the probability of a decryption failure can thus be upper bounded by:

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

There are 255 non-zero words in a [128,8,64] Reed-Muller code, 254 of weight 64 and one of weight 128. The contribution of the weight 128 vector is smaller than the weight 64 vectors, hence by applying the previous bound to duplicated Reed-Muller codes we obtain the result. □

Better upper bound for the DFR of the internal code. The previous simple bound pessimistically assumes that decoding fails when more than one codeword minimizes the distance to the received vector. The following bound improves the previous one by taking into account the fact that decoding can still succeed with probability 1/2 when exactly two codewords minimize the distance to the received vector.

Proposition 6.1.4 (Improved Upper Bound for the DFR 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:*

$$p_i = \sum_{\omega = d_i/2}^{n} \mathfrak{A}_\omega p^\omega (1-p)^{n-\omega}$$

where

$$\mathfrak{A}_\omega = \min \left[\binom{n}{\omega}, \frac{1}{2} 255 \binom{d_i}{d_i/2} \binom{d_i}{\omega - d_i/2} + 255 \sum_{j=d_i/2+1}^{d_i} \binom{d_i}{j} \binom{d_i}{\omega - j} + \frac{1}{2} \binom{255}{2} \sum_{j=0}^{d_i/2} \binom{d_i/2}{j}^3 \binom{d_i/2}{\omega - d_i + j} \right].$$

Proof. Let E be the decoding error event. Let $\mathbf{e}$ be the error vector.

- Let A be the event where the closest non-zero codeword $\mathbf{c}$ to the error is such that $d(\mathbf{e}, \mathbf{c}) = d(\mathbf{e}, \mathbf{0}) = \omega(\mathbf{e})$.
- Let B be the event where the closest non-zero codeword $\mathbf{c}$ to the error vector is such that $d(\mathbf{e}, \mathbf{c}) < \omega(\mathbf{e})$.

- Let $A' \subset A$ be the event where the closest non-zero codeword $\mathbf{c}$ to the error vector is such that $d(\mathbf{e}, \mathbf{c}) = \omega(\mathbf{e})$ and such a vector is unique, meaning that for every $\mathbf{c}' \in C, \mathbf{c}' \neq \mathbf{c}, \mathbf{c}' \neq \mathbf{0}$, we have $d(\mathbf{e}, \mathbf{c}') > \omega(\mathbf{e})$.
- Finally, let A'' be the event that is the complement of A' in A, meaning the event where the closest non-zero codeword $\mathbf{c}$ to the error is at distance $|\mathbf{e}|$ from $\mathbf{e}$, and there exists at least one codeword $\mathbf{c}', \mathbf{c}' \neq \mathbf{c}, \mathbf{c}' \neq \mathbf{0}$, such that $d(\mathbf{e}, \mathbf{c}') = d(\mathbf{e}, \mathbf{c}) = \omega(\mathbf{e})$.

The probability space is partitioned as $\Omega = A \cup B \cup C = A' \cup A'' \cup B \cup C$, where C is the complement of $A \cup B$. When C occurs, the decoder always decodes correctly, i.e. $P(E|C) = 0$. We therefore write:

$$P(E) = P(E|A')P(A') + P(E|A'')P(A'') + P(E|B)P(B)$$

When the event A' occurs, the decoder chooses at random between the two closest codewords and is correct with probability $1/2$, i.e. $P(E|A') = 1/2$. We have $P(E|B) = 1$ and writing $P(E|A'') \leqslant 1$, we have:

$$\begin{aligned} P(E_\omega) &\leqslant \frac{1}{2}P(A'_\omega) + P(A''_\omega) + P(B_\omega) \\ &= \frac{1}{2}(P(A'_\omega) + P(A''_\omega)) + \frac{1}{2}P(A''_\omega) + P(B_\omega) \\ P(E_w) &\leqslant \frac{1}{2}P(A_\omega) + \frac{1}{2}P(A''_\omega) + P(B_\omega) \end{aligned} \tag{4}$$

where for $X = A, A', A'', E$, the event X_ω signifies the intersection of the event X with the event "$\omega(\mathbf{e}) = \omega$". Now we have the straightforward union bounds:

$$P(B_\omega) \leqslant 255 \sum_{j=d_i/2+1}^{d_i} \binom{d_i}{j} \binom{d_i}{weight - j} p^\omega (1-p)^{n-\omega} \tag{5}$$

with $n = 2d_i$ the length of the inner code, and where we use the convention that a binomial coefficient $\binom{\ell}{k} = 0$ whenever $k < 0$ or $k > \ell$.

$$P(A_\omega) \leqslant 255 \binom{d_i}{d_i/2} \binom{d_i}{\omega - d_i/2} p^\omega (1-p)^{n-\omega} \tag{6}$$

and it remains to find an upper bound on $P(A'')$. We have:

$$P(A'') \leqslant \sum_{\mathbf{c},\mathbf{c}'} P(A_{\mathbf{c},\mathbf{c}'})$$

where the sum is over pairs of distinct non-zero codewords and where:

$$A_{\mathbf{c},\mathbf{c}'} = \{d(\mathbf{e}, \mathbf{c}) = d(\mathbf{e}, \mathbf{c}') = \omega(\mathbf{e})\}$$

