Hamming Quasi-Cyclic (HQC)

Fourth round version

Updated version 19/02/2025

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

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

Principal Submitters (by alphabetical order):

- Carlos Aguilar Melchor (SandboxAQ)
- Nicolas Aragon (Univ. of Limoges)
- Slim Bettaieb (TII)
- Loïc Bidoux (TII)
- Olivier Blazy (Ecole Polytechnique)
- Jurjen Bos (Worldline)
- Jean-Christophe Deneuville (ENAC)
- Arnaud Dion (ISAE Supaero)
- Philippe Gaborit (Univ. of Limoges)
- Jérôme Lacan (ISAE Supaero)
- Edoardo Persichetti (Florida Atlantic Univ.)
- Jean-Marc Robert (Univ. of Toulon)
- Pascal Véron (Univ. of Toulon)
- Gilles Zémor (Univ. of Bordeaux)

Inventors: Same as submitters

Developers: Same as submitters

Owners: Same as submitters

Main contact

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

Backup point of contact

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

Signatures

Digital copies of the signed statements were provided to NIST in the original submission on Nov. 30, 2017. The paper versions have been provided to NIST at the First PQC Standardization Conference on Apr. 13, 2018.
Paper versions of the signed statements for the team members added in round 3 will be provided to NIST during the next PQC Standardization Conference.

1 History of updates on HQC

1.1 Updates for February 19, 2025

- We fixed an implementation bug in both the reference and optimized versions of HQC, where an indexing error led to an incorrect interpretation of the public key during the decapsulation phase. This resulted in the decapsulation function returning an incorrect shared secret when given malformed ciphertext. The vulnerability was discovered by Célian Glénaz and Dahmun Goudarzi and brought to our attention by Spencer Wilson and Douglas Stebila.

1.2 Updates for October 30, 2024

- We have modified the order of variable sampling in both key generation and encryption, following the recommendations from the authors of [2], who demonstrated that this change results in significant performance gains in hardware implementation.

- We have updated the countermeasure against multi-target attacks by modifying the key binding process. Specifically, we changed the hashing mechanism to include only the first 32 bytes of the public key, instead of the entire public key, along with the message and a salt.

We have updated the implementation to reflect these improvements, and the Known Answer Test (KAT) files have been updated accordingly. The following table highlights the performance improvements resulting from the updates, with encapsulation improved by approximately 10 to 13% and decapsulation improved by approximately 3 to 12%.

Instance	KeyGen	Encaps	Decaps
hqc-128	75	177	323
hqc-192	175	404	669
hqc-256	356	799	1427

Table 1: Performance in kilocycles of the optimized implementation using `AVX2` instructions for different instances of HQC.

1.3 Updates for February the 23rd 2024

- We have updated our IND-CPA security proof and definitions of hard problems in order to fix an issue regarding the arguments of indistinguishability between Game 3 and 4 due to a lack of technical update after the introduction of truncation. One should note that neither the design, implementation nor parameters of the scheme are affected by this modification.

- We have updated our implementation replacing the modulo operator by the Barrett reduction as a counter-measure against the timing attack identified in [31].

1.4 Updates for April the 30th 2023

- We now consider the HHK transform with implicit rejection into our scheme. We provide an IND-CCA2 security proof in the HHK framework for this modification. We have updated the implementation to reflect the aforementioned changes and have also updated the KATs to align with these improvements..

- We provide a security analysis indicating that sampling vectors of small weights non-uniformly, yet close to uniform, has a negligible effect on HQC's IND-CCA2 security, following the approach of Nicolas Sendrier ("Secure Sampling of Constant-Weight Words, Application to BIKE". IACR Cryptol. ePrint Arch. 2021: 1631 (2021)).

1.5 Updates for October the 1st 2022

- **Multi-ciphertext attack**: for HQC-128 the ciphertext is generated deterministically from a seed of 128 bits, which allows a straightforward multi-ciphertext attack that allows an attacker to recover the shared secret for one out of N ciphertexts at a cost of $2^{128}/N$. This attack does not contradict the claim of category 1 IND-CCA2 security for this parameter set of HQC, meanwhile since it is an undesirable property, we modified our scheme at negligible cost by incorporating a public salt value into the ciphertext (for all security levels). So that the randomness θ is now computed from a salt together with the public key.

  $\theta = \text{SHAKE256-512}(\mathbf{m} \| \mathsf{pk} \| salt)$

  We modified our scheme and the proof accordingly.

- **Counter-measure to a timing attack**: In the paper:

  "Don't Reject This: Key-Recovery Timing Attacks Due to Rejection-Sampling in HQC and BIKE" by Qian Guo, Clemens Hlauschek, Thomas Johansson, Norman Lahr, Alexander Nilsson, and Robin Leander Schroder published a CHES 2022, the authors explain how it is possible to use the randomness generator of the small weight words to produce an attack. In order to counter this attack we included the counter-measure described by Nicolas Sendrier (Algo 5) in:

  "Secure Sampling of Constant-Weight Words, Application to BIKE". IACR Cryptol. ePrint Arch. 2021: 1631 (2021)

  In practice this counter-measures implies a loss of a few percentages points in our performance.

- **Constant time additional implementation**: we included a pure C constant time (not optimized) implementation.
- The hardware implementation does not currently incorporate the aforementioned mitigation related to multi-ciphertext attacks. We will provide an updated version in the next release.

We recall the main parameters of HQC and the last performances (in kilocycles):

	Public key size	Ciphertext size	KeyGen	Encaps	Decaps	DFR
hqc-128	2,249	4,497	87	204	362	$< 2^{-128}$
hqc-192	4,522	9,042	204	465	755	$< 2^{-192}$
hqc-256	7,245	14,485	409	904	1505	$< 2^{-256}$

1.6 Updates for June the 6th 2021

- Domain separation and the randomness generation are now performed using a KECCAK core rather than the functions `randombytes` and `seedexpander` provided by NIST.
- We provide a full HLS-compatible hardware implementation with two flavors: performance oriented and compactness oriented.
- A design of a common hardware-software architecture resulting in the same outputs for both the hardware and software reference implementation.

1.7 Updates for October the 1st 2020

- Since the RMRS decoder is strictly better than the BCH-Repetition decoder, we now only consider the RMRS decoder version of the HQC algorithm and we do not consider the BCH-Repetition decoder any more.
- In order to fit more precisely the Level 1 and 3 of NIST security categories, the sizes of the decoded messages for the concatenated RMRS code are set to the adequate security levels (*i.e.* dimension 128 and dimension 192 rather than 256 for level 1 and level 3), for Level 1 and Level 3 this modification improves on the decoding capacity of the RMRS code and hence improves parameters.
- We improved the theoretical lower bound for the Reed-Muller decoder (approaching optimality), which permits to lower our theoretical bound for the DFR and hence also improve on parameters (section 2.5).
- Based on the two previous improvements, we provide new sets of parameters, and we obtain the following sizes (in bytes) and performances (in kilocycles):

	Public key size	Ciphertext size	KeyGen	Encaps	Decaps	DFR
hqc-128	2,249	4,481	136	220	384	$< 2^{-128}$
hqc-192	4,522	9,026	305	501	821	$< 2^{-192}$
hqc-256	7,245	14,469	545	918	1538	$< 2^{-256}$

- All these changes have been implemented in constant time and we provide details on our implementations for multiplication and encoding/decoding (section 3.2).
- We give performances numbers for a hardware implementation of the scheme in section 3.3.
- We are pleased to welcome new members to our team: Jérôme Lacan and Arnaud Dion.

1.8 Updates for May the 4th 2020

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

- **(Improvement 1)** We provide in Section 2.4 a more precise analysis of the modelization of the error distribution. This new analysis permits to lower the DFR of our parameters and permits to decrease the size of our public keys by 3% (new parameters are given in Table 2 of Section 2.8.1). The size for 128 security bits is now (3,024 Bytes).
- **(Improvement 2)** We introduce in Section 2.6 a new decoding algorithm based on the concatenation of Reed-Muller and Reed-Solomon codes. This new algorithm does not change the general scheme nor its security and permits to decrease the size of the public key by 17% for 128 security bit (now of size 2,607 Bytes), a new set of parameters, HQC-RMRS, is given in Section 2.8.2 for 128, 192 and 256 bits of security.

- For parameters, we now only consider DFR corresponding to the security level and remove three parameters compared to the round 2 submission. We now only have one set of parameters for each level of security (both for HQC and the HQC-RMRS decoding variation).
- Our implementations gained in efficiency. Our optimized AVX2 implementation is now constant time and avoids secret dependent memory access. We provide new optimized implementations in C and AVX2 for the two sets of parameters HQC and HQC-RMRS (see Section 3.1 and 3.2). Moreover our implementations no longer rely on third party libraries.

- We highlight in Section 2.8.3 how it could be possible to further decrease by 10% the size of the public keys with a security reduction to a slight variation of the 3-QCSD problem.
- We welcome Jean-Marc Robert and Pascal Véron from the University of Toulon (France) as new members of our team.
- For 128 bits of security, we obtain the following sizes (in bytes) and performances (in kilocycles) for our optimized implementation leveraging AVX2:

	Public key size	Ciphertext size	KeyGen	Encaps	Decaps	DFR
HQC	3,024	6,017	175	286	486	$< 2^{-128}$
HQC-RMRS	2,607	5,191	160	272	556	$< 2^{-128}$

1.9 Modifications between Round 1 and Round 2

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

Contents

2 Specifications

In this section, we introduce HQC, an efficient encryption scheme based on coding theory. HQC stands for Hamming Quasi-Cyclic. This proposal has been published in IEEE Transactions on Information Theory [1].

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

- It is proved IND-CPA assuming the hardness of (a decisional version of) the Syndrome Decoding on structured codes. By construction, HQC perfectly fits the recent KEM-DEM transformation of [20], and allows to get an hybrid encryption scheme with strong security guarantees (IND-CCA2),
- In contrast with most code-based cryptosystems, the assumption that the family of codes being used is indistinguishable among random codes is no longer required, and
- It features a detailed and precise upper bound for the decryption failure probability analysis.

Organization of the Specifications. This section is organized as follows: we provide the required background in Section 2.1, we make some recalls on encryption and security in Section 2.2 then present our proposal in Section 2.3. An analysis of the decryption failure rate is proposed in Section 2.4. Details about codes being used are provided in Section 2.5, together with a specific analysis for these codes. Finally, concrete sets of parameters are provided in Section 2.7.

2.1 Preliminaries

2.1.1 General definitions

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

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

Let $\mathsf{truncate}(\mathbf{v}, \mathsf{num\_bits})$ be the function that takes as input a vector $\mathbf{v}$ of size n and an integer $\mathsf{num\_bits}$ and returns the truncation of $\mathbf{v}$ by $\mathsf{num\_bits}$ bits. Let $\mathsf{firstBytes}(\mathsf{var}, \mathsf{num\_bytes})$ be the function that takes as input a byte string var and an integer $\mathsf{num\_bytes}$. This function returns the first $\mathsf{num\_bytes}$ of var.

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

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

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

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

Definition 2.1.1 (Circulant Matrix)**.** *Let* $\mathbf{v} = (v_0, \ldots, v_{n-1}) \in \mathbb{F}_2^n$*. The* circulant matrix *induced by* $\mathbf{v}$ *is defined and denoted as follows:*

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

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

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

Coding Theory. We now recall some basic definitions and properties about coding theory that will be useful to our construction. We mainly focus on general definitions, and refer the reader to Section 2.3 the description of the scheme, and also to [21] for a complete survey on code-based cryptography.

Definition 2.1.2 (Linear Code)**.** *A* Linear Code $\mathcal{C}$ *of length* n *and dimension* k *(denoted* $[n, k]$*) is a subspace of* $\mathcal{R}$ *of dimension* k*. Elements of* $\mathcal{C}$ *are referred to as codewords.*

Definition 2.1.3 (Generator Matrix)**.** *We say that* $\mathbf{G} \in \mathbb{F}_2^{k \times n}$ *is a* Generator Matrix *for the* $[n, k]$ *code* $\mathcal{C}$ *if*

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

Definition 2.1.4 (Parity-Check Matrix)**.** *Given an* $[n, k]$ *code* $\mathcal{C}$*, we say that* $\mathbf{H} \in \mathbb{F}_2^{(n-k) \times n}$ *is a* Parity-Check Matrix *for* $\mathcal{C}$ *if* $\mathbf{H}$ *is a generator matrix of the dual code* $\mathcal{C}^\perp$*, or more formally, if*

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

Definition 2.1.5 (Syndrome)**.** *Let* $\mathbf{H} \in \mathbb{F}_2^{(n-k) \times n}$ *be a parity-check matrix of some* $[n, k]$ *code* $\mathcal{C}$*, and* $\mathbf{v} \in \mathbb{F}_2^n$ *be a word. Then the* syndrome *of* $\mathbf{v}$ *is* $\mathbf{Hv}^\top$*, and we have* $\mathbf{v} \in \mathcal{C} \Leftrightarrow \mathbf{Hv}^\top = \mathbf{0}$*.*

Definition 2.1.6 (Minimum Distance)**.** *Let* $\mathcal{C}$ *be an* $[n, k]$ *linear code over* $\mathcal{R}$ *and let* ω *be a norm on* $\mathcal{R}$*. The* Minimum Distance *of* $\mathcal{C}$ *is*

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

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

Code-based cryptography usually suffers from huge keys. In order to keep our cryptosystem efficient, we will use the strategy of Gaborit [14] for shortening keys. This results in Quasi-Cyclic Codes, as defined below.

