Hamming Quasi-Cyclic (HQC)

Version 5.1 (2026/11/09)

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-Brockmann (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).

Post-Selection

Version 5.1 (2026/10/12) and **(2026/11/09)**

- We have refactored this document to improve its readability, notably by adding detailed figures describing the sampling, encoding and decoding algorithms.
- We have improved our DFR analysis (Section 6.1) providing a more precise evaluation of the DFR following [11, 35, 2]. We have also updated our parameters increasing ω by 1 for HQC-1, HQC-3 and HQC-5, and ω_r by 1 for HQC-1 and HQC-5. This parameter update leverages the new DFR analysis to increase the scheme's security and compensates the minor technical cryptanalysis improvements from [14].
- We have updated HQC-PKE and HQC-KEM to use the compressed $\mathsf{dk}_{\mathsf{KEM}}$ format $\mathsf{dk}_{\mathsf{KEM}} = (\mathsf{seed}_{\mathsf{KEM}})$ as the default format for $\mathsf{dk}_{\mathsf{KEM}}$. This introduces a performance overhead in HQC-KEM.Decaps which can be mitigated by caching expanded keys.
- We have updated the irreducible polynomial used to compute arithmetic over $\mathbb{F}_{2^8}$ to $p(x) = x^8 + x^4 + x^3 + x + 1$.

Version 5.0 (2025/08/22)

- We have refactored this document to improve its readability, notably by 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 [39], using the salted $\mathsf{SFO}^{\not\perp}_m$ transform instead of the $\mathsf{FO}^{\not\perp}$ one [22] 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 $\mathsf{dk}_{\mathsf{KEM}} = (\mathsf{seed}_{\mathsf{KEM}})$ has been included.

- We have updated fixed weight vectors sampling in HQC-PKE.Keygen by using a sampler that outputs uniformly distributed vectors but relies on rejection sampling ($\mathsf{SampleFixedWeightVect}_{\$}$) instead of a sampler that doesn't rely on rejection sampling but outputs 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-Brockmann, Nicolas Sendrier, Jean-Pierre Tillich, and Valentin Vasseur have joined the HQC team.

Round 4

Version 4.3 (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 [3].
- 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.

Version 4.1 (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 weight vectors non-uniformly, yet close to uniform, has a negligible effect on HQC security following [41].

Version 4.0 (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 countermeasure to multi-ciphertext attacks [37].
- We have updated constant-weight words sampling using the technique from [41] as a countermeasure to the timing attack from [24].

Round 3

Version 3.0 (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 instead of 256) for levels 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 us to lower our theoretical bound for the DFR and improve our parameters.
- Jérôme Lacan and Arnaud Dion have joined the HQC team.

Round 2

Version 2.3 (2020/05/04)

- We have provided a more precise analysis of the modelization of the error distribution which allows 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.

Version 2.0 (2019/04/10)

- We have introduced the DQCSD problems with parity as a countermeasure 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 **6**

2 Preliminaries **6**
2.1 Notations . . . 6
2.2 Coding theory . . . 10
2.3 Security assumptions . . . 11
2.4 Security definitions . . . 13

3 Specifications **15**
3.1 XOF and Hash functions . . . 15
3.2 Auxiliary algorithms . . . 15
3.3 Vector sampling . . . 16
3.4 Vector multiplication . . . 20
3.5 Concatenated Reed-Muller and Reed-Solomon codes . . . 23
3.6 HQC-PKE . . . 38
3.7 HQC-KEM . . . 41

4 Parameters and Sizes **44**
4.1 Parameter sets . . . 44
4.2 Ciphertext and key sizes . . . 44

5 Performance Analysis **45**
5.1 Reference implementation . . . 45
5.2 Optimized implementation . . . 45
5.3 Known Answer Test values . . . 46

6 Security Analysis **46**
6.1 Decoding Failure Rate analysis . . . 46
6.2 Security proof . . . 61
6.3 Known attacks . . . 66

7 Advantages and Limitations **68**
7.1 Advantages . . . 68
7.2 Limitations . . . 68

References **69**

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 [26]. 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

Rings and fields. Let $(\mathbb{Z}, +, \times)$ denote the ring of integers. Let $\mathbb{F}_2$ denote the binary finite field and let $\mathbb{F}_{2^8} = (\mathbb{F}_2[X]/\langle p(X)\rangle, \oplus, \otimes)$ denote the field defined using the irreducible polynomial $p(X) = X^8 + X^4 + X^3 + X + 1$ and let alpha be a primitive element of $\mathbb{F}_{2^8}$. In addition, if $y \in \mathbb{F}_{2^8}$, we denote by y^{-1} the inverse of y. 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 seen as row vectors or polynomials in the ring $\mathcal{R} = (\mathbb{F}_2[X]/(X^n - 1), +, \cdot)$ where it will be clear from context whether $+$ denotes the addition in $\mathbb{Z}$ or $\mathcal{R}$. 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:

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

Hamming weight. Let $\omega(\cdot)$ denote the Hamming weight of a vector *i.e.* by the number of its non-zero coordinates. For a 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.

Byte representation. Let $\mathcal{B}$ denote the set of unsigned 8-bits integers $\{0, \ldots, 255\}$ and let $\mathcal{B}_0$ denote the byte whose value is equal to zero. Let $(\oplus, \wedge, \vee)$ denote respectively XOR, AND and OR operations over bytes and let $(\ll, \gg)$ denote respectively left and right logical shifts. A byte b is written as bits $(b_7 \ldots b_0)$ where b_0 is the least significant bit. A byte array $(a[0], ..., a[l-1])$ is little-endian when interpreted as an integer or polynomial coefficient array where index 0 is the least significant byte.

Element representation. Elements of $\mathcal{R}$ are represented as byte arrays. All arrays have indices that begin at zero. Let $\mathbf{v}[i]$ denote the entry at index i in the array $\mathbf{v}$. Let $\mathbf{v}[k:m]$ be the sub-array $(\mathbf{v}[k], \mathbf{v}[k+1], \ldots, \mathbf{v}[m-1])$ of the byte array $\mathbf{v}$. The field element $\sum_{i=0}^{7} b_i X^i \in \mathbb{F}_{2^8}$ is represented by the byte $b = \sum_{i=0}^{7} b_i 2^i$.

Symbols, parameters and constants. Mathematical symbols used in this document are summarized in Table 1. Parameters and constants presented in Table 2 are considered public parameters and are implicitly available to any algorithm presented in this document.

Interpretation of the pseudocode. Algorithms don't modify their input variables unless they are also outputs variables. The symbol "$\leftarrow$" is used to assign values to variables and tuple assignments are assumed to be simultaneous. Array assignment are deep copies and slice assignment in array copies the source slice before writing the destination. A loop whose range is not defined has zero iterations.

Symbol	Description
$\mathbb{N}$	Set of natural numbers
$\mathbb{Z}$	Set of integers
$+$, $-$	Addition and subtraction in $\mathbb{Z}$ whenever operands are integers
$\times$	Multiplication in $\mathbb{Z}$
$\lvert x \rvert$	Absolute value if x is an integer and byte size if x is a byte array
$\mathbb{F}_2$	Binary field
$\mathcal{V}$	Vector space of size n over $\mathbb{F}_2$
$\mathcal{R} = \mathbb{F}_2[X]/(X^n - 1)$	Polynomial quotient ring
$+$, $-$	Addition and subtraction in $\mathcal{R}$ whenever operands are polynomials
$\cdot$	Multiplication in $\mathcal{R}$
$\omega(\boldsymbol{x})$	Hamming weight of x
$\mathcal{R}_\omega$	Set of elements of $\mathcal{R}$ with Hamming weight equal to ω
$\mathbb{F}_{2^8} = \mathbb{F}_2[X]/\langle p(X) \rangle$	Field defined using the irreducible polynomial $p(X)$ where $p(X) = X^8 + X^4 + X^3 + X + 1$
alpha	Primitive element of $\mathbb{F}_{2^8}$
$\oplus$	Addition in $\mathbb{F}_{2^8}$
$\otimes$	Multiplication in $\mathbb{F}_{2^8}$
x^a, x^{-1}	Exponentiation of x to the power a and inverse of x computed in $\mathbb{F}_{2^8}$
$\mathcal{B}$	Set of unsigned 8-bits integers $\{0, \ldots, 255\}$
$\mathcal{B}_0$	Byte whose value is equal to zero
$\oplus$	Bitwise XOR operation
$\wedge$	Bitwise AND operation
$\vee$	Bitwise OR operation
$\ll$	Logical left bit shift
$\gg$	Logical right bit shift
$x \lvert\lvert y$	Concatenation of x and y
S^ℓ	ℓ-tuple over the set S
$\boldsymbol{v}^\ell, \boldsymbol{M}^{\ell_1 \times \ell_2}$	Vector of size ℓ and matrix of size $\ell_1 \times \ell_2$
$\boldsymbol{x}[i]$	Entry at index i in the array x and coefficient of X^i if $\boldsymbol{x}$ is a polynomial
$\boldsymbol{x}[i:j]$	Sub-array $(\boldsymbol{v}[i], \boldsymbol{v}[i+1], \ldots, \boldsymbol{v}[j-1])$
$x \leftarrow\!\!\$\, S$	Variable x is sampled uniformly at random in S
$\perp$	Absence of value or failure

Table 1: Mathematical symbols

Parameter	Description	Value
n	Length of vectors in $\mathcal{V}$	Table 8
$n_1 n_2'$	Length of the concatenated code $\mathcal{C}$	
k	Dimension of the concatenated code $\mathcal{C}$	
ω	Hamming weight of $\boldsymbol{x}$ and $\boldsymbol{y}$	
ω_r	Hamming weight of $\mathbf{r}_1$ and $\mathbf{r}_2$	
ω_e	Hamming weight of $\boldsymbol{e}$	
ℓ	Number of truncated bits	$n - n_1 n_2'$
n_1	Length of the shortened Reed-Solomon (sRS) code	Table 6
k_1	Dimension of the sRS code	
d_1	Minimum distance of the sRS code	
t_1	Error correction capacity of the sRS code	
alpha	Primitive element of $\mathbb{F}_{2^8}$	0x03
rs_generator_polynomial	Generator polynomial of the sRS code	Section 3.5.2
n_2'	Length of the duplicated Reed-Muller (dRM) code	Table 7
n_2	Length of the dRM code in bytes	
k_2	Dimension of the dRM code	
d_2	Minimum distance of the dRM code	
rm_multiplicity	Multiplicity of the duplicated dRM code	
n_3	Length of the RM$(1, 7)$ code	
k_3	Dimension of the RM$(1, 7)$ code	
d_3	Minimum distance of the RM$(1, 7)$ code	
rm_generator_matrix	Generator matrix of the RM$(1, 7)$ code	Section 3.5.3
hadamard_stages	Number of stages in the Hadamard transform	7
hadamard_register_size	Size of the registers in the Hadamard transform	64
G_domain_separator	Domain separator for the hash function G	0x00
H_domain_separator	Domain separator for the hash function H	0x01
I_domain_separator	Domain separator for the hash function I	0x02
J_domain_separator	Domain separator for the hash function J	0x03
mask_n	Mask used in the SampleVect algorithm	Table 4
rejection_threshold_n	Rejection threshold used in SampleFixedWeightVect$_\$$	

Table 2: Parameters and constants

2.2 Coding theory

In the following, we provide basic definitions and properties about coding theory and refer the reader to [27] 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 [20].

Definition 2.2.6 (Quasi-Cyclic Codes [36]). *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, $\ell = 1 \dots s-1$, 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 [8]. This problem can also be seen as the Learning Parity with Noise (LPN) problem with a fixed number of samples [4]. The DSD problem has been shown to be polynomially equivalent to its search version in [4]. 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 [21].

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 $\mathbf{x} = (\mathbf{x}_0, \dots, \mathbf{x}_{s-1}) \leftarrow\$ \mathbb{F}_2^{sn}$ 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-DQCSD(n, ω) 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 [40] 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 $\mathbb{F}_{2,b_1}^n = \{\mathbf{h} \in \mathbb{F}_2^n \text{ s.t. } \mathbf{h}(1) = b_1 \bmod 2\}$ *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 [23, 29, 30][1].

Definition 2.3.6 (2-$\mathcal{QCSD}$-$\mathcal{P}$ Distribution)**.** *Let n, ω, b_1 be positive integers and let $b_2 = \omega + b_1 \times \omega \bmod 2$. 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 $b_2 = \omega + b_1 \times \omega \bmod 2$. 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*

[1] The authors chose to use a parity version of the 2-DQCSD problem rather than a variable weight version as suggested in [30] for efficiency considerations.

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* $\mathbb{F}_{2,b_1}^{n\times 2n} \times \mathbb{F}_{2,b_2}^{n}$.

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 length $n_1 n_2'$ bits 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* $b_3 = \omega + b_1 \times \omega \bmod 2$. *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* $b_3 = \omega + b_1 \times \omega \bmod 2$. *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* PKE = (Keygen, Encrypt, 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:*

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

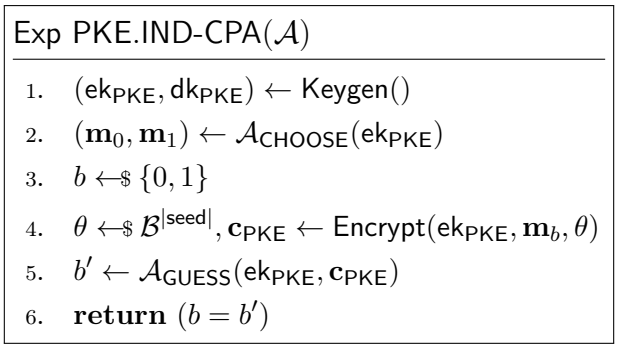


Figure 1: Experiment for the IND-CPA security of a PKE

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:*

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

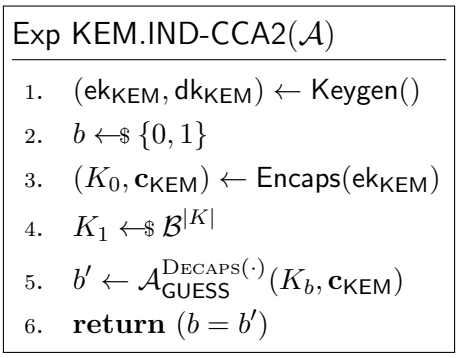


Figure 2: Experiment for the IND-CCA2 security of a KEM

3 Specifications

3.1 XOF and Hash functions

The extendable-output function XOF is instantiated using SHAKE256. The hash functions G and I are instantiated using SHA3-512 while the hash functions H and J are instantiated using SHA3-256. Each hash function uses its own domain separator as described in Table 3.