This event is equivalent to the error meeting the supports of $\mathbf{c}$ and $\mathbf{c}'$ on exactly half their coordinates. All codewords except the all-one vector have weight d_i, and any two codewords of weight d_i either have non-intersecting supports or intersect in exactly $d/2$ positions. $P(A_{\mathbf{c},\mathbf{c}'})$ is largest when $\mathbf{c}$ and $\mathbf{c}'$ have weight d and non-zero intersection. In this case we have:

$$P(A^{\omega}_{\mathbf{c},\mathbf{c}'}) = \sum_{j=0}^{d_i/2} \binom{d_i/2}{j}^3 \binom{d_i/2}{\omega - d_i + j} p^{\omega}(1-p)^{n-\omega}.$$

Hence,

$$P(A''_{\omega}) \leqslant \sum_{\mathbf{c},\mathbf{c}'} P(A_{\mathbf{c},\mathbf{c}'}) \leqslant \binom{255}{2} \sum_{j=0}^{d_i/2} \binom{d_i/2}{j}^3 \binom{d_i/2}{\omega - d_i + j} p^{\omega}(1-p)^{n-\omega}. \tag{7}$$

Plugging 6, 5 and 7 into 4 we obtain the result. □

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.

Simulation results. We performed simulations to compare the real decryption failure rate with the theoretical one from proposition 6.1.3 for $[512, 8, 256]$ and $[640, 8, 320]$ duplicated Reed-Muller codes using $p^{\star}$ values from actual parameters. Simulation results are presented table 11.

Security level	$p^{\star}$	Reed-Muller code	DFR from 6.1.4	Observed DFR
NIST-1	0.3398	$[384, 8, 192]$	-10.79	-10.96
NIST-3	0.3618	$[640, 8, 320]$	-14.14	-14.39
NIST-5	0.3725	$[640, 8, 320]$	-11.30	-11.48

Table 11: Comparison between the observed Decryption Failure Rate and the formula from proposition 6.1.3. Results are presented as $\log_2(DFR)$.

6.1.3 Decoding failure rate analysis

Using the lower bound p_i on the decoding probability of the Reed-Muller codes given in Section 6.1.2, one can deduce the DFR of the concatenated code used in HQC.

Theorem 6.1 (DFR of the concatenated code)**.** *The DFR of the concatenated code using a Reed-Solomon code $[n_e, k_e, d_e]_{\mathbb{F}_{256}}$ as the external code and a Reed-Muller code as the internal 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 6.1.3.*

Simulation results. In Figure 4, we tested the DFR of the concatenated codes against both symmetric binary channels and HQC vectors, and compared the results with the theoretical value obtained using Proposition 6.1.3 and Theorem 6.1.

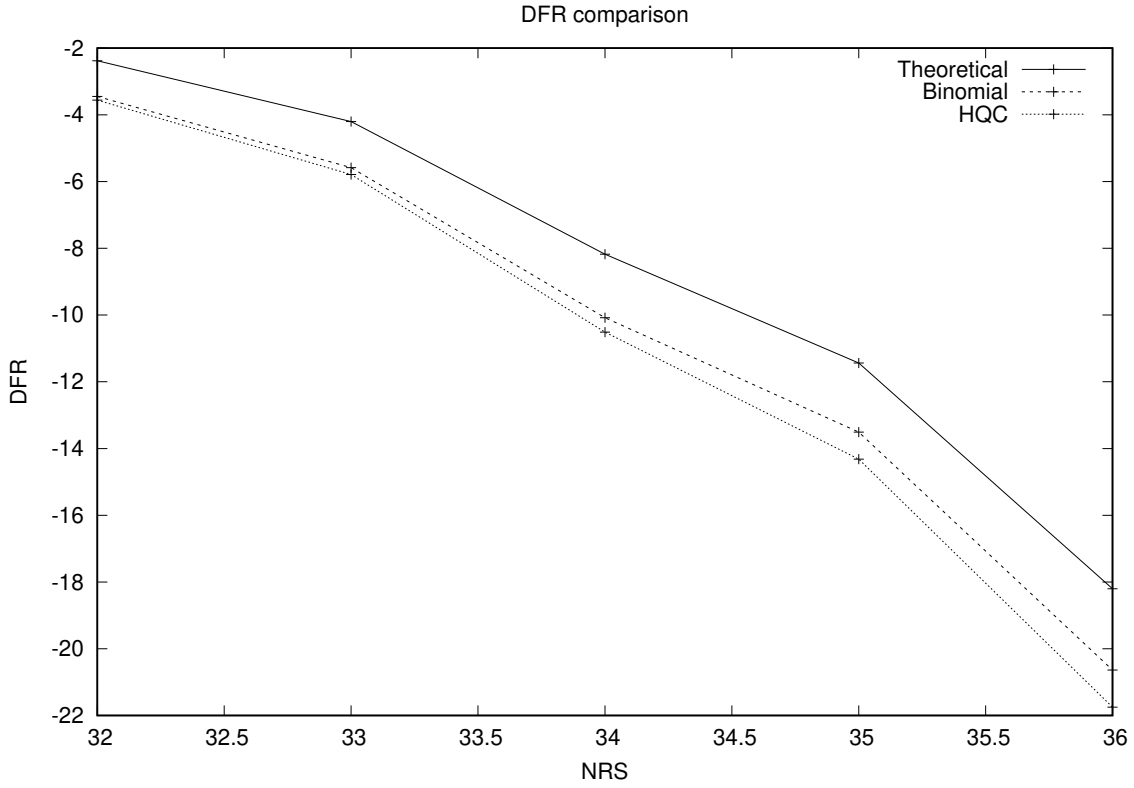

Figure 4: Comparison between the DFR from Theorem 6.1 (Theoretical) and the actual DFR of concatenated codes against approximation by a binary symmetric channel (Binomial) and against HQC error vectors (HQC). Parameters simulated are derived from those of HQC for NIST-1 security level: $\omega = 66$, $\omega_r = \omega_e = 75$, a $[384, 8, 192]$ duplicated Reed-Muller code for internal code and a $[NRS, 16]$ Reed-Solomon code for external code.

