Tailorable codes for lattice-based KEMs with applications to compact ML-KEM instantiations

Thales B. Paiva[1], Marcos A. Simplicio Jr[1,2], Syed Mahbub Hafiz[1], Bahattin Yildiz[1], Eduardo L. Cominetti[1], and Henrique S. Ogawa[1]

[1] Future Security Team, LG Electronics, USA
{thalespaiva, msimplicio, ecominetti}@larc.usp.br
{syedmahbub.hafiz, bahattin.yildiz, henrique1.ogawa}@lge.com
[2] Universidade de Sao Paulo, Brazil

Abstract. Compared to elliptic curve cryptography, a main drawback of lattice-based schemes is the larger size of their public keys and ciphertexts. A common procedure for compressing these objects consists essentially of dropping some of their least significant bits. Albeit effective for compression, there is a limit to the number of bits to be dropped before we get a noticeable decryption failure rate (DFR), which is a security concern. To address this issue, this paper presents a family of error-correction codes that, by allowing an increased number of dropped bits while preserving a negligible DFR, can be used for both ciphertext and public-key compression in modern lattice-based schemes. To showcase the impact and practicality of our proposal, we use the highly optimized ML-KEM, a post-quantum lattice-based scheme recently standardized by NIST. We provide detailed procedures for tailoring our codes to ML-KEM's specific noise distributions, and show how to analyze the DFR without independence assumptions on the noise coefficients. Among our results, we achieve between 4% and 8% ciphertext compression for ML-KEM. Alternatively, we obtain 8% shorter public keys compared to the current standard. We also present isochronous implementations of the decoding procedure, achieving negligible performance impact in the full ML-KEM decapsulation even when considering optimized implementations for AVX2, Cortex-M4, and Cortex-A53.

1 Introduction

In 2022, NIST started its post-quantum standardization process, aiming to select a set of schemes that are secure even if large-scale quantum computers are available to an attacker. Unlike traditional solutions like RSA [34] and schemes based on elliptic curves [24]), which can be broken by Shor's algorithm [38], post-quantum schemes are based on problems for which quantum algorithms are believed to have no significant advantage. After six years, NIST chose Kyber [7], a lattice-based key encapsulation mechanism (KEM), for standardization as ML-KEM in FIPS 203 [30]. Nevertheless, NIST still considers other lattice-based schemes, like Saber [15], NTRU [14], and NewHope [2], as promising options.

Modern lattice-based schemes are typically very computationally efficient, and often even faster than traditional cryptographic solutions in use today. Their main limitation, however, is that the sizes of their public keys and ciphertexts are orders of magnitude larger than schemes based on elliptic curves. While it is possible to compress public keys and ciphertexts by dropping a few of their least significant bits, there is a limit to how much one can compress them before seeing a noticeable decryption failure rate (DFR), which is a security concern [16,18,22]. This is a reason why designers of lattice-based schemes usually employ different error-correction strategies aiming to achieve a negligible DFR.

Following Regev's [33] work, ML-KEM and other efficient lattice-based schemes use the same encoding scheme during encryption: each bit b of the message is encoded into $\mathbb{Z}_q$ as $b\lceil q/2 \rfloor$. However, some schemes take additional steps to achieve better error correction. For example, some lattice-based candidates in the first round of NIST's post-quantum standardization process [1] apply distinct error-correction codes to the message before encryption: LAC [28] uses well-known BCH codes; Round5 [8] uses a custom code named XEf [35]; and NewHope [2] uses repetition codes. Interestingly, a previous version of NewHope [3] used more complex, 4-dimensional lattice codes, but those were superseded in favor of the simpler repetition codes, which are easier to understand and analyze.

Recent works explore ways to adapt Kyber for the use of higher-dimensional lattice codes [27, 36, 37]. Unfortunately, these techniques come with significant limitations. Liu and Sakzad's [27] approach assumes that the coefficients of the noise polynomial accumulated during decryption are independent, which does not hold in practice and may result in an overestimation of the scheme's security [16]. Meanwhile, although the work by Saliba et al. [36, 37] does not require independence assumptions, the resulting Kyber variant has a larger ciphertext size. Moreover, all of these approaches [27, 36, 37] require changing at least one of Kyber's core parameters: the polynomial degree n and the modulus q. Consequently, the resulting constructions are unable to take advantage of Kyber's NTT-based efficient polynomial multiplications, a major feature behind the scheme's high performance.

Contributions. We present a new family of higher-dimensional error-correction codes, called Minal codes, that can, in principle, be applied to most modern lattice-based KEMs, such as Kyber [7], Saber [15], or NewHope [2]. The most important property of our Minal codes is that they can be tailored for the specific distribution of the error to be corrected, which depends on the algorithms and parameters used by the target scheme. To evaluate our proposal in a concrete scenario, we chose ML-KEM due to its relevance. When compared to previous work, our proposal has the following benefits.

1. *Concrete improvements to ML-KEM without independence assumptions.* Unlike Liu and Sakzad [27], we do not rely on additional independence assumptions to obtain shorter ciphertexts. Also, unlike Saliba et al. [27, 36, 37], our

codes are able to provide ciphertext compression with negligible DFR. Moreover, our proposal allows 4% to 8% ciphertext compression while maintaining ML-KEM's DFR close to the values targeted by the current standard. Alternatively, our codes can be used for obtaining 8% compression of public keys in all security levels.

2. *Emphasis on crypto-agility.* All previous encoding proposals for Kyber require changes in its core parameters n and q, which define the polynomial ring $R_q = \mathbb{Z}_q[x]/(x^n + 1)$, used in most operations. This undermines crypto-agility, since most optimizations for NTT-based multiplication and modular operations from existing implementations would not be directly applicable, making them unlikely to be integrated into ML-KEM. In contrast, our codes do not require any change to the core parameters n and q.
3. *Negligible performance impact, while avoiding timing side-channels.* Unlike previous work, we evaluate our proposal's performance via an isochronous implementation – i.e., where the number of operations does not depend on any secret information. The performance impact of our codes on ML-KEM's decapsulation is evaluated considering highly optimized implementations using AVX2 instructions or running on Cortex M4 and A53 processors. In most of the platforms considered, the impact on decapsulation is below 1%.
4. *Fully reproducible.* To enable independent verification, the code and data associated with this paper are publicly available at (*omitted for anonymization, but these are provided to the referees as companion code*).

Paper organization. Section 2 reviews background concepts and notation. Section 3 describes ML-KEM and related works on alternative encoding methods. Section 4 presents our new family of codes. Section 5 introduces the techniques we use to obtain efficient isochronous implementation of the decoding operation. Section 6 describes how to compute ML-KEM's decryption failure rate when using our codes. Section 7 shows how our codes can be used to obtain shorter ciphertexts for ML-KEM, and discusses further applications. Section 8 concludes the discussion with open questions and ideas for future work.

2 Background and notation

For any prime q, we write $\mathbb{Z}_q$ to denote the field of integers modulo q. When n is a fixed positive integer, we let R_q denote the polynomial ring $\mathbb{Z}_q[x]/(x^n+1)$. Then, R_q^k is the free module[1] of rank k whose scalars are polynomials in R_q. Polynomials $a \in R_q$ are denoted using lowercase letters. Vectors $\mathbf{a} \in R_q^k$ and matrices $\mathbf{A} \in R_q^{k\times k}$ are denoted in bold using lowercase and uppercase, respectively, where $k \geqslant 1$. When $\mathbf{u}, \mathbf{v} \in R_q^k$, we let $\langle \mathbf{u}, \mathbf{v} \rangle \in R_q$ denote their dot product.

To get the vector-equivalent of a polynomial, we define the `poly_to_vec` function, which, given a polynomial $a \in R_q$, returns its n coefficients as a vector

[1] Modules are generalizations of vector spaces that allow scalars to be members of a ring instead of requiring a field.

in $\mathbb{Z}_q^n$. In other words, given the polynomial $a = a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$, we have $\mathbf{a} = \texttt{poly\_to\_vec}(a) = \left[a_0, a_1, \ldots, a_{n-1}\right]$. With a slight abuse of notation, we denote the i-th coefficient of a polynomial $a \in R_q$, associated with the power x^i, by either a_i (when discussing the polynomial form of a) or by $a[i]$ (when discussing its vector equivalent), where $0 \leqslant i < n$. Analogously, pairs of coefficients from polynomial a are denoted by $a[i, j] = (a[i], a[j])$. We denote by $\mathcal{B}_\eta$ the centered binomial distribution (CBD) with range $[-\eta, \eta]$.

The circulant matrix generated by a vector $\mathbf{a} \in \mathbb{Z}^n$ is the $n \times n$ matrix whose first row is $\mathbf{a}$, and each subsequent row is a right circular shift of the row above it. Let $\texttt{negashift}_i$ be the function that returns the i-th column of the negacyclic matrix generated by the coefficients of a given polynomial. For example, if $a = a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$, then $\texttt{negashift}_i(a) = \left[a_i, \ldots, a_0, -a_{n-1}, \ldots, -a_{i+1}\right]$. With this notation, we can represent the product of polynomials a and b in the negacyclic ring R_q using its vector form, whose i-th coefficient is given by

$$\texttt{poly\_to\_vec}(ab)[i] = \langle \texttt{poly\_to\_vec}(a), \texttt{negashift}_i(b) \rangle \,. \tag{1}$$

The centered modulo operation, denoted as $a' = a \bmod^{\pm} q$, returns the unique integer a' such that $a' \equiv a \bmod q$ and $-\lfloor (q-1)/2 \rfloor \leqslant a' \leqslant \lceil (q-1)/2 \rceil$. The distance modulo q between two points $\mathbf{v}_1$ and $\mathbf{v}_2$, is defined as $\text{dist}_q(\mathbf{v}_1, \mathbf{v}_2) = \left\| (\mathbf{v}_1 - \mathbf{v}_2) \bmod^{\pm} q \right\|$. We write $y \leftarrow \text{Compress}(x, d)$ to denote the lossy compression of x to d bits, where $d < \lceil \log_2 q \rceil$. The compression function is defined as $\text{Compress}(x, d) = \left\lfloor (2^d/q) x \right\rceil \bmod 2^d$, where $\lfloor \cdot \rceil$ is the rounding function that rounds up on ties. The decompression is defined as $x' = \text{Decompress}(y, d) = \left\lfloor (q/2^d) y \right\rceil$. The error $|x' - x|$ caused by (de)compression is then almost uniform over the set $\left\{ -\lfloor q/2^{d+1} \rfloor, \ldots, \lceil q/2^{d+1} \rceil \right\}$, with possibly some slight skewness when q is not a power of 2.

3 ML-KEM

This section briefly reviews ML-KEM's main procedures and selection of security parameters. We also discuss previous work on alternative encoding mechanisms proposed that are related to our construction.

3.1 Parameters and algorithms

ML-KEM is a lattice-based key encapsulation mechanism (KEM) whose security relies on the intractability of the module learning with errors (MLWE) problem. Essentially, it enables two parties to establish a shared 256-bit secret. In what follows, we present a slightly simplified version of ML-KEM that, although lacking some details, is sufficient for our discussion. In particular, we describe only the underlying algorithms that make the core of ML-KEM secure against chosen-plaintext attacks (CPA), ignoring the implicit-rejection Fujisaki-Okamoto (FO) transformation that protects the scheme against adaptive chosen-ciphertext attacks (CCA) [21,23]. Furthermore, we omit the optimizations based on the number theoretic transform (NTT) that are part of the ML-KEM specification.