Function	Instantiation
ctx $\leftarrow$ XOF.Init()	ctx $\leftarrow$ SHAKE256.Init()
ctx $\leftarrow$ XOF.Absorb(ctx, str, |str|)	SHAKE256.Absorb(ctx, str, |str|)
(ctx, out) $\leftarrow$ XOF.GetBytes(ctx, out, |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 3: Instantiation of XOF and Hash functions

3.2 Auxiliary algorithms

ByteToBit(byte, index)

Input: byte $\in \mathcal{B}$, index $\in [0, 7]$
Output: Bit $b \in \{0, 1\}$

1. **return** $b \leftarrow (\text{byte} \gg \text{index}) \wedge 1$

Truncate($\mathbf{v}, \ell$)

Input: Vector $\mathbf{v} \in \mathcal{B}^{\lceil n/8 \rceil}$
Output: Vector $\mathbf{v} \in \mathcal{B}^{n_1 n_2}$

1. **return** $\mathbf{v}[0 : (n - \ell)/8]$

3.3 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 and unused bits within the vector representation are set to zero using mask_n defined in Table 4.

SampleVect(ctx)

Input: XOF context ctx

Output: XOF context ctx, vector $\mathbf{v} \in \mathcal{R}$

1. size $\leftarrow \lceil n/8 \rceil$
2. $\mathbf{v} \leftarrow \mathcal{B}_0^{\text{size}}$
3. (ctx, $\mathbf{v}$) $\leftarrow$ XOF.GetBytes(ctx, $\mathbf{v}$, size)
4. $\mathbf{v}$[size $-$ 1] $\leftarrow$ $\mathbf{v}$[size $-$ 1] $\wedge$ mask_n
5. **return** $\mathbf{v}$

Sampling random fixed weight vectors. Sampling vectors from $\mathbb{F}_2^n$ of given Hamming weight is done using the SampleFixedWeightVect$_\$$ and SampleFixedWeightVect functions. SampleFixedWeightVect$_\$$ outputs uniformly distributed fixed weight vectors but relies on rejection sampling while SampleFixedWeightVect does not 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. SampleFixedWeightVect$_\$$ and SampleFixedWeightVect use the SampleSupport1 and SampleSupport2 algorithms to sample the support of the vector then transform it into the fixed weight vector using the ComputeVectorFromSupport algorithm. The SampleSupport1 uses the rejection_threshold_n defined in Table 4 while the SampleSupport2 function follows Algorithm 5 from [41].

SampleFixedWeightVect$_{\$}(\mathsf{ctx}, \omega)$

Input: XOF context ctx, weight ω

Output: XOF context ctx, vector $\mathbf{v} \in \mathcal{R}_\omega$

1. $(\mathsf{ctx}, \mathsf{support}) \leftarrow \mathsf{SampleSupport1}(\mathsf{ctx}, \omega)$
2. $\mathbf{v} \leftarrow \mathsf{ComputeVectorFromSupport}(\mathsf{support}, \omega)$
3. **return** $(\mathsf{ctx}, \mathbf{v})$

SampleFixedWeightVect(ctx, ω)

Input: XOF context ctx, weight ω

Output: XOF context ctx, vector $\mathbf{v} \in \mathcal{R}_\omega$

1. $(\mathsf{ctx}, \mathsf{support}) \leftarrow \mathsf{SampleSupport2}(\mathsf{ctx}, \omega)$
2. $\mathbf{v} \leftarrow \mathsf{ComputeVectorFromSupport}(\mathsf{support}, \omega)$
3. **return** $(\mathsf{ctx}, \mathbf{v})$

SampleSupport1(ctx, ω)

Input: XOF context ctx, weight ω

Output: XOF context ctx, support $\in (\mathcal{B}^2)^\omega$

1. support $\leftarrow (\mathcal{B}_0^2)^\omega$
2. candidate $\leftarrow \mathcal{B}_0^3$
3. $i \leftarrow 0$

4. **while** ($i < \omega$) **do**
   ▷ Sample next candidate position
5. (ctx, candidate) $\leftarrow$ XOF.GetBytes(ctx, candidate, 3)

   ▷ Check if candidate position is valid
6. **if** (candidate $\geq$ rejection_threshold_n) **then**
7. **continue**
8. **end**
9. candidate $\leftarrow$ candidate mod n

   ▷ Check if candidate position is available
10. is_position_available $= 1$
11. **for** ($j \leftarrow 0$; $j < i$; j++) **do**
12. **if** (candidate $=$ support$[j]$) **then**
13. is_position_available $= 0$
14. **break**
15. **end**
16. **end**

17. **if** (is_position_available $= 1$) **then**
18. support$[i] \leftarrow$ candidate
19. $i \leftarrow i + 1$
20. **end**
21. **end**

22. **return** (ctx, support)

SampleSupport2(ctx, ω)

Input: XOF context ctx, weight ω

Output: XOF context ctx, support $\in (\mathcal{B}^2)^\omega$

1. support $\leftarrow (\mathcal{B}_0^2)^\omega$
2. rnd_bytes $\leftarrow (\mathcal{B}_0^4)^\omega$
3. (ctx, rnd_bytes) $\leftarrow$ XOF.GetBytes(ctx, rnd_bytes, $4 \times \omega$)
4. **for** ($i \leftarrow 0$; $i < \omega$; i++) **do**
   ▷ Compute rnd_bytes[i] mod $(n - i)$ using Lemire sampling
5. support[i] $\leftarrow i + \lfloor$rnd_bytes[i] $\cdot\, (n - i)/2^{32} \rfloor$
6. **end**
7. **for** ($i \leftarrow \omega - 1$; $i \geq 0$; i−−) **do**
8. **for** ($j \leftarrow i + 1$; $j < \omega$; j++) **do**
9. **if** (support[i] = support[j]) **then**
10. support[i] $\leftarrow i$
11. **end**
12. **end**
13. **end**
14. **return** (ctx, support)

ComputeVectorFromSupport(support, w)

Input: support $\in (\mathcal{B}^2)^w$, an integer $w \in \mathbb{N}^*$

Output: vector $\mathbf{v} \in \mathcal{B}^{\lceil n/8 \rceil}$

1. $\mathbf{v} \leftarrow \mathcal{B}_0^{\lceil n/8 \rceil}$
2. index $\leftarrow 0$
3. **for** ($i \leftarrow 0$; $i < w$; i++) **do**
4. index $\leftarrow \lfloor \frac{\text{support}[i]}{8} \rfloor$
5. $\mathbf{v}$[index] $\leftarrow \mathbf{v}$[index] $\vee\, (1 \ll (\text{support}[i] \bmod 8))$
6. **end**
7. **return** $\mathbf{v}$

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 [3].

Instance	mask_n	rejection_threshold_n
HQC-1	0x1F	16 767 881
HQC-3	0x07	16 742 417
HQC-5	0x1F	16 772 367

Table 4: Sampling related parameters

3.4 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-1 and HQC-3, and a 5-part split (`5-Karatsuba`) as the Toom-Cook elementary multiplication for HQC-5. 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 5.

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
n	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 5: Implementation of HQC multiplications over $\mathbb{F}_2[X]$

3.5 Concatenated Reed-Muller and Reed-Solomon codes

HQC uses concatenated codes instantiated using Reed-Solomon and Reed-Muller codes [5].

3.5.1 Concatenated codes

Definition 3.5.1 (Concatenated codes)**.** *A concatenated code consists of an external code* $[n_1, k_1, d_1]$ *over* $\mathbb{F}_q$ *and an internal code* $[n_2', k_2, d_2]$ *over* $\mathbb{F}_2$*, with* $q = 2^{k_2}$*. One uses a bijection between elements of* $\mathbb{F}_q$ *and the words of the internal code to obtain a transformation* $\mathbb{F}_q^{n_1} \to \mathbb{F}_2^{n_1 n_2'}$*. The resulting concatenated code is a* $[n_1 n_2', k_1 k_2, d \geqslant d_1 d_2]$ *code over* $\mathbb{F}_2$*.*

HQC concatenated code ($\mathcal{C}$) is instantiated with a shortened Reed-Solomon (sRS) code as the external code and a duplicated Reed-Muller (dRM) code as the internal code. Encoding is performed using $\mathcal{C}$.Encode by calling RS.Encode then dRM.Encode. Decoding is performed using $\mathcal{C}$.Decode by calling dRM.Decode then RS.Decode.

$\mathcal{C}$.Encode($\mathbf{m}$)

Input: Message $\mathbf{m} \in \mathcal{B}^{k_1}$

Output: Codeword $\mathbf{v} \in \mathcal{B}^{n_1 \times n_2}$

1. $\mathbf{m}' \leftarrow$ RS.Encode($\mathbf{m}$)
2. $\mathbf{v} \leftarrow$ dRM.Encode($\mathbf{m}'$)
3. **return** $\mathbf{v}$

$\mathcal{C}$.Decode($\mathbf{w}$)

Input: Word $\mathbf{w} \in \mathcal{B}^{n_1 n_2}$

Output: Message $\mathbf{m} \in \mathcal{B}^{k_1}$

1. $\mathbf{w}' \leftarrow$ dRM.Decode($\mathbf{w}$)
2. $\mathbf{m} \leftarrow$ RS.Decode($\mathbf{w}'$)
3. **return** $\mathbf{m}$

3.5.2 Reed-Solomon codes

Definition 3.5.2 (Reed-Solomon codes [28])**.** *Let* $\mathbb{F}_q$ *be a finite field with* $q = 2^m$ *and let* α *be a primitive element of* $\mathbb{F}_q$*. A Reed-Solomon code* $\mathsf{RS}[n_1, k_1, d_1]$ *with symbols in* $\mathbb{F}_q$ *has the following parameters:*

- *Length* $n_1 = q - 1$ *and dimension* k_1*;*
- *Number of parity-check digits* $n_1 - k_1 = 2t_1$ *with* t_1 *the correcting capacity of the code;*
- *Minimum distance* $d_1 = 2t_1 + 1$.

The generator polynomial $\mathsf{rs_generator_polynomial}(x)$ *of the* $\mathsf{RS}[n_1, k_1, d_1]$ *code is given by:*

$$\mathsf{rs_generator_polynomial}(X) = (X + \alpha)(X + \alpha^2) \cdots (X + \alpha^{2t_1}).$$

RS codes considered in HQC. A shortened Reed-Solomon code sRS is a RS code featuring smaller length and dimension while preserving its error correcting capacity t_1. HQC uses sRS of dimension $k_1 = \lambda/8$ over $\mathbb{F}_q = \mathbb{F}_{2^8}$ that can encode messages of size λ bits. Arithmetic over $\mathbb{F}_{2^8} = (\mathbb{F}_2[X]/\langle p(X)\rangle, \oplus, \otimes)$ is performed using the irreducible polynomial $p(X) = X^8 + X^4 + X^3 + X + 1$. Table 6 provides the parameters for the considered RS and sRS codes while their associated generator polynomials are given below. One can use the functions provided in the file `reed_solomon.h` to reconstruct these generator polynomials.

Instance	RS Code	sRS Code	
		$[n_1, k_1, d_1]$	t_1
HQC-1	[255, 225, 31]	[46, 16, 31]	15
HQC-3	[255, 223, 33]	[56, 24, 33]	16
HQC-5	[255, 197, 59]	[90, 32, 59]	29

Table 6: Parameters of the Shortened Reed-Solomon codes used in HQC.

Generator polynomial of $\mathsf{sRS}[46, 16, 31]$. $g_1(X) = 74 + 81X + 224X^2 + 114X^3 + 221X^4 + 115X^5 + 110X^6 + 88X^7 + 91X^8 + 151X^9 + 139X^{10} + 151X^{11} + 84X^{12} + 184X^{13} + 21X^{14} + 242X^{15} + 97X^{16} + 199X^{17} + 87X^{18} + 113X^{19} + 104X^{20} + 184X^{21} + 197X^{22} + 111X^{23} + 89X^{24} + 81X^{25} + 156X^{26} + 71X^{27} + 128X^{28} + 217X^{29} + X^{30}$.

Generator polynomial of $\mathsf{sRS}[56, 24, 33]$. $g_3(X) = 56 + 180X + 145X^2 + 30X^3 + 131X^4 + 105X^5 + 28X^6 + 60X^7 + 107X^8 + 33X^9 + 206X^{10} + 184X^{11} + 155X^{12} + 249X^{13} + 153X^{14} + 231X^{15} + 112X^{16} + 11X^{17} + 231X^{18} + X^{19} + 144X^{20} + 201X^{21} + 71X^{22} + 62X^{23} + 8X^{24} + 150X^{25} + 142X^{26} + 246X^{27} + 33X^{28} + 214X^{29} + 26X^{30} + 150X^{31} + X^{32}$.