6.2 Security proof

Hereafter, we present the approach used to prove the security of HQC. We provide an IND-CPA security proof for HQC-PKE in Section 6.2.1. Then, we prove the IND-CCA2 security of HQC-KEM using the salted Fujisaki-Okamoto transformation with implicit rejection ($\mathsf{SFO}^{\not\perp}$) [14, 22, 18] in Section 6.2.2. For the sake of simplicity, we assume in Sections 6.2.1 and 6.2.2 that the vectors $\mathbf{r}_1, \mathbf{r}_2$ and $\mathbf{e}$ are generated using $\mathsf{SampleFixedWeightVect}_{\$}$ instead of SampleFixedWeightVect (see Section 3.2). Finally, we show in Section 6.2.3 that replacing $\mathsf{SampleFixedWeightVect}_{\$}$ with SampleFixedWeightVect has a minimal impact on the validity of the aforementioned security proofs.

6.2.1 IND-CPA security

In this section we prove the IND-CPA security of the HQC-PKE scheme. Let $b_1 = \mathbf{h}(1) \bmod 2$, $b_2 = \omega + b_1 \times \omega \bmod 2$, $b_3 = \omega_r + b_1 \times \omega_r \bmod 2 = \omega_e + b_1 \times \omega_e \bmod 2$ and $\ell = n - n_1 \times n_2$.

Theorem 6.2. *For any* IND-CPA *adversary* $\mathcal{A}$ *against the* HQC-PKE *scheme, there exists adversaries* $\mathcal{B}_1$ *and* $\mathcal{B}_2$ *such that:*

$$\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{HQC\text{-}PKE}}(\mathcal{A}) \leq \mathsf{Adv}_{\mathsf{2\text{-}DQCSD\text{-}P}}(\mathcal{B}_1) + \mathsf{Adv}_{\mathsf{3\text{-}DQCSD\text{-}PT}}(\mathcal{B}_2). \tag{8}$$

Proof of Theorem 6.2. We build a sequence of games transitioning from the PKE.IND-CPA game to a similar game in which the distribution of $\mathbf{c}_{\mathsf{PKE}}$ is independent of b and show that if the adversary manages to distinguish one from the other, then one can build a simulator breaking the 2-DQCSD-P assumption or the 3-DQCSD-PT assumption. For simplicity, we introduce a small variation in both distinguishers regarding the encryption key $\mathsf{ek}_{\mathsf{PKE}}$. Instead of using as input the seed $\mathsf{seed}_{\mathsf{PKE.ek}}$ used to generate $\mathbf{h}$ and subsequently $\mathbf{H}$, we directly use the matrix $\mathbf{H}$ as input of the distinguishers.

Game_1: This game is the PKE.IND-CPA game following the protocol.

$\mathsf{Game}_1(\lambda)$

1. $\mathsf{seed}_{\mathsf{PKE}} \leftarrow\!\!\$\ \mathcal{B}^{|\mathsf{seed}|}$
2. $(\mathsf{ek}_{\mathsf{PKE}}, \mathsf{dk}_{\mathsf{PKE}}) \leftarrow \mathsf{HQC\text{-}PKE.Keygen}(\mathsf{seed}_{\mathsf{PKE}})$
3. $(\mathbf{m}_0, \mathbf{m}_1) \leftarrow \mathcal{A}_{\mathsf{CHOOSE}}(\mathsf{ek}_{\mathsf{PKE}})$
4. $b \leftarrow\!\!\$\ \{0, 1\}$
5. $\theta \leftarrow\!\!\$\ \mathcal{B}^{|\theta|}, \mathbf{c}_{\mathsf{PKE}} \leftarrow \mathsf{HQC\text{-}PKE.Encrypt}(\mathsf{ek}_{\mathsf{PKE}}, \mathbf{m}_b, \theta)$
6. $b' \leftarrow \mathcal{A}_{\mathsf{GUESS}}(\mathsf{ek}_{\mathsf{PKE}}, \mathbf{c}_{\mathsf{PKE}})$
7. **return** $(b = b')$

Game_2: In this game, we forget the decryption key $\mathsf{dk}_{\mathsf{PKE}}$, take $\mathbf{s}$ at random with parity b_2 and then follow the protocol as in Game_1 for the remaining steps:

$\mathsf{Game}_2(\lambda)$

1. $\mathsf{seed}_{\mathsf{PKE}} \leftarrow\$ \mathcal{B}^{|\mathsf{seed}|}$
2.
   1. $(\mathsf{ek}_{\mathsf{PKE}}, \mathsf{dk}_{\mathsf{PKE}}) \leftarrow \mathsf{HQC\text{-}PKE.Keygen}(\mathsf{seed}_{\mathsf{PKE}})$
   2. $\mathbf{s} \leftarrow\$ \mathbb{F}_{2,b_2}^n$
   3. $(\mathsf{ek}_{\mathsf{PKE}}, \mathsf{dk}_{\mathsf{PKE}}) \leftarrow ((\mathsf{seed}_{\mathsf{PKE.ek}}, \mathbf{s}), \mathbf{0})$
3. $(\mathbf{m}_0, \mathbf{m}_1) \leftarrow \mathcal{A}_{\mathsf{CHOOSE}}(\mathsf{ek}_{\mathsf{PKE}})$
4. $b \leftarrow\$ \{0,1\}$
5. $\theta \leftarrow\$ \mathcal{B}^{|\theta|}, \mathbf{c}_{\mathsf{PKE}} \leftarrow \mathsf{HQC\text{-}PKE.Encrypt}(\mathsf{ek}_{\mathsf{PKE}}, \mathbf{m}_b, \theta)$
6. $b' \leftarrow \mathcal{A}_{\mathsf{GUESS}}(\mathsf{ek}_{\mathsf{PKE}}, \mathbf{c}_{\mathsf{PKE}})$
7. **return** $(b = b')$

Suppose that an adversary $\mathcal{B}_1$ is able to distinguish Game_1 from Game_2 with advantage ϵ for some security parameter λ. Then one can build an algorithm $\mathcal{D}^{\lambda}_{\text{2-DQCSD-P}}$ solving the 2-DQCSD-P problem with the same advantage ϵ.