Table 1. ML-KEM parameters for each security level [30].

NIST security	Parameter set	k	η_1	η_2	$d_{\mathbf{u}}$	d_v	Ciphertext size (bytes)	DFR
Level 1	ML-KEM-512	2	3	2	10	4	768	$2^{-139.1}$
Level 3	ML-KEM-768	3	2	2	10	4	1088	$2^{-165.2}$
Level 5	ML-KEM-1024	4	2	2	11	5	1568	$2^{-175.2}$

Setup. ML-KEM supports three (out of the five) security levels defined by NIST, namely levels 1, 3, and 5 [30]. For all security levels, it fixes parameters $q = 3329$ and $n = 256$, defining the polynomial ring $R_q = \mathbb{Z}_q[x]/(x^n + 1)$ over which most operations are performed. This benefits crypto-agility, as any optimization or hardware acceleration for operations in R_q can be reused for all security levels.

Given a desired security level, the setup takes public parameters $k, \eta_1, \eta_2, d_{\mathbf{u}}$, and d_v from Table 1: k defines the sizes of the modules used in the scheme; η_1 and η_2 define the centered binomial distributions $\mathcal{B}_{\eta_1}$ and $\mathcal{B}_{\eta_2}$, used to generate coefficients with small norm in $\mathbb{Z}_q$; and integers $d_{\mathbf{u}}$ and d_v are the number of bits into which coefficients from the two parts of the ciphertext are compressed. Table 1 also shows the upper bounds on the decryption failure rate (DFR) for each parameter set, as computed using the Kyber security scripts [17].

Key generation. Let $\mathbf{A}$ be a $k \times k$ matrix of polynomials sampled uniformly at random from R_q. Sample two vectors $\mathbf{s}$ and $\mathbf{e}$ from $\mathcal{B}_{\eta_1}\left(R_q^k\right)$, i.e., the coefficients of their polynomials are sampled according to the centered binomial distribution $\mathcal{B}_{\eta_1}$. Compute vector $\mathbf{t} = \mathbf{A}\mathbf{s} + \mathbf{e} \in R_q^k$. The resulting public key is the pair $(\mathbf{A}, \mathbf{t})$, while the private key is the vector $\mathbf{s} \in R_q^k$.

Encryption. Let $\mathbf{m}$ be an n-bit message to be encrypted using public-key $(\mathbf{A}, \mathbf{t})$. Sample vectors $\mathbf{r}$ and $\mathbf{e}_1$ from $\mathcal{B}_{\eta_1}\left(R_q^k\right)$ and $\mathcal{B}_{\eta_2}\left(R_q^k\right)$, respectively. Similarly, sample a polynomial e_2 from $\mathcal{B}_{\eta_2}(R_q)$. Let $\mathbf{u} = \mathbf{A}^{\mathrm{T}}\mathbf{r} + \mathbf{e}_1$. Compute polynomial $z = \langle \mathbf{t}, \mathbf{r} \rangle + e_2$. Let $v = \mathrm{Encode}\,(\mathbf{m}) + z$, where the encoding function, when applied to each bit b of $\mathbf{m}$, returns $\mathrm{Encode}\,(b) = b\lceil q/2 \rceil$. Compress the coefficients of vector $\mathbf{u}$ and polynomial v to $d_{\mathbf{u}}$ and d_v bits, respectively, obtaining $\mathbf{c_u} = \mathrm{Compress}(\mathbf{u}, d_{\mathbf{u}})$ and $c_v = \mathrm{Compress}(v, d_v)$. Finally, return the ciphertext $(\mathbf{c_u}, c_v)$.

Decryption. To decrypt ciphertext $(\mathbf{c_u}, c_v)$ using secret key $\mathbf{s}$, first decompress the ciphertext components to obtain $\mathbf{u}' = \mathrm{Decompress}(\mathbf{c_u}, d_{\mathbf{u}})$ and $v' = \mathrm{Decompress}(c_v, d_v)$. Compute $m' = v' - \langle \mathbf{u}', \mathbf{s} \rangle$. Let $\Delta\mathbf{u} = \mathrm{Decompress}(\mathbf{c_u}, d_{\mathbf{u}}) - \mathbf{u}$ and $\Delta v = \mathrm{Decompress}(c_v, d_v) - v$. We can then write $m' = \mathrm{Encode}\,(\mathbf{m}) + \Delta m$, where the accumulated noise polynomial Δm is given by

$$\Delta m = \langle \mathbf{e}, \mathbf{r} \rangle - \langle \mathbf{s}, \mathbf{e}_1 + \Delta\mathbf{u} \rangle + e_2 + \Delta v.$$

ML-KEM's parameters are carefully chosen to ensure that polynomial Δm has only relatively small coefficients. Therefore, the message can be recovered by computing $\hat{m} = \text{Decode}\,(m')$, where the decoding function, applied to each coefficient $m'[i]$ of the noisy message polynomial m', returns

$$\text{Decode}\,(m'[i]) = \begin{cases} 0, \text{ if } |m'[i] \bmod^{\pm} q| < \lceil q/4 \rceil, \text{ and} \\ 1, \text{ otherwise.} \end{cases}$$

In the next section, we briefly discuss how ML-KEM's parameters are chosen to ensure security and a negligible decryption failure rate.

3.2 Security and decryption failure rate

ML-KEM's design and security analysis revolve around finding parameters that ensure the MLWE problems protecting the secret key and the ciphertext are hard to solve while maintaining a negligible DFR. To facilitate the scheme's security analysis, the Kyber team provided public scripts [17] that compute the complexity of known attacks, which is obtained through the core-SVP hardness measure [3], and the resulting DFR for a given parameter set.

The parameters having the most impact on the MLWE security are the modulus q, the degree n, and the module dimension k, together with the parameters η_1 and η_2 that control the noise added to the LWE samples. Albeit not as much, the ciphertext compression parameters $d_{\mathbf{u}}$ and d_v can also affect security. In particular, under the learning with rounding (LWR) hardness assumption, increasing the deterministic compression noise improves security. This is explored in the choice of ML-KEM-512 parameters to reduce ciphertext sizes, making it the only parameter set whose security is based both on the hardness of LWE and on an LWR-like assumption.

For a given parameter set, the DFR is algorithmically computed as follows. Since each coefficient $\Delta m[i]$ follows the same distribution, we can start by computing the distribution of $\Delta m[0]$. This is done by considering the sums of the distributions corresponding to the right-hand side of the following equation

$$\Delta m[0] = \langle \mathbf{e}, \mathbf{r} \rangle\,[0] - \langle \mathbf{s}, \mathbf{e}_1 + \Delta \mathbf{u} \rangle\,[0] + e_2[0] + \Delta v[0],$$

which are easy to compute. Then, an upper bound on the DFR is computed using the union bound as

$$\begin{aligned} \Pr\left(\text{Decryption fails}\right) &= \Pr\left(\left|\Delta m[i] \bmod^{\pm} q\right| \geqslant q/4 \text{ for any } 0 \leqslant i < n\right) \\ &\leqslant n \Pr\left(\left|\Delta m[0] \bmod^{\pm} q\right| \geqslant q/4\right). \end{aligned}$$

A necessary condition for a KEM to provide CCA security is to resist attacks exploiting decryption failures [18, 22]. Unfortunately, apart from being negligible, there is no clear guidance on how the DFR should be selected. Code-base schemes such as BIKE [4] and HQC [29] target very low DFRs of $2^{-\lambda}$, where λ is the scheme's security in bits, while ML-KEM and Saber are more permissive.

Namely, in Kyber's round 2 specification, the authors state that the DFR target was at most 2^{-160} for all security levels [6, §1.5]. This was later relaxed in round 3, with DFR targets of 2^{-128} for level 1, and kept at 2^{-160} for levels 3 and 5 [7, §1.4 and §4.4]. In its specification, Saber [15] does not state DFR targets, but only the achieved DFRs of $2^{-120}, 2^{-136}$ and 2^{-165} for NIST security levels 1, 3, and 5.

In an effort to deliver DFR levels close to the state of the art in lattice-based schemes, we hereby define our DFR targets for each security level as the maximum between the DFR targets set by Kyber and the concrete DFR provided by Saber parameter sets. This gives us $\mathrm{DFR}_1 = 2^{-120}$, $\mathrm{DFR}_3 = 2^{-136}$ and $\mathrm{DFR}_5 = 2^{-160}$, for levels 1, 3, and 5, respectively. These targets are used in Section 6 to select viable parameter sets for each security level using the alternative encoding mechanisms proposed in this paper.

3.3 Previous work on alternative encoding methods for Kyber

Like our work, some recent studies present strategies to use higher-dimensional codes in Kyber variants. One example is [27], which relies on lattice codes with dimensions 16 and 24. The authors claim to improve both the DFR and the ciphertext size, but they require n to be changed, preventing the NTT-based multiplication. Moreover, their work, like most proposals for error correction in lattice-based schemes, requires independence assumptions on the coefficients of the noise polynomial Δm, which do not hold in practice and are known to cause underestimation of the DFR when error-correction codes are used [16]. In particular, these assumptions would break Kyber's DFR arguments from Section 3.2, so it would be hard (if at all feasible) to adapt [27] to Kyber's design.

More closely related to our work is the approach taken by Saliba et al. [37], which is explained in depth in Saliba's PhD thesis [36]. They propose a variant of Kyber based on reconciliation, which, in lattice-based schemes, refers to a procedure in which the sender and receiver produce the same shared string from different noisy versions of it. This contrasts with the encoding-decoding paradigm, where the intended shared message is predefined. Their construction uses 8-dimensional lattice codes and does not require independence assumptions, so it can be seen as an extension of the original NewHope's DFR analysis [3,31] to Kyber. While [37] delivers between 10 to 15 extra bits of LWE security for Kyber's 3 security levels, the proposal has a few practical shortcomings, listed in Table 2. One of the main issues is that the values of the modulus q are powers of two, which means they cannot use the NTT for polynomial multiplication. Furthermore, their scheme increases the ciphertext size in all security levels, while the DFR is increased in levels 1 and 5. For example, compared with Kyber, there is a noticeable increase in the DFR for level 5, by a factor of 2^{37}.

In summary, since the approaches found in the literature [27,36,37] on alternative Kyber encoding mechanisms require either changing n or q, they do not benefit from a core feature in Kyber: the fast NTT-based multiplication. For instance, Saliba et al.'s proposal [37] requires different values of q for each security level, hindering Kyber's crypto-agility properties. We note that their proposal's

Table 2. The DFR and ciphertext sizes obtained by Saliba et al. [36,37].

Security	q	Ciphertext size (bytes)	DFR	Relative ciphertext size	Relative DFR	Advantages (ciphertext size and DFR)
Level 1	2^{11}	832	2^{-133}	108.3%	2^{6}	**None**
Level 3	2^{11}	1184	2^{-174}	108.8%	2^{-10}	Better DFR
Level 5	2^{11}	1600	2^{-137}	102.0%	2^{37}	**None**

performance impact is not reported, though, and we were unable to find any publicly available implementation for conducting an independent evaluation.

4 Minal codes: tailorable codes for lattice-based schemes

In this section, we introduce the family of Minal codes.[2] We start by providing motivation for higher-dimensional codes and listing desirable properties for codes suitable to lattice-based schemes. We then present the formal definition of Minal codes and their core properties.