Definition 2.1.7 (Quasi-Cyclic Codes [29])**.** *View a vector* $\mathbf{c} = (\mathbf{c}_0, \ldots, \mathbf{c}_{s-1})$ *of* $\mathbb{F}_2^{sn}$ *as* s *successive blocks (*n*-tuples). An* $[sn, k, d]$ *linear code* $\mathcal{C}$ *is* Quasi-Cyclic (QC) *of index* s *if, for any* $\mathbf{c} = (\mathbf{c}_0, \ldots, \mathbf{c}_{s-1}) \in \mathcal{C}$*, the vector obtained after applying a simultaneous circular shift to every block* $\mathbf{c}_0, \ldots, \mathbf{c}_{s-1}$ *is also a codeword.*

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

Definition 2.1.8 (Systematic Quasi-Cyclic Codes)**.** *A* systematic *Quasi-Cyclic* $[sn, n]$ *code of index* s *and rate* $1/s$ *is a quasi-cyclic code with an* $(s-1)n \times sn$ *parity-check matrix of the form:*

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

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

Remark 2.1. *The definition of systematic quasi-cyclic codes of index* s *can of course be generalized to all rates* ℓ/s*,* $\ell = 1 \ldots s-1$*, but we shall only use systematic QC-codes of rates* $1/2$ *and* $1/3$ *and wish to lighten notation with the above definition. In the sequel, referring to a systematic QC-code will imply by default that it is of rate* $1/s$*. Note that arbitrary QC-codes are not necessarily equivalent to a systematic QC-code.*

2.1.2 Difficult problems for cryptography

In this section we describe difficult problems which 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 2.1.9 ($\mathcal{SD}$ Distribution)**.** *Let* n*,* k *and* w *be positive integers. The Syndrome Decoding Distribution* $\mathcal{SD}(n, k, w)$ *samples* $\mathbf{H} \xleftarrow{\$} \mathbb{F}_2^{(n-k)\times n}$ *and* $\mathbf{x} \xleftarrow{\$} \mathbb{F}_2^n$ *such that* $\omega(\mathbf{x}) = w$*, computes* $\mathbf{y}^\top = \mathbf{H}\mathbf{x}^\top$ *and outputs* $(\mathbf{H}, \mathbf{y})$*.*

Definition 2.1.10 (Computational SD Problem)**.** *Let* n*,* k *and* w *be positive integers. Given* $(\mathbf{H},\mathbf{y}) \in \mathbb{F}_2^{(n-k)\times n} \times \mathbb{F}_2^{n-k}$ *from the* $\mathcal{SD}(n, k, w)$ *distribution, the Syndrome Decoding Problem* $\mathsf{SD}(n, k, w)$ *asks to find* $\mathbf{x} \in \mathbb{F}_2^n$ *such that* $\mathbf{y}^\top = \mathbf{H}\mathbf{x}^\top$ *and* $\omega(\mathbf{x}) = w$*.*

Definition 2.1.11 (DSD Problem)**.** *Let n, k and w 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* $\mathsf{DSD}(n, k, w)$ *asks to decide with non-negligible advantage whether $(\mathbf{H}, \mathbf{y})$ came from the $\mathcal{SD}(n, k, w)$ distribution or the uniform distribution over $\mathbb{F}_2^{(n-k)\times n} \times \mathbb{F}_2^{n-k}$.*

For the Hamming distance, the SD problem has been proven NP-complete [7]. This problem can also be seen as the Learning Parity with Noise (LPN) problem with a fixed number of samples [3]. The DSD problem has been shown to be polynomially equivalent to its search version in [3]. As mentioned above, this problem is the problem of decoding random linear codes from random errors. The random errors are often taken as independent Bernoulli variables acting independently on vector coordinates, rather than uniformly chosen from the set of errors of a given weight, but this hardly makes any difference and one model rather than the other is a question of convenience.

As our cryptosystem use QC-codes, the following definitions describe the DSD problems in the QC setting. QC codes are very useful in cryptography since their compact description allows to decrease considerably the size of the keys. In particular, the case $s = 2$ corresponds to double circulant codes with generator matrices of the form $(\mathbf{I}_n \ \mathbf{A})$ for $\mathbf{A}$ a circulant matrix. Such double circulant codes have been used for more than 15 years in cryptography [15].

Definition 2.1.12 (s-$\mathcal{QCSD}$ Distribution)**.** *Let n, s and w be positive integers. The s-Quasi-Cyclic Syndrome Decoding Distribution s-$\mathcal{QCSD}(n, w)$ samples a parity-check matrix $\mathbf{H} \xleftarrow{\$} \mathbb{F}_2^{(sn-n)\times sn}$ of a systematic QC code $\mathcal{C}$ of index s and rate $1/s$ (see Definition 2.1.8) and a vector $\mathbf{x} = (\mathbf{x}_0, \ldots, \mathbf{x}_{s-1}) \xleftarrow{\$} \mathbb{F}_2^{sn}$ such that $\omega(\mathbf{x}_i) = w \ \forall i \in [0, s-1]$, computes $\mathbf{y}^\top = \mathbf{H}\mathbf{x}^\top$ and outputs $(\mathbf{H}, \mathbf{y})$.*

Definition 2.1.13 (s-DQCSD Problem)**.** *Let n, s and w be positive integers. Given $(\mathbf{H}, \mathbf{y}) \in \mathbb{F}_2^{(sn-n)\times sn} \times \mathbb{F}_2^{sn-n}$, the Decisional s-Quasi-Cyclic Syndrome Decoding Problem* s-$\mathsf{DQCSD}(n, w)$ *asks to decide with non-negligible advantage whether $(\mathbf{H}, \mathbf{y})$ came from the s-$\mathcal{QCSD}(n, w)$ 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 (7). We choose this distribution so as to make the security reduction to follow less technical. It is readily seen that, for fixed s, when choosing quasi-cyclic codes with this more general distribution, one obtains with non-negligible probability, a quasi-cyclic code that admits a parity-check matrix of the form (7). Therefore requiring quasi-cyclic codes to be systematic does not hurt the generality of the decoding problem for quasi-cyclic codes. A similar remark holds for the slightly special form of weight distribution of the vector $\mathbf{x}$.

Assumption 1. *Although there is no general complexity result for quasi-cyclic codes, decoding these codes is considered hard by the community. There exist general attacks which*

uses the cyclic structure of the code [32] but these attacks have only a small (sub-linear in the code length) impact 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 needs to be appended. For $b_1 \in \{0,1\}$, we define the finite set $\mathbb{F}^n_{2,b_1} = \{\mathbf{h} \in \mathbb{F}^n_2 \text{ s.t. } \mathbf{h}(1) = b_1 \mod 2\}$, *i.e.* binary vectors of length n and parity b_1. Similarly for matrices, we define the finite sets

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

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

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 [17, 23, 24][1].

Definition 2.1.14 (2-$\mathcal{QCSD}$-$\mathcal{P}$ Distribution)**.** *Let n, w, b_1 be positive integers and $b_2 = w + b_1 \times w \mod 2$. The 2-Quasi-Cyclic Syndrome Decoding with Parity Distribution 2-$\mathcal{QCSD}$-$\mathcal{P}(n, w, b_1, b_2)$ samples $\mathbf{H} \in \mathbb{F}^{n\times 2n}_{2,b_1}$ and $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2) \xleftarrow{\$} \mathbb{F}^{2n}_2$ such that $\omega(\mathbf{x}_1) = \omega(\mathbf{x}_2) = w$, compute $\mathbf{y}^\top = \mathbf{H}\mathbf{x}^\top$ and outputs $(\mathbf{H}, \mathbf{y}) \in \mathbb{F}^{n\times 2n}_{2,b_1} \times \mathbb{F}^n_{2,b_2}$.*

Definition 2.1.15 (2-DQCSD-P Problem)**.** *Let n, w, b_1 be positive integers and $b_2 = w + b_1 \times w \mod 2$. Given $(\mathbf{H}, \mathbf{y}) \in \mathbb{F}^{n\times 2n}_{2,b_1} \times \mathbb{F}^n_{2,b_2}$, the Decisional 2-Quasi-Cyclic Syndrome Decoding with Parity Problem 2-DQCSD-P(n, w, b_1, b_2) asks to decide with non-negligible advantage whether $(\mathbf{H}, \mathbf{y})$ came from the 2-$\mathcal{QCSD}$-$\mathcal{P}(n, w, b_1, b_2)$ distribution or the uniform distribution over $\mathbb{F}^{n\times 2n}_{2,b_1} \times \mathbb{F}^n_{2,b_2}$.*

In order to thwart structural attacks, we need to work with a code of primitive prime length n, so that $X^n - 1$ has only two irreducible factors mod q. However for the considered parameters and codes (concatenated Reed-Muller and Reed-Solomon codes), the encoding of a message $\mathbf{m}$ has size $n_1 n_2$ which is obviously not prime. Therefore we use as ambient length n which is the first primitive prime greater than $n_1 n_2$ and truncate the last $\ell = n - n_1 n_2$ bits wherever needed. This results in a slightly modified version of the 3-DQCSD problem that we define hereafter.

Definition 2.1.16 (3-$\mathcal{QCSD}$-$\mathcal{PT}$ Distribution)**.** *Let n, w, b_1, b_2, ℓ be positive integers and $b_3 = w + b_1 \times w \mod 2$. The 3-Quasi-Cyclic Syndrome Decoding with Parity and Truncation Distribution 3-$\mathcal{QCSD}$-$\mathcal{PT}(n, w, b_1, b_2, b_3, \ell)$ samples $\mathbf{H} \xleftarrow{\$} \mathbb{F}^{2n\times 3n}_{2,b_1,b_2}$ and $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3) \xleftarrow{\$} \mathbb{F}^{3n}_2$ such that $\omega(\mathbf{x}_1) = \omega(\mathbf{x}_2) = \omega(\mathbf{x}_3) = w$, computes $\mathbf{y}^\top = \mathbf{H}\mathbf{x}^\top$ where $\mathbf{y} = (\mathbf{y}_1, \mathbf{y}_2)$ and outputs $(\mathbf{H}, (\mathbf{y}_1, \mathsf{truncate}(\mathbf{y}_2, \ell))) \in \mathbb{F}^{2n\times 3n}_{2,b_1,b_2} \times (\mathbb{F}^n_{2,b_3} \times \mathbb{F}^{n-\ell}_2)$.*

[1] The authors chose to use a parity version of the 2-DQCSD problem rather than a variable weight version as suggested in [24] for efficiency considerations.

Definition 2.1.17 (3-QCSD-PT Problem)**.** *Let n, w, b_1, b_2, ℓ be positive integers and $b_3 = w + b_1 \times w \bmod 2$. 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$(n, w, b_1, b_2, b_3, \ell)$ *asks to decide with non-negligible advantage whether $(\mathbf{H}, (\mathbf{y}_1, \mathbf{y}_2))$ came from the 3-$\mathcal{QCSD}$-$\mathcal{PT}(n, w, b_1, b_2, b_3, \ell)$ 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.

2.2 Encryption and security

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

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

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

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

IND-CPA [18]: This notion formalized by the game depicted in Fig. 1, states that an adversary should not be able to efficiently guess which plaintext has been encrypted even if he knows it is one among two plaintexts of his choice.

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

$$\mathsf{Adv}_{\mathsf{PKE}}^{\mathsf{IND\text{-}CPA}}(t) = \max_{|\mathcal{A}|\leq t} \mathsf{Adv}_{\mathsf{PKE}}^{\mathsf{IND\text{-}CPA}}(\mathcal{A}), \tag{8}$$

where $\mathsf{Adv}_{\mathsf{PKE}}^{\mathsf{IND\text{-}CPA}}(\mathcal{A})$ is the advantage the adversary $\mathcal{A}$ has in winning game $\mathbf{Exp}_{\mathsf{PKE}}^{\mathsf{ind}-b}(\mathcal{A})$:

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

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

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

2.3 Presentation of the scheme

HQC is an IND-CCA2 Key Encapsulation Mechanism (KEM) build from an IND-CPA PKE construction on top of which the HHK transform [20] is performed. Parameter sets can be found in Section 2.7. We further use $\mathcal{M}$ to denote the message space. Let w be a positive integer, we denote by $\mathcal{R}_\omega := \{\mathbf{v} \in \mathcal{R} \text{ such that } \omega(\mathbf{v}) = w\}$ the set of vectors having Hamming weight w.

2.3.1 Public key encryption version (HQC.PKE)

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

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

Figure 2: Description of our proposal HQC.PKE.

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

$\mathcal{C}$ is instantiated using concatenated Reed-Muller and Reed-Solomon codes: see Section 2.5 for more details. Furthermore, we will have $\mathbf{G} \in \mathbb{F}_2^{n_1 n_2}$ and $\mathbf{h} \in \mathbb{F}_2^n$, with n the

smallest primitive prime greater than $n_1 n_2$. All computations are made in the ambient space $\mathbb{F}_2^n$ and the remaining $\ell = n - n_1 n_2$ bits are truncated whenever required.

In the secret key $(\mathbf{x}, \mathbf{y})$ is represented as $(\mathbf{seed1})$ where $\mathbf{seed1}$ is used to generate $\mathbf{x}$ and $\mathbf{y}$. The public key $\mathsf{pk} = (\mathbf{h}, \mathbf{s})$ is represented as $\mathsf{pk} = (\mathbf{seed2}, \mathbf{s})$ where $\mathbf{seed2}$ is used to generate $\mathbf{h}$.

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

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

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

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

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

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

In order to provide an upper bound on the decryption failure probability, an analysis of the distribution of the error vector $\mathbf{e}' = \mathbf{x} \cdot \mathbf{r}_2 - \mathbf{r}_1 \cdot \mathbf{y} + \mathbf{e}$ is provided in Section 2.4.