$\mathcal{D}^{\lambda}_{\text{2-DQCSD-P}}(\mathbf{H}, \mathbf{s})$

1. Compute $\mathbf{h}$ from $\mathbf{H} = (\mathbf{I}_n \ \mathbf{rot}(\mathbf{h}))$
2. Compute $\mathsf{ek}_{\mathsf{PKE}} \leftarrow (\mathbf{h}, \mathbf{s})$
3. Get $(\mathbf{m}_0, \mathbf{m}_1) \leftarrow \mathcal{B}_{1,\mathsf{CHOOSE}}(\mathsf{ek}_{\mathsf{PKE}})$
4. Sample $b \leftarrow\$ \{0,1\}$ and $\theta \leftarrow\$ \mathcal{B}^{|\theta|}$
5. Compute $\mathbf{c}_{\mathsf{PKE}} \leftarrow \mathsf{HQC\text{-}PKE.Encrypt}(\mathsf{ek}_{\mathsf{PKE}}, \mathbf{m}_b, \theta)$
6. Get $b' \leftarrow \mathcal{B}_{1,\mathsf{GUESS}}(\mathsf{ek}_{\mathsf{PKE}}, \mathbf{c}_{\mathsf{PKE}})$
7. If $b' = \mathsf{Game}_1$, output 2-$\mathcal{QCSD}$-$\mathcal{P}(n, \omega, b_1, b_2)$ distribution
8. If $b' = \mathsf{Game}_2$, output $\mathcal{U}(\mathbb{F}_{2,b_1}^{n \times 2n} \times \mathbb{F}_{2,b_2}^n)$ distribution

Note that $\mathsf{ek}_{\mathsf{PKE}}$ is sampled from the 2-$\mathcal{QCSD}$-$\mathcal{P}(n, \omega, b_1, b_2)$ distribution in Game_1 while it is sampled from the uniform distribution over $\mathbb{F}_{2,b_1}^{n \times 2n} \times \mathbb{F}_{2,b_2}^n$ in Game_2 therefore the advantage of $\mathcal{D}^{\lambda}_{\text{2-DQCSD-P}}$ is the same as the advantage of $\mathcal{B}_1$.

Game_3: In this game, instead of picking correctly weighted $\mathbf{r}_1, \mathbf{r}_2, \mathbf{e}$, the simulator picks random vectors in $\mathbb{F}_2^n$ thus generating a random ciphertext with expected parity.

$\mathsf{Game}_3(\lambda)$

1. $\mathsf{seed}_{\mathsf{PKE}} \leftarrow\!\$\ \mathcal{B}^{|\mathsf{seed}|}$
2.
   1. $(\mathsf{ek}_{\mathsf{PKE}}, \mathsf{dk}_{\mathsf{PKE}}) \leftarrow \mathsf{HQC\text{-}PKE.Keygen}(\mathsf{seed}_{\mathsf{PKE}})$
   2. $\mathbf{s} \leftarrow\!\$\ \mathbb{F}_{2,b_2}^n$
   3. $(\mathsf{ek}_{\mathsf{PKE}}, \mathsf{dk}_{\mathsf{PKE}}) \leftarrow ((\mathsf{seed}_{\mathsf{PKE.ek}}, \mathbf{s}), \mathbf{0})$
3. $(\mathbf{m}_0, \mathbf{m}_1) \leftarrow \mathcal{A}_{\mathsf{CHOOSE}}(\mathsf{ek}_{\mathsf{PKE}})$
4. $b \leftarrow\!\$\ \{0,1\}$
5.
   1. $\mathbf{e} \leftarrow\!\$\ \mathbb{F}_2^n, (\mathbf{r}_1, \mathbf{r}_2) \leftarrow\!\$\ \mathbb{F}_{2,\omega_r}^n \times \mathbb{F}_{2,\omega_r}^n$
   2. $\mathbf{u} \leftarrow \mathbf{r}_1 + \mathbf{h}\mathbf{r}_2$
   3. $\mathbf{v} \leftarrow \mathcal{C}.\mathsf{Encode}(\mathbf{m}_b) + \mathsf{Truncate}(\mathbf{s} \cdot \mathbf{r}_2 + \mathbf{e}, \ell)$
   4. $\mathbf{c}_{\mathsf{PKE}} \leftarrow (\mathbf{u}, \mathbf{v})$
6. $b' \leftarrow \mathcal{A}_{\mathsf{GUESS}}(\mathsf{ek}_{\mathsf{PKE}}, \mathbf{c}_{\mathsf{PKE}})$
7. **return** $(b = b')$

Suppose that an adversary $\mathcal{B}_2$ is able to distinguish Game_2 from Game_3 with advantage ϵ for some security parameter λ. Then one can build an algorithm $\mathcal{D}^{\lambda}_{\text{3-DQCSD-PT}}$ solving the 3-DQCSD-PT problem with the same advantage ϵ.