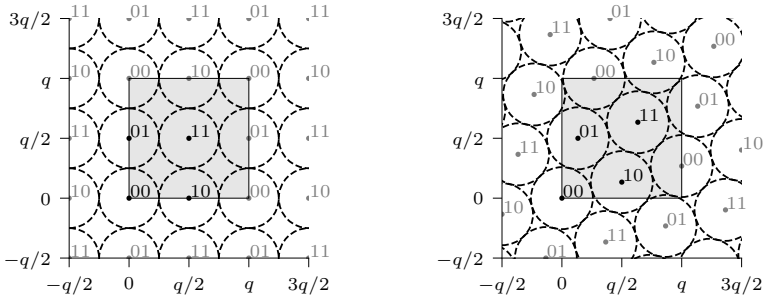
4.1 Motivation

Consider ML-KEM's mechanism for encoding a message into a polynomial. We can treat it as a two-dimensional code by pretending it encodes a pair (b_0, b_1) of message bits into coefficients $(b_0\lceil q/2\rceil, b_1\lceil q/2\rceil) \in \mathbb{Z}_q^2$. This code is illustrated in Figure 1a, where dots denote the codewords, and the circles around them show the radius of minimum distance decoding (i.e., any point falling into the area of a given circle is corrected to the valid codeword at its center).

This 2D view of ML-KEM's code in Figure 1a highlights one possible problem: it leaves too much uncovered space under its minimum distance. To support denser codes in 2 or more dimensions, one solution is to use lattice codes. For example, Figure 1b shows a lattice code with minimum distance ≈ 1722. Lattice codes are well-known to be useful for correcting Gaussian noise, or errors that have a short Euclidean norm. However, it can be difficult to use them directly in lattice-based schemes because, in general, they are not periodic in $\mathbb{Z}_q^n$. To see why this is a problem, consider what happens when one adds (q, q) to the $(0, 0)$ point to the lattice shown in Figure 1b. Ideally, since ML-KEM's operations are done in $\mathbb{Z}_q$, we would like (q, q) to result in a point encoding $(0, 0)$, however, the resulting point is not a codeword, and the closest codewords to it are encodings of $(0, 1)$ and $(1, 0)$. Notice that, since the 2D view of ML-KEM's code illustrated in Figure 1a is periodic in $\mathbb{Z}_q^2$, this problem is avoided.

Previous proposals [36, 37] handle this issue by changing the parameter q to powers of 2, so it is easy to employ an 8D lattice that is periodic in $\mathbb{Z}_q^8$. While this allows such proposals to exploit the lattice structure when proving the

[2] The name Minal is an acronym for *Minal is not a lattice.*



(a) ML-KEM's code with minimum distance of $\lfloor q/2 \rfloor = 1664$.

(b) A lattice code with minimum distance of about 1722.

Figure 1. Comparison between the original ML-KEM code seen as a 2D code and a denser lattice code. The shaded areas represent the $\mathbb{Z}_q^2$ square.

DFR of the resulting scheme, these values of q significantly impact ML-KEM's performance, because fast NTT multiplication would no longer be available. Furthermore, the resulting scheme has larger ciphertexts than ML-KEM.

In what follows, we introduce a new family of higher-dimensional codes, called Minal codes, that can be seen as an intermediate between lattice codes and ML-KEM's code. On the one hand, we enforce, by construction, that the resulting n-dimensional code is periodic in $\mathbb{Z}_q^n$. On the other hand, our code's structure is not as rigid as ML-KEM's code: it uses a tailoring parameter that allows the position of the codewords to change, allowing the code to be more efficient for the particular error distribution observed on the target lattice-based scheme.

4.2 Definitions and properties

We begin with a formal definition of Minal codes.

Definition 1 (Minal code). Given an integer $n \geqslant 2$, a prime number q, and a non-negative integer $\beta < q/2$, the n-dimensional Minal code with parameter β over alphabet $\mathbb{F}_q$ is the infinite set of points defined as

$$\mathcal{M} = \{\mathbf{Gm} + q\mathbf{z} : \mathbf{z} \in \mathbb{Z}^n \text{ and } \mathbf{m} \in \mathbb{Z}_2^n\},$$

where $\mathbf{G} \in \mathbb{Z}^{n \times n}$, the generator matrix of $\mathcal{M}$, is the circulant matrix generated by $[\lfloor q/2 \rfloor, \beta, 0, \ldots, 0]$. Parameter β is called the tailoring parameter. We say that $\mathbf{c}$ encodes an n-bit message $\mathbf{m} \in \mathbb{Z}_2^n$ when $\mathbf{c} = \mathbf{Gm} + q\mathbf{z}$, for some $\mathbf{z} \in \mathbb{Z}^n$. □

A natural consequence of Definition 1 is that, by setting $(q, \beta) = (3329, 0)$, we get n-dimensional Minal codes that are equivalent to the code used by ML-KEM. Furthermore, notice that the simple definition of Minal codes can mislead one to think that they have a linear structure. However, unlike linear codes or lattice codes, Minal codes do not even form groups, as these sets are not closed

under addition. For concreteness, take two codewords $\mathbf{c}_1 = \mathbf{G}\mathbf{m}_1 + q\mathbf{z}_1$ and $\mathbf{c}_2 = \mathbf{G}\mathbf{m}_2 + q\mathbf{z}_2$. Their sum $\mathbf{c}_1 + \mathbf{c}_2 = \mathbf{G}(\mathbf{m}_1 + \mathbf{m}_2) + q(\mathbf{z}_1 + \mathbf{z}_2)$ is not always a codeword because $\mathbf{m}_1 + \mathbf{m}_2$ is not guaranteed to be in $\mathbb{Z}_2^n$.

The structure of Minal codes in $\mathbb{Z}_q^n$ is repeated over all $\mathbb{Z}^n$. Therefore, we consider that the main representative of each codeword lies in $\mathbb{Z}_q^n$. In addition, to measure the distance between elements of $\mathbb{Z}^n$, we use the distance modulo q metric, defined as $\mathrm{dist}_q(\mathbf{v}_1, \mathbf{v}_2) = \left\|(\mathbf{v}_1 - \mathbf{v}_2) \bmod^{\pm} q\right\|$. We can then define the minimum distance decoding under the dist_q metric as follows.

Definition 2 (Minimum distance decoding). Let $\mathcal{M}$ be an n-dimensional Minal code over $\mathbb{F}_q$ with generator $\mathbf{G}$, and suppose we are given a vector $\mathbf{t} \in \mathbb{Z}^n$. A minimum distance decoder is an algorithm that finds the point $\mathbf{m} = \mathrm{Decode}(\mathbf{t}) \in \mathbb{Z}_2^n$ that minimizes $\mathrm{dist}_q(\mathbf{t}, \mathbf{G}\mathbf{m})$, that is, the distance between $\mathbf{t}$ and the codeword corresponding to $\mathbf{m}$. □

Interestingly, Definition 2 implicitly defines a simple algorithm to decode a vector $\mathbf{t} \in \mathbb{Z}^n$: iterate over all possible $\mathbf{m} \in \mathbb{Z}_2^n$ to find the closest codeword to $\mathbf{t}$. While this algorithm's complexity is clearly exponential on the dimension n, it is efficient in small dimensions. In fact, this is the decoder we use in Section 5.1 for decoding 4D Minal codes that can be applied to ML-KEM. Moreover, in Section 5.2, we also show a more efficient decoder that is specific for $n = 2$.

Naturally, minimum distance decoding is more effective when codewords are farther apart. This motivates us to compare different codes based on the widely used minimum distance property, which is defined next for Minal codes.

Definition 3 (Minimum distance). The minimum distance of an n-dimensional Minal code $\mathcal{M}$ over $\mathbb{F}_q$, denoted as $\mathrm{dist}(\mathcal{M})$, is the smallest distance between different points in $\mathcal{M}$, that is

$$\begin{aligned}\mathrm{dist}(\mathcal{M}) &= \min\{\mathrm{dist}_q(\mathbf{c}_1, \mathbf{c}_2) : \mathbf{c}_1, \mathbf{c}_2 \in \mathcal{M} \text{ and } \mathbf{c}_1 \neq \mathbf{c}_2\} \\ &= \min\{\mathrm{dist}_q(\mathbf{G}\mathbf{m}_1, \mathbf{G}\mathbf{m}_2) : \mathbf{m}_1, \mathbf{m}_2 \in \mathbb{Z}_2^n \text{ and } \mathbf{m}_1 \neq \mathbf{m}_2\}.\end{aligned}$$ □

The main feature of Minal codes is that, by tuning the tailoring parameter β, we get better codes for different error distributions. The next section explores this in more detail, showing tailoring procedures for finding the best value of β in a way that minimizes the code's DFR for a given error distribution.

4.3 Tailoring Minal codes for lattice-based schemes

In most lattice-based schemes, the accumulated noise that needs to be corrected during decryption is a sum of two components. The first is an approximately normal distribution stemming from the sums of products of polynomials with a small norm. The second, present in schemes allowing ciphertext compression, is an approximately uniform component resulting from the decompression error. For concreteness, notice that the error polynomial Δm in ML-KEM indeed consists of an approximately normal factor, coming from $(\langle \mathbf{e}, \mathbf{r} \rangle - \langle \mathbf{s}, \mathbf{e}_1 + \Delta\mathbf{u} \rangle + e_2)$, and a somewhat uniform term resulting from Δv.

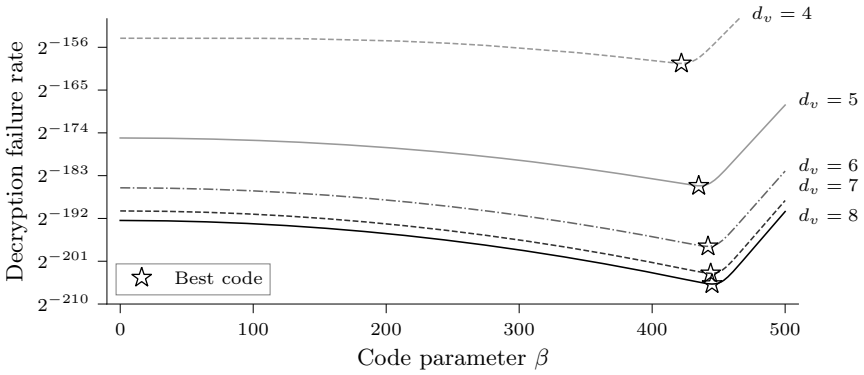


Figure 2. The effect of d_v in the DFR and best code, for ML-KEM in Level 5.

While it may be appealing to simply pick the code with the largest minimum distance, the best choice actually depends on the nature of the error. For example, when the noise is normally distributed, the best code should indeed be the one with the largest minimum distance. However, if the noise is uniform in a region, then ML-KEM's original code would be a better choice.

In what follows, we evaluate how to find the optimal tailoring parameter β under two settings. First, we consider the case in which we are able to fully determine the distribution of the noise to be corrected. In this case, a simple exhaustive search for the parameter that minimizes the DFR gives us very good results, but this can only be done for small dimensions. In particular, for ML-KEM, this approach only works for 2D. We then show how to find β in larger dimensions without the need for the full noise distribution.

Tailoring in 2D using exhaustive search. In this case, we assume that the multidimensional noise distribution is efficiently computable and known. Using the noise distribution, we find the parameter β that minimizes the DFR using a simple exhaustive search. While this simple approach can be very effective, it does not scale well for dimensions higher than 2 because, in these cases, it is very expensive to compute the full noise distribution.