2.3.2 A Key Encapsulation Mechanism (HQC.KEM)

Let $\mathcal{G}$ and $\mathcal{K}$ be hash functions. Following the HHK framework [20], we construct the HQC.KEM IND-CCA2-secure key encapsulation mechanism (see Figure 3) from the IND-CPA-secure public-key encryption scheme described in Figure 2. More details regarding the security proof is provided in section 5.2.

2.4 Analysis of the error vector distribution for Hamming distance

In this section we provide a more precise analysis of the error distribution approximation compared to the Round 2 submission. This analysis is taken from [4]. We first compute exactly the probability distribution of each fixed coordinate e'_k of the error vector

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

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

To compute decoding error probabilities, we will then need the probability distribution of the weight of the error vector $\mathbf{e}'$ restricted to given sets of coordinates that correspond to codeword supports. We will make the simplifying assumption that the coordinates e'_k of $\mathbf{e}'$ are independent variables, which will let us work with the binomial distribution of parameter p^* for the weight distributions of $\mathbf{e}'$. In other words we modelize the error vector as a binary symmetric channel with parameters p^*. This working assumption is justified by remarking that, in the high weight regime relevant to us, since the component vectors $\mathbf{x}, \mathbf{y}, \mathbf{e}$

- $\mathsf{Setup}(1^\lambda)$: as before, except that k will be the length of the shared key being exchanged.
- $\mathsf{KeyGen}(\mathsf{param})$: samples $\mathbf{h} \xleftarrow{\$} \mathcal{R}$, $\sigma \xleftarrow{\$} \mathbb{F}_2^k$, the generator matrix $\mathbf{G} \in \mathbb{F}_2^{k\times n}$ of $\mathcal{C}$, $\mathsf{sk} = (\mathbf{x}, \mathbf{y}, \sigma) \xleftarrow{\$} \mathcal{R}_w \times \mathcal{R}_w \times \mathbb{F}_2^k$, sets $\mathsf{pk} = (\mathbf{h}, \mathbf{s} = \mathbf{x} + \mathbf{h}\cdot\mathbf{y})$, and returns $(\mathsf{pk}, \mathsf{sk})$.
- $\mathsf{Encapsulate}(\mathsf{pk})$: generates $\mathbf{m} \xleftarrow{\$} \mathbb{F}_2^k$, $salt \xleftarrow{\$} \mathbb{F}_2^{128}$. Using the first 32 bytes of the public key pk, derive the randomness $\theta \leftarrow \mathcal{G}(\mathbf{m}\|\mathsf{firstBytes}(\mathsf{pk}, 32)\|salt)$ and use θ to generates $(\mathbf{e}, \mathbf{r}_1, \mathbf{r}_2)$ such that $\omega(\mathbf{e}) = w_\mathbf{e}$ and $\omega(\mathbf{r}_1) = \omega(\mathbf{r}_2) = w_\mathbf{r}$, sets $\mathbf{u} = \mathbf{r}_1 + \mathbf{h}\cdot\mathbf{r}_2$ and $\mathbf{v} = \mathsf{truncate}(\mathbf{mG} + \mathbf{s}\cdot\mathbf{r}_2 + \mathbf{e}, \ell)$, sets $\mathbf{c} = (\mathbf{u}, \mathbf{v})$. Computes $K \leftarrow \mathcal{K}(\mathbf{m}, \mathbf{c})$, and return $(K, \mathbf{c}, salt)$.
- $\mathsf{Decapsulate}(\mathsf{sk}, \mathbf{c}, salt)$: Decrypt $\mathbf{m}' \leftarrow \mathsf{Decrypt}(\mathsf{sk}, \mathbf{c})$, compute the randomness $\theta' \leftarrow \mathcal{G}(\mathbf{m}'\|\mathsf{firstBytes}(\mathsf{pk}, 32)\|salt)$, and (re-)encrypt $\mathbf{m}'$ by using θ' to generates $(\mathbf{e}', \mathbf{r}_1', \mathbf{r}_2')$ such that $\omega(\mathbf{e}') = w_\mathbf{e}$ and $\omega(\mathbf{r}_1') = \omega(\mathbf{r}_2') = w_\mathbf{r}$, sets $\mathbf{u}' = \mathbf{r}_1' + \mathbf{h}\cdot\mathbf{r}_2'$ and $\mathbf{v}' = \mathsf{truncate}(\mathbf{m}'\mathbf{G} + \mathbf{s}\cdot\mathbf{r}_2' + \mathbf{e}', \ell)$, sets $\mathbf{c}' = (\mathbf{u}', \mathbf{v}')$. If $\mathbf{m}' = \perp$ or $\mathbf{c} \neq \mathbf{c}'$ then $K \leftarrow \mathcal{K}(\sigma, \mathbf{c})$. Otherwise, $K \leftarrow \mathcal{K}(\mathbf{m}, \mathbf{c})$.

Figure 3: Description of our proposal HQC.KEM derived from Figures 2, 6 and 7.

have fixed weights, the probability that a given coordinate e_k' takes the value 1 conditioned on abnormally many others equalling 1 can realistically only be $\leq p^*$. We support this modeling of the otherwise intractable weight distribution of $\mathbf{e}'$ by extensive simulations: these back up our assumption that our computations of decoding error probabilities and DFRs can only be upper bounds on their real values.

The vectors $\mathbf{x}, \mathbf{y}, \mathbf{r}_1, \mathbf{r}_2, \mathbf{e}$ have been taken uniformly random and independently chosen among vectors of weight w, $w_\mathbf{r}$ and $w_\mathbf{e}$. We first evaluate the distributions of the products $\mathbf{x}\cdot\mathbf{r}_2$ and $\mathbf{r}_1\cdot\mathbf{y}$.

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

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

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

Proof. The total number of ordered pairs $(\mathbf{x}, \mathbf{r})$ is $\binom{n}{w}\binom{n}{w_\mathbf{r}}$. Among those, we need to count

how many are such that $z_k = 1$. We note that

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

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

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

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

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

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

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

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

Proposition 2.4.2 gives us the probability that a coordinate of the error vector $\mathbf{e}'$ is 1. In our simulations, which occur in the regime $w = \alpha\sqrt{n}$ with constant α, we make the simplifying assumption that the coordinates of $\mathbf{e}'$ are independent, meaning that the weight of $\mathbf{e}'$ follows a binomial distribution of parameter $p^\star$, where $p^\star$ is defined as in Eq. (15): $p^\star = 2\tilde{p}(1-\tilde{p})(1-\frac{w_{\mathbf{e}}}{n}) + ((1-\tilde{p})^2 + \tilde{p}^2)\,\frac{w_{\mathbf{e}}}{n}$. This approximation will give us, for $0 \le d \le \min(2 \times w \times w_{\mathbf{r}} + w_{\mathbf{e}}, n)$,

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

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

Examples of simulations. We consider a parameter set that corresponds to cryptographic parameters and for which we simulate the error distribution versus the binomial approximation together with the probability of obtaining large error weights. We computed vectors of length n and then truncated the last $l = n - n_1 n_2$ bits before measuring the Hamming weight of the vectors.

Parameter set	w	$w_{\mathbf{e}} = w_{\mathbf{r}}$	n	$n_1 n_2$	$p^\star$
hqc-128	66	75	17669	17664	0.3398

Simulation results

Simulation results are shown figure 4. We computed the weights such that 0.1%, 0.01% and 0.001% of the vectors are of weight greater than this value, to study how often extreme weight values occur. Results are presented table 2.

	0.1%	0.01%	0.001%	0.0001%
Error vectors	6169	6203	6232	6257
Binomial approximation	6197	6237	6272	6301

Table 2: Simulated probabilities of large weights for hqc-128 for the distributions of the error vector and the binomial approximation

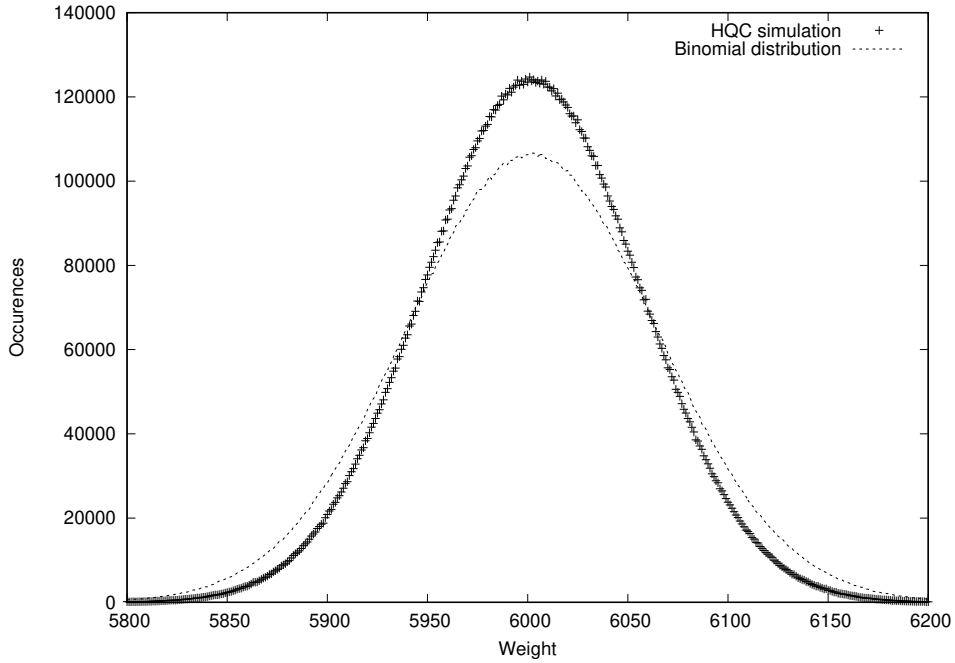

Figure 4: Comparison between error $\mathbf{e}'$ generated using hqc-128 parameters and its binomial approximation.

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

In this section taken from [4] we propose to consider a new decoding algorithm based on Reed-Muller and Reed-Solomon concatenated codes.

2.5.1 Definitions

Definition 2.5.1 (Concatenated codes)**.** *A concatenated code consists of an external code* $[n_e, k_e, d_e]$ *over* $\mathbb{F}_q$ *and an internal code* $[n_i, k_i, d_i]$ *over* $\mathbb{F}_2$*, with* $q = 2^{k_i}$*. We use a bijection between elements of* $\mathbb{F}_q$ *and the words of the internal code, this way we obtain a transformation:*

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

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

For the external code, we chose a Reed-Solomon code of dimension 32 over $\mathbb{F}_{256}$ and, for the internal code, we chose the Reed-Muller code $[128, 8, 64]$ that we are going to duplicate 3 or 5 times (i.e duplicating each bit to obtain codes of parameters $[384, 8, 192]$ and $[640, 8, 320]$).

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

2.5.2 Reed-Solomon codes

Let p be a prime number and q is any power of p. Following [22], a Reed-Solomon code with symbols in $\mathbb{F}_{q^p}$ has the following parameters:

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

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

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

Depending on HQC parameters, we construct shortened Reed-Solomon (RS-S1, RS-S2 and RS-S3) codes such that k is equal to 16, 24 or 32 from the following RS codes RS-1, RS-2 and RS-3 (codes from [22]).

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

Table 3: Original and shortened Reed-Solomon codes.

The shortened codes are obtained by subtracting 209 from the parameters n and k of the code RS-1 and subtracting 199 from the parameters n and k of the code RS-2 and by subtracting 165 from the parameters n and k of the code RS-3. Notice that shortening the Reed-Solomon code does not affect the correcting capacity, thus we have the following shortened Reed-Solomon codes :

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

In our case, we will be working in $\mathbb{F}_{2^m}$ with $m = 8$. To do so, we use the primitive polynomial $1 + x^2 + x^3 + x^4 + x^8$ of degree 8 to build this field (polynomial from [22]). We denote by $g_1(x)$, $g_2(x)$ and $g_3(x)$ the generator polynomials of RS-S1, RS-S2 and RS-S3 respectively, which are equal to the generator polynomials of Reed-Solomon codes RS-1, RS-2 and RS-3 respectively. We precomputed the generator polynomials $g_1(x)$, $g_2(x)$ and $g_3(x)$ of the code RS-S1, RS-S2 and RS-S3 and we included them in the file `parameters.h`. One can use the functions provided in the file `reed_solomon.h` to reconstruct the generator polynomials for those codes.

Generator polynomial of RS-1. $g_1(x) = 9 + 69x + 153x^2 + 116x^3 + 176x^4 + 117x^5 + 111x^6 + 75x^7 + 73x^8 + 233x^9 + 242x^{10} + 233x^{11} + 65x^{12} + 210x^{13} + 21x^{14} + 139x^{15} + 103x^{16} + 173x^{17} + 67x^{18} + 118x^{19} + 105x^{20} + 210x^{21} + 174x^{22} + 110x^{23} + 74x^{24} + 69x^{25} + 228x^{26} + 82x^{27} + 255x^{28} + 181x^{29} + x^{30}$.

Generator polynomial of RS-2. $g_2(x) = 45 + 216x + 239x^2 + 24x^3 + 253x^4 + 104x^5 + 27x^6 + 40x^7 + 107x^8 + 50x^9 + 163x^{10} + 210x^{11} + 227x^{12} + 134x^{13} + 224x^{14} + 158x^{15} + 119x^{16} + 13x^{17} + 158x^{18} + 1x^{19} + 238x^{20} + 164x^{21} + 82x^{22} + 43x^{23} + 15x^{24} + 232x^{25} + 246x^{26} + 142x^{27} + 50x^{28} + 189x^{29} + 29x^{30} + 232x^{31} + x^{32}$.