$\mathcal{D}^{\lambda}_{\text{3-DQCSD-PT}}(\mathbf{H}, (\mathbf{u}, \mathbf{v}))$

1. Compute $\mathbf{h}$ and $\mathbf{s}$ from $\mathbf{H} = \begin{pmatrix} \mathbf{I}_n & \mathbf{0} & \mathbf{rot}(\mathbf{h}) \\ \mathbf{0} & \mathbf{I}_n & \mathbf{rot}(\mathbf{s}) \end{pmatrix}$
2. Compute $\mathsf{ek}_{\mathsf{PKE}} \leftarrow (\mathbf{h}, \mathbf{s})$
3. Get $(\mathbf{m}_0, \mathbf{m}_1) \leftarrow \mathcal{B}_{2,\mathsf{CHOOSE}}(\mathsf{ek}_{\mathsf{PKE}})$
4. Sample $b \leftarrow\!\$\ \{0,1\}$
5. Compute $\mathbf{c}_{\mathsf{PKE}} \leftarrow (\mathbf{u}, \mathcal{C}.\mathsf{Encode}(\mathbf{m}_b) + \mathbf{v})$
6. Get $b' \leftarrow \mathcal{B}_{2,\mathsf{GUESS}}(\mathsf{ek}_{\mathsf{PKE}}, \mathbf{c}_{\mathsf{PKE}})$
7. If $b' = \mathsf{Game}_2$, output 3-$\mathcal{QCSD}$-$\mathcal{PT}(n, \omega, b_1, b_2, b_3, \ell)$ distribution
8. If $b' = \mathsf{Game}_3$, output $\mathcal{U}(\mathbb{F}_{2,b_1,b_2}^{2n\times 3n} \times (\mathbb{F}_{2,b_3}^n \times \mathbb{F}_2^{n-\ell}))$ distribution

As we have:

$$\left(\mathbf{u}, \mathbf{v} - \mathcal{C}.\mathsf{Encode}(\mathbf{m}_b)\right)^\top = \left((\mathbf{I}_n\ \mathbf{0}\ \mathbf{rot}(\mathbf{h})) \cdot (\mathbf{r}_1\ \mathbf{e}\ \mathbf{r}_2)^\top, \mathsf{Truncate}((\mathbf{0}\ \mathbf{I}_n\ \mathbf{rot}(\mathbf{s})) \cdot (\mathbf{r}_1\ \mathbf{e}\ \mathbf{r}_2)^\top, \ell)\right)$$

the difference between Game_2 and Game_3 is that in the former

$$\left(\begin{pmatrix} \mathbf{I}_n & \mathbf{0} & \mathbf{rot}(\mathbf{h}) \\ \mathbf{0} & \mathbf{I}_n & \mathbf{rot}(\mathbf{s}) \end{pmatrix}, (\mathbf{u}, \mathbf{v} - \mathcal{C}.\mathsf{Encode}(\mathbf{m}_b))\right)$$

follows the 3-$\mathcal{QCSD}$-$\mathcal{PT}(n, \omega, b_1, b_2, b_3, \ell)$ distribution while in the latter it follows a uniform distribution with parity over $\mathbb{F}_{2,b_1,b_2}^{2n\times 3n} \times (\mathbb{F}_{2,b_3}^n \times \mathbb{F}_2^{n-\ell})$. Hence, the advantage of $\mathcal{D}^{\lambda}_{\text{3-DQCSD-PT}}$ is the same as the advantage of $\mathcal{B}_2$.

One can see that in Game_3, the distribution of $\mathbf{c}_{\mathsf{PKE}}$ is independent of b as $\mathbf{u}$ is computed independently of b and $\mathbf{v}$ is masked by the random vector $\mathbf{s}\cdot\mathbf{r}_2+\mathbf{e}$ hence $\Pr[\mathsf{Game}_3 \Rightarrow 1] = \frac{1}{2}$. As a result, the advantage of any IND-CPA adversary $\mathcal{A}$ against the HQC-PKE scheme is bounded by:

$$\mathsf{Adv}^{\text{IND-CPA}}_{\text{HQC-PKE}}(\mathcal{A}) \leq \mathsf{Adv}_{\text{2-DQCSD-P}}(\mathcal{B}_1) + \mathsf{Adv}_{\text{3-DQCSD-PT}}(\mathcal{B}_2).$$

□

6.2.2 IND-CCA2 security

HQC-KEM is obtained by applying the salted Fujisaki–Okamoto ($\mathsf{SFO}^{\not\perp}$) [14, 22, 18] transform to HQC-PKE. In this section, we discuss the IND-CCA2 security of HQC-KEM.

Definition 6.2.1 (δ-correct PKE [22])**.** *A* $\mathsf{PKE} = (\mathsf{PKE.Keygen}, \mathsf{PKE.Encrypt}, \mathsf{PKE.Decrypt})$ *is* δ*-correct if:*

$$\mathbb{E}\left(\max_{\mathbf{m}\in\mathcal{M}} \Pr\left[\mathsf{PKE.Decrypt}(\mathsf{dk}_{\mathsf{PKE}}, \mathbf{c}_{\mathsf{PKE}}) \neq \mathbf{m} \mid \mathbf{c}_{\mathsf{PKE}} \leftarrow \mathsf{PKE.Encrypt}(\mathsf{ek}_{\mathsf{PKE}}, \mathbf{m})\right]\right) \leq \delta. \quad (9)$$

where the expectation is taken over $(\mathsf{ek}_{\mathsf{PKE}}, \mathsf{dk}_{\mathsf{PKE}}) \leftarrow \mathsf{PKE.Keygen}(\mathsf{param})$.