Generator polynomial of $\mathsf{sRS}[90, 32, 59]$. $g_5(X) = 35 + 203X + 35X^2 + 52X^3 + 165X^4 + 121X^5 + 125X^6 + 73X^7 + 136X^8 + 42X^9 + 235X^{10} + 82X^{11} + 232X^{12} + 122X^{13} + 67X^{14} + 98X^{15} + 51X^{16} + 188X^{17} + 230X^{18} + 82X^{19} + 164X^{20} + 117X^{21} + 103X^{22} + 184X^{23} + 209X^{24} + 218X^{25} + 244X^{26} + 181X^{27} + 122X^{28} + 10X^{29} + 25X^{30} + 138X^{31} + 216X^{32} + 182X^{33} + 225X^{34} + 145X^{35} + 100X^{36} + 244X^{37} + 5X^{38} + 142X^{39} + 213X^{40} + 238X^{41} +$

$15X^{42} + 150X^{43} + 59X^{44} + 28X^{45} + 244X^{46} + 222X^{47} + 252X^{48} + 85X^{49} + 125X^{50} + 59X^{51} + 52X^{52} + 215X^{53} + 180X^{54} + 34X^{55} + 173X^{56} + 208X^{57} + X^{58}$.

Encoding RS codes. Encoding Shortened Reed-Solomon codes can be done in the same way as encoding Reed-Solomon codes. We denote by $u(X) = u_0 + \cdots + u_{k_1-1}X^{k_1-1}$ the polynomial corresponding to the message $u = (u_0, \cdots, u_{k_1-1})$. We use the systematic form of encoding where the rightmost k_1 positions of the codeword polynomial are the message symbols and the leftmost $n_1 - k_1$ bits are the parity-check symbols. The codeword is given by $c(X) = b(x) + X^{n_1-k_1}u(X)$, where $b(X)$ is the remainder of the division of the polynomial $X^{n_1-k_1}u(X)$ by the generator polynomial. In consequence, the encoding in systematic form consists of three steps :

1. Multiply the message $u(X)$ by $X^{n_1-k_1}$.
2. Compute the remainder $b(X)$ by dividing $X^{n_1-k_1}u(X)$ by the generator polynomial.
3. Combine $b(X)$ and $X^{n_1-k_1}u(X)$ to obtain the codeword $c(X) = b(X) + X^{n_1-k_1}u(X)$.

The RS.Encode algorithm implements these steps using an LFSR as described in [28].

RS.Encode(message)

Input: message $\in \mathcal{B}^{k_1}$
Output: codeword $\in \mathcal{B}^{n_1}$

```
1.  codeword ← B_0^{n_1}
2.  gate_value ← B_0

    ▷ Compute the redundancy bytes
3.  for ( i ← 0 ; i < k_1 ; i++ ) do
4.      gate_value ← message[k_1 − 1 − i] ⊕ codeword[n_1 − k_1 − 1]

5.      for ( j ← n_1 − k_1 − 1 ; j > 0 ; j−− ) do
6.          codeword[j] ← codeword[j − 1] ⊕ (gate_value ⊗ rs_generator_polynomial[j])
7.      end
8.      codeword[0] ← gate_value ⊗ rs_generator_polynomial[0]
9.  end

    ▷ Compute the message bytes
10. codeword[n_1 − k_1 : n_1] ← message[0 : k_1]
11. return codeword
```

Decoding RS codes. Similarly to encoding, the decoding of classical Reed-Solomon codes can be used to decode shortened Reed-Solomon codes. Hereafter, we describe the process of decoding classical Reed-Solomon codes following [28]. Consider the Reed-Solomon code defined by $[n_1, k_1, d_1]$, with $n_1 = 2^m - 1$ (where $m \geq 0$ is a positive integer) and suppose that a codeword $v(X) = v_0 + v_1 X + \cdots + v_{n_1-1} X^{n_1-1}$ is transmitted. Let $e(X) = e_0 + e_1 X + \cdots + e_{n_1-1} X^{n_1-1}$ be an error polynomial meaning that there is an error in position j whenever $e_j \neq 0$. Let $r(X) = r_0 + r_1 X + \cdots + r_{n_1-1} X^{n_1-1}$ be the received word, potentially altered by some errors such that $r(X) = v(X) + e(X)$.

We define the set of syndromes $S_1, S_2, \cdots, S_{2t_1}$ 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 $v(\alpha^i) = 0$ as v is a codeword. Suppose that $e(x)$ has t_1 errors at locations $j_1, j_2, \cdots, j_{t_1}$, *i.e.* $e(x) = e_{j_1} x^{j_1} + e_{j_2} x^{j_2} + \cdots + e_{j_{t_1}} x^{j_{t_1}}$. We obtain the following set of equations where $\alpha^{j_1}, \cdots, \alpha^{j_{t_1}}$ are unknown:

$$\begin{cases} S_1 &= e_{j_1}(\alpha^{j_1})^1 + e_{j_2}(\alpha^{j_2})^1 + \cdots + e_{j_{t_1}}(\alpha^{j_{t_1}})^1 \\ S_2 &= e_{j_1}(\alpha^{j_1})^2 + e_{j_2}(\alpha^{j_2})^2 + \cdots + e_{j_{t_1}}(\alpha^{j_{t_1}})^2 \\ &\vdots \\ S_{2t_1} &= e_{j_1}(\alpha^{j_1})^{2t_1} + e_{j_2}(\alpha^{j_2})^{2t_1} + \cdots + e_{j_{t_1}}(\alpha^{j_{t_1}})^{2t_1} \end{cases}$$

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. Hence, the set of equations can be expressed as follows:

$$\begin{cases} S_1 &= e_{j_1}\beta_1 + e_{j_2}\beta_2 + \cdots + e_{j_{t_1}}\beta_{t_1} \\ S_2 &= e_{j_1}\beta_1^2 + e_{j_2}\beta_2^2 + \cdots + e_{j_{t_1}}\beta_{t_1}^2 \\ &\vdots \\ S_{2t_1} &= e_{j_1}\beta_1^{2t_1} + e_{j_2}\beta_2^{2t_1} + \cdots + e_{j_{t_1}}\beta_{t_1}^{2t_1} \end{cases}$$

We define the error location polynomial as:

$$\sigma(X) = (1 + \beta_1 X)(1 + \beta_2 X) \cdots (1 + \beta_{t_1} X) = 1 + \sigma_1 X + \sigma_2 X^2 + \cdots + \sigma_{t_1} X^{t_1}$$

The roots of $\sigma(X)$ are $\beta_1^{-1}, \beta_2^{-1}, \cdots, \beta_{t_1}^{-1}$ are the inverses of the error location numbers.

After retrieving the coefficients of $\sigma(x)$, one can compute the error values. Let $Z(X) = 1 + (S_1 + \sigma_1)X + (S_2 + \sigma_1 S_1 + \sigma_2)X^2 + \cdots + (S_{t_1} + \sigma_1 S_{t_1-1} + \sigma_2 S_{t_1-2} + \cdots + \sigma_{t_1})X^{t_1}$, the error value at location β_l is given [7] by:

$$e_{j_l} = \frac{Z(\beta_l^{-1})}{\prod_{\substack{i=1 \\ i \neq l}}^{t_1} (1 + \beta_i \beta_l^{-1})}.$$

Forney's algorithm [18] provides an efficient way to compute the error values using the formal derivative of the error location polynomial:

$$\sigma'(X) = \frac{d}{dX}\prod_{i=1}^{t_1}(1+\beta_i X) = \sum_{i=1}^{t_1}\beta_i \prod_{\substack{j=1\\ j\neq i}}^{t_1}(1+\beta_j X).$$

As $\sigma'(\beta_l^{-1}) = \beta_l \prod_{j=1, j\neq l}^{t_1}(1+\beta_j X)$, the error magnitude is simply:

$$e_{j_l} = \frac{Z(\beta_l^{-1})}{\beta_l^{-1}\sigma'(\beta_l^{-1})}.$$

One can define the error locator polynomial from its even-part and odd-part polynomials namely $\sigma(X) = \sigma_e(X) + \sigma_o(X)$ with:

$$\sigma_e(X) := \sum_{i=0}^{\lfloor t_1/2 \rfloor} \sigma_{2i} X^{2i} \text{ and } \sigma_o(X) := \sum_{i=1}^{\lceil t_1/2 \rceil} \sigma_{2i-1} X^{2i-1}.$$

One can also write $X\sigma'(X) = \sigma_1 X + 2\sigma_2 X^2 + \cdots + t_1\sigma_{t_1}X^{t_1} = \sum_{i=1}^{t_1-1} i\sigma_i X^i$ which in characteristic 2 simplifies to $X\sigma'(X) = \sum_{i=1}^{\lceil t_1/2 \rceil} \sigma_{2i-1}X^{2i-1}$ thus:

$$e_{j_l} = \frac{Z(\beta_l^{-1})}{\sigma_o(\beta_l^{-1})}.$$

This way, both $\sigma(X)$ and the denominator of the error magnitude can be computed at the cost of evaluating only $\sigma(X)$ and $Z(X)$ at the code support $\beta_1^{-1} \ldots \beta_{t_1}^{-1}$ along with one field inversion for each evaluation. The decoding is completed by computing $r(X) - e(X)$.

The decoding procedure RS.Decode returns a deterministic best-effort message and does not signal failure. One can summarize it as follows:

1. RS.Decode.ComputeSyndromes computes the $2t_1$ syndromes from the received word $r(X)$ by evaluating $r(\alpha^i)$ for each value of i.
2. RS.Decode.ComputeErrorLocatorPolynomial computes the error-location polynomial $\sigma(X)$ from the syndromes using the Berlekamp's algorithm [28].
3. RS.Decode.ComputeErrorPositions finds the error positions within the first k_1 symbols of the codeword by computing the inverses of the roots of the polynomial $\sigma(X)$.
4. RS.Decode.CorrectErrors computes the polynomial $Z(X)$ and $\sigma_o(X)$, the error values and correct the errors to recompute the message.

RS.Decode(received_word)

Input: received_word $\in \mathcal{B}^{n_1}$

Output: message $\in \mathcal{B}^{k_1}$

1. syndrome $\leftarrow$ RS.Decode.ComputeSyndromes(received_word)
2. sigma $\leftarrow$ RS.Decode.ComputeErrorLocatorPolynomial(syndrome)
3. error_positions $\leftarrow$ RS.Decode.ComputeErrorPositions(sigma)
4. message $\leftarrow$ RS.Decode.CorrectErrors(received_word, syndrome, sigma, error_positions)
5. **return** message

RS.Decode.EvaluatePoly(polynomial, degree, point)

Input: polynomial $\in \mathcal{B}^{\text{degree}+1}$, highest index examined degree $\in \mathbb{N}$, point $\in \mathcal{B}$

Output: eval $\in \mathcal{B}$

1. point_pow $\leftarrow$ point
2. eval $\leftarrow$ polynomial$[0]$
3. **for** ($i \leftarrow 1$; $i \leq$ degree ; $i{+}{+}$) **do**
4. eval $\leftarrow$ eval $\oplus$ (polynomial$[i] \otimes$ point_pow)
5. point_pow $\leftarrow$ point_pow $\otimes$ point
6. **end**
7. **return** eval

RS.Decode.ComputeSyndromes(received_word)

Input: received_word $\in \mathcal{B}^{n_1}$

Output: syndrome $\in \mathcal{B}^{2 \times t_1}$

1. syndrome $\leftarrow \mathcal{B}_0^{2 \times t_1}$
2. **for** ($i \leftarrow 0$; $i < 2 \times t_1$; $i{+}{+}$) **do**
3. syndrome$[i] \leftarrow$ RS.Decode.EvaluatePoly(received_word, $n_1 - 1$, alpha^{i+1})
4. **end**
5. **return** syndrome

RS.Decode.ComputeErrorLocatorPolynomial(syndrome)

Input: syndrome $\in \mathcal{B}^{2\times t_1}$

Output: Error locator polynomial sigma $\in \mathcal{B}^{t_1+1}$

1. $(\mathsf{sigma}, d) \leftarrow (\mathcal{B}_0^{t_1+1}, 0)$
2. $(\mathsf{sigma}_{\mathsf{mu}}, \mathsf{deg_sigma}_{\mathsf{mu}}) \leftarrow (\mathcal{B}_0^{t_1+1}, 0)$
3. $(\mathsf{sigma}_{\mathsf{rho}}, \mathsf{deg_sigma}_{\mathsf{rho}}) \leftarrow (\mathcal{B}_0^{t_1+1}, 0)$
4. $(\mathsf{sigma}[0], \mathsf{d}_{\mathsf{mu}}) \leftarrow (1, \mathsf{syndrome}[0])$
5. $(\mathsf{rho}, \mathsf{sigma}_{\mathsf{rho}}[0], \mathsf{d}_{\mathsf{rho}}) \leftarrow (-1, 1, 1)$

6. **for** ($\mathsf{mu} \leftarrow 0$; $\mathsf{mu} < 2 \times t_1$; $\mathsf{mu}{+}{+}$) **do**
7. **if** ($\mathsf{d}_{\mathsf{mu}} \neq 0$) **then**

▷ Update polynomial sigma

8. $(\mathsf{sigma}_{\mathsf{mu}}, \mathsf{deg_sigma}_{\mathsf{mu}}) \leftarrow (\mathsf{sigma}, d)$
9. **for** ($i \leftarrow 0$; $i \leq \min(\mathsf{deg_sigma}_{\mathsf{rho}}, t_1 - (\mathsf{mu} - \mathsf{rho}))$; $i{+}{+}$) **do**
10. $\mathsf{pos} \leftarrow i + \mathsf{mu} - \mathsf{rho}$
11. $\mathsf{sigma}[\mathsf{pos}] \leftarrow \mathsf{sigma}[\mathsf{pos}] \oplus (\mathsf{d}_{\mathsf{mu}} \otimes \mathsf{d}_{\mathsf{rho}}^{-1} \otimes \mathsf{sigma}_{\mathsf{rho}}[i])$
12. **end**
13. $d \leftarrow \min(t_1, \max(d, (\mathsf{mu} - \mathsf{rho} + \mathsf{deg_sigma}_{\mathsf{rho}})))$