For a concrete example, we consider the noise distribution for ML-KEM, which can be computed using the approach described in Section 6.2. The following experiment shows how the best value of β varies depending on the noise distribution. For ML-KEM-1024, we changed parameter d_v from 1 to 12, and computed the DFR of 2D Minal codes using $\beta = 0$ to 500. Figure 2 depicts our results.

We can make two important observations. First, we clearly see that the best values of β get larger for increasing d_v, although with diminishing returns. Second, we see that the DFR improvement compared to ML-KEM's code ($\beta = 0$) is more noticeable for higher values of d_v. Both of these stem from the fact that, by increasing d_v, we progressively lower the uniform component of the noise, increasing the effectiveness of codes with higher minimum distances. Interestingly,

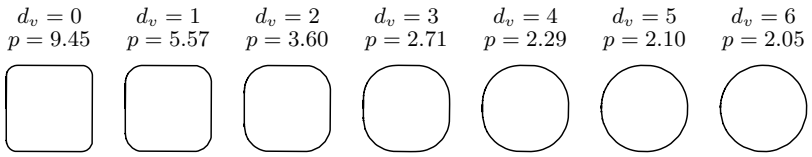


Figure 3. Level curves of the 2D noise distribution corresponding to probability 2^{-128} for different values of d_v. The values of p show the p-norms where the shape is closely approximated by a circle.

even for a code with only 2 dimensions, we can see that tailoring has a major impact on the DFR, taking the DFR of ML-KEM's original code ($\beta = 0$) from $2^{-192.4}$ down to $2^{-205.8}$ for the best tailored code ($\beta = 445$) when $d_v = 8$.

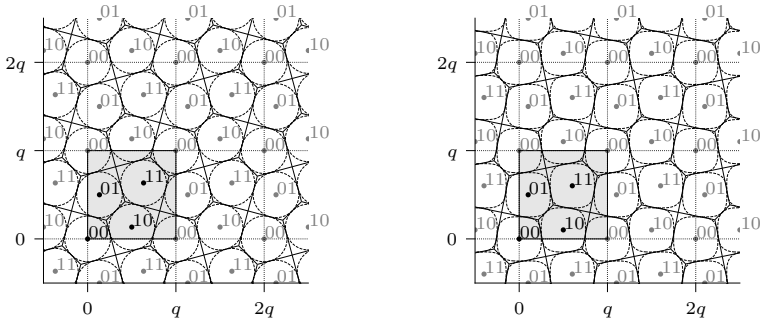
Tailoring in higher-dimensions using p-norms. We now discuss how to find a good parameter β for n-dimensional codes without having to compute the joint distribution in n dimensions. Our main observation is that the shape of the discrete level curves in the joint distribution of the noise can be approximated by circles in the p-norm. Concretely, if the noise distribution is approximately normal, then p is close to 2, corresponding to more circular level curves. Alternatively, if the uniform component of the noise is very strong, then p will be higher, leading to level curves shaped as squared circles.

Figure 3 shows how d_v, which is the main parameter that controls the intensity of the uniform component of the noise, impacts the overall shape of the level curves corresponding to 2^{-128}, when the other parameters are those adopted by ML-KEM-1024 (Level 5). It is interesting to consider both Figures 2 and 3 together, which provides a more clear picture of the relation between the optimal values of β and the overall shape of the noise distribution.

Figure 4 shows how the p-norm that best approximates the shape of the noise distribution impacts the shape of the best code. Intuitively, for a given error distribution whose shape is an approximate circle in the p-norm, the best n-dimensional Minal code should provide a good packing of n-dimensional spheres defined in the corresponding norm.

We then propose the following steps to find the optimal value of β.

1. Compute the 1D noise distribution $\mathcal{D}$. This is efficient for most modern schemes – e.g., Kyber [7] and Saber [15] provide scripts for this task.
2. Build the set of 2D points $P = \{(x_1, x_2) : \mathcal{D}[x_1] \cdot \mathcal{D}[x_2] \approx 2^{-\kappa}\}$. The set $P \in \mathbb{Z}^2$ approximates the level curve corresponding to probability $2^{-\kappa}$ in 2D, where $2^{-\kappa}$ is close to the DFR values we want to achieve.
3. Find the value of p that best approximate the points in P as a circle in the p-norm. For this step, we take p as the value that minimizes the variance of the p-norm for the points in P.
4. Find the parameter β of the n-dimensional Minal code that maximizes the minimum distance with respect to the p-norm.



(a) Minal code with $\beta = 446$, which is the best for $p = 2$.

(b) Minal code with $\beta = 339$, which is the best for $p = 3$.

Figure 4. Best codes depending on the p-norm closest to the noise distribution.

In this paper, when implementing step 2, we use $\kappa = 128$ for all parameters as it makes the procedure easier to implement and appears to provide reasonably good results. Notice that while there is an implicit independence assumption to approximate the 2D distribution using the 1D distribution $\mathcal{D}$, it does not have any security impact because we are only using it to get a reasonable value for β. The DFR of the resulting code will be derived later without requiring any independence assumptions.

To efficiently implement step 4, it is possible to use a ternary search if we assume that the minimum distance is an unimodal function of β for any p-norm. This appears to be the case in our empirical tests, and Figure 2 can provide some intuition on why unimodality may hold. However, it does not seem to be trivial to formally prove unimodality for any dimension and p-norm. Since our Python implementation of this procedure provides good results for β in dimensions $n \leqslant 10$, we leave a more formal treatment of our heuristic assumptions for future work.

5 Implementation and performance

This section describes the practical aspects of our Minal codes, allowing us to achieve efficient implementations by following isochronous programming practices to protect against timing attacks. We remark that the encoding using Minal codes is rather trivial: every row of the generator matrix has only two non-null entries, so the encoding complexity per bit is independent of the dimension. Hence, this section only describes how decoding can be efficiently implemented, whereas the companion implementation contains both operations.

5.1 General decoding algorithm

For concreteness, in this section, we describe the general decoding algorithm induced from Definition 2 by using 4D Minal codes as basis – see Algorithm 1.

We emphasize, however, that the same approach can easily be extended to other dimensions while still following isochronous implementation practices. Indeed, in the accompanying code, we provide a generic implementation of the decoding algorithm that works for 2D up to 16D.

To decode a 4D point $\mathbf{t} \in \mathbb{Z}_q^4$, we need to find $\mathbf{m} \in \mathbb{Z}_2^4$ that minimizes $\text{dist}_q(\mathbf{t}, \mathbf{Gm}) = \left\|(\mathbf{t} - \mathbf{Gm}) \bmod^{\pm} q\right\|$, where $\mathbf{G}$ is the generator matrix of the Minal code. For the reference ML-KEM implementation, we can assume that $\mathbf{t} \in [-\lfloor q/2 \rfloor, \lceil q/2 \rceil]^4$ is already reduced modulo q and centered at zero.[3] Our implementation is based on two observations. First, we notice that since the generator matrix is sparse and circulant, there are only four possible values for the coordinates of each codeword, represented by the array `CW_COORD_VALUES`. Therefore, the distance between the target and each codeword can be computed more efficiently by using memorization of the partial squared distances between their coordinates, which are stored in the 4×4 matrix `dist_sqr_matrix`. The second observation is that we do not need a generic reduction algorithm, such as Barrett's reduction, because the difference between coordinates in $[-\lfloor q/2 \rfloor, \lceil q/2 \rceil]$ is in $[-q, q]$ and we only need the squares of the distances. Therefore, we use a custom function `centered_mod_sqr` that, on input x, returns $\left(x \bmod^{\pm} q\right)^2$.

We can then implement function `get_distance_sqr_to_codeword` that uses the memorization matrix to compute, for a given message index `idx` $\in \{0, 15\}$, the square of the distance modulo q between the target and the codeword associated with the binary expansion of `idx`. Using a `secure_min` function that isochronously computes the minimum between two values, the general decoding is done by iterating over each of the 16 possible messages (represented by `idx`).

Interestingly, even for more than 4 dimensions, the performance of this decoder is comparable to state-of-the-art error correction codes used in PQC. For example, using HQC's [29] optimized AVX2 implementation, we observed that their decoder uses around 660 cycles per decoded bit. This value is close to the 600 cycles per bit we observed when decoding 9D Minal codes.

5.2 Decoding 2D codewords using approximate Voronoi cells

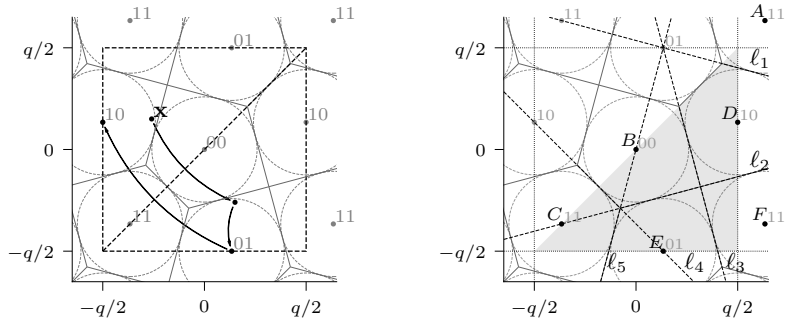
For the 2D decoding, we propose a custom algorithm that is more efficient than the generic approach from Section 5.1. We begin by observing a symmetry, illustrated in Figure 5a, that can be exploited for decoding. We can see that, by construction, the codewords of the Minal code $\mathcal{M}$ with parameter β are symmetric over the identity line, which separates the $[-q/2, q/2]^2$ square in two triangles. Because of this property, if a point is closer to a codeword associated with $(1, 0)$ in the upper triangle, it will be closer to a codeword associated with $(0, 1)$ in the lower triangle and vice-versa – e.g., see point $\mathbf{x}$ in Figure 5a. However, we can also see that closeness to codewords associated with $(0, 0)$ and $(1, 1)$ is preserved by reflection around the identity line. Building upon this symmetry, any point in the upper triangle can be reflected, decoded in the lower triangle, and then

[3] In some implementations (e.g., AVX2 and pqm4), it is more convenient to use $\mathbf{t} \in [0, q)^4$, but this can be accommodated by the algorithm with minor changes.

```
int16_t CW_COORD_VALUES[4] = {0, CODE_BETA, KYBER_Q/2, CODE_BETA + KYBER_Q/2 - KYBER_Q};

int32_t centered_mod_sqr(int32_t value) {
  uint32_t mask_sign = value >> 31;
  value ^= mask_sign;
  value += mask_sign & 1; // Result: value = abs(value)
  value -= KYBER_Q & lower_than_mask(KYBER_Q/2, value); // Result:+-(value center_mod q)
  return value * value; // Result: (+- (value center_mod q))^2 = (value center_mod q))^2
}
uint32_t get_distance_sqr_to_codeword(uint16_t idx, uint32_t dist_sqr_matrix[4][4]) {
  int32_t a0 = dist_sqr_matrix[0][CODEWORDS[idx][0]];
  int32_t a1 = dist_sqr_matrix[1][CODEWORDS[idx][1]];
  int32_t a2 = dist_sqr_matrix[2][CODEWORDS[idx][2]];
  int32_t a3 = dist_sqr_matrix[3][CODEWORDS[idx][3]];
  return (a0 + a1 + a2 + a3) << 4 | idx;  // Returns 'distance_sqr | codeword_index'
}
int decode_minal_4d(int16_t target[4]) {
  // Build memorization matrix with square distances to target coordinates
  uint32_t dist_sqr_matrix[4][4] = {0};
  for (int i = 0; i < 4; i++) {
    for (int j = 0; j < 4; j++)
      dist_sqr_matrix[i][j] = centered_mod_sqr(target[i] - CW_COORD_VALUES[j]);
  }
  uint32_t min_dist_codeword = get_distance_sqr_to_codeword(0, dist_sqr_matrix);
  for (size_t i = 1; i < 16; i++)
    min_dist_codeword = secure_min(get_distance_sqr_to_codeword(i, dist_sqr_matrix),
                                   min_dist_codeword);
  return min_dist_codeword & 0xF;  // Extracts the codeword index part
}
```

Algorithm 1. Isochronous implementation of decoding in 4D Minal codes.