Definition 6.2.2 (δ-correct KEM [22])**.** *A* $\mathsf{KEM} = (\mathsf{KEM.Keygen}, \mathsf{KEM.Encaps}, \mathsf{KEM.Decaps})$ *is* δ*-correct if:*

$$\Pr = \left[\mathsf{KEM.Decaps}(\mathsf{dk}_{\mathsf{KEM}}, \mathbf{c}_{\mathsf{KEM}}) \neq K \;\middle|\; \begin{array}{l}(\mathsf{ek}_{\mathsf{KEM}}, \mathsf{dk}_{\mathsf{KEM}}) \leftarrow \mathsf{KEM.Keygen}();\\ (K, \mathbf{c}_{\mathsf{KEM}}) \leftarrow \mathsf{KEM.Encaps}(\mathsf{ek}_{\mathsf{KEM}})\end{array}\right] \leq \delta. \quad (10)$$

In HQC-PKE the failure to decrypt a ciphertext $\mathbf{c}_{\mathsf{PKE}}$ occurs if and only if

$$\omega\left(\mathbf{x}\cdot\mathbf{r}_2 - \mathbf{r}_1\cdot\mathbf{y} + \mathbf{e}\right) > \Delta.$$

Note that the aforementioned equation does not depend on the message $\mathbf{m}$. Therefore, the probability in Equation 9 simplifies to

$$\Pr\left[\mathsf{PKE.Decrypt}(\mathsf{dk}_{\mathsf{PKE}}, \mathbf{c}_{\mathsf{PKE}}) \neq \mathbf{m} \mid \mathbf{c}_{\mathsf{PKE}} \leftarrow \mathsf{PKE.Encrypt}(\mathsf{ek}_{\mathsf{PKE}}, \mathbf{m})\right] \leq \delta. \quad (11)$$

This probability is equivalent to the probability that is analyzed in section 6.1 namely:

$$\Pr\left[\omega\left(\mathbf{x}\cdot\mathbf{r}_2 - \mathbf{r}_1\cdot\mathbf{y} + \mathbf{e}\right) > \Delta \;\middle|\; \begin{array}{l}\mathsf{ctx}_{\mathsf{PKE.dk}} \leftarrow \mathsf{XOF.Init}(\mathsf{seed}_{\mathsf{PKE.dk}});\\ (\mathsf{ctx}_{\mathsf{PKE.dk}}, \mathbf{y}) \leftarrow \mathsf{SampleFixedWeightVect}_{\$}(\mathsf{ctx}_{\mathsf{PKE.dk}}, \mathcal{R}_{\omega});\\ (\mathsf{ctx}_{\mathsf{PKE.dk}}, \mathbf{x}) \leftarrow \mathsf{SampleFixedWeightVect}_{\$}(\mathsf{ctx}_{\mathsf{PKE.dk}}, \mathcal{R}_{\omega});\\ \mathsf{ctx}_{\theta} \leftarrow \mathsf{XOF.Init}(\theta);\\ (\mathsf{ctx}_{\theta}, \mathbf{r}_2) \leftarrow \mathsf{SampleFixedWeightVect}_{\$}(\mathsf{ctx}_{\theta}, \mathcal{R}_{\omega_r});\\ (\mathsf{ctx}_{\theta}, \mathbf{e}) \leftarrow \mathsf{SampleFixedWeightVect}_{\$}(\mathsf{ctx}_{\theta}, \mathcal{R}_{\omega_e});\\ (\mathsf{ctx}_{\theta}, \mathbf{r}_1) \leftarrow \mathsf{SampleFixedWeightVect}_{\$}(\mathsf{ctx}_{\theta}, \mathcal{R}_{\omega_r})\end{array}\right] \leq \delta. \quad (12)$$

The next theorem shows that HQC-KEM achieves IND-CCA2 security in the random-oracle model, with its concrete advantage bound obtained by combining the estimates of Theorem 13 from [18] and Theorem 6.2.

Theorem 6.3. *If* HQC-PKE *is* δ *correct, for any* IND-CCA2 *adversary* $\mathcal{A}$ *against the* HQC-KEM *scheme issuing at most* q_{RO} *queries to* G *and* q_D *queries to the* HQC-KEM.Decaps *oracle, there exists adversaries* $\mathcal{B}_1$ *and* $\mathcal{B}_2$ *such that:*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{IND\text{-}CCA2}}_{\mathsf{HQC\text{-}KEM}}(\mathcal{A}) \leq \frac{1}{2^{|k|}\cdot 2^{|\mathsf{salt}|}} + \frac{3q_{RO}}{2^{|k|}} + (q_{RO}+q_D)\cdot\delta \\ + 2\cdot\left(\mathsf{Adv}_{\text{2-DQCSD-P}}(\mathcal{B}_1) + \mathsf{Adv}_{\text{3-DQCSD-PT}}(\mathcal{B}_2)\right).\end{aligned} \tag{13}$$

6.2.3 Security proof with non uniform randomness sampling

In this section, we show that replacing the uniform sampling from $\mathsf{SampleFixedWeightVect}_{\$}$ with the biased distribution from SampleFixedWeightVect has a minimal impact on the IND-CCA2 security of HQC-KEM following the approach described in [36].