▷ Update polynomial $\mathsf{sigma}_{\mathsf{rho}}$

14. **if** ($(\mathsf{mu} - \mathsf{deg_sigma}_{\mathsf{mu}}) > (\mathsf{rho} - \mathsf{deg_sigma}_{\mathsf{rho}})$) **then**
15. $(\mathsf{sigma}_{\mathsf{rho}}, \mathsf{deg_sigma}_{\mathsf{rho}}) \leftarrow (\mathsf{sigma}_{\mathsf{mu}}, \mathsf{deg_sigma}_{\mathsf{mu}})$
16. $(\mathsf{rho}, \mathsf{d}_{\mathsf{rho}}) \leftarrow (\mathsf{mu}, \mathsf{d}_{\mathsf{mu}})$
17. **end**
18. **end**

▷ Update discrepancy d_{mu}

19. **if** ($\mathsf{mu} + 1 < 2 \times t_1$) **then**
20. $\mathsf{d}_{\mathsf{mu}} \leftarrow \mathsf{syndrome}[\mathsf{mu} + 1]$
21. **for** ($i \leftarrow 1$; $i \leq d$; $i{+}{+}$) **do**
22. $\mathsf{d}_{\mathsf{mu}} \leftarrow \mathsf{d}_{\mathsf{mu}} \oplus (\mathsf{sigma}[i] \otimes \mathsf{syndrome}[\mathsf{mu} + 1 - i])$
23. **end**
24. **end**
25. **end**

26. **return** sigma

RS.Decode.ComputeErrorPositions(sigma)

Input: Error locator polynomial sigma $\in \mathcal{B}^{t_1+1}$

Output: error_positions $\in \{0,1\}^{k_1}$

1. error_positions $\leftarrow 0^{k_1}$
2. pos $\leftarrow 0$

3. **for** ($i \leftarrow 0$; $i < k_1$; $i{+}{+}$) **do**
4. pos $\leftarrow n_1 - k_1 + i$
5. **if** (RS.Decode.EvaluatePoly(sigma, t_1, (alpha$^{\text{pos}}$)$^{-1}$) $= 0$) **then**
6. error_positions$[i] \leftarrow 1$
7. **end**
8. **end**

9. **return** error_positions

RS.Decode.CorrectErrors(codeword, syndrome, sigma, error_positions)

Input: codeword $\in \mathcal{B}^{n_1}$, syndrome $\in \mathcal{B}^{2\times t_1}$, ELP sigma $\in \mathcal{B}^{t_1+1}$, error_positions $\in \{0,1\}^{k_1}$
Output: message $\in \mathcal{B}^{k_1}$

1. z $\leftarrow \mathcal{B}_0^{t_1+1}$
2. z$[0] \leftarrow 1$
3. sigma_odd $\leftarrow \mathcal{B}_0^{t_1+1}$
4. message $\leftarrow \mathcal{B}_0^{k_1}$
5. pos $\leftarrow 0$

▷ Compute the error-evaluator polynomial z(x)

6. **for** ($i \leftarrow 0$; $i < t_1$; i++) **do**
7. z$[i+1] \leftarrow$ syndrome$[i]$
8. **for** ($j \leftarrow 0$; $j < i$; j++) **do**
9. z$[i+1] \leftarrow$ z$[i+1] \oplus$ (sigma$[j+1] \otimes$ syndrome$[i-1-j]$)
10. **end**
11. z$[i+1] \leftarrow$ z$[i+1] \oplus$ sigma$[i+1]$
12. **end**

▷ Compute polynomial sigma_odd(x)

13. **for** ($i \leftarrow 1$; $i \leq t_1$; $i \leftarrow i+2$) **do**
14. sigma_odd$[i] \leftarrow$ sigma$[i]$
15. **end**

▷ Correct errors

16. **for** ($i \leftarrow 0$; $i < k_1$; i++) **do**
17. pos $\leftarrow n_1 - k_1 + i$
18. numerator $\leftarrow$ RS.Decode.EvaluatePoly(z, t_1, (alpha$^{\text{pos}}$)$^{-1}$)
19. denominator $\leftarrow$ RS.Decode.EvaluatePoly(sigma_odd, t_1, (alpha$^{\text{pos}}$)$^{-1}$)
20. message$[i] \leftarrow$ codeword[pos]
21. **if** (error_positions$[i] = 1$ **and** denominator $\neq 0$) **then**
22. message$[i] \leftarrow$ message$[i] \oplus$ (numerator $\otimes$ denominator^{-1})
23. **end**
24. **end**

25. **return** message

3.5.3 Duplicated Reed-Muller codes

Definition 3.5.3 (Reed-Muller codes [32]). *Let m and r be positive integers with $0 \leq r \leq m$. The binary r^{th} order Reed-Muller code denoted $\mathsf{RM}(r, m)$ has the following parameters:*

- *Length* $n_3 = 2^m$;
- *Dimension* $k_3 = \sum_{i=0}^{r} \binom{m}{i}$;
- *Minimum distance* $d_3 = 2^{m-r}$.

RM codes considered in HQC. HQC uses the first-order $\mathsf{RM}(1, 7)$ code namely the binary $[128, 8, 64]$ code. The codeword is duplicated rm_multiplicity times to obtain the $[n_2' = \mathsf{rm_multiplicity} \times n_3, k_2 = k_3, d_2 = \mathsf{rm_multiplicity} \times d_3]$ code as shown in Table 7.

Instance	RM Code	rm_multiplicity	dRM Code	
	$[n_3, k_3, d_3]$		$[n_2', k_2, d_2]$	n_2
HQC-1	[128, 8, 64]	3	[384, 8, 192]	48
HQC-3		5	[640, 8, 320]	80
HQC-5		5	[640, 8, 320]	80

Table 7: Parameters of the Reed-Muller and Duplicated Reed-Muller codes used in HQC.

Encoding dRM codes. Encoding duplicated Reed-Muller codes is done by encoding each byte of the message using RM.Encode yielding a $n_1\lceil n_3/8 \rceil$ byte long string that is duplicated rm_multiplicity times in order to obtain the $n_1 n_2$ bytes long codeword. Following [32], RM.Encode performs the vector matrix multiplication between the message $\boldsymbol{m} = (\boldsymbol{m}_0, \ldots, \boldsymbol{m}_7)$ and the $k_3 \times n_3$ generator matrix rm_generator_matrix where (i) denotes the column-wise bit representation of $i \in [0, 127]$ where the most significant bit of i is at the topmost position. As bytes are represented as $(b_7, \ldots, b_0)$ with b_0 the least significant bit, one should note that the position i in the message $\boldsymbol{m}$ represented as a vector corresponds to the position $7 - i$ in the message message represented as a byte.

$$\mathsf{rm_generator_matrix} = \begin{bmatrix} 1 & \cdots & 1 \\ (0) & \cdots & (127) \end{bmatrix}$$

RM.Encode uses rm_generator_matrix represented as a $k_3 \times \lceil n_3/8 \rceil$ byte matrix namely:

rm_generator_matrix =

$$\begin{bmatrix}
\texttt{0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF} \\
\texttt{0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF} \\
\texttt{0x00 0x00 0x00 0x00 0xFF 0xFF 0xFF 0xFF 0x00 0x00 0x00 0x00 0xFF 0xFF 0xFF 0xFF} \\
\texttt{0x00 0x00 0xFF 0xFF 0x00 0x00 0xFF 0xFF 0x00 0x00 0xFF 0xFF 0x00 0x00 0xFF 0xFF} \\
\texttt{0x00 0xFF 0x00 0xFF 0x00 0xFF 0x00 0xFF 0x00 0xFF 0x00 0xFF 0x00 0xFF 0x00 0xFF} \\
\texttt{0x0F 0x0F 0x0F 0x0F 0x0F 0x0F 0x0F 0x0F 0x0F 0x0F 0x0F 0x0F 0x0F 0x0F 0x0F 0x0F} \\
\texttt{0x33 0x33 0x33 0x33 0x33 0x33 0x33 0x33 0x33 0x33 0x33 0x33 0x33 0x33 0x33 0x33} \\
\texttt{0x55 0x55 0x55 0x55 0x55 0x55 0x55 0x55 0x55 0x55 0x55 0x55 0x55 0x55 0x55 0x55}
\end{bmatrix}.$$

dRM.Encode(message)

Input: message $\in \mathcal{B}^{n_1}$

Output: codeword $\in \mathcal{B}^{n_1 \times n_2}$

1. codeword $\in \mathcal{B}_0^{n_1 \times n_2}$
2. **for** ($i \leftarrow 0$; $i < n_1$; $i{+}{+}$) **do**
   ▷ Encode message one byte at a time
3. pos $\leftarrow i \times (16 \times$ rm_multiplicity$)$
4. codeword[pos : pos $+ 16$] $\leftarrow$ RM.Encode(message[i])

   ▷ Duplicate encoding rm_multiplicity times
5. **for** ($j \leftarrow 1$; $j <$ rm_multiplicity ; $j{+}{+}$) **do**
6. codeword[pos $+ 16 \times j$: pos $+ 16 \times j + 16$] $\leftarrow$ codeword[pos : pos $+ 16$]
7. **end**
8. **end**
9. **return** codeword

RM.Encode(message)

Input: message $\in \mathcal{B}$

Output: codeword $\in \mathcal{B}^{(n_3/8)}$

```
1.  codeword ∈ B_0^(n3/8)

2.  for ( i ← 0 ; i < k3 ; i++ ) do
3.    if ( ByteToBit(message, 7 − i) = 1 ) then
4.      for ( j ← 0 ; j < n3/8 ; j++ ) do
5.        codeword[j] ← codeword[j] ⊕ rm_generator_matrix[i][j]
6.      end
7.    end
8.  end

9.  return codeword
```

Decoding dRM code. First order Reed-Muller codes can be decoded using the fast Hadamard transform (see [32], chapter 14 for example). Duplicated Reed-Muller codes can be decoded similarly by slightly adapting the procedure to take into account the duplication. The dRM.Decode returns a deterministic best-effort message and does not signal failure. It manipulates the $n_1 n_2$ bytes long word by blocks of n_2 bytes sequentially in order to retrieve the n_1 bytes long message. The decoding of each block follows these three steps:

1. dRM.Decode.ExpandAndSum takes a duplicated word $\boldsymbol{w} = (\boldsymbol{w}^0, \ldots, \boldsymbol{w}^{\mathsf{rm_multiplicity}})$ and computes $\mathsf{bit_sums} = (\sum_{i=0}^{\mathsf{rm_multiplicity}} \boldsymbol{w}_0^i, \ldots, \sum_{i=0}^{\mathsf{rm_multiplicity}} \boldsymbol{w}_{127}^i)$. One should note that $\mathsf{bit_sums} \in [0, \mathsf{rm_multiplicity}]^{n_3}$ which differs from F in [32] as codewords are mapped to $\{0,1\}$ instead of $\{-1,1\}$ and the computation is adjusted for rm_multiplicity duplications.

2. dRM.Decode.Hadamard computes hadamard_coeffs by applying the Hadamard transform to the bit_sums values. After the computation of the Hadamard transform, hadamard_coeffs[0] is corrected to take into account that dRM.Decode.ExpandAndSum uses $\{0,1\}$ values instead of $\{-1,1\}$ ones.

3. dRM.Decode.FindPeaks computes the peak namely the position of the highest absolute value of the hadamard_coeffs and interprets it as as the decoded_byte. If the peak's value is positive, the most significant bit of the decoded_byte is set before returning it. If the peak is not unique, the decoded_byte corresponding to the smallest position is returned.

dRM.Decode($\mathbf{w}$)

Input: Word $\mathbf{w} \in \mathcal{B}^{n_1 \times n_2}$

Output: Message $\mathbf{m} \in \mathcal{B}^{n_1}$

1. **for** ($i \leftarrow 0$; $i < n_1$; $i{+}{+}$) **do**
2. duplicated_word $\leftarrow \mathbf{w}[i \times n_2 : (i+1) \times n_2]$
3. bit_sums $\leftarrow$ dRM.Decode.ExpandAndSum(duplicated_word)
4. hadamard_coeffs $\leftarrow$ dRM.Decode.Hadamard(bit_sums)
5. $\mathbf{m}[i] \leftarrow$ dRM.Decode.FindPeaks(hadamard_coeffs)
6. **end**
7. **return m**

dRM.Decode.ExpandAndSum(duplicated_word)

Input: duplicated_word $\in \mathcal{B}^{n_2}$

Output: bit_sums $\in \mathbb{Z}^{n_3}$

1. bit_sums $\leftarrow 0^{n_3}$
2. **for** ($i \leftarrow 0$; $i <$ rm_multiplicity ; $i{+}{+}$) **do**
3. **for** ($j \leftarrow 0$; $j < n_3/8$; $j{+}{+}$) **do**
4. **for** ($k \leftarrow 0$; $k < k_3$; $k{+}{+}$) **do**
5. tmp $\leftarrow$ ByteToBit(duplicated_word$[i \times n_3/8 + j], 7 - k)$
6. bit_sums$[8 \times j + k] \leftarrow$ bit_sums$[8 \times j + k] +$ tmp
7. **end**
8. **end**
9. **end**
10. **return** bit_sums

dRM.Decode.Hadamard(bit_sums)

Input: bit_sums $\in \mathbb{Z}^{n_3}$

Output: hadamard_coeffs $\in \mathbb{Z}^{n_3}$

```
1.  bit_sums_copy ← bit_sums
2.  hadamard_coeffs ← 0^{n_3}
3.  (a, b) ← (0, 0)

    ▷ Compute Hadamard transform
4.  for ( i ← 0 ; i < hadamard_stages ; i++ ) do
5.      for ( j ← 0 ; j < hadamard_register_size ; j++ ) do
6.          a = bit_sums_copy[2 × j]
7.          b = bit_sums_copy[2 × j + 1]
8.          hadamard_coeffs[j] = a + b
9.          hadamard_coeffs[j + hadamard_register_size] = a − b
10.     end

11.     if ( i ≠ hadamard_stages − 1 ) then
12.         (hadamard_coeffs, bit_sums_copy) ← (bit_sums_copy, hadamard_coeffs)
13.     end
14. end

    ▷ Correct first coefficient
15. hadamard_coeffs[0] ← hadamard_coeffs[0] − (hadamard_register_size × rm_multiplicity)

16. return hadamard_coeffs
```

dRM.Decode.FindPeaks(hadamard_coeffs)

Input: hadamard_coeffs $\in \mathbb{Z}^{n_3}$

Output: decoded_byte $\in \mathcal{B}$

```
1.  decoded_byte ← B_0
2.  current_magnitude ← 0
3.  peak_magnitude ← 0

    ▷ Find the peak_magnitude
4.  for ( i ← 0 ; i < n_3 ; i++ ) do
5.     current_magnitude ← |hadamard_coeffs[i]|
6.     if ( current_magnitude > peak_magnitude ) then
7.        peak_magnitude ← current_magnitude
8.        decoded_byte ← i
9.     end
10. end

    ▷ Compute the most significant bit of decoded_byte
11. if ( hadamard_coeffs[decoded_byte] > 0 ) then
12.    decoded_byte ← decoded_byte ∨ 0x80
13. end

14. return decoded_byte
```

3.6 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 3.5. 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.