(a) Symmetry used for decoding a point $\mathbf{x}$ in the upper triangle.

(b) Lines ℓ_i for decoding in the lower triangle.

Figure 5. The geometric properties of our codes used during decoding.

reflected once again. Consequently, we only need to devise an efficient decoding mechanism for the lower triangle.

Suppose we are given a point in the lower triangle and want to find the closest codeword to this point. One simple way to accomplish that task would be to compute the distance to all six codewords whose Voronoi cells overlap the lower triangle, and then output the closest one. Although we considered this simple strategy, the resulting isochronous implementation was not very efficient due to the number of comparisons to the closest codeword.

For the sake of building our argument for our optimized strategy, assume for a moment that q is divisible by 2. In this setting, we can construct the Voronoi

cells of each codeword relevant for decoding points in the lower triangle, as illustrated in Figure 5b. By the definition of $\mathcal{M}$, the points whose Voronoi cells intersect the lower triangle, which are shown in Figure 5b, are defined as:

$$A = (q/2 + \beta, q/2 + \beta), \quad B = (0, 0), \quad C = (\beta - q/2, \beta - q/2),$$
$$D = (q/2, \beta), \quad E = (\beta, -q/2), \quad F = (q/2 + \beta, -q/2 + \beta).$$

The Voronoi cells intersecting the lower triangle can be defined by the perpendicular bisector lines, which we call ℓ_i, between the codewords and their neighbors. First we define ℓ_1, ℓ_4 and ℓ_5 as the bisectors between pairs $(A, D), (B, C)$, and (C, E), respectively. Now, since we assume q is even, the set of codewords $\{B, D, E, F\}$ forms a square, so line ℓ_2 is the bisector of the pairs of points (B, E) and (D, F). Similarly, line ℓ_3 is simultaneously the bisector of both pairs (B, D) and (E, F). This means that, for an even q, we can characterize the Voronoi cells of these codewords using only 5 lines. To effectively use these lines to decode a point (x, y) in the lower triangle, we can verify whether (x, y) is above or below ℓ_i for each i. For example, if (x, y) is above lines ℓ_4 and ℓ_2, but below ℓ_3, then it should be decoded as $(0, 0)$.

Now that we have explained how to handle an even q, we have to deal with the real-world q, which is an odd prime. Since q is not divisible by 2, we must use $\alpha = \lfloor q/2 \rfloor$. This impacts the definition of some of the points. In particular, we now have $D = (\lfloor q/2 \rfloor, \beta)$, $E = (\beta, -\lfloor q/2 \rfloor - 1)$, and $F = (\lfloor q/2 \rfloor + \beta, -\lfloor q/2 \rfloor - 1 + \beta)$. Therefore, the set of points $\{B, D, E, F\}$ does not form a rectangle anymore. However, since $q = 3329$ is relatively large, we observe that $\{B, D, E, F\}$ can be reasonably well approximated by a square. More specifically, if we define ℓ_3 as the bisector between points (B, D), and ℓ_2 as the bisector of points (B, E), then we can say that lines $\ell_1, \ldots, \ell_5$ give an approximate characterization of the Voronoi cells when q is prime.

We use this approach based on approximate Voronoi cells for decoding when computing all DFR results involving 2D codes. Since we compute the 2D DFR based on the probability of Algorithm 2 returning the wrong codeword (see Algorithm 3 from Section 6.2), there is no security problem in using this approximate decoder as long as its DFR is negligible.

Implementation of the 2D decoding procedure. Algorithm 2 shows the full algorithm for decoding using these ideas. It builds upon macros `ABOVE_Li`, that return `0xffffffff` if point (x, y) is above ℓ_i and `0x0` otherwise. Notice that the equations that define lines ℓ_i have only integer coefficients because the representatives of all codewords themselves have integer coefficients. Furthermore, because all points are integers, the implementation of `ABOVE_Li` based on the lines' equations uses only 32-bit integer multiplications, which most implementations, including the ones of ML-KEM, assume to be isochronous. Furthermore, we note that a reflection mask is used to reflect (x, y) whenever needed, and then to reflect the result in case codewords corresponding to $(0, 1)$ or $(1, 0)$ are found.

One remark regarding this algorithm is that decoding could be slightly more efficient if a different square of representatives was employed. In particular, us-

```
int decode_minal_2d(int32_t x, int32_t y) {
  uint32_t reflect_mask = mask_lower_than(x, y); // -1 if (x < y) and 0x0 otherwise
  int32_t x_prime = (x & ~reflect_mask) | (y & reflect_mask);
  int32_t y_prime = (y & ~reflect_mask) | (x & reflect_mask);
  uint8_t abovel1 = ABOVE_L1(x_prime, y_prime);
  uint8_t abovel2 = ABOVE_L2(x_prime, y_prime);
  uint8_t abovel3 = ABOVE_L3(x_prime, y_prime);
  uint8_t abovel4 = ABOVE_L4(x_prime, y_prime);
  uint8_t abovel5 = ABOVE_L5(x_prime, y_prime);
  // It is unnecessary to check for (00), but: c00 = (~abovel3 & abovel2 & abovel4);
  uint8_t c01 = ~abovel2 & ~abovel5 & ~abovel3;
  uint8_t c10 = abovel2 & abovel3 & ~abovel1;
  uint8_t c11 = abovel1 | (abovel3 & ~abovel2) | (abovel5 & ~abovel4);
  c01 &= (1 ^ reflect_mask);
  c10 &= (2 ^ reflect_mask);
  return (c01 | c10 | c11) & 3;
}
```

Algorithm 2. Isochronous implementation of decoding in 2D Minal codes.

ing the $q \times q$ square whose bottom left point is $(\lfloor -q/2 \rfloor - \epsilon, \lfloor -q/2 \rfloor - \epsilon)$, the comparison with line ℓ_1 is not necessary. However, since this would lead to a more complex description, we leave possible extra optimizations for future work.

5.3 Performance evaluation

To evaluate the performance of the encoding and decoding operations with Minal codes, we considered three platforms for which highly optimized ML-KEM implementations are available: AVX2 [7], ARM Cortex-M4 [26], and ARM Cortex-A53 [12]. We integrated our isochronous implementations of the 2D and 4D Minal encoding and decoding into the existing Kyber implementations in which the encoding and decoding of the full 256-bit message are done by the `poly_frommsg` and `poly_tomsg` functions, respectively. We remark that we did not write any manual optimizations of the Minal code operations for the given targets.

To obtain the AVX2 cycle counts, we considered the latest AVX2 implementation provided by the Kyber team [7] running on an Intel Core i7-8700 (Coffee Lake) CPU with a base frequency of 3.20GHz. The code was compiled using `gcc` version 14.2.1 with flags `-O3`, `-march=native`, `-mtune=native`, and `-fomit-frame-pointer`. We report the median cycle count of 10,000 executions.

For Cortex-A53, we used the aarch64-optimized Kyber implementation from *PQClean* [25, commit `0c5bb14`], an run it on Raspberry Pi Zero 2W board [32]. The compilation was done using `aarch64-none-linux-gnu-gcc` v.13.3.1 with the following flags: `-O3`, `-mcpu=cortex-a53`, and `-mtune=cortex-a53`. For cycle count benchmarking, we employed the Performance Monitors Cycle Counter (`PMCCNTR_EL0`) to measure the average of 10,000 runs.

For Cortex-M4, we used the *m4fspeed* version of the pqm4 library [26, commit `cda61fb`], and measured the performance on an STMicroelectronics Nucleo-F439ZI board [39]. The compilation employed `arm-none-eabi-gcc` v.13.2.1 with the following flags: `-O3`, `-mcpu=cortex-m4`, `-mfpu=fpv4-sp-d16`, `-mfloat-abi=hard`, and `-mthumb`. For cycle counts, we used the Data Watchpoint and Trace (DWT) registers to get the average of 100 runs.

Table 3 shows the decoding and decoding cycle counts. Notice that, for the 2D decoding, we used our custom decoder from Section 5.2, which was about

Table 3. Number of cycles for encoding and decoding under different targets.

Code	Dimension	Encoding (`poly_frommsg`)			Decoding (`poly_tomsg`)		
		AVX2	M4	A53	AVX2	M4	A53
ML-KEM's code	1D	24	6676	441	17	3587	816
Minal code	2D	79	2017	332	418	9248	2724
Minal code	4D	62	1771	337	986	20,142	7937

35% faster than the general decoder in 2D (e.g., 418 cycles instead of 646 in our AVX2 setup). We can see that, unsurprisingly, decoding 2D and 4D codes is more complex under all architectures. However, since the cycle count for decapsulation[4] is much larger than this difference, the overall performance impact of Minal codes on the decapsulation procedure should be very small. The impact of our codes in ML-KEM's decapsulation time is evaluated in Section 7.2.

6 Analyzing Minal codes' DFR when used in ML-KEM

In this section, we discuss how to compute the DFR for our Minal codes when they are used in a lattice-based scheme. For concreteness, we focus specifically on ML-KEM, but the procedure can be extended to other schemes such as Saber [15] and NewHope [2]. While one of the main features of our analysis is that it does not require any independence assumptions on the coefficients of Δm, designers who are willing to make such assumptions can also benefit from our approach. We begin by exploring the source of the dependence between coefficients in Δm.

Consider the noise polynomial $\Delta m = \langle \mathbf{e}, \mathbf{r} \rangle - \langle \mathbf{s}, \mathbf{e}_1 + \Delta \mathbf{u} \rangle + e_2 + \Delta v$. Notice that, by definition, all coefficients from e_2 and Δv are independent. However, the coefficients of the polynomials resulting from the two dot products $\langle \mathbf{e}, \mathbf{r} \rangle$ and $\langle \mathbf{s}, \mathbf{e}_1 + \Delta \mathbf{u} \rangle$ cannot be assumed to be independent, because they are computed through sums of polynomial multiplications. If ignored, this dependence is known to cause issues in scenarios where error-correction codes are used, leading to significantly underestimating the scheme's DFR [16].

Now, let us focus on $\langle \mathbf{e}, \mathbf{r} \rangle$, which is the simplest of the two dot products that determine Δm in Kyber. It is defined as

$$\langle \mathbf{e}, \mathbf{r} \rangle = \mathbf{e}[0]\mathbf{r}[0] + \ldots + \mathbf{e}[k-1]\mathbf{r}[k-1].$$

We start by noticing that every product of polynomials $\mathbf{e}[i]\mathbf{r}[i]$ is independent of $\mathbf{e}[j]\mathbf{r}[j]$ for $j \neq i$. Also, because all $\mathbf{e}[i]$ and $\mathbf{r}[i]$ are sampled from the same $\mathcal{B}_{\eta_1}$, every product $\mathbf{e}[i]\mathbf{r}[i]$ follows the same distribution for all i. Therefore, in what follows, we focus our attention on the joint distribution of $\mathbf{e}[i]\mathbf{r}[i]$ for any particular i to analyze the distribution of $\langle \mathbf{e}, \mathbf{r} \rangle$.