Proposition 6.2.1 ([36])**.** *When* $x \leftarrow \mathsf{randbits}(B, \mathsf{prng})$ *behaves as a random oracle which yields uniformly distributed integers* $0 \leq x < 2^B$ *for any integer* $B > 0$*, one has:*

$$\prod_{i=0}^{w-1}\left(1-\frac{n_i}{2^B}\right) = \tau^{\omega}_{min} \leq \frac{\Pr\left[\mathbf{e} \mid \mathbf{e} \leftarrow \mathsf{SampleFixedWeightVect}(\mathsf{ctx}_{\mathbf{e}}, \mathcal{R}_\omega)\right]}{\Pr\left[\mathbf{e} \mid \mathbf{e} \leftarrow \mathsf{SampleFixedWeightVect}_{\$}(\mathsf{ctx}_{\mathbf{e}}, \mathcal{R}_\omega)\right]} \leq \tau^{\omega}_{max} = \prod_{i=0}^{w-1}\left(1+\frac{(n-i)-n_i}{2^B}\right) \tag{14}$$

where $n_i = 2^B \bmod (n-i)$ *for all* i*,* $0 \leq i < w$*.*

For HQC-KEM, $(\tau^{\omega}_{min}, \tau^{\omega}_{max})$ and $(\tau^{\omega_r}_{min}, \tau^{\omega_r}_{max})$ are very close to 1 as shown in Table 12.

Security	n	w	τ^{ω}_{min}	τ^{ω}_{max}	$\omega_r = \omega_e$	$\tau^{\omega_r}_{min}$	$\tau^{\omega_r}_{max}$
NIST-1	17,669	66	0.99987	1.00014	75	0.99985	1.00015
NIST-3	35,851	100	0.99958	1.00041	114	0.99952	1.00047
NIST-5	57,637	131	0.99913	1.00089	149	0.99901	1.00101

Table 12: Bias between the uniform distribution and the output of Algorithm SampleFixedWeightVect (using $B = 32$) for the vectors of weight ω and $\omega_r = \omega_e$.

The following lemma (Equation 5 from [36]) shows that the advantage of any adversary when a vector $\mathbf{e}$ of weight w is sampled following Algorithm SampleFixedWeightVect instead of the uniform distribution cannot increase by a factor larger than τ^{ω}_{max}.

Lemma 6.4. *For any real-valued random variable* $V : \mathcal{R}_\omega \to \mathbb{R}$*, one has:*

$$\sum_{\mathbf{e}\in\mathcal{R}_\omega} \Pr\left[\mathbf{e} \mid \mathbf{e} \leftarrow \mathsf{SampleFixedWeightVect}(\mathsf{ctx}_{\mathbf{e}}, \mathcal{R}_\omega)\right] V(\mathbf{e}) \leq \tau^{\omega}_{max} \cdot \sum_{\mathbf{e}\in\mathcal{R}_\omega} \Pr\left[\mathbf{e} \mid \mathbf{e} \leftarrow \mathsf{SampleFixedWeightVect}_{\$}(\mathsf{ctx}_{\mathbf{e}}, \mathcal{R}_\omega)\right] V(\mathbf{e}).$$

Impact of the security proof. Following [36], we explain how to modify the Equation 13 to take into account the non uniform randomness sampling:

- The first two terms $\frac{1}{2^{|k|}.2^{|\mathsf{salt}|}}$ and $\frac{3q_{RO}}{2^{|k|}}$ remain unchanged since they are independent of the output of the sampler ;
- The third term $(q_{RO}+q_D)\cdot\delta$ is related to the δ-correctness of the scheme. When $(\mathbf{x},\mathbf{y})$ and $(\mathbf{r}_1,\mathbf{r}_2,\mathbf{e})$ are sampled using SampleFixedWeightVect rather than the uniform distribution, δ must be such that:

$$\Pr\left[\omega\left(\mathbf{x}\cdot\mathbf{r}_2-\mathbf{r}_1\cdot\mathbf{y}+\mathbf{e}\right)>\Delta \;\middle|\; \begin{array}{l}\mathsf{ctx}_{\mathsf{PKE.dk}}\leftarrow\mathsf{XOF.Init}(\mathsf{seed}_{\mathsf{PKE.dk}});\\(\mathsf{ctx}_{\mathsf{PKE.dk}},\mathbf{y})\leftarrow\mathsf{SampleFixedWeightVect}_{\$}(\mathsf{ctx}_{\mathsf{PKE.dk}},\mathcal{R}_\omega);\\(\mathsf{ctx}_{\mathsf{PKE.dk}},\mathbf{x})\leftarrow\mathsf{SampleFixedWeightVect}_{\$}(\mathsf{ctx}_{\mathsf{PKE.dk}},\mathcal{R}_\omega);\\\mathsf{ctx}_\theta\leftarrow\mathsf{XOF.Init}(\theta);\\(\mathsf{ctx}_\theta,\mathbf{r}_2)\leftarrow\mathsf{SampleFixedWeightVect}(\mathsf{ctx}_\theta,\mathcal{R}_{\omega_r});\\(\mathsf{ctx}_\theta,\mathbf{e})\leftarrow\mathsf{SampleFixedWeightVect}(\mathsf{ctx}_\theta,\mathcal{R}_{\omega_e});\\(\mathsf{ctx}_\theta,\mathbf{r}_1)\leftarrow\mathsf{SampleFixedWeightVect}(\mathsf{ctx}_\theta,\mathcal{R}_{\omega_r})\end{array}\right]\leq\delta.$$

  Using Lemma 6.4, the above probability increases by at most $(\tau_{max}^{\omega_r})^3$;
- The fourth term $2\cdot(\mathsf{Adv}_{\text{2-DQCSD-P}}(\mathcal{B}_1)+\mathsf{Adv}_{\text{3-DQCSD-PT}}(\mathcal{B}_2))$ must be adjusted to account for a slight change in the sequence of games. One can add a game Game_0 where the small Hamming weight vectors are sampled according to the biased distribution rather than uniformly at random before resuming with Game_1. Using Lemma 6.4, the advantage related to this term increases by a factor at most $(\tau_{max}^{\omega_r})^3$.