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

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

7. $\mathsf{ctx}_{\mathsf{PKE.ek}} \leftarrow \mathsf{XOF.Init}()$
8. $\mathsf{ctx}_{\mathsf{PKE.ek}} \leftarrow \mathsf{XOF.Absorb}(\mathsf{ctx}_{\mathsf{PKE.ek}}, \mathsf{seed}_{\mathsf{PKE.ek}}, |\mathsf{seed}|)$
9. $(\mathsf{ctx}_{\mathsf{PKE.ek}}, \mathbf{h}) \leftarrow \mathsf{SampleVect}(\mathsf{ctx}_{\mathsf{PKE.ek}})$
10. $\mathbf{s} \leftarrow \mathbf{x} + \mathbf{h} \cdot \mathbf{y}$
11. $\mathsf{ek}_{\mathsf{PKE}} \leftarrow (\mathsf{seed}_{\mathsf{PKE.ek}}, \mathbf{s})$
12. **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. $(\mathsf{seed}_{\mathsf{PKE.ek}}, \mathbf{s}) \leftarrow \mathsf{ek}_{\mathsf{PKE}}$
2. $\mathsf{ctx}_{\mathsf{PKE.ek}} \leftarrow \mathsf{XOF.Init}()$
3. $\mathsf{ctx}_{\mathsf{PKE.ek}} \leftarrow \mathsf{XOF.Absorb}(\mathsf{ctx}_{\mathsf{PKE.ek}}, \mathsf{seed}_{\mathsf{PKE.ek}}, |\mathsf{seed}|)$
4. $(\mathsf{ctx}_{\mathsf{PKE.ek}}, \mathbf{h}) \leftarrow \mathsf{SampleVect}(\mathsf{ctx}_{\mathsf{PKE.ek}})$

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

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

13. **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 $\mathbf{m}$

▷ Recompute $\mathbf{y}$ from decryption key $\mathsf{dk}_{\mathsf{PKE}}$

1. $\mathsf{ctx}_{\mathsf{PKE.dk}} \leftarrow \mathsf{XOF.Init}()$
2. $\mathsf{ctx}_{\mathsf{PKE.dk}} \leftarrow \mathsf{XOF.Absorb}(\mathsf{ctx}_{\mathsf{PKE.dk}}, \mathsf{dk}_{\mathsf{PKE}}, |\mathsf{seed}|)$
3. $(\mathsf{ctx}_{\mathsf{PKE.dk}}, \mathbf{y}) \leftarrow \mathsf{SampleFixedWeightVect}_{\$}(\mathsf{ctx}_{\mathsf{PKE.dk}}, \omega)$

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

4. $(\mathbf{u}, \mathbf{v}) \leftarrow \mathbf{c}_{\mathsf{PKE}}$

▷ Compute plaintext $\mathbf{m}$

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

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

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.

3.7 HQC-KEM

HQC-KEM is derived from HQC-PKE using the Fujisaki-Okamoto transformation [19] instantiated following the HHK framework with implicit rejection [26].

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 $\mathsf{seed}_{\mathsf{PKE}}$

2. $\mathsf{seed}_{\mathsf{PKE}} \leftarrow \mathcal{B}_0^{|\mathsf{seed}|}$
3. $\mathsf{ctx}_{\mathsf{KEM}} \leftarrow \mathsf{XOF.Init}()$
4. $\mathsf{ctx}_{\mathsf{KEM}} \leftarrow \mathsf{XOF.Absorb}(\mathsf{ctx}_{\mathsf{KEM}}, \mathsf{seed}_{\mathsf{KEM}} \,||\, k/8, |\mathsf{seed}| + 1)$
5. $(\mathsf{ctx}_{\mathsf{KEM}}, \mathsf{seed}_{\mathsf{PKE}}) \leftarrow \mathsf{XOF.GetBytes}(\mathsf{ctx}_{\mathsf{KEM}}, \mathsf{seed}_{\mathsf{PKE}}, |\mathsf{seed}|)$

▷ Compute HQC-PKE keypair

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

▷ Compute HQC-KEM keypair

7. $\mathsf{ek}_{\mathsf{KEM}} \leftarrow \mathsf{ek}_{\mathsf{PKE}}$
8. $\mathsf{dk}_{\mathsf{KEM}} \leftarrow \mathsf{seed}_{\mathsf{KEM}}$

9. **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. $\mathbf{m} \leftarrow\$ \mathcal{B}^{\lceil k/8 \rceil}$
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($\mathsf{dk_{KEM}}, \mathbf{c}_{\mathsf{KEM}}$)

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

Output: Shared secret key K'

▷ Recompute $(\mathsf{ek_{PKE}}, \mathsf{dk_{PKE}})$

1. $\mathsf{seed_{PKE}} \leftarrow \mathcal{B}_0^{|\mathsf{seed}|}$
2. $\mathsf{ctx_{KEM}} \leftarrow \mathsf{XOF.Init}()$
3. $\mathsf{ctx_{KEM}} \leftarrow \mathsf{XOF.Absorb}(\mathsf{ctx_{KEM}}, \mathsf{dk_{KEM}} \,||\, k/8, |\mathsf{seed}| + 1)$
4. $(\mathsf{ctx_{KEM}}, \mathsf{seed_{PKE}}) \leftarrow \mathsf{XOF.GetBytes}(\mathsf{ctx_{KEM}}, \mathsf{seed_{PKE}}, |\mathsf{seed}|)$
5. $(\mathsf{ek_{PKE}}, \mathsf{dk_{PKE}}) \leftarrow \mathsf{HQC\text{-}PKE.Keygen}(\mathsf{seed_{PKE}})$

▷ Compute σ

6. $(\mathsf{ctx_{KEM}}, \sigma) \leftarrow \mathsf{XOF.GetBytes}(\mathsf{ctx_{KEM}}, \sigma, \lceil k/8 \rceil)$

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

7. $(\mathbf{c}_{\mathsf{PKE}}, \mathsf{salt}) \leftarrow \mathbf{c}_{\mathsf{KEM}}$

▷ Compute message $\mathbf{m}'$

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

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

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

▷ Compute rejection key $\bar{K}$

12. $\bar{K} \leftarrow \mathsf{J}(\mathsf{H}(\mathsf{ek_{PKE}}) \| \sigma \| \mathbf{c}_{\mathsf{KEM}})$
13. **if** $\mathbf{m}' = \perp$ **or** $\mathbf{c}'_{\mathsf{KEM}} \neq \mathbf{c}_{\mathsf{KEM}}$
14. $\quad K' \leftarrow \bar{K}$
15. **endif**
16. **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 based on the minimal workfactor of the best known attacks (see Section 6.3 for more details).

4.1 Parameter sets

The main parameters of HQC are given in Table 8 where n_1 denote the length of the shortened Reed-Solomon code and n_2 denote the length of the duplicated 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. Additional parameters for shortened Reed-Solomon and duplicated Reed-Muller related parameters are given in Tables 6 and 7 respectively.

Instance	Security	n_1	n_2	n	k	ω	$\omega_r = \omega_e$	DFR
HQC-1	NIST-1	46	48	17 669	128	67	76	$< 2^{-127}$
HQC-3	NIST-3	56	80	35 851	192	101	114	$< 2^{-192}$
HQC-5	NIST-5	90	80	57 637	256	132	150	$< 2^{-256}$

Table 8: 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 $|\mathsf{dk}_{\mathsf{KEM}}| = |\mathsf{seed}|$. 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 9.

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	32	4 433	32
HQC-3	NIST-3	4 514	32	8 978	32
HQC-5	NIST-5	7 237	32	14 421	32

Table 9: 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. Benchmarks were performed on a machine running Ubuntu 22.04.2 LTS with 32 GB of memory and an Intel® Core™ i7-11850H CPU @ 2.50GHz. For all experiments, Hyper-Threading and Turbo Boost features were disabled. Implementations were compiled using `gcc` (version `15.2.0`). For each parameter set, the results have been obtained by computing the mean over 1000 random instances. In order to minimize any bias from background tasks running on the benchmark platform, each instance have been repeated 100 times and averaged.

Constant time. The optimized implementation has been implemented in constant time hence it is expected that it does not leak any sensitive data whenever compiled on the aforementioned benchmark platform with the compiler and compiler flags mentioned hereafter.

5.1 Reference implementation

The performances of our reference implementation on the aforementioned benchmark platform are given in Table 10. The reference implementation is written in `C` and have been compiled using the flags `-O3 -std=c99 -funroll-all-loops -pedantic -Wall -Wextra`.

Instance	KeyGen	Encaps	Decaps
HQC-1	5 010	10 014	20 280
HQC-3	15 156	30 288	60 915
HQC-5	37 468	74 080	148 929

Table 10: 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 11. Our optimized implementation leverages AVX2 instructions and have been compiled using the flags `-O3 -std=c99 -funroll-all-loops -mavx -mavx2 -mpclmul -pedantic -Wall -Wextra`.

Instance	KeyGen	Encaps	Decaps
HQC-1	75	152	375
HQC-3	175	349	812
HQC-5	355	712	1 624

Table 11: 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. In addition, examples with intermediate values are also provided. 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

This section analyzes the Decoding Failure Rate (DFR) of HQC, *i.e.* the probability that the error vector $\mathbf{e}' = \mathbf{x}\cdot\mathbf{r}_2 - \mathbf{r}_1\cdot\mathbf{y} + \mathbf{e}$ is not corrected by the public concatenated code. It is organized in two parts. Sections 6.1.1 to 6.1.3 recall the original HQC's DFR analysis: the coordinates of $\mathbf{e}'$ are modeled as independent Bernoulli variables, *i.e.* as a binary symmetric channel (Section 6.1.1, Model 1); the DFR of the inner duplicated Reed-Muller code is upper bounded by a union bound over its codewords (Section 6.1.2); and the DFR of the concatenated code follows by a binomial argument over the inner blocks (Section 6.1.3). Simulations show that this first approach is conservative. Section 6.1.4 then presents a finer modeling introduced in [11] (Model 2), which keeps the exact weight of the integer products $\mathbb{Z}(\mathbf{x})\cdot\mathbb{Z}(\mathbf{r}_2)$ and $\mathbb{Z}(\mathbf{r}_1)\cdot\mathbb{Z}(\mathbf{y})$ and takes the dependence between the inner blocks into account exactly. This refined modeling is validated by simulations on the whole range of weights that can be observed. The DFR reported in Table 8 are based on Model 2. Finally, Section 6.1.5 shows that the deterministic tie-breaking rule used in dRM.Decode is covered by both analyses.

6.1.1 Analysis of the error vector distribution (Model 1)

We provide a precise analysis of the error distribution approximation following [5]. 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 $p^\star = P[e'_k = 1]$ 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 $p^\star$ for the weight distributions of $\mathbf{e}'$. In other words, we model the error vector as a binary symmetric channel with parameter $p^\star$. 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 equaling 1 can realistically only be $\leq p^\star$. 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:*

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

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

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

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 $\ell > \min(\omega, \omega_r)$ we clearly have $C_\ell = 0$. For $\ell \leq \min(\omega, \omega_r)$ 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 ω 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 $\mathbf{e}' = \mathbf{x} \cdot \mathbf{r}_2 - \mathbf{r}_1 \cdot \mathbf{y} + \mathbf{e} = (e'_0, \ldots, e'_{n-1})$. Then for any $k = 0 \ldots n-1$, 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 Equation (2): $p^\star = 2\tilde{p}(1-\tilde{p})(1-\frac{\omega_e}{n}) + ((1-\tilde{p})^2 + \tilde{p}^2)\,\frac{\omega_e}{n}$. The actual error satisfies $\omega(\mathbf{e}') \leq \min(2\omega\omega_r + \omega_e, n)$; under the independence assumption its weight is approximated by Binomial$(n, p^\star)$, *i.e.* for $0 \leq d \leq n$,

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

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 parameter $p^\star$. 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 $\mathbf{e}'$ rather than for the binomial approximation. This supports our model and the fact that computing the decoding failure probability with this binomial approximation permits to obtain an upper bound on the real DFR. This is confirmed hereafter by simulations with real weight parameters (but smaller lengths).

We consider the HQC-1 parameter set 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 $\ell = n - n_1 n_2'$ 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	67	76	17 669	17 664	0.3449

Table 12: Probability $p^\star$ 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 in Table 13.