[4] For example, the decapsulation cycle counts in our AVX2 setup are 20725, 31748, and 46104, for levels 1, 3, and 5, respectively.

6.1 The joint distribution of coefficients in ML-KEM's noise

We now present two results that allow us to separate the joint probability distribution of s coefficients of a polynomial product in $R_q = \mathbb{Z}_q[x]/(x^n+1)$ into a sum of independent distributions, provided that three conditions are met: (i) n and s are powers of 2 with $1 < s < n$; (ii) the distance between adjacent pairs of coefficients is n/s; and (iii) the probability distribution for the coefficients of at least one of the polynomials must be symmetric. In particular, by the end of this section, we will characterize the s-dimensional noise distribution

$$\Pr\left(\Delta m[i, i+n/2, \ldots, i+(s-1)n/2]\right),$$

which enables us to analyze the DFR of s-dimensional codes without relying on independence assumptions.

Lemma 1. Let $n \geqslant 4$ be a power of 2, and let s be a positive nontrivial divisor of n. Define $\nu = n/s$ and consider the probability distribution $\mathcal{P}_n$ defined over s-tuples as

$$\mathcal{P}_n = \Pr\left(c[0, \nu, 2\nu, \ldots, n-\nu]\right),$$

where the polynomial c is defined by the following experiment. Let $\mathcal{D}_1$ be any distribution over $\mathbb{Z}_q$, and $\mathcal{D}_2$ be a symmetric distribution also over $\mathbb{Z}_q$. Choose polynomials a and b in $R_q = \mathbb{Z}_q[x]/(x^n+1)$ by taking its coefficients from distributions $\mathcal{D}_1$ and $\mathcal{D}_2$, respectively. Compute their product $c = ab \in R_q$, and output the tuple $c[0, \nu, 2\nu, \ldots, (n-\nu)]$ consisting of the coefficients of c associated with powers $x^{i\nu}$, for $i = 0$ up to $s-1$. Then $\mathcal{P}_n$ can be split as $\mathcal{P}_n = \mathcal{P}_{n/2} + \mathcal{P}_{n/2}$.

Proof. Let $n = 2^\ell$ for some $\ell \geqslant 2$. Take polynomials a and b from R_q. If $c = ab$, we can use Equation 1 from Section 2 to write each coefficient of the product as

$$c_i = \texttt{poly\_to\_vec}(c)[i] = \langle \texttt{poly\_to\_vec}(a), \texttt{negashift}_i(b) \rangle .$$

Let $\mathbf{a} = \texttt{poly\_to\_vec}(a)$ and $\mathbf{b} = \texttt{negashift}_0(b)$, i.e., vector $\mathbf{b}$ is the first column of the negacyclic matrix generated by the coefficients of b. Notice that, since $\mathcal{D}_2$ is symmetric, $\mathbf{b} = (b_0, -b_{n-1}, \ldots, -b_1)$ has the same distribution as b.

Because $n \geqslant 4$ is a power of 2 and s divides n, $\mathbf{a}$ and $\mathbf{b}$ can be split into s parts, each of length ν, such that

$$c_0 = \left\langle \left[\mathbf{a}_1\ ,\ \mathbf{a}_2\ , \ldots,\ \mathbf{a}_{(s-1)}\ ,\ \mathbf{a}_s\right], \left[\mathbf{b}_1\ ,\ \mathbf{b}_2\ , \ldots,\ \mathbf{b}_{(s-1)}\ ,\ \mathbf{b}_s\right] \right\rangle .$$

More generally, for $i = 0$ up to s, we can write

$$c_{i\nu} = \left\langle \left[\mathbf{a}_1\ ,\ \mathbf{a}_2\ , \ldots,\ \mathbf{a}_{(s-1)}\ ,\ \mathbf{a}_s\right], \left[-\mathbf{b}_{(s-i+1)}\ , \ldots,\ \mathbf{b}_{(s-i-1)}\ ,\ \mathbf{b}_{(s-i)}\right] \right\rangle .$$

Remember that, since s is a nontrivial divisor of n, then $\nu = n/s$ is even, so we can split each $\mathbf{a}_i$ and $\mathbf{b}_i$ into two halves, such that $\mathbf{a}_i = [\mathbf{a}'_i, \mathbf{a}''_i]$ and $\mathbf{b}_i = [\mathbf{b}'_i, \mathbf{b}''_i]$, respectively. Now we write $c_{i\nu} = c'_{i\nu} + c''_{i\nu}$ where each term is defined as

$$c'_{i\nu} = \left\langle \left[\mathbf{a}'_1\ ,\ \mathbf{a}'_2\ , \ldots,\ \mathbf{a}'_{(s-1)}\ ,\ \mathbf{a}'_s\right], \left[-\mathbf{b}'_{(s-i+1)}\ , \ldots,\ \mathbf{b}'_{(s-i-1)}\ ,\ \mathbf{b}'_{(s-i)}\right] \right\rangle, \text{ and}$$

$$c''_{i\nu} = \left\langle \left[\mathbf{a}''_1\ ,\ \mathbf{a}''_2\ , \ldots,\ \mathbf{a}''_{(s-1)}\ ,\ \mathbf{a}''_s\right], \left[-\mathbf{b}''_{(s-i+1)}\ , \ldots,\ \mathbf{b}''_{(s-i-1)}\ ,\ \mathbf{b}''_{(s-i)}\right] \right\rangle .$$

But notice that the entries of $\mathbf{a}$ and $\mathbf{b}$ that appear in terms $c'_0, c'_\nu, \ldots, c'_{s-1}$, that come from the left halves of the blocks of length ν, are completely independent from those appearing in $c''_0, c''_\nu, \ldots, c''_{s-1}$, which come from the right halves. Furthermore, notice that tuples $\left(c'_0, c'_\nu, \ldots, c'_{s-1}\right)$ and $\left(c''_0, c''_\nu, \ldots, c''_{s-1}\right)$ are not only equally distributed, but, by the definition of $\mathcal{P}_j$, both of them follow the s-dimensional distribution $\mathcal{P}_{n/2}$. Therefore, $\mathcal{P}_n = \mathcal{P}_{n/2} + \mathcal{P}_{n/2}$. □

We now present an accompanying result showing that we can use the distribution $\mathcal{P}_n$ from Lemma 1 to characterize all probability distributions $\Pr\left(c[i, i+\nu, i+2\nu, \ldots, i+n-\nu]\right)$, for $i = 0$ to $\nu - 1$.

Proposition 1. Let s, ν, and n be positive integers such that $\nu = n/s$, and $1 < s < n$. Let a and b be two polynomials in $R_q = \mathbb{Z}_q[x]/(x^n + 1)$. Suppose that the coefficients of a and b are sampled from two distributions $\mathcal{D}_1$ and $\mathcal{D}_2$, respectively, and assume that $\mathcal{D}_2$ is symmetric. If $c = ab$ is their product, then, for all $0 \leqslant i < \nu$, we have

$$\Pr\left(c[i, i+\nu, i+2\nu, \ldots, i+n-\nu]\right) = \Pr\left(c[0, \nu, 2\nu, \ldots, n-\nu]\right).$$

Proof. Since $c = ab$, we use Equation 1 to write each coefficient of c as

$$c_i = \texttt{poly\_to\_vec}(c)[i] = \left\langle \texttt{poly\_to\_vec}(a), \texttt{negashift}_i(b) \right\rangle.$$

Notice that this means that the s-tuple $c[i, i+\nu, i+2\nu, \ldots, i+n-\nu]$ is completely defined by the sequence of vectors

$$S_i = \left(\texttt{negashift}_i(b), \ldots, \texttt{negashift}_{(i+n-\nu)}(b)\right).$$

But remember that the elements of b come from a symmetric distribution. Therefore, by the definition of the negacyclic shifts, the sequence S has exactly the same distribution as $S_0 = \left(\texttt{negashift}_0(b), \ldots, \texttt{negashift}_{(n-\nu)}(b)\right)$. □

We notice that Lemma 1 is analogous to the *polynomial splitting for recovery* used in NewHope's original analysis [3, §C]. The main difference is that we provide a broader presentation for supporting a more direct construction of the joint distribution, whose effective computation is not needed in related works.

Let us now discuss the applicability of Lemma 1 to ML-KEM. Fix parameters $q = 3329$ and $n = 256$. Consider the two dot products $\langle \mathbf{e}, \mathbf{r} \rangle$ and $\langle \mathbf{s}, \mathbf{e}_1 + \Delta\mathbf{u} \rangle$ that appear in the computation of Δm. The first one is done with polynomials whose coefficients are taken from the centered binomial $\mathcal{B}_{\eta_1}$, which is symmetric. In the second one, elements from $\mathbf{s}$ are also drawn from $\mathcal{B}_{\eta_1}$. Therefore, we can swap the operands of the commutative product so that all of the lemma's hypotheses are satisfied. We now provide a simple corollary that explicitly states the distribution of the coefficients of the ML-KEM noise.

Corollary 1. Fix integers s and ν such that $\nu = n/s$ and $1 < s < n$. Let $\mathcal{P}_{\text{prod}}^{(\phi_a, \phi_b)}$ denote the probability distribution of a product $c = ab$ of two polynomials

$a, b \in \mathbb{Z}_q/(x^s+1)$, whose coefficients are selected according to distributions ϕ_a and ϕ_b, correspondingly. Let $\mathcal{D}_{\Delta\mathbf{u}}$ denote the distribution of the coefficients of $\Delta\mathbf{u}$ and $\mathcal{P}_{(\Delta v+e_2)}$ be the probability distribution of $(\Delta v + e_2)[i, i+\nu, \ldots, i+n-\nu]$. Then, by Lemma 1 and Proposition 1, we have:

$$\Delta m[i, i+\nu, \ldots, i+n-\nu] \sim \frac{kn}{s}\mathcal{P}_{\text{prod}}^{(\mathcal{B}_{\eta_1}, \mathcal{B}_{\eta_1})} + \frac{kn}{s}\mathcal{P}_{\text{prod}}^{(\mathcal{B}_{\eta_2}, \mathcal{B}_{\eta_1}+\mathcal{D}_{\Delta\mathbf{u}})} + \mathcal{P}_{(\Delta v+e_2)}. \quad \square$$

When s is sufficiently small, we can calculate the base distributions over coefficient pairs, $\mathcal{P}_{\text{prod}}^{(\mathcal{B}_{\eta_1}, \mathcal{B}_{\eta_1})}$ and $\mathcal{P}_{\text{prod}}^{(\mathcal{B}_{\eta_2}, \mathcal{B}_{\eta_1}+\mathcal{D}_{\Delta\mathbf{u}})}$, by enumerating the corresponding polynomials in $\mathbb{Z}_q/(x^s+1)$ and computing their products while keeping track of the associated probabilities. Also, the product rule can be used to directly compute $\mathcal{P}_{(\Delta v+e_2)}$, as the coefficients in both Δv and e_2 are all independent.

The above analysis has an important consequence to how we encode messages using s-dimensional Minal codes. Since $n = 256$ in ML-KEM, we are bound to use codes whose dimension is a power of 2. Additionally, since we need to compute the base s-dimensional distributions, the complexity grows exponentially, making it harder to use $s \geqslant 8$ for ML-KEM parameters. Moreover, to use Corollary 1 when evaluating the DFR of s-dimensional codes applied to ML-KEM, we must encode each s-bit string in the message $\mathbf{m} \in \mathbb{Z}_2^n$ into coefficients separated by n/s entries. In the following sections, we show how to evaluate the DFR when using 2D and 4D Minal codes in ML-KEM.