6.3 Known attacks

Attacks against Syndrome Decoding. 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 [33] and improved later by Stern [37], then Dumer [12]. Recent works [29, 5, 30] suggest a complexity of order $2^{c\omega(1+\mathsf{negl}(1))}$, for some constant c. A particular work focusing on the regime $\omega=\mathsf{negl}(n)$ confirms this formula, with a close dependence between c and the rate k/n of the code being used [11].

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 [35] 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 [21, 27, 35], 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 $x^{n-1}+x^{n-2}+\ldots+x+1$, 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.

Security of the 2-DQCSD-P and 3-DQCSD-PT problems. Concerning the security of the 2-DQCSD-P problem, there is one security bit lost in the reduction to the 2-DQCSD

problem. Regarding the security of the 3-DQCSD-PT problem, whenever the number of truncated positions is very small compared to the block length n, the impact on the security is negligible with respect to the 3-DQCSD problem since the best attack is the ISD attack. Moreover since the truncation breaks the quasi-cyclicity, it also weakens the advantage of quasi-cyclicity for the attacker.

Choice of parameters. We proposed different sets of parameters in Section 4 that fit security levels 1, 3 and 5, as defined by NIST. The quantum-safe security is obtained by dividing the security bits by two (taking the square root of the complexity) [8]. Best known attacks include the works from [10, 9, 13, 29, 5, 30] and for quantum attacks, the work of [8]. In the setting $\omega = \mathcal{O}(\sqrt{n})$, best known attacks have a complexity in $2^{-t\ln(1-R)(1+o(1))}$ where $t = \mathcal{O}(\omega)$ and R is the rate of the code [11]. In our configuration, we have $t = 2\omega$ and $R = 1/2$ for the reduction to the 2-DQCSD-P problem, and $t = 3\omega_r$ and $R = 1/3$ for the 3-DQCSD-PT problem. By taking into account the DOOM attack [35], 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 $\log\left(\binom{n}{\omega}^2/\binom{2n}{2\omega}\right)$ and $\log\left(\binom{n}{\omega_r}^3/\binom{3n}{3\omega_r}\right)$ for the 2-DQCSD-P and 3-DQCSD-PT problems.

7 Advantages and Limitations

7.1 Advantages

The HQC scheme features several advantages:

- Immunity against attacks aiming at recovering the hidden structure of the code being used contrarily to many code-based cryptosystems;
- A security reduction to a well-understood problem on coding theory (the Quasi-Cyclic Syndrome Decoding problem) along with a precise estimate of its decoding failure rate;
- Efficient implementations based on classical decoding algorithms;
- Reasonably small ciphertext and key sizes.

7.2 Limitations

The HQC scheme has also some limitations:

- HQC-PKE has a low encryption rate hence encrypting large plaintexts require to increase the parameters;
- In contrast with lattices and the Ring Learning With Errors problem, code-based cryptography does not benefit from search to decision reduction for structured codes;
- In comparison with some lattice-based KEM such as ML-KEM, HQC features larger ciphertext and key sizes as well as less efficient implementations.

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.

[2] Francesco Antognazza, Alessandro Barenghi, Gerardo Pelosi, and Ruggero Susella. A High Efficiency Hardware Design for the Post-Quantum KEM HQC. In *2024 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 431–441. IEEE, 2024.

[3] 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, Berlin, Heidelberg, August 2007.

[4] Nicolas Aragon, Philippe Gaborit, and Gilles Zémor. HQC-RMRS, an instantiation of the HQC encryption framework with a more efficient auxiliary error-correcting code. https://arxiv.org/abs/2005.10741.

[5] 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, Berlin, Heidelberg, April 2012.

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

[7] 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.

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

[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.

[10] 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/.

[11] 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.

[12] 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.

[13] 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, Berlin, Heidelberg, December 2009.

[14] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 537–554. Springer, Berlin, Heidelberg, August 1999.

[15] 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.

[16] 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.

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

[18] Lewis Glabush, Kathrin Hövelmanns, and Douglas Stebila. Tight Multi-challenge Security Reductions for Key Encapsulation Mechanisms. *Cryptology ePrint Archive, Paper 2025/343*, 2025. https://eprint.iacr.org/2025/343.

[19] 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.

[20] Qian Guo, Clemens Hlauschek, Thomas Johansson, Norman Lahr, Alexander Nilsson, and Robin Leander Schröder. Don't reject this: Key-recovery timing attacks due to rejection-sampling in HQC and BIKE. *IACR TCHES*, 2022(3):223–263, 2022.

[21] 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.

[22] 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.

[23] 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.

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

[25] 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.

[26] 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.

[27] 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.

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

[29] 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.

[30] 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.

[31] 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.

[32] Ray Perlner. Security strength categories for Code Based Crypto. Public comment on NIST pqc-forum mailing list, 2021.

[33] 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/.

[34] Markku-Juhani O. Saarinen. IND-CCA2 issue in HQC. Public comment on NIST pqc-forum mailing list, 2025.

[35] 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.

[36] Nicolas Sendrier. Secure Sampling of Constant Weight Words – Application to BIKE. Cryptology ePrint Archive, Report 2021/1631, 2021. https://eprint.iacr.org/2021/1631.

[37] 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.