Generator polynomial of RS-3. $g_3(x) = 49 + 167x + 49x^2 + 39x^3 + 200x^4 + 121x^5 + 124x^6 + 91x^7 + 240x^8 + 63x^9 + 148x^{10} + 71x^{11} + 150x^{12} + 123x^{13} + 87x^{14} + 101x^{15} + 32x^{16} + 215x^{17} + 159x^{18} + 71x^{19} + 201x^{20} + 115x^{21} + 97x^{22} + 210x^{23} + 186x^{24} + 183x^{25} + 141x^{26} + 217x^{27} + 123x^{28} + 12x^{29} + 31x^{30} + 243x^{31} + 180x^{32} + 219x^{33} + 152x^{34} + 239x^{35} + 99x^{36} + 141x^{37} + 4x^{38} + 246x^{39} + 191x^{40} + 144x^{41} + 8x^{42} + 232x^{43} + 47x^{44} + 27x^{45} + 141x^{46} + 178x^{47} + 130x^{48} + 64x^{49} + 124x^{50} + 47x^{51} + 39x^{52} + 188x^{53} + 216x^{54} + 48x^{55} + 199x^{56} + 187x^{57} + x^{58}$.

2.5.3 Encoding shortened Reed-Solomon codes

In the following we present the encoding of Reed-Solomon codes which can also be used to encode shortened Reed-Solomon codes. We denote by $u(x) = u_0 + \cdots + u_{k-1}x^{k-1}$ the polynomial corresponding to the message $u = (u_0, \cdots, u_{k-1})$ to be encoded and $g(x)$ the generator polynomial. We use the systematic form of encoding where the rightmost k elements of the code word polynomial are the message bits and the leftmost $n - k$ bits are the parity-check bits. Following [22], the code word is given by $c(x) = b(x) + x^{n-k}u(x)$, where $b(x)$ is the reminder of the division of the polynomial $x^{n-k}u(x)$ by $g(x)$. In consequence, the encoding in systematic form consists of three steps :

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

2.5.4 Decoding shortened Reed-Solomon codes

The decoding of classical Reed-Solomon codes can be used to decode shortened Reed-Solomon codes. For sake of simplicity, we will detail the process of decoding classical Reed-Solomon codes. Following [22], consider the Reed-Solomon code defined by $[n, k, d_{min}]$, with $n = 2^m - 1$ ($m \geq 0$ of positive integer) and suppose that a codeword $v(x) = v_0 + v_1 x + \cdots + v_{n-1}x^{n-1}$ is transmitted. We denote $r(x) = r_0 + r_1 x + \cdots + r_{n-1}x^{n-1}$ the received word, potentially altered by some errors.

We denote the error polynomial $e(x) = e_0 + e_1 x + \cdots + e_{n-1}x^{n-1}$, meaning that there is an error in position i whenever $e_i \neq 0$. Hence, $r(x) = v(x) + e(x)$.

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

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

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

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

we define the error location polynomial as:

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

We can see that the roots of $\sigma(x)$ are $\beta_1^{-1}, \beta_2^{-1}, \cdots, \beta_t^{-1}$ which are the inverses of the error location numbers. After retrieving the coefficients of $\sigma(x)$, we can compute the error values. Let

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

The error value at location β_l is given by [6]

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

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

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

2. The second step is the computation of the error-location polynomial $\sigma(x)$ from the 2δ syndromes computed in the first step. Here we use Berlekamp's algorithm [22].

3. The third step is to find the error-location numbers by calculating the roots of the polynomial $\sigma(x)$ and returning their inverses. We implement this step with an additive Fast Fourier Transform algorithm from [16].

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

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

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

2.5.5 Duplicated Reed-Muller codes

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

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

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

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

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

We then apply the Hadamard transform to the function F. We take the maximum value in $\hat{F}$ and $x \in \mathbb{F}_2^3$ that maximizes the value of $|\hat{F}|$. If $\hat{F}(x)$ is positive, then the closest codeword is xG where G is the generator matrix of the Hadamard code (without the all-one-vector). If $\hat{F}(x)$ is negative, then we need to add the all-one-vector to it.

2.5.6 Encoding Duplicated Reed-Muller codes

Following [26], the encoding is done in classical way by using a matrix vector multiplication. The codeword is then duplicated depending on the used parameter (see Table 4).

Scheme	Reed-Muller Code	Multiplicity	Duplicated Reed-Muller Code
hqc-128	[128, 8, 64]	3	[384, 8, 192]
hqc-192	[128, 8, 64]	5	[640, 8, 320]
hqc-256	[128, 8, 64]	5	[640, 8, 320]

Table 4: Duplicated Reed-Muller codes.

2.5.7 Decoding Duplicated Reed-Muller codes

Following [26] (Chapter 14), the decoding of duplicated Reed-Muller codes is done in three steps:

1. The first step is the computation of the function F described in Section 2.5.5. We apply F on the received codeword. We give details about how this process is done where the multiplicity is equal to 2. Let v a duplicated Reed-Muller codeword, it can be seen as $v = (a_1 b_1, \cdots, a_{n_2} b_{n_2})$ where each a_i, b_i has 128 bits size ($a_i = (a_{i_0}, \cdots, a_{i_{128}})$ and $b_i = (b_{i_0}, \cdots, b_{i_{128}})$). The transformation F is applied to each element in v as follows $((-1)^{a_{i_0}} + (-1)^{b_{i_0}}, \cdots, (-1)^{a_{i_{128}}} + (-1)^{b_{i_{128}}})$. The cases when multiplicity is equal to 4 follow a similar process.

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

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

2.5.8 Decryption failure rate analysis

In this section we analyze the DFR of the concatenated codes. We use the binomial law approximation p^* of the error vector of Section 2.4.

It is only possible to obtain an exact decoding probability formula for the Reed-Solomon codes as for Reed-Muller codes we consider a maximum-likelihood decoding for which there is no exact formula. We provide in the following proposition a lower bound on the decoding probability in that case.

Proposition 2.5.1. [Simple Upper Bound for the DFR of the internal code] *Let p be the transition probability of the binary symmetric channel. Then the DFR of a duplicated Reed-Muller code of dimension 8 and minimal distance d_i can be upper bounded by:*

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

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

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

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

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

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

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

Proposition 2.5.2. [Improved Upper Bound for the DFR of the internal code] *Let p be the transition probability of the binary symmetric channel. Then the DFR of a Reed-Muller code of dimension 8 and minimal distance d_i can be upper bounded by:*

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

where

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

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

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

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

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

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

$$\begin{aligned} P(E_w) &\leqslant \frac{1}{2} P(A'_w) + P(A''_w) + P(B_w) \\ &= \frac{1}{2}(P(A'_w) + P(A''_w)) + \frac{1}{2} P(A''_w) + P(B_w) \\ P(E_w) &\leqslant \frac{1}{2} P(A_w) + \frac{1}{2} P(A''_w) + P(B_w) \end{aligned} \tag{17}$$

where for $X = A, A', A'', E$, the event X_w signifies the intersection of the event X with the event "$\omega(\mathbf{e}) = w$".

Now we have the straightforward union bounds:

$$P(B_w) \leqslant 255 \sum_{j=d_i/2+1}^{d_i} \binom{d_i}{j}\binom{d_i}{w-j} p^w (1-p)^{n-w} \tag{18}$$

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 $k > \ell$.

$$P(A_w) \leqslant 255 \binom{d_i}{d_i/2}\binom{d_i}{w-d_i/2} p^w (1-p)^{n-w} \tag{19}$$

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

We have:

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

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

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

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

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

Hence

$$P(A''_w) \leqslant \sum_{\mathbf{c},\mathbf{c}'} P(A_{\mathbf{c},\mathbf{c}'}) \leqslant \binom{255}{2} \sum_{j=0}^{d_i/2} \binom{d_i/2}{j}^3 \binom{d_i/2}{w-d_i+j} p^w (1-p)^{n-w}. \tag{20}$$

Plugging 19, 18 and 20 into 17 we obtain the result. □

Remark 2.2. *The previous formula permits to obtain a lower bound on the decoding probability; when the error rate gets smaller the bound becomes closer to the real value of the decoding probability. For cryptographic parameters the approximation is less precise, which means that the DFR obtained will be conservative compared to what happens in practice. We performed simulations to compare the real decryption failure rate with the theoretical one from proposition 2.5.1 for* $[512, 8, 256]$ *and* $[640, 8, 320]$ *duplicated Reed-Muller codes using* $p^\star$ *values from actual parameters. Simulation results are presented table 5.*

Security level	$p^\star$	Reed-Muller code	DFR from 2.5.2	Observed DFR
128	0.3398	$[384, 8, 192]$	-10.79	-10.96
192	0.3618	$[640, 8, 320]$	-14.14	-14.39
256	0.3725	$[640, 8, 320]$	-11.30	-11.48

Table 5: Comparison between the observed Decryption Failure Rate and the formula from proposition 2.5.1. Results are presented as $\log_2(DFR)$.

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

Theorem 2.3. *Decryption Failure Rate of the concatenated code Using a Reed-Solomon code $[n_e, k_e, d_e]_{\mathbb{F}_{256}}$ as the external code, the DFR of the concatenated code can be upper bounded by:*

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

Where $d_e = 2\delta_e + 1$ and p_i is defined as in proposition 2.5.1.

2.5.9 Simulation results

In Fig. 5, we tested the Decryption Failure rate of the concatenated codes against both symmetric binary channels and HQC vectors, and compared the results with the theoretical value obtained using proposition 2.5.1 and 2.3.

2.6 Representation of objects

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

Seeds. The considered seed-expander is based on the SHAKE256 function. It is initialized with a byte string of length 40 which are used as the `seed`.

2.6.1 Keys and ciphertext representation

In the secret key $(\mathbf{x}, \mathbf{y})$ is represented as $(\mathbf{seed1})$ where **seed1** is used to generate $\mathbf{x}$ and $\mathbf{y}$. The public key $\mathsf{pk} = (\mathbf{h}, \mathbf{s})$ is represented as $\mathsf{pk} = (\mathbf{seed2}, \mathbf{s})$ where **seed2** is used to generate $\mathbf{h}$. The ciphertext $\mathbf{c}$ is represented as $(\mathbf{u}, \mathbf{v}, salt)$ where *salt* is generated using SHAKE256-512. The secret key has size $40+\lceil k/8 \rceil$ bytes, the public key has size $40+\lceil n/8 \rceil$ bytes and the ciphertext has size $\lceil n/8 \rceil + \lceil n_1 n_2/8 \rceil + 16$ bytes.

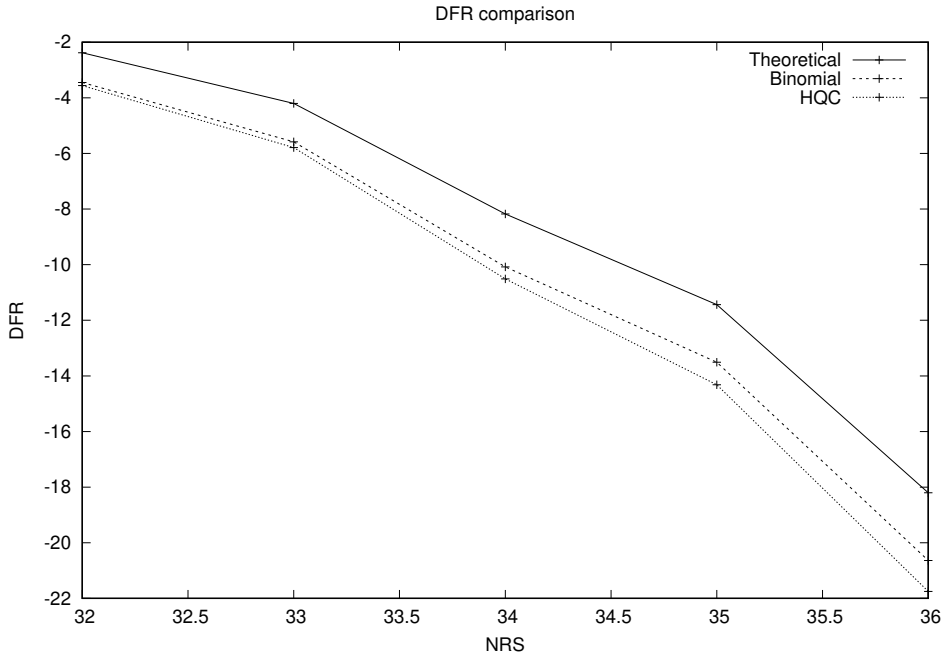

Figure 5: Comparison between the Decryption Failure Rate from 2.3 (Theoretical) and the actual Decryption Failure Rate of concatenated codes against approximation by a binary symmetric channel (Binomial) and against HQC error vectors (HQC). Parameters simulated are derived from those of HQC for 128 security bits: $w = 66$, $w_{\mathbf{r}} = w_{\mathbf{e}} = 75$, a $[384, 8, 192]$ duplicated Reed-Muller code for internal code and a $[NRS, 16]$ Reed-Solomon code for external code.

2.6.2 Randomness and vector generation

Random bytes are generated using the SHAKE256 based `shake_prng` or `seedexpander` functions. The `shake_prng` function is used to generate **seed1**, **seed2**, **m** as well as *salt*. The `seedexpander` function is used to generate **x**, **y** (using **seed1** as seed), **h** (using **seed2** as seed) and $\mathbf{r}_1$, $\mathbf{r}_2$, **e** (using θ as seed). For key generation, the randomized access is done using the `seedexpander` with **seed1** as seed. For encryption process, randomized access is done using the `seedexpander` function with θ as seed.