6.2 Computing the DFR using 2D Minal codes

The core observation allowing us to compute the DFR for 2D Minal codes is that it is possible to use Lemma 1 to fully compute the 2D joint noise distribution $\Delta m[i, i+n/2]$, which is then used for computing the DFR. While the Python scripts [17] provided by the Kyber team are fast enough to compute the distribution of a single coefficient of Δm, their approach is highly inefficient when computing sums of joint distributions. To address this problem, we implemented a custom 2-dimensional FFT with multiprecision complex arithmetic using the MPC [19] and MPFR [20] libraries. Running in a standard PC using 6 threads, the computation of the joint probability distribution $\Pr(\Delta m[i, i+n/2])$ with 260 bits of precision takes less than 3 minutes for each parameter set.

Using $\Delta m[i, i+n/2]$, Algorithm 3 provides the DFR for a 2D Minal code $\mathcal{M}$. First, it computes the error probability p_{failure} for the 2-dimensional code $\mathcal{M}$. This is done by accumulating the probabilities that noise coefficients drawn from $\Pr(\Delta m[i, i+n/2])$ cause decoding failures for each of the 4 possible codewords. Since each codeword appears with probability $1/4$, the error probabilities have to be weighed by this factor when updating the value of p_{failure} in Line 6. The algorithm then returns the upper bound on the DFR by using the union bound over the decoding failure for the $n/2 = 128$ encoded pairs.

6.3 Computing the DFR using 4D Minal codes

If we try to use the approach from Section 6.2 to compute the DFR in the 4D setting, we need to compute the joint distribution $\Pr(\Delta m[0, n/4, n/2, 3n/4])$.

procedure DFR 2D(code $\mathcal{M}$, distribution $\Pr(\Delta m[i, i+n/2])$)
  $p_{\text{failure}} \leftarrow 0$ ▷ Accumulates the decoding error probability for $\mathcal{M}$
  for each codeword $\mathbf{c} \in \mathcal{M}$ **do**
    for each possible error $\mathbf{e} \in \mathbb{Z}_q^2$ **do**
      if Decode $(\mathbf{c}+\mathbf{e}) \neq \mathbf{c}$ **then**
        $p_{\text{failure}} \leftarrow p_{\text{failure}} + \frac{1}{4}\Pr(\Delta m[i, i+n/2] = \mathbf{e})$
  return $\frac{n}{2} p_{\text{failure}}$ ▷ Union bound over the $n/2 = 128$ pairs.

Algorithm 3. Computation of the DFR for a 2D Minal code $\mathcal{M}$.

procedure DFR 4D(code $\mathcal{M}$)
  Compute the 4D Voronoi cells of the main codewords of $\mathcal{M}$
  $p_{\text{failure}} \leftarrow 0$ ▷ Accumulates the decoding error probability for $\mathcal{M}$
  for each main codeword $\mathbf{c} \in \mathcal{M} \cap \mathbb{Z}_q^4$ **do**
    $\text{vor}(\mathbf{c}) \leftarrow$ Voronoi cell centered in $\mathbf{c}$
    $\text{neigh}(\mathbf{c}) \leftarrow$ set of codewords whose cells share a hyperplane with $\text{vor}(\mathbf{c})$
    for each adjacent codeword $\mathbf{v}$ in $\text{neigh}(\mathbf{c}) \in \mathcal{M}$ **do**
      $\hat{\mathbf{v}} \leftarrow \mathbf{v} - \mathbf{c}$
      Compute the 1D distribution $\Pr(\langle \Delta m[0, n/4, 2n/4, 3n/4], \hat{\mathbf{v}} \rangle)$
      $p_{\text{failure}} \leftarrow p_{\text{failure}} + \frac{1}{16}\Pr\left(\langle \Delta m[0, n/4, 2n/4, 3n/4], \hat{\mathbf{v}} \rangle \geqslant \frac{\|\hat{\mathbf{v}}\|^2}{2}\right)$
  return $\frac{n}{4} p_{\text{failure}}$ ▷ Union bound over the $n/4 = 64$ pairs.

Algorithm 4. Computation of the DFR for a 4D Minal code $\mathcal{M}$.

The problem is that the 4D FFTs, which have to be computed with padding so that the convolutions do not cause cyclic interference, would incur prohibitively large memory and processing costs. Our solution is then to make a different use of Lemma 1 in a way which is akin to how the DFR of NewHope's first variant was computed [3]. Algorithm 4 briefly describes the 4D DFR computation, while a detailed explanation is given in what follows.

First, we define the parameters q and β of the Minal code $\mathcal{M}$ whose DFR we want to evaluate. Usually, the field size q is defined by the cryptographic scheme (e.g., $q = 3329$ for ML-KEM) and β can be found using the p-norm approach discussed in Section 4.3. Using these parameters, we can compute the 4D Voronoi cells for all main codewords using the Quickhull [11] algorithm from the Qhull library [10]. From the 4D Voronoi cells, we can see which are the Voronoi-relevant vectors of each codeword, i.e., the neighboring codewords that characterize the Voronoi cells of a given codeword.

Based on the Voronoi cells, we define a function $\text{neigh}(\mathbf{c})$ that, for a main codeword $\mathbf{c} \in \mathcal{M} \cap \mathbb{Z}_q^4$, returns the set of points whose Voronoi cells share a common hyperplane with the Voronoi cell centered in $\mathbf{c}$. Let $p_{\text{error}}(\mathbf{c}, \mathbf{v})$ denote the probability that $\mathbf{c}' = \mathbf{c} + \Delta m[0, n/4, 2n/4, 3n/4]$ is closer to $\mathbf{v}$ than to $\mathbf{c}$. The DFR is then upper-bounded by

$$\text{DFR} \leqslant \frac{1}{16} \sum_{\mathbf{c} \in \mathcal{M} \cap \mathbb{Z}_q^4} \left(\sum_{\mathbf{v} \in \text{neigh}(\mathbf{c})} p_{\text{error}}(\mathbf{c}, \mathbf{v}) \right),$$

where the 1/16 factor accounts for the probability of getting each codeword $\mathbf{c}$.

Now, we know from elementary linear algebra that $\|\mathbf{c}' - \mathbf{c}\| \geqslant \|\mathbf{c}' - \mathbf{v}\|$ if, and only if, $\langle \mathbf{c}', \mathbf{v} - \mathbf{c} \rangle \geqslant \frac{1}{2}\left(\|\mathbf{v}\|^2 - \|\mathbf{c}\|^2\right)$. Then, if we expand $\mathbf{c}'$, we get that $p_{\text{error}}(\mathbf{c}, \mathbf{v})$ can be computed as

$$p_{\text{error}}(\mathbf{c}, \mathbf{v}) = \Pr\left(\langle \Delta m[0, n/4, 2n/4, 3n/4], \mathbf{v} - \mathbf{c} \rangle \geqslant \tfrac{1}{2}\|\mathbf{v} - \mathbf{c}\|^2\right).$$

While we cannot compute the noise distribution $\Pr\left(\Delta m[0, n/4, 2n/4, 3n/4]\right)$ in 4D, we can compute $p_{\text{error}}(\mathbf{c}, \mathbf{v})$ as follows. Consider the following vectors and their associated probability distributions: $\mathbf{x}_1 \sim \mathcal{P}_{\text{prod}}^{\left(\mathcal{B}_{\eta_1}, \mathcal{B}_{\eta_1}\right)}$, $\mathbf{x}_2 \sim \mathcal{P}_{\text{prod}}^{\left(\mathcal{B}_{\eta_2}, \mathcal{B}_{\eta_1}\right)}$, $\mathbf{x}_3 \sim \mathcal{D}_{\Delta\mathbf{u}}$, and $\mathbf{x}_4 \sim \mathcal{P}_{(\Delta v + e_2)}$. Define probability distributions $\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3, \mathcal{D}_4$ such that each $\mathcal{D}_i = \Pr\langle \mathbf{x}_i, \mathbf{v} - \mathbf{c} \rangle$, for $i = 1$ up to 4. Then, using Corollary 1 together with the linearity of the dot product, we have

$$\Pr\left(\langle \Delta m[0, n/4, 2n/4, 3n/4], \mathbf{v} - \mathbf{c} \rangle\right) = \frac{kn}{s}\mathcal{D}_1 + \frac{kn}{s}\mathcal{D}_2 + \mathcal{D}_3 + \mathcal{D}_4.$$

Notice that each $\mathcal{P}_i$ is efficiently computable because the dimension 4 is relatively small and is a simple 1D distribution. Now, to compute the multiple convolutions above, we can again use arbitrary precision FFTs. We notice that, because $\mathbf{v} - \mathbf{c}$ may have large coordinates, the padding required for the FFT-based convolution is relatively high. Therefore, we use Bailey's [9] trick that transforms an FFT over a large vector into a 2D FFT with shorter vectors to make the implementation more efficient. For a given ML-KEM parameter set, together with the 4D Minal code parameters, our implementation gives the corresponding DFR in about 1 hour in a standard computer using 6 threads.

7 Applications to compact ML-KEM instantiations

This section shows a key benefit of our proposed codes: with them, ML-KEM parameter sets associated with shorter ciphertext sizes can achieve failure rates lower than the DFR targets, thus making them viable. We begin by remarking that the size of a ciphertext $(\mathbf{c_u}, c_v)$ in ML-KEM is completely determined by the parameters $n, k, d_{\mathbf{u}}$, and d_v as $\left\lceil \frac{1}{8} n(k d_{\mathbf{u}} + d_v) \right\rceil = 32(k d_{\mathbf{u}} + d_v)$ bytes. Hence, to obtain shorter ciphertexts without changing the core parameters n and k, we need to use smaller parameters for $d_{\mathbf{u}}$ and d_v than those adopted by the current standard. In this section, we show how this can be accomplished for each security level supported by ML-KEM.

7.1 Finding compression parameters that yield shorter ciphertexts

First, we consider ML-KEM settings with values of $(d_{\mathbf{u}}, d_v)$, allowing for shorter ciphertexts than the current standard. We then compute the DFR for the regular ML-KEM 1D code, and for our 2D and 4D Minal codes tailored for the error

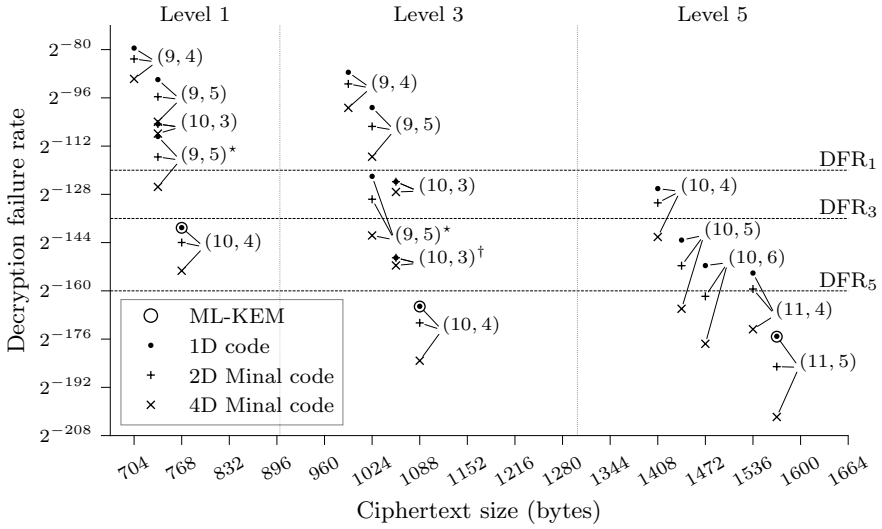


Figure 6. The impact of our 2D and 4D Minal codes in obtaining viable compression parameters $(d_{\mathbf{u}}, d_v)$ allowing for shorter ciphertexts.

distribution induced by these parameters. Then, we can select points with shorter ciphertext sizes, as long as their DFR lies below the target for each security level.