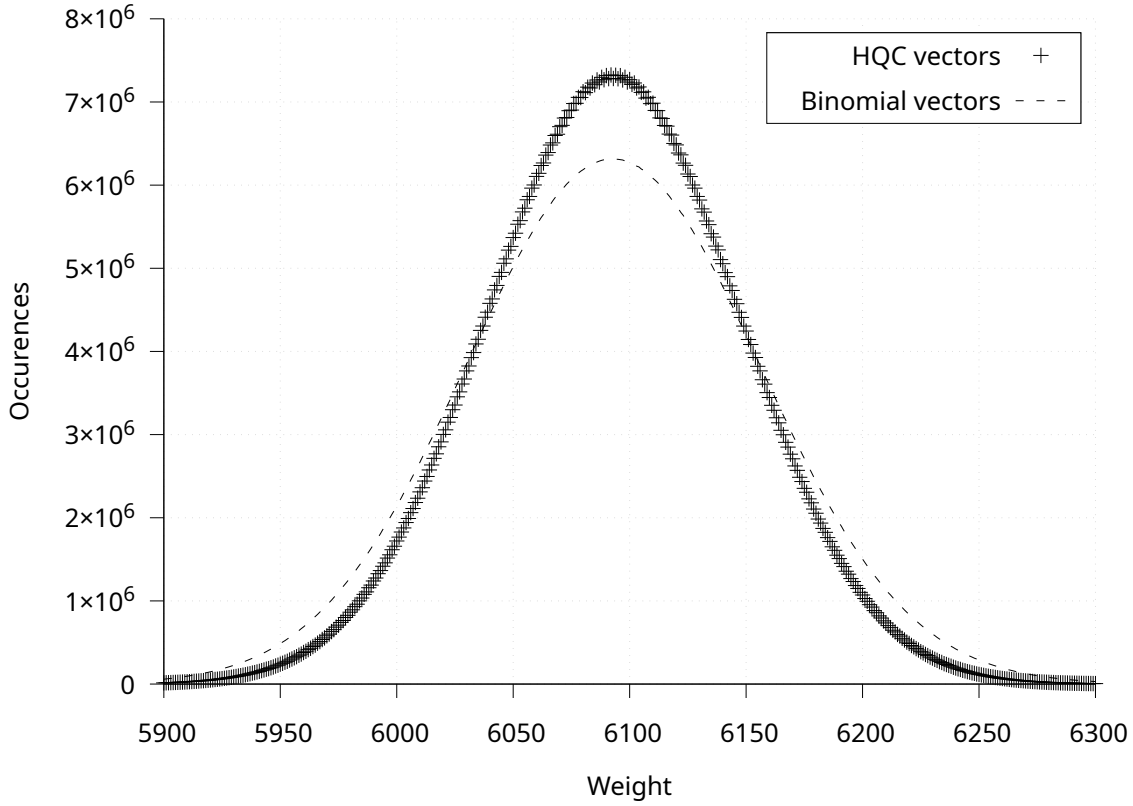


Figure 3: Comparison between error $\mathbf{e}'$ generated using HQC-1 parameter set and its binomial approximation. 1 billion samples were generated for both distributions.

	0.1%	0.01%	0.001%
Error vectors	6 261	6 297	6 330
Binomial approximation	6 289	6 328	6 372

Table 13: Weights exceeded by 0.1%, 0.01% and 0.001% of the vectors for HQC-1. Simulated error vectors computed from 10^8 samples and binomial approximation computed exactly.

6.1.2 Upper bound for the DFR of the RM code

One can only 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 an upper bound on the DFR in that case.

Proposition 6.1.3 (Simple Upper Bound for the DFR of the RM 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 $\mathcal{C}$ 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 $\mathcal{C}$ and $\omega(\mathbf{x})$ the weight of $\mathbf{x}$) is:

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

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

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

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 RM 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 still succeeds with probability exactly 1/2 when exactly two codewords minimize the distance to the received vector, whatever the tie-breaking rule of the decoder (see Lemma 6.2 below).

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 duplicated Reed-Muller code of dimension 8, minimal distance d_i and length $n = 2d_i$ 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 $\mathbf{c}' \in \mathcal{C}, \mathbf{c}' \neq \mathbf{c}, \mathbf{c}' \neq \mathbf{0}$, 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, exactly two codewords are at minimal distance from the received word and the decoder is correct with probability exactly $1/2$, whatever its tie-breaking rule, by Lemma 6.2 below; i.e. $P(E|A') = 1/2$. We have $P(E|B) = 1$ and writing $P(E|A'') \leqslant 1$, we have:

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

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 following union bounds over the codewords of weight d_i, which are the only ones that can contribute when $\omega < d_i$ (the all-one codeword satisfies $d(\mathbf{e}, \mathbf{1}) \leqslant \omega(\mathbf{e})$ only when $\omega(\mathbf{e}) \geqslant d_i$):

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

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 $d_i/2$ positions. When $\omega < d_i$, only pairs of weight-d_i codewords with intersecting supports contribute to $A_{\mathbf{c},\mathbf{c}'}$: two complementary codewords, or a codeword and the all-one vector, can both be at distance $\omega(\mathbf{e})$ from $\mathbf{e}$ only when $\omega(\mathbf{e}) = d_i$. For a pair of weight-d_i codewords with intersecting supports 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 for $\omega < d_i$.

It remains to deal with $\omega \geqslant d_i$, for which the all-one codeword and the complementary pairs are not covered by (5)–(7). For such ω the sum of the three terms inside the minimum defining $\mathfrak{A}_\omega$ exceeds $\binom{n}{\omega}$, so that $\mathfrak{A}_\omega = \binom{n}{\omega}$ and the bound reduces to the trivial $P(E_\omega) \leqslant$

$P(\omega(\mathbf{e}) = \omega)$. Indeed, writing $S_+ = \sum_{j>d_i/2} \binom{d_i}{j}\binom{d_i}{\omega-j}$ and $S_- = \sum_{j<d_i/2} \binom{d_i}{j}\binom{d_i}{\omega-j}$, the map $j \mapsto \omega - j$ sends $\{j < d_i/2\}$ injectively into $\{j > d_i/2\}$ when $\omega \geqslant d_i$, so $S_- \leqslant S_+$ and, by Vandermonde's identity,

$$\begin{aligned}\binom{n}{\omega} = S_+ + S_- + \binom{d_i}{d_i/2}\binom{d_i}{\omega - d_i/2} &\leqslant 2S_+ + \binom{d_i}{d_i/2}\binom{d_i}{\omega - d_i/2} \\ &\leqslant \frac{2}{255}\left[\frac{1}{2}255\binom{d_i}{d_i/2}\binom{d_i}{\omega - d_i/2} + 255S_+\right].\end{aligned}$$

Hence the all-one codeword and the complementary pairs need no separate treatment, and the formula holds for every ω. □

The previous formula permits to obtain an upper bound on the DFR of the internal code. When the error rate gets smaller, the bound becomes closer to the real value of the DFR. 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 6.1.4 for $[384, 8, 192]$ and $[640, 8, 320]$ duplicated Reed-Muller codes using the $p^\star$ values from provided parameter sets (Table 14).

Security level	$p^\star$	RM code	DFR (Prop. 6.1.4)	Observed DFR
NIST-1	0.3449	$[384, 8, 192]$	-9.82	-10.05
NIST-3	0.3636	$[640, 8, 320]$	-13.65	-13.88
NIST-5	0.3750	$[640, 8, 320]$	-10.72	-10.88

Table 14: Comparison between the observed Decryption Failure Rate and the formula from Proposition 6.1.4. Results are presented as $\log_2(\text{DFR})$.

6.1.3 DFR of the concatenated code (Model 1)

Using the upper bound p_i on the DFR 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)**.** *Under Model 1, let p_i be any upper bound on the DFR of the internal code (Proposition 6.1.3 or 6.1.4). The DFR of the concatenated code using a Reed-Solomon code $[n_1, k_1, d_1]_{\mathbb{F}_{256}}$ as the external code and a duplicated Reed-Muller code as the internal code can be upper bounded by:*

$$\sum_{l=t_1+1}^{n_1} \binom{n_1}{l} p_i^l (1-p_i)^{n_1-l}$$

where $d_1 = 2t_1 + 1$.

Proof. Under Model 1, the inner blocks see independent errors. With a randomized tie-breaking the inner decoding failures are therefore independent events, each of probability at most p_i, and the bounded-distance decoder of the Reed-Solomon code fails if and only if more than t_1 inner blocks fail. The binomial tail is increasing in p_i. Section 6.1.5 covers the deterministic tie-breaking of dRM.Decode. □

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.

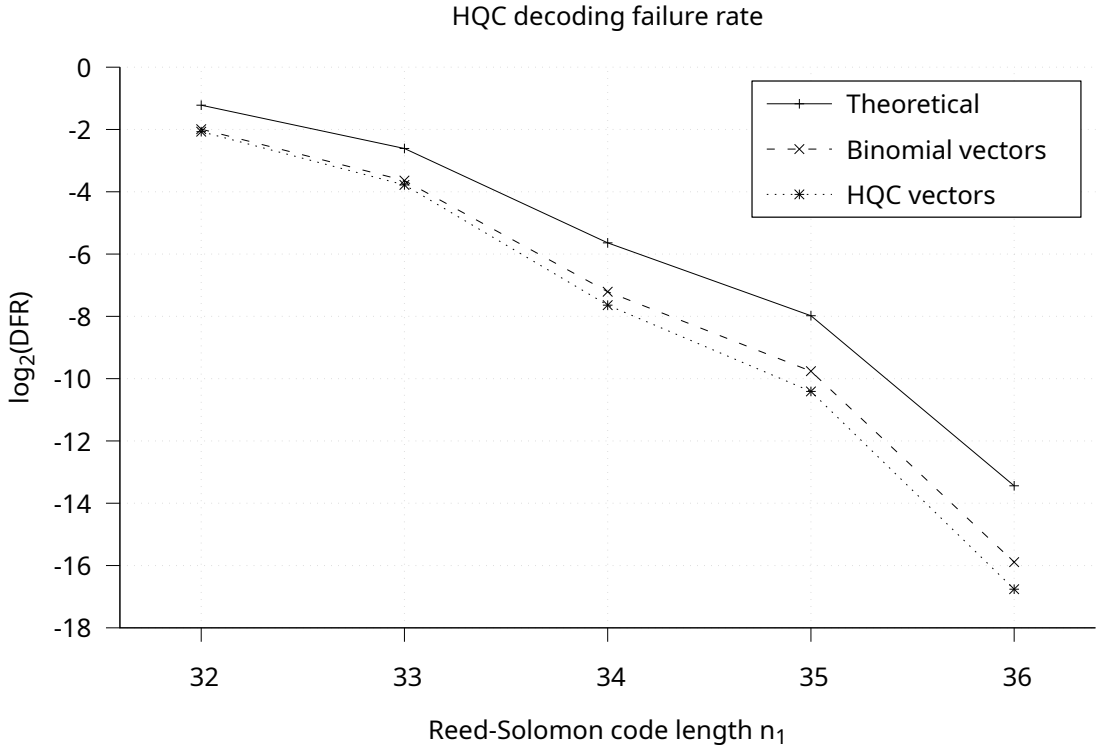


Figure 4: Comparison between the DFR from Theorem 6.1 (Theoretical, with the bound of Proposition 6.1.3) and the actual DFR of concatenated codes against approximation by a binary symmetric channel (Binomial) and against HQC error vectors (HQC). The horizontal axis represents n_1, the Reed-Solomon codeword length. The simulated parameters are derived from the HQC-1 parameter set ($\omega = 67$, $\omega_r = \omega_e = 76$), a duplicated $[384, 8, 192]$ Reed-Muller inner code and a $[n_1, 16]$ Reed-Solomon outer code.

6.1.4 Correlated model (Model 2) and the fundamental decoding domain

A refinement of the long-standing simplifying assumption that the coordinates of the error vector $\mathbf{e}' = \mathbf{x} \cdot \mathbf{r}_2 - \mathbf{r}_1 \cdot \mathbf{y} + \mathbf{e}$ behave as independent and identically distributed variables (Model 1) was proposed in [11]. This refinement (Model 2) features two independent improvements: it captures a weight constraint observable by lifting the polynomial products to the ring of integers, and it describes the set of correctable errors of the concatenated code through its fundamental decoding domain [15]. This subsection recalls Model 2 from [11] along with the resulting weight distribution of HQC errors and DFR bound.

Models. Let $\mathbb{Z}(\mathbf{x}), \mathbb{Z}(\mathbf{r}_2)$ be the liftings of $\mathbf{x}, \mathbf{r}_2$ to $\mathbb{Z}[X]/(X^n - 1)$. The k-th coefficient of $\mathbb{Z}(\mathbf{x}) \cdot \mathbb{Z}(\mathbf{r}_2)$ is a sum of ω coordinates of $\mathbb{Z}(\mathbf{r}_2)$ drawn without replacement, hence follows the hypergeometric distribution $h_j = \binom{\omega}{j}\binom{n-\omega}{\omega_r - j}\binom{n}{\omega_r}^{-1}$, $0 \le j \le \min(\omega, \omega_r)$. The probability $\tilde{p} = \sum_{j \text{ odd}} h_j$ that a coordinate of $\mathbf{x} \cdot \mathbf{r}_2$ equals 1, and the probability $p^\star$ that a coordinate of $\mathbf{e}'$ equals 1, are the ones of Propositions 6.1.1 and 6.1.2.

Definition 6.1.1 (Model 1: independence assumption)**.** *The model 1 assumes that* $\mathbf{e}' \sim \text{Bernouilli}(p^\star)^n$.

Definition 6.1.2 (Model 2: correlated-hypergeometric assumption)**.** *Let* $\mathbf{t}_1^{\mathbb{Z}} = (t_{1,0}^{\mathbb{Z}}, \dots, t_{1,n-1}^{\mathbb{Z}})$ *and* $\mathbf{t}_2^{\mathbb{Z}} = (t_{2,0}^{\mathbb{Z}}, \dots, t_{2,n-1}^{\mathbb{Z}})$ *be two independent integer vectors of* $\mathbb{Z}^n$*, each sampled from* $\text{Hypergeometric}(n, \omega, \omega_r)^n$ *conditioned on* $\sum_i t_{1,i}^{\mathbb{Z}} = \sum_i t_{2,i}^{\mathbb{Z}} = \omega\,\omega_r$*; these vectors model the integer products* $\mathbb{Z}(\mathbf{x}) \cdot \mathbb{Z}(\mathbf{r}_2)$ *and* $\mathbb{Z}(\mathbf{r}_1) \cdot \mathbb{Z}(\mathbf{y})$*. Let* $\mathbf{t}_1, \mathbf{t}_2 \in \mathbb{F}_2^n$ *be the coordinate-wise reductions modulo 2,* $\mathbf{t}_j = \mathbf{t}_j^{\mathbb{Z}} \bmod 2$*, which model the binary products* $\mathbf{x} \cdot \mathbf{r}_2$ *and* $\mathbf{r}_1 \cdot \mathbf{y}$*, and let* $\mathbf{e} \xleftarrow{\$} \mathcal{R}_{\omega_e}$*. The model 2 models the HQC error as* $\mathbf{e}' = \mathbf{t}_1 + \mathbf{t}_2 + \mathbf{e}$.