Random vectors are sampled uniformly from $\mathbb{F}_2^k$, $\mathbb{F}_2^n$ or from $\mathbb{F}_2^n$ with a given Hamming weight. Sampling from $\mathbb{F}_2^k$ and $\mathbb{F}_2^n$ is performed by filling the mathematical representation of the vector with random bits. Sampling a vector from $\mathbb{F}_2^n$ of a given weight starts by generating the support using Algorithm 1. Next, the sampled support is converted to an n-dimensional array. The distribution of the resulting vector is biased away from the uniform distribution, however we show in section 5.3, that this bias does not affect the security of the scheme.

Algorithm 1: Fixed Hamming weight vector sampling (Algorithm 5 in [33])

Input: n, w, seed
Output: w distinct elements of $\{0, \cdots, n\}$
1: $\mathsf{prng} \leftarrow$ **prng-init**(seed)
2: **for** $i = w - 1$ **downto** 0 **do**
3: $\quad l \leftarrow i +$ **rand**$(n - i, \mathsf{prng})$
4: $\quad \mathrm{pos}[i] \leftarrow (l \in \{\mathrm{pos}[j], i < j < t\})\ ?\ i : l$
5: **return** $\mathrm{pos}[0], \cdots, \mathrm{pos}[w-1]$

rand(n, prng);
1: $x \leftarrow$ **randBits**(B, prng)
2: **return** $x \bmod n$

2.6.3 Sampling order for key generation and encryption

To optimize performance, the sampling order of variables in both key generation and encryption processes follows the strategy outlined for hardware implementations in [2]. As demonstrated by the authors of [2], this adjusted sampling order leads to significant performance improvements, particularly in hardware environments. For key generation, variables are sampled in the order: first $\mathbf{y}$, then $\mathbf{x}$. Similarly, during encryption, the randomness values are sampled in the following sequence: $\mathbf{r}_2$, then $\mathbf{e}$, and finally $\mathbf{r}_1$.

2.6.4 The functions $\mathcal{G}$ and $\mathcal{K}$

The concrete instantiation of $\mathcal{G}$ and $\mathcal{K}$ is defined using SHAKE256 with 512 bits output customized with a domain separation tag that we denote SHAKE256-512. In particular, we have SHAKE256-512$(\cdot \| \texttt{G_FCT_DOMAIN})$ for $\mathcal{G}(\cdot)$ and SHAKE256-512$(\cdot \| \texttt{K_FCT_DOMAIN})$ for $\mathcal{K}(\cdot)$ where the final constant (`G_FCT_DOMAIN` or `K_FCT_DOMAIN`) is encoded over one byte.

2.7 Parameters

In this section, we specify which codes are used for HQC and give concrete sets of parameters.

We propose several sets of parameters, targeting different levels of security with DFR related to these security levels. The proposed sets of parameters cover security categories 1, 3, and 5 (for respectively 128, 192, and 256 bits of security). For each parameter set, the parameters are chosen so that the minimal workfactor of the best known attack exceeds the security parameter. For classical attacks, best known attacks include the works from [10, 9, 13, 5] and for quantum attacks, the work of [8]. We consider $w = \mathcal{O}(\sqrt{n})$ and follow the complexity described in [11] (see Section 6 for more details).

2.7.1 Concatenated codes

When we use a Concatenated code (Def. 2.5.1). A message $\mathbf{m} \in \mathbb{F}_2^k$ is encoded into $\mathbf{m}_1 \in \mathbb{F}_{2^8}^{n_1}$ with the Reed-Solomon code, then each coordinate $\mathbf{m}_{1,i}$ of $\mathbf{m}_1$ is encoded into $\tilde{\mathbf{m}}_{1,i} \in \mathbb{F}_2^{n_2}$ with the duplicated Reed-Muller code. In the latter step, the encoding is done in two

phases. First, we use the $RM(1,7)$ to encode $\mathbf{m}_{1,i}$ and we obtain $\bar{\mathbf{m}}_{1,i} \in \mathbb{F}_2^{128}$. Then, $\bar{\mathbf{m}}_{1,i}$ is duplicated depending on the multiplicity of the Reed-Muller code (see Tab. 4).

To match the description of our cryptosystem in Section 2.3, we have $\mathbf{mG} = \tilde{\mathbf{m}} = (\tilde{\mathbf{m}}_{1,0}, \ldots, \tilde{\mathbf{m}}_{1,n_1-1}) \in \mathbb{F}_2^{n_1 n_2}$. To obtain the ciphertext, $\mathbf{r} = (\mathbf{r}_1, \mathbf{r}_2) \xleftarrow{\$} \mathcal{R}^2$ and $\mathbf{e} \xleftarrow{\$} \mathcal{R}$ are generated and the encryption of $\mathbf{m}$ is $\mathbf{c} = (\mathbf{u} = \mathbf{r}_1 + \mathbf{h} \cdot \mathbf{r}_2, \mathbf{v} = \mathbf{mG} + \mathbf{s} \cdot \mathbf{r}_2 + \mathbf{e})$.

In Tab. 6, n_1 denotes the length of the Reed-Solomon code, n_2 the length of the Reed-Muller code so that the length of the concatenated code $\mathcal{C}$ is $n_1 n_2$ (the ambient space has length n, the smallest primitive prime greater than $n_1 n_2$ to avoid algebraic attacks). w is the weight of the n-dimensional vectors $\mathbf{x}$, $\mathbf{y}$, $w_{\mathbf{r}}$ the weight of $\mathbf{r}_1$, and $\mathbf{r}_2$ and similarly $w_{\mathbf{e}} = \omega(\mathbf{e})$ for our cryptosystem.

Instance	n_1	n_2	n	w	$w_{\mathbf{r}} = w_{\mathbf{e}}$	security	p_{fail}
hqc-128	46	384	17,669	66	75	128	$< 2^{-128}$
hqc-192	56	640	35,851	100	114	192	$< 2^{-192}$
hqc-256	90	640	57,637	131	149	256	$< 2^{-256}$

Table 6: Parameter sets for HQC. The concatenated code used is consists of a $[n_2, 8, n_2/2]$ Reed-Muller code as the internal code, and a $[n_1, k, n_1 - k + 1]$ Reed-Solomon code as the external code. The resulting public key, secret key and ciphertext sizes, are given in Tab. 7. The aforementioned sizes are the ones used in our reference implementation except that we also concatenate the public key within the secret key in order to respect the NIST API.

Instance	pk size	sk size	ct size	ss size
hqc-128	2,249	56	4,497	64
hqc-192	4,522	64	9,042	64
hqc-256	7,245	72	14,485	64

Table 7: Sizes in bytes for HQC (see section 2.6).

3 Performance Analysis

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

Benchmark platform. The benchmarks have been performed on a machine that has 32GB of memory and an Intel® Core™ i7-11850H CPU @ 2.50GHz for which the Hyper-Threading, Turbo Boost and SpeedStep features were disabled. The scheme have been compiled with gcc (version 11.3.0). For each parameter set, the results have been obtained by computing the mean from 1000 random instances. In order to minimize biases from

background tasks running on the benchmark platform, each instances have been repeated 100 times and averaged.

Constant time. The provided optimized AVX implementations have been implemented in constant time. We have thoroughly analyzed the code to check that only unused randomness (i.e. rejected based on public criteria) or otherwise nonsensitive data may be leaked. The reference implementation is provided to help understanding the scheme and thus is not implemented to be constant time in any way.

3.1 Reference implementation

The reference implementation is written in `C++` and have been compiled with `g++` (version 8.2.1) using the compilation flags `-O3 -pedantic -pthread`. The following third party libraries have been used: `gmp` (version 6.1.2), `NTL` (version 11.5.1) [34] and `GF2X` (version 1.3.0). The performances of our reference implementation on the aforementioned benchmark platform are described Tab. 8.

Instance	KeyGen	Encaps	Decaps
hqc-128	189	393	783
hqc-192	424	890	1724
hqc-256	860	2067	3624

Table 8: Performance in kilocycles of the reference implementation for different instances of HQC.

3.2 Optimized constant-time implementation

A constant-time optimized implementation leveraging AVX2 instructions have been provided. Its performances on the aforementioned benchmark platform are described in Tab. 10. The following optimization flags have been used during compilation: `-O3 -std=c99 -funroll-all-loops -flto -mavx -mavx2 -mbmi -mpclmul -pedantic -Wall -Wextra`. There are two main differences between the reference and the optimized implementation. Firstly, the multiplication of two polynomial is vectorized. Secondly, we added a vectorized version of the Reed-Muller decoding algorithm.

In the sequel we give some details on the optimizations done in this version.

Multiplication over $\mathbb{F}_2[X]/(X^n - 1)$ (dense-dense multiplication) In this version we do not take into account the sparsity of one of the polynomial. We use a classical dense-dense multiplication to avoid some possible leakage of information. This multiplication is done using a combination of Toom-Cook multiplication and Karatsuba multiplication.

About 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 :

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

About 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 reproduce in Algorithm 2. In this algorithm, the polynomials A and B are split into two parts, however, variants with other splits can be extrapolated. In particular, we used a 3-part split (`3-Karatsuba`) as the Toom-Cook elementary multiplication for hqc-128 and hqc-192, and a 5-part split (`5-Karatsuba`) as the Toom-Cook elementary multiplication for hqc-256. The multiplication line 2 (denoted `Mult64`) is performed using a single processor instruction (`pclmul` for carry-less multiplier): this is the case for the Intel Cores i3, i5 and i7 and above.

Algorithm 2: `KaratRec`(A,B,t)

Require: A and B on $t = 2^r$ computer words.
Ensure: $R = A \times B$
1: **if** $t = 1$ **then**
2: **return** ($Mult64(A, B)$)
3: **else**
4: // Split in two halves of word size $t/2$.
5: $A = A_0 + x^{64t/2} A_1$
6: $B = B_0 + x^{64t/2} B_1$
7: // Recursive multiplication
8: $R_0 \leftarrow$ `KaratRec`$(A_0, B_0, t/2)$
9: $R_1 \leftarrow$ `KaratRec`$(A_1, B_1, t/2)$
10: $R_2 \leftarrow$ `KaratRec`$(A_0 + A_1, B_0 + B_1, t/2)$
11: // Reconstruction
12: $R \leftarrow R_0 + (R_0 + R_1 + R_2)X^{64t/2} + R_1 X^{64t}$
13: **return** (R)
14: **end if**

Application to the HQC multiplication over $\mathbb{F}_2[X]$ The set of parameters for the HQC protocols leads to the following construction of the multiplications over $\mathbb{F}_2[X]$ depicted in table 9.

3.3 Hardware Implementation

We have implemented HQC in its entirety on an Artix-7 FPGA, using High-Level Synthesis (HLS). In order to be compatible with HLS, we have produced an alternative version of our software library, that can be compiled in C and run in software or transformed by HLS into VHDL code. This greatly simplifies the maintainability of the code with respect to a pure VHDL implementation. The implementation is available in the folder `Hardware_Implementation` and has detailed readme files explaining its usage. It provides a set of test benchs for the key generation, encapsulation and decapsulation functions that verify that the hardware implementation provides exactly the same output as the reference implementation.

Table 9: Implementation of the multiplications over $\mathbb{F}_2[X]$

Multiplication over $\mathbb{F}_2[X]$			
Version	hqc-128	hqc-192	hqc-256
HQC Size (bits)	17669	35851	57637
Main multiplication Size (bits)	`Toom3-Karat3` 18048	`Toom3-Karat3` 36480	`Toom-Cook 3` 59904
Elementary mult. Size (bits)	`3-Karatsuba` 6144	`3-Karatsuba` 12288	`5-Karatsuba` 20480

Instance	KeyGen	Encaps	Decaps
hqc-128	75	177	323
hqc-192	175	404	669
hqc-256	356	799	1427

Table 10: Performance in kilocycles of the optimized implementation using `AVX2` instructions for different instances of HQC.

The HLS-compatible C implementation[2] can be automatically translated in two VHDL implementations, one high-throughput (called perf) and one compact. It is also possible to implement only one function (key generation, encapsulation and decapsulation) or to implement all of them with the benefit of resource sharing (i.e. the cost of implementing the three functions together is quite below the sum of the costs of the functions taken independently). For the moment we have only optimized and studied the performance for the VHDL generated for HQC L1.

The performance figures can be resumed as follows: the perf implementation requires 6.6k slices in an Artix-7 and provides key generation/encapsulation/decapsulation in 0.27/0.52/1.2 milliseconds; the compact implementation requires 3.1k slices in the same FPGA and provides the same functionalities in 4.8/12/16 milliseconds. More detailed figures can be found in the following tables. First we provide the results for our perf implementation.

HQC L1 function	Area (slices)	LUTs	FF	BRAM	Cycles	Freq. (MHz)	Time (ms)
All functions	6.6k	20k	16k	12.5	320k	148	2.2
Keygen	3.9k	12k	9k	3	40k	150	0.27
Encaps	5.5k	16k	13k	5	89k	151	0.59
Decaps	6.2k	19k	15k	9	190k	152	1.2

As the figures highlight, the implementation is quite compact for a throughput oriented

[2]Note that the files of our implementation have the extension .cpp as we use C++ datatypes that make data fiddling easier, but besides this bit manipulations inside the data all of our code is pure C as in the original library.

implementation, requiring just six thousand slices, including the area taken by the Keccak functions. The throughput obtained is also well balanced with a remarkably fast key generation. HLS has the reputation in cryptography of providing large and slow implementations. Whereas the result is probably suboptimal and it is possible to provide a pure-VHDL implementation that is faster and smaller, these figures show that HQC is hardware friendly enough to have at the same time compacity, high throughput, and easy maintainability with an HLS implementation.

The compact implementation increases significantly (around a factor ten) the time required by each function while dividing the surface required by two. It may be interesting in niche settings in which the FPGA surface has other important usages and doing a few transactions per second is enough (e.g. a satellite). The performance figures of the compact implementation are as follows.