One should note that $\mathbf{t}_1, \mathbf{t}_2$, $\mathbf{t}_1^{\mathbb{Z}}, \mathbf{t}_2^{\mathbb{Z}}$ denote the aforementioned vectors, while the scalar t_1 denotes the error-correcting capacity of the RS code hence the two notations are unrelated.

The two models have the same coordinate marginals up to the $O(1/n)$ perturbation induced by the sum constraint (which is why the mean of Model 2 in Table 15 is marginally larger than $n_1 n_2' p^\star$). They differ in the correlations: the constraint $\sum_i t_{j,i}^{\mathbb{Z}} = \omega\,\omega_r$ is exact for the actual products, since $(\mathbb{Z}(\mathbf{x}) \cdot \mathbb{Z}(\mathbf{r}_2))(1) = \omega\,\omega_r$, and it makes the integer coefficients slightly negatively correlated. Model 2 reproduces this correlation and the exact parity of $\omega(\mathbf{x} \cdot \mathbf{r}_2)$, while Model 1 ignores both. The effect on the weight is of first order. For HQC-1, the variance of $\omega(\mathbf{x} \cdot \mathbf{r}_2)$ is 3035.4 under Model 1, 1403.3 under Model 2 and 1403.7 in simulations (10^8 samples). Model 2 thus predicts lighter tails for $\omega(\mathbf{e}')$ and a smaller DFR, and explains why Model 1 was observed to be conservative (Section 6.1.1).

Analysis of the error vector distribution (Model 2). Under Model 2, $\mathbf{e}'$ and its truncation to $n_1 n_2'$ coordinates are radially symmetric. All vectors of a given weight are equally likely [11, Lemmas 2, 3] hence the model is entirely described by weight distributions, obtained in three steps.

Weight of $\mathbf{t}_i$. Following [42], let $H(z) = \sum_j h_j z^j$ and $O(z), E(z)$ its odd and even parts. Counting integer vectors with m odd coordinates and sum $s = \omega\,\omega_r$ gives

$$\Pr\big(\omega(\mathbf{t}_i) = m\big) = \binom{n}{m} \frac{[z^s]\, O(z)^m E(z)^{n-m}}{[z^s]\, H(z)^n}, \tag{8}$$

which is the binomial distribution of Model 1 multiplied by a correction factor accounting for the sum constraint [11, Equation (5)]. Numerically, after the change of variable $z \mapsto e^\tau z$ with τ chosen so that the tilted mean equals s, each coefficient $[z^s]$ becomes the probability that a sum of n independent variables equals its mean, which is computed accurately in double precision by FFT (multi-precision arithmetic [11] or a saddle-point approximation [35] can also be used).

Sums. If $\mathbf{a}, \mathbf{b}$ are independent, radially symmetric, of weights a and b, the size i of the intersection of their supports is hypergeometric, so that $\omega(\mathbf{a} + \mathbf{b}) = a + b - 2i$ with

$$\Pr(i) = \binom{a}{i}\binom{n-a}{b-i}\binom{n}{b}^{-1}.$$

Applied to $\mathbf{t}_1 + \mathbf{t}_2$, then to $(\mathbf{t}_1 + \mathbf{t}_2) + \mathbf{e}$ with $\omega(\mathbf{e}) = \omega_e$, this gives the weight distribution of $\mathbf{e}'$ [11, Proposition 1].

Truncation. With $\ell = n - n_1 n_2'$,

$$\Pr\big(\omega(\mathsf{Truncate}(\mathbf{e}', \ell)) = u\big) = \sum_W \binom{n_1 n_2'}{u}\binom{\ell}{W-u}\binom{n}{W}^{-1} \Pr\big(\omega(\mathbf{e}') = W\big) \quad [11, \text{Lemma 3}].$$

Table 15 and Figure 5 compare both models with simulations for HQC parameters. Model 2 matches the simulations in variance and in the tails (its mean is marginally larger than the exact mean $n_1 n_2' p^\star$, which is conservative), whereas Model 1 overestimates the variance. The same holds in the tails: for HQC-1 as the error vectors are $6261, 6295, 6325$ in simulations, $6262, 6296, 6326$ under Model 2 and $6289, 6329, 6363$ under Model 1.

Instance	Model/Simulation	Mean	Variance	D_{KL} (bit)
HQC-1	Simulation	6 092.9	2 980.9	–
	Model 1	6 092.9	3 991.3	$2.80 \cdot 10^{-2}$
	Model 2	6 093.1	2 980.9	$1.3 \cdot 10^{-5}$
HQC-3	Simulation	13 030.2	6 540.0	–
	Model 1	13 030.2	8 292.9	$1.88 \cdot 10^{-2}$
	Model 2	13 030.4	6 539.6	$1.0 \cdot 10^{-5}$
HQC-5	Simulation	21 598.5	10 980.5	–
	Model 1	21 598.5	13 499.6	$1.44 \cdot 10^{-2}$
	Model 2	21 598.6	10 980.7	$1.0 \cdot 10^{-5}$

Table 15: Weight of the truncated error using parameters from Table 8. Simulations computed from 10^8 samples per parameter set and compared with respect to Models 1 and 2. D_{KL} is the Kullback–Leibler divergence of the empirical distribution from the model.

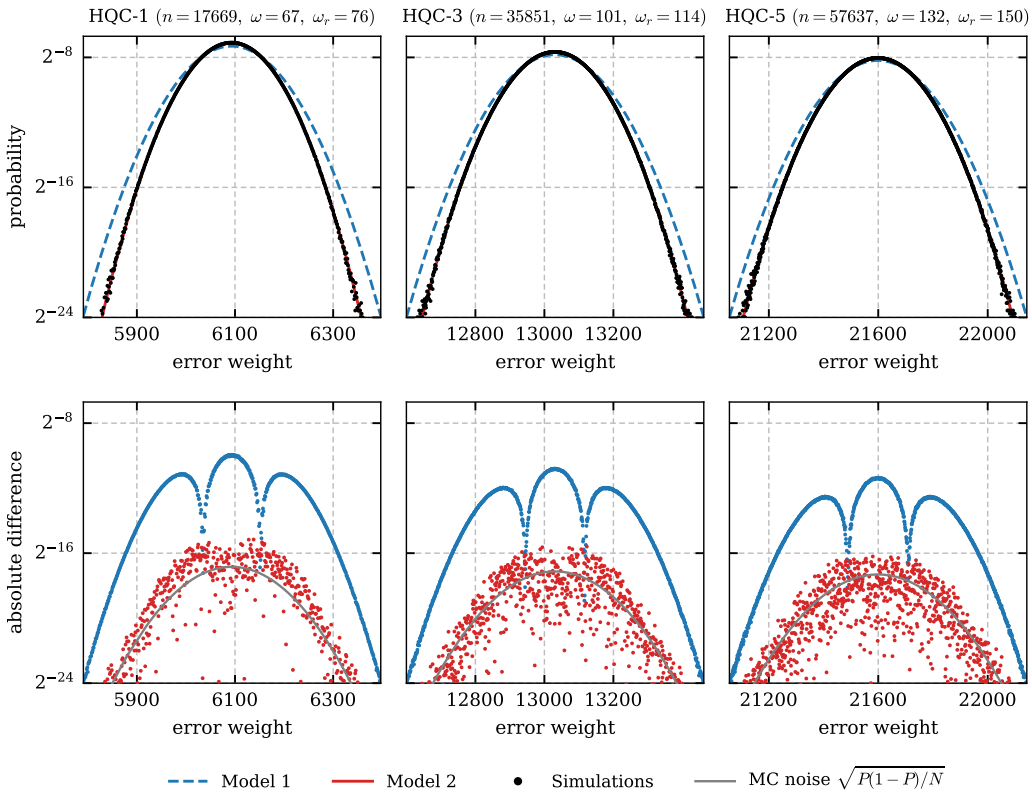


Figure 5: Weight distribution of the truncated error under Models 1 and 2 and in simulations ($N = 10^8$ samples per parameter set). Top: probabilities. Bottom: absolute difference with the simulations; the grey line is the Monte-Carlo standard deviation $\sqrt{P(1-P)/N}$. As in [11, Figure 2], both vertical axes are cut at 2^{-24} and the horizontal range is the one where Model 1 exceeds 2^{-24}.

Fundamental decoding domain. Assuming, as in [11], that the decoder of the concatenated code $\mathcal{C}$ is translation invariant, decryption succeeds if and only if the truncated error lies in the fundamental decoding domain $\mathcal{F}_{\mathcal{C}} = \{\mathbf{v} : \mathcal{C}.\mathsf{Decode}(\mathbf{v}) = \mathbf{0}\}$. For a radially symmetric error, the DFR is thus $\sum_u \Pr[\omega(\mathsf{Truncate}(\mathbf{e}', \ell)) = u]\,\mathrm{DFR}(u)$, where $\ell = n - n_1 n_2'$ is the number of truncated coordinates and $\mathrm{DFR}(u)$ is the proportion of weight-u vectors outside $\mathcal{F}_{\mathcal{C}}$. Let $\mathfrak{A}_j$ be the quantity $\mathfrak{A}_\omega$ of Proposition 6.1.4 for $\omega = j$, which is an upper bound on the expected number of weight-j patterns not decoded by the inner code (because of the factors $1/2$ accounting for two-way ties, $\mathfrak{A}_j$ need not be an integer), $U(X) = \sum_j \mathfrak{A}_j X^j$ and $L(X) = \sum_j \left(\binom{n_2'}{j} - \mathfrak{A}_j\right) X^j$. Counting the weight-$u$ errors for which more than t_1 inner blocks may fail (with $d_1 = 2t_1 + 1$) gives [35, Theorem 4.7], [11, Proposition 2]

$$\mathrm{DFR} \le \sum_{u=0}^{n_1 n_2'} \Pr\big(\omega(\mathsf{Truncate}(\mathbf{e}', \ell)) = u\big) \binom{n_1 n_2'}{u}^{-1} \sum_{\lambda=t_1+1}^{n_1} \binom{n_1}{\lambda} [X^u]\, U(X)^\lambda L(X)^{n_1-\lambda}. \quad (9)$$

Unlike Theorem 6.1, this bound takes the dependence between the n_1 blocks into account exactly. When the binomial weight distribution of Model 1 is plugged into (9), one recovers exactly the bound of Theorem 6.1 with the p_i of Proposition 6.1.4, the only change is the error model.

DFR of the concatenated code (Model 2). The DFR of HQC reported in Tables 8 and 16 are based on this refined DFR analysis and are obtained in two steps. Firstly, the weight distribution of the truncated error is computed under Model 2 as described above (Equation (8), sums and truncation); secondly, it is combined with Equation (9), where the numbers $\mathfrak{A}_j$ are given by Proposition 6.1.4.

Instance	Security	DFR
HQC-1	NIST-1	$2^{-127.9}$
HQC-3	NIST-3	$2^{-196.4}$
HQC-5	NIST-5	$2^{-257.0}$

Table 16: DFR of HQC under Model 2 using parameters from Table 8.

Related works. Three independent and concurrent works [11, 35, 2] revisited the DFR analysis of HQC. They follow different approaches namely [11] introduces Model 2 and the fundamental decoding domain, [35] computes the exact variance of the syndrome weight in quasi-cyclic codes and uses a Gaussian approximation, and [2] relies on observed failure rates of the inner decoder. Nonetheless, they converge to the same conclusion: the initial DFR analysis (Model 1, Sections 6.1.1 to 6.1.3) is conservative, and the refined analyses all yield the same type of gain over it. The analysis adopted in Table 16 is the one of [11], combined with the bound of Proposition 6.1.4 on the inner code.

6.1.5 Deterministic tie-breaking for the RM code

The bounds of Proposition 6.1.4, Theorem 6.1 and Equation (9) count a block whose received word is at equal minimal distance from exactly two codewords as failing with probability 1/2. For convenience of implementation, it may be desirable however to use a deterministic tie-breaking procedure. Indeed, dRM.Decode breaks such ties deterministically using first peak of the Hadamard transform. The following lemma shows that the factor 1/2 is exact whatever the tie-breaking rule, because the distribution of the transmitted inner codewords is uniform.