HQC L1 function	Area (slices)	LUTs	FF	BRAM	Cycles	Freq. (MHz)	Time (ms)
All functions	3.1k	8.9k	6.4k	14	4.3m	132	32
Keygen	1.5k	4.7k	2.7k	3	630k	129	4.8
Encaps	2.1k	6.4k	4.1k	5	1.5m	127	12
Decaps	2.7k	7.7k	5.6k	10.5	2.1m	130	16

4 Known Answer Test Values

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

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

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

5 Security

5.1 IND-CPA security

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

Theorem 5.1. *For any* IND-CPA *adversary* $\mathcal{A}$ *against the* HQC.PKE *scheme (Figure 2), there exists adversaries* $\mathcal{B}_1$ *and* $\mathcal{B}_2$ *both running in about the same time as* $\mathcal{A}$ *such that:*

$$\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{HQC.PKE}}(\mathcal{A}) \leq 2 \cdot \left(\mathsf{Adv}_{\mathsf{2\text{-}DQCSD\text{-}P}}(\mathcal{B}_1) + \mathsf{Adv}_{\mathsf{3\text{-}DQCSD\text{-}PT}}(\mathcal{B}_2)\right). \tag{21}$$

Proof of Theorem 5.1. We build a sequence of games transitioning from an adversary receiving an encryption of message $\mathbf{m}_0$ to an adversary receiving an encryption of a message $\mathbf{m}_1$, and show that if the adversary manages to distinguish one from the other, then one can build a simulator breaking the 2-DQCSD-P assumption or the 3-DQCSD-PT assumption.

Game $\boldsymbol{G}_1$: In this game, we encrypt $\mathbf{m}_0$ following the protocol:

$\underline{\mathbf{Game}^1_{\mathsf{HQC.PKE},\mathcal{A}}(\lambda)}$
1. $\mathsf{param} \leftarrow \mathsf{Setup}(1^\lambda)$
2. $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{param})$ with $\mathsf{pk} = (\mathbf{h}, \mathbf{s} = \mathbf{x} + \mathbf{h} \cdot \mathbf{y})$ and $\mathsf{sk} = (\mathbf{x}, \mathbf{y})$
3. $(\mathbf{m}_0, \mathbf{m}_1) \leftarrow \mathcal{A}_{\mathsf{CHOOSE}}(\mathsf{pk})$
4. $\mathbf{c} = (\mathbf{u}, \mathbf{v}) \leftarrow \mathsf{Encrypt}(\mathsf{pk}, \mathbf{m}_0)$
5. $b \leftarrow \mathcal{A}_{\mathsf{GUESS}}(\mathsf{pk}, \mathbf{c})$
6. `RETURN` b

Game $\boldsymbol{G}_2$: In this game we forget the decryption key $\mathsf{sk} = (\mathbf{x}, \mathbf{y})$, take $\mathbf{s}$ at random with parity b_2 and then follow the protocol as in Game $\boldsymbol{G}_1$ for the remaining steps:

$\underline{\mathbf{Game}^2_{\mathsf{HQC.PKE},\mathcal{A}}(\lambda)}$
1. $\mathsf{param} \leftarrow \mathsf{Setup}(1^\lambda)$

2a. $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{param})$ with $\mathsf{pk} = (\mathbf{h}, \mathbf{s} = \mathbf{x} + \mathbf{h} \cdot \mathbf{y})$ and $\mathsf{sk} = (\mathbf{x}, \mathbf{y})$

2b. $\mathbf{s} \xleftarrow{\$} \mathbb{F}^n_{2,b_2}$

2c. $(\mathsf{pk}, \mathsf{sk}) \leftarrow ((\mathbf{h}, \mathbf{s}), \mathbf{0})$

3. $(\mathbf{m}_0, \mathbf{m}_1) \leftarrow \mathcal{A}_{\mathsf{CHOOSE}}(\mathsf{pk})$
4. $\mathbf{c} = (\mathbf{u}, \mathbf{v}) \leftarrow \mathsf{Encrypt}(\mathsf{pk}, \mathbf{m}_0)$
5. $b \leftarrow \mathcal{A}_{\mathsf{GUESS}}(\mathsf{pk}, \mathbf{c})$
6. `RETURN` b

Suppose that an adversary $\mathcal{B}_1$ is able to distinguish Game $\boldsymbol{G}_1$ from Game $\boldsymbol{G}_2$ 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.

$\underline{\mathcal{D}^\lambda_{\text{2-DQCSD-P}}\,(\mathbf{H}, \mathbf{s})}$
1. Set $\mathsf{param} \leftarrow \mathsf{Setup}(1^\lambda)$
2. Compute $\mathbf{h}$ from $\mathbf{H} = (\mathbf{I}_n \ \mathbf{rot}(\mathbf{h}))$
3. Compute $\mathsf{pk} \leftarrow (\mathbf{h}, \mathbf{s})$
4. Compute $(\mathbf{m}_0, \mathbf{m}_1) \leftarrow \mathcal{B}_{1,\mathsf{CHOOSE}}(\mathsf{pk})$
5. Compute $\mathbf{c} = (\mathbf{u}, \mathbf{v}) \leftarrow \mathsf{Encrypt}(\mathsf{pk}, \mathbf{m}_0)$
6. Get $b' \leftarrow \mathcal{B}_{1,\mathsf{GUESS}}(\mathsf{pk}, \mathbf{c})$
7. If $b' = \boldsymbol{G}_1$, output 2-$\mathcal{QCSD}$-$\mathcal{P}(n, w, b_1, b_2)$ distribution
8. If $b' = \boldsymbol{G}_2$, output $\mathcal{U}(\mathbb{F}^{n \times 2n}_{2,b_1} \times \mathbb{F}^n_{2,b_2})$ distribution

Note that pk is sampled from the 2-$\mathcal{QCSD}$-$\mathcal{P}(n, w, b_1, b_2)$ distribution in Game $\boldsymbol{G}_1$ 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 $\boldsymbol{G}_2$ therefore the advantage of $\mathcal{D}^{\lambda}_{\text{2-DQCSD-P}}$ is the same as the advantage of $\mathcal{B}_1$.

Game $\boldsymbol{G}_3$: 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.

$\underline{\mathbf{Game}^{3}_{\mathsf{HQC.PKE},\mathcal{A}}(\lambda)}$
1. $\mathsf{param} \leftarrow \mathsf{Setup}(1^\lambda)$
2a. $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{param})$ with $\mathsf{pk} = (\mathbf{h}, \mathbf{s} = \mathbf{x} + \mathbf{h} \cdot \mathbf{y})$ and $\mathsf{sk} = (\mathbf{x}, \mathbf{y})$
2b. $\mathbf{s} \xleftarrow{\$} \mathbb{F}_{2,b_2}^n$
2c. $(\mathsf{pk}, \mathsf{sk}) \leftarrow ((\mathbf{h}, \mathbf{s}), \mathbf{0})$
3. $(\mathbf{m}_0, \mathbf{m}_1) \leftarrow \mathcal{A}_{\mathsf{CHOOSE}}(\mathsf{pk})$
4a. $\mathbf{e} \xleftarrow{\$} \mathbb{F}_2^n, \mathbf{r} = (\mathbf{r}_1, \mathbf{r}_2) \xleftarrow{\$} \mathbb{F}_{2,w_{\boldsymbol{r}}}^n \times \mathbb{F}_{2,w_{\boldsymbol{r}}}^n$
4b. $\mathbf{u} \leftarrow \mathbf{r}_1 + \mathbf{h}\mathbf{r}_2$ and $\mathbf{v} \leftarrow \mathsf{truncate}(\mathbf{m}_0\mathbf{G} + \mathbf{s} \cdot \mathbf{r}_2 + \mathbf{e}, \ell)$
4c. $\mathbf{c} \leftarrow (\mathbf{u}, \mathbf{v})$
5. $b \leftarrow \mathcal{A}_{\mathsf{GUESS}}(\mathsf{pk}, \mathbf{c})$
6. `RETURN` b

Suppose that an adversary $\mathcal{B}_2$ is able to distinguish Game $\boldsymbol{G}_2$ from Game $\boldsymbol{G}_3$ 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.

$\underline{\mathcal{D}^{\lambda}_{\text{3-DQCSD-PT}}(\mathbf{H}, (\mathbf{u}, \mathbf{v}))}$
1. Set $\mathsf{param} \leftarrow \mathsf{Setup}(1^\lambda)$
2. Compute $\mathbf{G}$ from param
3. Compute $\mathbf{h}$ and $\mathbf{s}$ from $\mathbf{H} = \begin{pmatrix} \mathbf{I}_n & \mathbf{0} & \mathbf{rot}(\mathbf{h}) \\ \mathbf{0} & \mathbf{I}_n & \mathbf{rot}(\mathbf{s}) \end{pmatrix}$
4. Compute $\mathsf{pk} \leftarrow (\mathbf{h}, \mathbf{s})$
5. Get $(\mathbf{m}_0, \mathbf{m}_1) \leftarrow \mathcal{B}_{2,\mathsf{CHOOSE}}(\mathsf{pk})$
6. Compute $\mathbf{c} \leftarrow (\mathbf{u}, \mathbf{m}_0\mathbf{G} + \mathbf{v})$
7. Get $b' \leftarrow \mathcal{B}_{2,\mathsf{GUESS}}(\mathsf{pk}, \mathbf{c})$
8. If $b' = \boldsymbol{G}_2$, output 3-$\mathcal{QCSD}$-$\mathcal{PT}(n, w, b_1, b_2, b_3, \ell)$ distribution
9. If $b' = \boldsymbol{G}_3$, output $\mathcal{U}(\mathbb{F}_{2,b_1,b_2}^{2n\times 3n} \times (\mathbb{F}_{2,b_3}^n \times \mathbb{F}_2^{n-\ell}))$ distribution

As we have:

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

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

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

follows the 3-$\mathcal{QCSD}$-$\mathcal{PT}(n, w, b_1, b_2, b_3, \ell)$ 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 $\mathcal{B}_2$.

Game $\boldsymbol{G}_4$: In this game, we encrypt the message $\mathbf{m}_1$ instead of $\mathbf{m}_0$.

$\underline{\mathbf{Game}^4_{\mathsf{HQC.PKE},\mathcal{A}}(\lambda)}$
1. $\mathsf{param} \leftarrow \mathsf{Setup}(1^\lambda)$
2a. $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{param})$ with $\mathsf{pk} = (\mathbf{h}, \mathbf{s} = \mathbf{x} + \mathbf{h} \cdot \mathbf{y})$ and $\mathsf{sk} = (\mathbf{x}, \mathbf{y})$
2b. $\mathbf{s} \xleftarrow{\$} \mathbb{F}_{2,b_2}^n$
2c. $(\mathsf{pk}, \mathsf{sk}) \leftarrow ((\mathbf{h}, \mathbf{s}), \mathbf{0})$
3. $(\mathbf{m}_0, \mathbf{m}_1) \leftarrow \mathcal{A}_{\mathsf{CHOOSE}}(\mathsf{pk})$
4a. $\mathbf{e} \xleftarrow{\$} \mathbb{F}_2^n, \mathbf{r} = (\mathbf{r}_1, \mathbf{r}_2) \xleftarrow{\$} \mathbb{F}_{2,w_{\boldsymbol{r}}}^n \times \mathbb{F}_{2,w_{\boldsymbol{r}}}^n$
4b. $\mathbf{u} \leftarrow \mathbf{r}_1 + \mathbf{h}\mathbf{r}_2$ and $\mathbf{v} \leftarrow \mathsf{truncate}(\mathbf{m}_1\mathbf{G} + \mathbf{s} \cdot \mathbf{r}_2 + \mathbf{e}, \ell)$
4c. $\mathbf{c} \leftarrow (\mathbf{u}, \mathbf{v})$
5. $b \leftarrow \mathcal{A}_{\mathsf{GUESS}}(\mathsf{pk}, \mathbf{c})$
6. `RETURN` b

The outputs from Game $\boldsymbol{G}_3$ and Game $\boldsymbol{G}_4$ follow the exact same distribution, and therefore the two games are indistinguishable from an information-theoretic point of view. Indeed, $\mathbf{u}$ is computed identically in both Game $\boldsymbol{G}_3$ and Game $\boldsymbol{G}_4$. In addition, $\mathbf{v}$ is indistinguishable between both games as it is masked by the random vector $\mathbf{e}$ and any parity difference between the Hamming weights of $\mathbf{m}_0\mathbf{G}$ and $\mathbf{m}_1\mathbf{G}$ is hidden by its truncation.

Game $\boldsymbol{G}_5$: In this game, we pick $\mathbf{r}_1, \mathbf{r}_2$ and $\mathbf{e}$ with their expected weight. We do not explicit this game as Game $\boldsymbol{G}_4$ and Game $\boldsymbol{G}_5$ are equivalent to Game $\boldsymbol{G}_3$ and Game $\boldsymbol{G}_2$ except that $\mathbf{m}_1$ is used instead of $\mathbf{m}_0$. Hence, a distinguisher between these two games breaks the 3-DQCSD-PT assumption.

Game $\boldsymbol{G}_6$: In this game, the public key is sampled as expected in the protocol. We do not explicit this game as Game $\boldsymbol{G}_5$ and Game $\boldsymbol{G}_6$ are equivalents to Game $\boldsymbol{G}_2$ and Game $\boldsymbol{G}_1$. Hence, a distinguisher between these two games breaks the 2-DQCSD-P assumption.