Lemma 6.2 (Tie-breaking)**.** *Let* $\mathbf{c}$ *be uniformly distributed in the inner code* $\mathcal{C}$*, and let* $\mathbf{e}$ *be independent of* $\mathbf{c}$*. Assume that* $\mathbf{e}$ *is radially symmetric,* i.e. *conditionally on* $\omega(\mathbf{e}) = w$*,* $\mathbf{e}$ *is uniformly distributed over all binary vectors of weight* w*. Equivalently,*

$$P[\mathbf{e} = \mathbf{u}] = P[\mathbf{e} = \mathbf{v}] \quad \textit{whenever} \quad \omega(\mathbf{u}) = \omega(\mathbf{v}).$$

This property holds for the binary symmetric channel of Model 1 (Section 6.1.1) and for the error distribution defined by Model 2 (Section 6.1.4). Let D *be any decoder returning a codeword at minimal distance from* $\mathbf{y} = \mathbf{c} + \mathbf{e}$*, ties being broken by an arbitrary rule that depends only on* $\mathbf{y}$ *for instance the smallest index (as in* dRM.Decode.FindPeaks*) or a uniformly random choice. Let* $T(\mathbf{y})$ *be the set of codewords at minimal distance from* $\mathbf{y}$*. Then*

$$P\big(D(\mathbf{y}) = \mathbf{c} \;\big|\; \mathbf{y},\ \mathbf{c} \in T(\mathbf{y})\big) = \frac{1}{|T(\mathbf{y})|}.$$

In particular $P(E|A') = 1/2$ *exactly, whatever the tie-breaking rule, and Proposition 6.1.4 applies to the* dRM.Decode *deterministic decoder.*

Proof. Conditionally on $\mathbf{y}$, the posterior law of $\mathbf{c}$ is proportional to $P(\mathbf{c} = \mathbf{c}')\, P(\mathbf{e} = \mathbf{y} + \mathbf{c}')$, $\mathbf{c}' \in \mathcal{C}$. The first factor is constant and, by radial symmetry, the second depends only on $\omega(\mathbf{y} + \mathbf{c}') = d(\mathbf{y}, \mathbf{c}')$, which takes the same value for all $\mathbf{c}' \in T(\mathbf{y})$. The posterior is therefore uniform on $T(\mathbf{y})$, while $D(\mathbf{y})$ is, once $\mathbf{y}$ is fixed, either a fixed element of $T(\mathbf{y})$ or a uniformly random one drawn independently of $\mathbf{c}$; in both cases it equals $\mathbf{c}$ with probability $1/|T(\mathbf{y})|$. The event A' is the event $\{\mathbf{c} \in T(\mathbf{y}),\ |T(\mathbf{y})| = 2\}$, and on this event $\omega(\mathbf{e}) = d(\mathbf{y}, \mathbf{c})$ is a function of $\mathbf{y}$; hence $P(E \cap A'_\omega) = \frac{1}{2} P(A'_\omega)$ for every ω. Uniformity of $\mathbf{c}$ holds in HQC as the inner codeword encodes one coordinate of the Reed-Solomon encoding of the uniform message $\mathbf{m}$, and every coordinate of a RS codeword is a surjective linear function of $\mathbf{m}$. □

Lemma 6.2 shows that Theorem 6.1 and Equation (9) still hold if we assume that every inner codeword is uniformly distributed and if there are all independent. However, we only have uniform distribution for the codewords of the concatenated code hence the uniform inner codeword blocks are s coupled through the Reed-Solomon code. Since any k_1 symbols of a uniform Reed-Solomon codeword are independent and uniform, this coupling has no effect as long as at most k_1 blocks are tied with their transmitted codeword simultaneously. Under Model 1, the probability that more than k_1 blocks are tied is at most $\binom{n_1}{k_1+1} P_{\text{tie}}^{k_1+1}$,

where P_{tie}, the probability that an inner block is tied with its transmitted codeword, is bounded by (6) summed over ω (capped at $\binom{n}{\omega}$) namely below 2^{-133} for HQC-1 and 2^{-296} for HQC-3 and HQC-5. Under Model 2 the blocks are not independent and this probability has to be computed jointly over the blocks, as in Equation (9); it remains far below the DFR and does not affect Table 16.

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 ($\mathsf{SFO}^{\not\perp}$) [19, 26, 22] 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 3.3). 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 6.3. *For any* IND-CPA *adversary* $\mathcal{A}$ *against the* HQC-PKE *scheme, there exists adversaries* $\mathcal{B}_1$ *and* $\mathcal{B}_2$ *such that:*

$$\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{HQC\text{-}PKE}}(\mathcal{A}) \leq \mathsf{Adv}_{\mathsf{2\text{-}DQCSD\text{-}P}}(\mathcal{B}_1) + \mathsf{Adv}_{\mathsf{3\text{-}DQCSD\text{-}PT}}(\mathcal{B}_2). \tag{10}$$

Proof of Theorem 6.3. 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. Return $(b = b')$

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. $\mathbf{e} \leftarrow\!\$\ \mathbb{F}_{2,\omega_e}^{n}, (\mathbf{r}_1, \mathbf{r}_2) \leftarrow\!\$\ \mathbb{F}_{2,\omega_r}^{n} \times \mathbb{F}_{2,\omega_r}^{n}$
   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. Return $(b = b')$

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 ($\mathsf{SFO}^{\not\perp}$) [19, 26, 22] transform to HQC-PKE. In this section, we discuss the IND-CCA2 security of HQC-KEM.

Definition 6.2.1 (δ-correct PKE [26])**.** *A* $\mathsf{PKE} = (\mathsf{PKE.Keygen}, \mathsf{PKE.Encrypt}, \mathsf{PKE.Decrypt})$ *is* δ*-correct if:*

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

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

$$\Pr = \left[\mathsf{KEM.Decaps}(\mathsf{dk}_{\mathsf{KEM}}, \mathbf{c}_{\mathsf{KEM}}) \neq K \;\middle|\; \begin{array}{l}(\mathsf{ek}_{\mathsf{KEM}}, \mathsf{dk}_{\mathsf{KEM}}) \leftarrow \mathsf{KEM.Keygen}();\\ (K, \mathbf{c}_{\mathsf{KEM}}) \leftarrow \mathsf{KEM.Encaps}(\mathsf{ek}_{\mathsf{KEM}})\end{array}\right] \leq \delta. \quad (12)$$

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 11 simplifies to

$$\Pr\left[\mathsf{PKE.Decrypt}(\mathsf{dk}_{\mathsf{PKE}}, \mathbf{c}_{\mathsf{PKE}}) \neq \mathbf{m} \mid \mathbf{c}_{\mathsf{PKE}} \leftarrow \mathsf{PKE.Encrypt}(\mathsf{ek}_{\mathsf{PKE}}, \mathbf{m})\right] \leq \delta. \quad (13)$$

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

$$\Pr\left[\omega\left(\mathbf{x}\cdot\mathbf{r}_2-\mathbf{r}_1\cdot\mathbf{y}+\mathbf{e}\right)>\Delta \;\middle|\; \begin{array}{l} \mathsf{ctx}_{\mathsf{PKE.dk}} \leftarrow \mathsf{XOF.Init}(); \\ \mathsf{ctx}_{\mathsf{PKE.dk}} \leftarrow \mathsf{XOF.Absorb}(\mathsf{ctx}_{\mathsf{PKE.dk}}, \mathsf{seed}_{\mathsf{PKE.dk}}, |\mathsf{seed}|); \\ (\mathsf{ctx}_{\mathsf{PKE.dk}}, \mathbf{y}) \leftarrow \mathsf{SampleFixedWeightVect}_{\$}(\mathsf{ctx}_{\mathsf{PKE.dk}}, \omega); \\ (\mathsf{ctx}_{\mathsf{PKE.dk}}, \mathbf{x}) \leftarrow \mathsf{SampleFixedWeightVect}_{\$}(\mathsf{ctx}_{\mathsf{PKE.dk}}, \omega); \\ \mathsf{ctx}_{\theta} \leftarrow \mathsf{XOF.Init}(); \\ \mathsf{ctx}_{\theta} \leftarrow \mathsf{XOF.Absorb}(\mathsf{ctx}_{\theta}, \theta, |\mathsf{seed}|); \\ (\mathsf{ctx}_{\theta}, \mathbf{r}_2) \leftarrow \mathsf{SampleFixedWeightVect}_{\$}(\mathsf{ctx}_{\theta}, \omega_r); \\ (\mathsf{ctx}_{\theta}, \mathbf{e}) \leftarrow \mathsf{SampleFixedWeightVect}_{\$}(\mathsf{ctx}_{\theta}, \omega_e); \\ (\mathsf{ctx}_{\theta}, \mathbf{r}_1) \leftarrow \mathsf{SampleFixedWeightVect}_{\$}(\mathsf{ctx}_{\theta}, \omega_r) \end{array}\right] \leq \delta. \tag{14}$$

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 [22] and Theorem 6.3.

Theorem 6.4. *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:*

$$\begin{aligned} \mathsf{Adv}^{\mathsf{IND\text{-}CCA2}}_{\mathsf{HQC\text{-}KEM}}(\mathcal{A}) \;\leq\; & \frac{1}{2^{|k|}\cdot 2^{|\mathsf{salt}|}} + \frac{3q_{RO}}{2^{|k|}} + (q_{RO}+q_D)\cdot\delta \\ & + 2\cdot\left(\mathsf{Adv}_{\mathsf{2\text{-}DQCSD\text{-}P}}(\mathcal{B}_1) + \mathsf{Adv}_{\mathsf{3\text{-}DQCSD\text{-}PT}}(\mathcal{B}_2)\right). \end{aligned} \tag{15}$$

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 [41].

Proposition 6.2.1 ([41])**.** *When* $\mathsf{XOF.GetBytes}(\mathsf{ctx}, \mathsf{rnd_bytes}, B)$ *behaves as a random oracle which yields uniformly distributed integers* $0 \leq x < 2^B$ *for any integer* $B > 0$*, one has:*

$$\prod_{i=0}^{w-1}\left(1-\frac{n_i}{2^B}\right) = \tau^{\omega}_{min} \leq \frac{\Pr\left[\mathbf{e} \mid \mathbf{e} \leftarrow \mathsf{SampleFixedWeightVect}(\mathsf{ctx}_{\mathbf{e}}, \omega)\right]}{\Pr\left[\mathbf{e} \mid \mathbf{e} \leftarrow \mathsf{SampleFixedWeightVect}_{\$}(\mathsf{ctx}_{\mathbf{e}}, \omega)\right]} \leq \tau^{\omega}_{max} = \prod_{i=0}^{w-1}\left(1+\frac{(n-i)-n_i}{2^B}\right) \tag{16}$$

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 17.

The following lemma (Equation 5 from [41]) 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 6.5. *For any real-valued random variable* $V : \mathcal{R}_{\omega} \to \mathbb{R}$*, one has:*

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

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	67	0.99986	1.00014	76	0.99985	1.00016
NIST-3	35,851	101	0.99957	1.00042	114	0.99952	1.00047
NIST-5	57,637	132	0.99913	1.00090	150	0.99900	1.00101

Table 17: Bias between the uniform distribution and the output of Algorithm SampleFixedWeightVect (using $B = 32$) for the vectors of weight ω and $\omega_r = \omega_e$.

Impact of the security proof. Following [41], we explain how to modify the Equation 15 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 $(\mathbf{r}_1, \mathbf{r}_2, \mathbf{e})$ are sampled using SampleFixedWeightVect rather than the uniform fixed-weight sampler $\mathsf{SampleFixedWeightVect}_{\$}$, while $(\mathbf{x}, \mathbf{y})$ are sampled uniformly, δ must be such that:

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

  Using Lemma 6.5, the above probability increases by at most $(\tau^{\omega_r}_{max})^3$;
- 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 6.5, the advantage related to this term increases by a factor at most $(\tau^{\omega_r}_{max})^3$.

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 [38] and improved later by Stern [43], then Dumer [16]. Recent works [33, 6, 34, 14] 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 [13].

Specific structural attacks. Quasi-cyclic codes have a special structure which may potentially open the door to specific structural attacks. A first generic attack is the DOOM attack [40] 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 [25, 31, 40], and are especially efficient when the polynomial $x^n - 1$ has many low degree factors. These attacks become inefficient as soon as $x^n - 1$ has only two irreducible factors of the form $(x - 1)$ and $x^{n-1} + x^{n-2} + ... + x + 1$, which is the case when n is prime and q generates the multiplicative group $(\mathbb{Z}/n\mathbb{Z})^*$. Such numbers are known up to very large values. We consider such primitive n for our parameters.

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 correspond to 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) [9]. Best known attacks include the works from [12, 10, 17, 33, 6, 34, 14] and for quantum attacks, the work of [9]. 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 [13]. 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 [40], 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] Alessandro Annechini, Alessandro Barenghi, and Gerardo Pelosi. On the Decoding Failure Rate of HQC. Cryptology ePrint Archive, Paper 2026/795, 2026.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

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

[8] 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.

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

[10] 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.

[11] Sebastian Bitzer, Jean-Christophe Deneuville, Emma Munisamy, Bharath Purtipli, Stefan Ritterhoff, and Antonia Wachter-Zeh. HQC beyond the standard: ciphertext compression and refined DFR analysis. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 544–574. Springer, 2026.

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

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

[14] Kévin Carrier, Valérian Hatey, Laura Luzzi, and Jean-Pierre Tillich. Multilevel Amortized Gaussian Elimination in Information-Set Decoding: Applications to HQC and PCG. Cryptology ePrint Archive, Paper 2026/1498, 2026.

[15] Thomas Debris-Alazard, Léo Ducas, and Wessel P. J. van Woerden. An Algorithmic Reduction Theory for Binary Codes: LLL and More. *IEEE Transactions on Information Theory*, 68(5):3426–3444, 2022.

[16] 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.

[17] 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.

[18] G Forney. On decoding BCH codes. *IEEE Transactions on Information Theory*, 11(4):549–557, 1965. https://doi.org/10.1109/TIT.1965.1053825.

[19] 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.

[20] 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.

[21] 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.

[22] Lewis Glabush, Kathrin Hövelmanns, and Douglas Stebila. Tight multi-challenge security reductions for key encapsulation mechanisms. Cryptology ePrint Archive, Report 2025/343, 2025.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

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

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

[30] Zhen Liu, Yanbin Pan, and Tianyuan Xie. Breaking the hardness assumption and IND-CPA security of HQC submitted to NIST PQC project. In *International Conference on Cryptology and Network Security*, pages 344–356. Springer, 2018.

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

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

[33] 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.

[34] 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.

[35] Antoine Mesnard, Jean-Pierre Tillich, and Valentin Vasseur. The syndrome weight distribution in quasi-cyclic codes, applications to BIKE and HQC. In *International Conference on Post-Quantum Cryptography*, pages 156–187. Springer, 2026.

[36] 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.

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

[38] 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/.

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

[40] 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.

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

[42] Nicolas Sendrier and Valentin Vasseur. On the decoding failure rate of QC-MDPC bit-flipping decoders. In Jintai Ding and Rainer Steinwandt, editors, *Post-Quantum Cryptography - 10th International Conference, PQCrypto 2019*, pages 404–416. Springer, Cham, 2019.

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