We have built a sequence of games allowing a simulator to transform a ciphertext of a message $\mathbf{m}_0$ into a ciphertext of a message $\mathbf{m}_1$. As a result, the advantage of an adversary against the IND-CPA experiment is bounded by:

$$\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{HQC.PKE}}(\mathcal{A}) \leq 2 \cdot (\mathsf{Adv}_{\text{2-DQCSD-P}}(\mathcal{B}_1) + \mathsf{Adv}_{\text{3-DQCSD-PT}}(\mathcal{B}_2)) \,. \tag{22}$$

□

5.2 IND-CCA2 security

In this section we provide the IND-CCA2 proof for HQC.

5.2.1 HQC.PKE correction and DFR

Definition 5.2.1 (δ-correct PKE [20]). *A* PKE $(\mathsf{Keygen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ *is* δ*-correct if*

$$\mathbb{E}\left(\max_{\mathbf{m}\in\mathcal{M}} \Pr\left[\mathsf{Decrypt}(\mathsf{sk}, \mathbf{c}) \neq \mathbf{m} \mid \mathbf{c} \leftarrow \mathsf{Encrypt}(\mathsf{pk}, \mathbf{m})\right]\right) \leq \delta. \tag{23}$$

where the expectation is taken over $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{param})$.

Definition 5.2.2 (δ-correct KEM [20]). *A* KEM $(\mathsf{Keygen}, \mathsf{Encapsulate}, \mathsf{Decapsulate})$ *is* δ*-correct if*

$$\Pr = \left[\mathsf{Decapsulate}(\mathsf{sk}, \mathbf{c}) \neq K \;\middle|\; \begin{array}{l}(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{param});\\ (K, \mathbf{c}) \leftarrow \mathsf{Encapsulate}(\mathsf{pk})\end{array}\right] \leq \delta \tag{24}$$

In HQC.PKE the failure to decrypt a ciphertext $(\mathbf{u}, \mathbf{v})$ 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 23 simplifies to

$$\Pr\left[\mathsf{Decrypt}(\mathsf{sk}, \mathbf{c}) \neq \mathbf{m} \mid \mathbf{c} \leftarrow \mathsf{Encrypt}(\mathsf{pk}, \mathbf{m})\right] \leq \delta. \tag{25}$$

Which is equivalent to the following probability that we analyze in section 2.5.8,

$$\Pr\left[\omega\left(\mathbf{x}\cdot\mathbf{r}_2 - \mathbf{r}_1\cdot\mathbf{y} + \mathbf{e}\right) > \Delta \;\middle|\; \begin{array}{l}(\mathbf{x}, \mathbf{y}) \xleftarrow{\$} \mathcal{R}_w \times \mathcal{R}_w;\\ \mathbf{e} \xleftarrow{\$} \mathcal{R}_{w_\mathbf{e}};\\ \mathbf{r} = (\mathbf{r}_1, \mathbf{r}_2) \xleftarrow{\$} \mathcal{R}_{w_\mathbf{r}} \times \mathcal{R}_{w_\mathbf{r}}\end{array}\right] \leq \delta. \tag{26}$$

5.2.2 A CCA proof for HQC

Following the HHK framework [20], the public-key encryption scheme HQC.PKE is converted to a deterministic public-key encryption scheme $\mathsf{HQC.PKE}_1$ (see Figure 6). The security of $\mathsf{HQC.PKE}_1$ is reduced to the security of HQC.PKE. It is worth mentioning, that to prevent multi-ciphertext attack, we introduced a minor modification to $\mathsf{HQC.PKE}_1$ by incorporating a public salt value into the ciphertext. So that the randomness θ is computed from a salt together with the first 32 bytes of the public key. More formally we have the following Lemma.

Lemma 5.2 (Theorem 3.2 in [20]). *If* HQC.PKE *is a δ-correct public-key encryption scheme, for any* OW-PCA *adversary $\mathcal{B}'$ against* $\mathsf{HQC.PKE}_1$ *issuing at most $q_{\mathcal{G}}$ queries to the sampler (modeled as a random oracle), there exists an* IND-CPA *adversary $\mathcal{A}$ against* HQC.PKE *such that*

$$\mathsf{Adv}^{\mathsf{OW\text{-}PCA}}_{\mathsf{HQC.PKE}_1}(\mathcal{B}') \leq q_{\mathcal{G}} \cdot \delta + \frac{2 \cdot q_{\mathcal{G}} + 1}{|\mathcal{M}|} + 3 \cdot \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{HQC.PKE}}(\mathcal{A}). \tag{27}$$

HHK defines a transform from a deterministic public-key encryption scheme $\mathsf{HQC.PKE}_1$ to a key encapsulation mechanism $\mathsf{HQC.KEM}^{\not\perp}$ (see Figure 7). The IND-CCA2 security of the $\mathsf{HQC.KEM}^{\not\perp}$ is reduced to the OW-PCA security of $\mathsf{HQC.PKE}_1$, more formally, using Theorem 3.4 in [20] we have the following result.

Lemma 5.3 (Theorem 3.4 in [20]). *If* $\mathsf{HQC.PKE}_1$ *is δ-correct then* $\mathsf{HQC.KEM}^{\not\perp}$ *is also δ-correct. For any* IND-CCA2 *adversary $\mathcal{B}$ against* $\mathsf{HQC.KEM}^{\not\perp}$ *issuing at most $q_{\mathcal{K}}$ queries to the key generation function $\mathcal{K}$ (modeled as a random oracle), there exists an* OW-PCA *adversary $\mathcal{B}'$ against* $\mathsf{HQC.PKE}_1$ *such that*

$$\mathsf{Adv}^{\mathsf{IND\text{-}CCA2}}_{\mathsf{HQC.KEM}^{\not\perp}}(\mathcal{B}) \leq \frac{q_{\mathcal{K}}}{|\mathcal{M}|} + \mathsf{Adv}^{\mathsf{OW\text{-}PCA}}_{\mathsf{HQC.PKE}_1}(\mathcal{B}'). \tag{28}$$

Notice that we have that $\mathsf{HQC.PKE}_1$ is OW-PCA in the sense of HHK, therefore Lemmas 5.2 and 5.3 hold in their framework.

Theorem 5.4. *If* HQC.PKE *is a δ-correct public-key encryption scheme, then, for all* IND-CCA2 *adversary $\mathcal{B}$ against* $\mathsf{HQC.KEM}^{\not\perp}$ *issuing at most $q_{\mathcal{K}}$ queries to $\mathcal{K}$ and $q_{\mathcal{G}}$ to $\mathcal{G}$ (where $\mathcal{K}$ and $\mathcal{G}$ are modeled as random oracles), there exits an* IND-CPA *adversary $\mathcal{A}$ against* HQC.PKE*, running about the same time, such that*

$$\mathsf{Adv}^{\mathsf{IND\text{-}CCA2}}_{\mathsf{HQC.KEM}^{\not\perp}}(\mathcal{B}) \leq q_{\mathcal{G}} \cdot \delta + \frac{2 \cdot q_{\mathcal{G}} + 1 + q_{\mathcal{K}}}{|\mathcal{M}|} + 3 \cdot \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{HQC.PKE}}(\mathcal{A}). \tag{29}$$

Proof. The proof combines Lemmas 5.2 and 5.3. □

- $\mathsf{Setup}(1^\lambda)$: outputs the global parameters $\mathsf{param} = (n, k, \delta, w, w_{\mathbf{r}}, w_{\mathbf{e}}, \ell)$.
- $\mathsf{KeyGen}(\mathsf{param})$: samples $\mathbf{h} \xleftarrow{\$} \mathcal{R}$, the generator matrix $\mathbf{G} \in \mathbb{F}_2^{k \times n}$ of $\mathcal{C}$, $\mathsf{sk} = (\mathbf{x}, \mathbf{y}) \xleftarrow{\$} \mathcal{R}_w \times \mathcal{R}_w$, sets $\mathsf{pk} = (\mathbf{h}, \mathbf{s} = \mathbf{x} + \mathbf{h} \cdot \mathbf{y})$, and returns $(\mathsf{pk}, \mathsf{sk})$.
- $\mathsf{Encrypt}_1(\mathsf{pk}, \mathbf{m}, salt)$: Derive the randomness $\theta \leftarrow \mathcal{G}(\mathbf{m} \| \mathsf{firstBytes}(\mathsf{pk}, 32) \| salt)$ and use θ to generates $(\mathbf{e}, \mathbf{r}_1, \mathbf{r}_2)$ such that $\omega(\mathbf{e}) = w_{\mathbf{e}}$ and $\omega(\mathbf{r}_1) = \omega(\mathbf{r}_2) = w_{\mathbf{r}}$, sets $\mathbf{u} = \mathbf{r}_1 + \mathbf{h} \cdot \mathbf{r}_2$ and $\mathbf{v} = \mathsf{truncate}(\mathbf{mG} + \mathbf{s} \cdot \mathbf{r}_2 + \mathbf{e}, \ell)$, returns $\mathbf{c} = (\mathbf{u}, \mathbf{v})$.
- $\mathsf{Decrypt}_1(\mathsf{sk}, \mathbf{c}, salt)$: computes $\mathbf{m} \leftarrow \mathsf{Decrypt}(\mathsf{sk}, \mathbf{c})$. If $\mathbf{m} = \perp$ or $\mathbf{c} \neq \mathsf{Encrypt}_1(\mathsf{pk}, \mathbf{m}, salt)$ then returns $\perp$ otherwise returns $\mathbf{m}$.

Figure 6: $\mathsf{HQC.PKE}_1$ - A deterministic version of HQC.PKE

- $\mathsf{Setup}(1^\lambda)$: outputs the global parameters $\mathsf{param} = (n, k, \delta, w, w_{\mathbf{r}}, w_{\mathbf{e}}, \ell)$.
- $\mathsf{KeyGen}(\mathsf{param})$: samples $\mathbf{h} \xleftarrow{\$} \mathcal{R}$, the generator matrix $\mathbf{G} \in \mathbb{F}_2^{k\times n}$ of $\mathcal{C}$, $\mathsf{sk} = (\mathbf{x}, \mathbf{y}, \sigma) \xleftarrow{\$} \mathcal{R}_w \times \mathcal{R}_w \times \mathcal{M}$, sets $\mathsf{pk} = (\mathbf{h}, \mathbf{s} = \mathbf{x} + \mathbf{h} \cdot \mathbf{y})$, and returns $(\mathsf{pk}, \mathsf{sk})$.
- $\mathsf{Encapsulate}(\mathsf{pk})$: generates $\mathbf{m} \xleftarrow{\$} \mathcal{M}$, $salt \xleftarrow{\$} \mathbb{F}_2^{128}$, computes $\mathbf{c} \leftarrow \mathsf{Encrypt}_1(\mathsf{pk}, \mathbf{m}, salt)$. Computes $K \leftarrow \mathcal{K}(\mathbf{m}, \mathbf{c})$, and return $(K, \mathbf{c}, salt)$.
- $\mathsf{Decapsulate}(\mathsf{sk}, \mathbf{c}, salt)$: compute $\mathbf{m} \leftarrow \mathsf{Decrypt}_1(\mathsf{sk}, \mathbf{c}, salt)$. If $\mathbf{m} \neq \perp$ then $K \leftarrow \mathcal{K}(\mathbf{m}, \mathbf{c})$ else $K \leftarrow \mathcal{K}(\sigma, \mathbf{c})$.

Figure 7: $\mathsf{HQC.KEM}^{\not\perp}$ - HQC.KEM with implicit rejection from $\mathsf{HQC.PKE}_1$

Theorem 5.5. *If* HQC.PKE *is a δ-correct public-key encryption scheme, for any* IND-CCA2 *adversary $\mathcal{B}$ against* $\mathsf{HQC.KEM}^{\not\perp}$ *issuing at most q queries to $\mathcal{K}$ or $\mathcal{G}$ (where $\mathcal{K}$ and $\mathcal{G}$ are modeled as random oracles), there there exists adversaries $\mathcal{B}_1$ and $\mathcal{B}_2$ both running in about the same time as $\mathcal{B}$ such that:*

$$\mathsf{Adv}^{\mathsf{IND\text{-}CCA2}}_{\mathsf{HQC.KEM}^{\not\perp}}(\mathcal{B}) \leq q_{\mathcal{G}} \cdot \delta + \frac{2 \cdot q_{\mathcal{G}} + 1 + q_{\mathcal{K}}}{|\mathcal{M}|} + 6 \cdot (\mathsf{Adv}_{\mathsf{2\text{-}DQCSD\text{-}P}}(\mathcal{B}_1) + \mathsf{Adv}_{\mathsf{3\text{-}DQCSD\text{-}PT}}(\mathcal{B}_2)) \,. \quad (30)$$

Proof. The proof combines Theorems 5.1 and 5.4. □

5.3 Security proof with non uniform randomness generation

In this section, we show that there is no effective impact on the IND-CCA2 security of HQC if vectors of small weights are sampled non uniformly, but close enough to uniform. In order to do so, we use a similar approach as in [33]. Let us start by recalling the following results from [33].

Proposition 5.3.1 (Proposition 3 in [33])**.** *Let $\mathcal{S}$ be the distribution over $\mathcal{R}_w$ when sampling using Algorithm 1, when $x \leftarrow \mathsf{randbits}(B, \mathsf{prng})$ behaves as a random oracle which yields uniformly distributed integers, $0 \leq x < 2^B$. for any integer $B > 0$, we have*

$$\prod_{i=0}^{w-1} \left(1 - \frac{n_i}{2^B}\right) = \tau_{min} \leq \frac{\Pr\left[\mathbf{e} \,\middle|\, \mathbf{e} \xleftarrow{\mathcal{S}} \mathcal{R}_w\right]}{\Pr\left[\mathbf{e} \,\middle|\, \mathbf{e} \xleftarrow{\$} \mathcal{R}_w\right]} \leq \tau_{max} = \prod_{i=0}^{w-1} \left(1 + \frac{(n-i) - n_i}{2^B}\right). \quad (31)$$

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

For HQC parameters, the ratios τ_{min} and τ_{max} are very close to 1 (see Table 11).

Using the following Lemma adapted from [33], we have that the advantage of any adversary when a vector $\mathbf{e}$ of weight w is sampled following Algorithm 1 instead of uniform distribution, cannot increase by a factor larger than τ_{max}.

$B = 32$				
Security	n	$w_{\mathbf{r}}$	τ_{min}	τ_{max}
128	17,669	75	0.99938	1.00061
192	35,851	114	0.99808	1.00188
256	57,637	149	0.99803	1.00202

Table 11: Bias between the uniform distribution and the output of Algorithm 1 for encryption randomness vectors of weight $w_{\mathbf{r}}$ or $w_{\mathbf{e}}$.

Lemma 5.6 (Adapted from [33]). *For any real-valued random variable $V : \mathcal{R}_w \to \mathbb{R}$, we have*

$$\sum_{\mathbf{e}\in\mathcal{R}_w} \Pr\left[\mathbf{e} \;\middle|\; \mathbf{e} \xleftarrow{\mathcal{S}} \mathcal{R}_w\right] V(\mathbf{e}) \leq \tau_{max} \cdot \sum_{\mathbf{e}\in\mathcal{R}_w} \Pr\left[\mathbf{e} \;\middle|\; \mathbf{e} \xleftarrow{\$} \mathcal{R}_w\right] V(\mathbf{e}). \tag{32}$$

5.3.1 Arguments related to the security reduction

Following [33], we show that the IND-CCA2 security proof of HQC is not impacted when the encryption randomness is sampled using Algorithm 1 rather than the uniform distribution. More precisely, we check that Equations 27 and 28 still holds.

In Equation 28, the inequality holds independently of the distribution of the vectors obtained from $\mathcal{G}$. Therefore, this equation still holds if we switch to Algorithm 1. On the other hand, and as shown in [33], to prove the inequality in Equation 27, one should revisit the proof. Indeed, in the third term in Equation 27, the derandomization of $\mathsf{HQC.PKE}$ would not lead to $\mathsf{HQC.PKE}_1$. We rather consider a variant $\mathsf{HQC.PKE}^{\mathcal{S}}$ (Figure 8) in which the vectors $(\mathbf{e}, \mathbf{r}_1, \mathbf{r}_2)$ are sampled using Algorithm 1.

1. The first term $q_{\mathcal{G}} \cdot \delta$ in Equation 27 is related to the δ-correctness and by consequence to the DFR of our scheme. Let $(\mathbf{e}, \mathbf{r}_1, \mathbf{r}_2)$ be sampled using Algorithm 1 rather than the uniform distribution. Then, δ must be such that

$$\Pr = \left[\omega\left(\mathbf{x}\cdot\mathbf{r}_2 - \mathbf{r}_1\cdot\mathbf{y} + \mathbf{e}\right) > \Delta \;\middle|\; \begin{array}{l} (\mathbf{x},\mathbf{y}) \xleftarrow{\$} \mathcal{R}_w \times \mathcal{R}_w; \\ \mathbf{e} \xleftarrow{\mathcal{S}} \mathcal{R}_{w_{\mathbf{e}}}; \\ \mathbf{r} = (\mathbf{r}_1,\mathbf{r}_2) \xleftarrow{\mathcal{S}} \mathcal{R}_{w_{\mathbf{r}}} \times \mathcal{R}_{w_{\mathbf{r}}} \end{array}\right] \leq \delta. \tag{33}$$

   Using Lemma 5.6, we have that the above probability, increases by a factor at most τ_{max}.

2. As showed [33], the middle term remain unchanged since it is independent of the output distribution of $\mathcal{G}$.

3. In the right most term, the advantage become $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{HQC.PKE}^{\mathcal{S}}}(A)$. Using Lemma 5.6, we have that $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{HQC.PKE}^{\mathcal{S}}}(A) \leq \tau_{max} \cdot \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{HQC.PKE}}(A)$.

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

Figure 8: $\mathsf{HQC.PKE}^{\mathcal{S}}$ a modified HQC.PKE with non uniform encryption randomness.

5.3.2 Arguments related to the public key generation

Figure 9 presents two games associated with HQC, which vary solely in the method of sampling for the secret key - either through uniform sampling or utilizing a specific distribution.

Lemma 5.7. *If there is a polynomial time adversary A that can succeed in winning the biased game $\mathbf{Exp}_1$ with a probability of p, it can also achieve victory in game $\mathbf{Exp}'_1$ with probability $\leq \tau p$ where*

$$\tau = \frac{\Pr\left[(\mathbf{x}, \mathbf{y}) \;\middle|\; (\mathbf{x}, \mathbf{y}) \xleftarrow{\mathcal{S}} \mathcal{R}_w \times \mathcal{R}_w\right]}{\Pr\left[(\mathbf{x}, \mathbf{y}) \;\middle|\; (\mathbf{x}, \mathbf{y}) \xleftarrow{\$} \mathcal{R}_w \times \mathcal{R}_w\right]}. \tag{34}$$

Proof. The proof is straightforward application of Lemma 5.6. □

In the security games associated with public key encryption and key encapsulation mechanism discussed in the sections 5.1 and 5.2, a single key pair is randomly selected at the start of the game, and a single instance of distribution $\mathcal{S}$ is used. Consequently, no adversary can achieve a success probability greater than a factor of τ when a biased key is utilized instead of a uniform one. Using Proposition 5.3.1 we have that

$$\tau \leq 2 \prod_{i=0}^{w-1} \left(1 + \frac{(n-i) - n_i}{2^{32}}\right), n_i = 2^{32} \bmod (n-i), 0 \leq i \leq w. \tag{35}$$

6 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 [30] and improved later by Stern [35], then Dumer [12]. Recent works [27, 5, 28] suggest a complexity of order $2^{cw(1+\mathsf{negl}(1))}$, for some constant c. A particular work focusing on the

$\mathbf{Exp}_1$ (biased key)	$\mathbf{Exp}'_1$ (uniform key)
1. $(\mathbf{x}, \mathbf{y}) \xleftarrow{\mathcal{S}} \mathcal{R}_w \times \mathcal{R}_w$	1. $(\mathbf{x}, \mathbf{y}) \xleftarrow{\$} \mathcal{R}_w \times \mathcal{R}_w$
.	.
.	.
.	.

Figure 9: Experiments using biased and uniform key sampling.

$B = 32$				
Security	n	w	τ_{min}	τ_{max}
128	17,669	66	0.99945	1.00054
192	35,851	100	0.99832	1.00166
256	57,637	131	0.99651	1.00353

Table 12: Bias between the uniform distribution and the output of Algorithm 1 for the secret key vectors of weight w.

regime $w = \mathsf{negl}(n)$ confirms this formula, with a close dependence between c and the rate k/n of the code being used [11].

Specific structural attacks. Quasi-cyclic codes have a special structure which may potentially open the door to specific structural attacks. A first generic attack is the DOOM attack [32] 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 [19, 25, 32], 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 2.7 that fit security levels 1, 3 and 5, as defined by NIST. The quantum-safe security is obtained by dividing the security bits by two (taking the square root of the complexity) [8]. Best known

attacks include the works from [10, 9, 13, 27, 5, 28] and for quantum attacks, the work of [8]. In the setting $w = \mathcal{O}(\sqrt{n})$, best known attacks have a complexity in $2^{-t\ln(1-R)(1+o(1))}$ where $t = \mathcal{O}(w)$ and R is the rate of the code [11]. In our configuration, we have $t = 2w$ and $R = 1/2$ for the reduction to the 2-DQCSD problem, and $t = 3w_{\mathbf{r}}$ and $R = 1/3$ for the 3-DQCSD problem. By taking into account the DOOM attack [32], and also the fact that we consider balanced vectors $(\mathbf{x}, \mathbf{y})$ and $(\mathbf{r}_1, \mathbf{e}, \mathbf{r}_2)$ for the attack (which costs only a very small factor, since random words have a good probability to be balanced on each block), we need to divide this complexity by approximately $\sqrt{n}$ (up to polylog factor). The term $o(1)$ is respectively $\log\left(\binom{n}{w}^2/\binom{2n}{2w}\right)$ and $\log\left(\binom{n}{w_{\mathbf{r}}}^3/\binom{3n}{3w_{\mathbf{r}}}\right)$ for the 2-DQCSD and 3-DQCSD problems.

7 Advantages and Limitations

7.1 Advantages

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

- its IND-CPA reduction to a well-understood problem on coding theory: the Quasi-Cyclic Syndrome Decoding problem,
- its immunity against attacks aiming at recovering the hidden structure of the code being used,
- small public key size
- close estimations of its decryption failure rate.
- efficient implementations based on classical decoding algorithms.

The fourth item allows to achieve a tight reduction for the IND-CCA2 security of the KEM-DEM version through the recent transformation of [20].

7.2 Limitations

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

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

References

[1] Carlos Aguilar-Melchor, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, and Gilles Zémor. Efficient encryption from random quasi-cyclic codes. *IEEE Transactions on Information Theory*, 64(5):3927–3943, 2018. *10*, *11*

[2] Francesco Antognazza, Alessandro Barenghi, Gerardo Pelosi, and Ruggero Susella. A High Efficiency Hardware Design for the Post-Quantum KEM HQC. In *2024 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 431–441. IEEE, 2024. *3*, *32*

[3] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography with constant input locality. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 92–110. Springer, Heidelberg, August 2007. *13*

[4] Nicolas Aragon, Philippe Gaborit, and Gilles Zémor. Hqc-rmrs, an instantiation of the hqc encryption framework with a more efficient auxiliary error-correcting code. https://arxiv.org/abs/2005.10741. *17*, *21*

[5] Anja Becker, Antoine Joux, Alexander May, and Alexander Meurer. Decoding random binary linear codes in $2^{n/20}$: How $1 + 1 = 0$ improves information set decoding. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 520–536. Springer, Heidelberg, April 2012. *32*, *46*, *48*

[6] Elwyn Berlekamp. *Algebraic coding theory*. World Scientific, 1968. *24*

[7] Elwyn R Berlekamp, Robert J McEliece, and Henk CA van Tilborg. On the inherent intractability of certain coding problems. *IEEE Transactions on Information Theory*, 24(3):384–386, 1978. http://authors.library.caltech.edu/5607/1/BERieeetit78.pdf. *13*

[8] Daniel J Bernstein. Grover vs. mceliece. In *Post-Quantum Cryptography*, pages 73–80. Springer, 2010. https://cr.yp.to/codes/grovercode-20091123.pdf. *32*, *47*, *48*

[9] Daniel J Bernstein, Tanja Lange, and Christiane Peters. Attacking and defending the mceliece cryptosystem. In *Post-Quantum Cryptography*, pages 31–46. Springer, 2008. https://cr.yp.to/codes/mceliece-20080807.pdf. *32*, *48*

[10] Anne Canteaut and Florent Chabaud. A new algorithm for finding minimum weight words in a linear code: application to mceliece cryptosystem and to narrow-sense bch codes of length 511. *IEEE Transactions on Information Theory*, 44(1):367–378, 1998. http://ieeexplore.ieee.org/document/651067/. *32*, *48*

[11] Rodolfo Canto Torres and Nicolas Sendrier. Analysis of information set decoding for a sub-linear error weight. In Tsuyoshi Takagi, editor, *Post-Quantum Cryptography - 7th*

International Workshop, PQCrypto 2016, Fukuoka, Japan, February 24-26, 2016, Proceedings, volume 9606 of *Lecture Notes in Computer Science*, pages 144–161. Springer, 2016. https://hal.inria.fr/hal-01244886. *32*, *47*, *48*

[12] Ilya Dumer. On minimum distance decoding of linear codes. In *Proc. 5th Joint Soviet-Swedish Int. Workshop Inform. Theory*, pages 50–52, 1991. https://www.researchgate.net/publication/296573348_On_minimum_distance_decoding_of_linear_codes. *46*

[13] Matthieu Finiasz and Nicolas Sendrier. Security bounds for the design of code-based cryptosystems. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 88–105. Springer, Heidelberg, December 2009. *32*, *48*

[14] 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. *12*

[15] 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. *13*

[16] Shuhong Gao and Todd Mateer. Additive fast fourier transforms over finite fields. *IEEE Transactions on Information Theory*, 56(12):6265–6272, 2010. *25*

[17] 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. *14*

[18] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984. *15*

[19] 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. *47*

[20] 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. *10*, *16*, *17*, *42*, *43*, *48*

[21] 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. *11*

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

[23] 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. *14*

[24] 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. *14*

[25] 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. *47*

[26] Florence Jessie MacWilliams and Neil James Alexander Sloane. *The theory of error-correcting codes*, volume 16. Elsevier, 1977. *26*

[27] 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. *46*, *48*

[28] 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. *46*, *48*

[29] 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. *12*

[30] 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/. *46*

[31] Robin Leander Schröder, Stefan Gast, and Qian Guo. Divide and surrender: Exploiting Variable Division Instruction Timing in HQC Key Recovery Attacks. *Cryptology ePrint Archive, Paper 2024/299*, 2024. https://eprint.iacr.org/2024/299. *4*

[32] 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. *14*, *47*, *48*

[33] Nicolas Sendrier. Secure Sampling of Constant Weight Words – Application to BIKE. Cryptology ePrint Archive, Report 2021/1631, 2021. https://eprint.iacr.org/2021/1631. *32*, *44*, *45*

[34] Victor Shoup. NTL: A library for doing number theory. 2001. http://www.shoup.net/ntl. *34*

[35] 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. *46*