Figure 6 shows our results. Note that results for the experiment described above are the regular $(d_{\mathbf{u}}, d_v)$ points, i.e., points marked as $(d_{\mathbf{u}}, d_v)^\star$ or $(d_{\mathbf{u}}, d_v)^\dagger$ are not part of this experiment, as they require additional changes that are described later in this section. We can see that the impact of our codes is greater in Level 5, as there are multiple points using 2D and 4D codes with significantly shorter ciphertexts – namely, up to 8% compression. This results from the fact that Level 5 parameters, in general, use higher values of d_v, so the uniform part of the noise is less relevant, and our codes are better at error correction. To allow for an easier comparison of the actual numbers, we collect in Table 4 the relevant parameters discussed in this section.

Now, to obtain compressed ciphertexts for levels 1 and 3, we proceed as follows. First, remember that the security evaluation of ML-KEM-512 uses not only the LWE hardness but also considers the deterministic compression noise when evaluating the Core-SVP hardness associated with the ciphertext security, which is akin to also considering the LWR in the security analysis. This allows ML-KEM-512 to use a slightly lower value[5] for η_2, without compromising security, because the extra noise introduced by the compression accounts for the decrease in η_2. If we extend this idea further, we can use even more aggressive compression factors than the values $(d_{\mathbf{u}}, d_v) = (10, 4)$ used in ML-KEM-512 and ML-KEM-768, and get rid of η_2 completely. This idea is actually not new, as it is even mentioned in Kyber's round 1 specification [5, §6.4.6]. However, for this

[5] Remember that η_2 defines the centered binomial distribution used for generating vector $\mathbf{e}_1$ and polynomial e_2 used in encryption.

Table 4. Relevant parameters allowing for ciphertext compression. Non-viable codes whose DFR lie above the DFR target are marked with a slash.

NIST Level	DFR target	$d_{\mathbf{u}}$	d_v	η_2	Parameter set name	Requires LWR?	Ciphertext size (bytes)	Minimum DFR found 1D	2D	4D
1	2^{-120}	10	4	2	ML-KEM-512	Yes	768	$2^{-139.1}$	$2^{-144.0}$	$2^{-153.3}$
		9	5	0	None	Yes	736	~~$2^{-108.9}$~~	~~$2^{-115.6}$~~	$2^{-125.5}$
3	2^{-136}	10	4	2	ML-KEM-768	No	1088	$2^{-165.2}$	$2^{-170.6}$	$2^{-183.2}$
		10	3	1	None	Yes	1056	$2^{-149.2}$	$2^{-149.2}$	$2^{-151.6}$
		9	5	0	None	Yes	1024	~~$2^{-122.1}$~~	~~$2^{-129.6}$~~	$2^{-141.6}$
5	2^{-160}	11	5	2	ML-KEM-1024	No	1568	$2^{-175.2}$	$2^{-185.1}$	$2^{-201.8}$
		11	4	2	None	No	1536	~~$2^{-154.2}$~~	~~$2^{-159.4}$~~	$2^{-172.7}$
		10	6	2	None	No	1472	~~$2^{-151.7}$~~	$2^{-161.8}$	$2^{-177.6}$
		10	5	2	None	No	1440	~~$2^{-143.3}$~~	~~$2^{-151.7}$~~	$2^{-165.9}$

approach to be successful for ML-KEM, one needs to use a better error correction code because it is more difficult to deal with the higher compression noise, which is uniform in nature, than with the noise from the centered binomials.

The result of this approach is illustrated in Figure 6 by the points $(9,5)^\star$ and $(10,3)^\dagger$ that define $\eta_2 = 0$ and $\eta_2 = 1$, respectively. Note that these are only considered for levels 1 and 3. We can see that our 4D Minal codes allow for viable points, providing 4% and 6% shorter ciphertexts for both levels 1 and 3, respectively. Finally, we remark that all parameters provide the same Core-SVP hardness as ML-KEM's standards for the corresponding security level, as computed by the security scripts provided by the Kyber team [17].

7.2 Proposed parameters and their performance impact

Section 5.3 shows that the decoding operation of Minal codes is significantly more costly than that of ML-KEM's 1D code, due to the higher-dimensional nature of our proposal. However, we emphasize that the time taken by our isochronous decoding algorithms is still orders of magnitude lower than the full decapsulation time. Moreover, since our Minal codes allow $\eta_2 = 0$ to be a viable setting, together with $(d_{\mathbf{u}}, d_v)$ in levels 1 and 3, the full setup also benefits from not having to make the additional samplings related to η_2 in the encryption procedure.

To show this reduced impact, we benchmarked ML-KEM's operations using Minal codes for the same targets and settings from Section 5.3. We also use a general isochronous implementation of the Minal code operations, rather than optimizing it for the given targets, while integrating our code into highly optimized implementations for AVX2 [7], Cortex M4 [26] and Cortex-A53 [12].

The results are depicted in Table 5, which shows the speedup of our proposal compared with the encapsulation and decapsulation times for ML-KEM implementations using standard parameters. As expected, the performance impact is minor. For levels 1 and 3, there is even a significant speedup resulting from the

Table 5. ML-KEM instantiations with compressed ciphertexts and speedups for full encapsulation and decapsulation, for different security levels and platforms.

NIST Level	$(d_{\mathbf{u}}, d_v, \eta_2)$	Minal code	Ciphertext compression	Encaps speedup			Decaps speedup		
				AVX2	M4	A53	AVX2	M4	A53
1	(9, 5, 0)	4D ($\beta = 745$)	4.17%	0.96	1.14	1.13	0.99	1.08	1.02
3	(9, 5, 0)	4D ($\beta = 741$)	5.88%	1.07	1.10	1.07	1.05	1.07	1.01
5	(10, 6, 2)	2D ($\beta = 442$)	6.12%	0.99	1.00	1.00	1.00	1.00	0.99
5	(10, 5, 2)	4D ($\beta = 741$)	8.16%	1.01	1.00	1.00	0.99	0.99	0.97

fact that some of the sampling operations are not needed in encryption[6] since $\eta_2 = 0$. For level 5, no speedup was observed since there is no change to η_2, but the performance impact is still negligible, specially when 2D codes are used. Note that all parameters in Table 5 provide the same core-SVP hardness as the standard ML-KEM parameters, for the corresponding security levels, as computed by Kyber's security scripts [17]. Furthermore, if we compare our results with the ones obtained in [36,37] with 8D lattice codes (see Table 2), we can see the power of tailoring. For concreteness, consider level 5: even the low-dimensional 2D Minal codes already provide ciphertext compression, requiring a simple change in the compression factors; in contrast, the 8D codes from [36, 37] require changes to core ML-KEM parameters, and actually increase the ciphertext size.

7.3 An overview of additional applications of our codes

Due to space limitations, the results presented so far focus more on applications to ciphertext compression, which are arguably more immediately applicable to ML-KEM. However, we argue that there are other important applications to which our codes also have a relevant impact, as briefly discussed next.

Eliminating the LWR assumption from ML-KEM-512. Remember that ML-KEM-512 requires a hardness assumption similar to LWR to achieve its claimed security using $\eta_2 = 2 < \eta_1$. Using our 4D codes, though, we can instantiate ML-KEM-512 using $\eta_2 = \eta_1 = 3$, and get a DFR of $2^{-135.6}$ that is very close to the one observed for ML-KEM-512. This would not only eliminate the need for the LWR assumption but it would also result in an easier specification by removing the need of one additional parameter. Moreover, it would allow for simpler vectorized implementations of the sampling, since only one function would be needed for generating all centered binomial values.

Public-key compression for free. Kyber's round 1 specification [5] allowed compression of the public key. Using a slightly different formulation of the noise

[6] In ML-KEM, the decapsulation also calls the encryption procedure due to the reencryption step of the Fujisaki-Okamoto transform.

polynomial Δm to take public key compression into account [13, Thm. 1], we can easily extend our analysis for this case. Namely, our Minal 4D codes allow for 8% compression of the ML-KEM public key while keeping the DFR very close[7] to the current standards without compression.

Exploring the parameter space considering our codes. From our results, we can see that the usage of higher-dimensional Minal codes empowers designers of lattice-based KEMs by providing a richer set of trade-offs between failure rate and both ciphertext and public key sizes. In this paper, we build upon this flexibility to show how Minal codes can be directly applied to the ML-KEM scheme without changing its core parameters. Nevertheless, our results suggest that Minal codes can be explored much beyond this particular case, leading to interesting parameter sets that yield significantly more compact schemes.

8 Conclusion and future work

We present a new family of error-correction codes called Minal codes. By proposing a novel way to model the accumulated noise in lattice-based schemes as n-dimensional circles under different p-norms, we show how Minal codes can be tailored to improve their error correction capability for each particular application without requiring any change to the target scheme's parameters. We then demonstrate how to compute the decryption failure rate (DFR) of our Minal codes without making independence assumptions on the real noise distribution.

We use ML-KEM, which is arguably the most relevant KEM today, to illustrate how our proposed codes can impact even state-of-the-art constructions by obtaining up to 8% ciphertext compression for this highly optimized scheme. Additionally, we provide efficient isochronous implementations of our code's decoding procedure that cause only a minor impact on the full decapsulation execution time, even compared with optimized implementations of ML-KEM under three relevant targets: Intel's AVX2, ARM Cortex-M4, and ARM Cortex-A53.

This work also raises several questions, both in theory and practice. We believe the most important theoretical question is whether we can build more efficient decoders, possibly borrowing results from decoding of lattice codes and adapting them to our case. However, there are other interesting research directions, such as formalizing and possibly proving some of our heuristic[8] assumptions used for tailoring, or trying to find even better error-correction codes by considering more complex constructions with additional parameters.

With respect to the practical side, it would be interesting to devise efficient implementation using SIMD instructions for CPUs supporting AVX2/AVX512 or ARM NEON. It would also be interesting to extend our analysis to ML-KEM using 8D Minal codes. If, as in our present work, one is not willing to make simplifying independence assumptions because these can yield underestimated

[7] Less than 2 bits of impact in the corresponding DFR.
[8] The heuristic assumptions are only used for tailoring and have no security impact.

DFR values [16], the main challenge is the increased complexity of computing the joint 8D base distributions (before the self-convolutions). We also think it would be important to understand how our codes might improve other prominent lattice-based schemes such as Saber [15] and NewHope [3].

References

1. Alagic, G., Alperin-Sheriff, J., Apon, D., Cooper, D., Dang, Q., Liu, Y.K., Miller, C., Moody, D., Peralta, R., Perlner, R., Robinson, A., Smith-Tone, D.: Status report on the first round of the NIST post-quantum cryptography standardization process. US Department of Commerce, National Institute of Standards and Technology (2019). https://doi.org/10.6028/NIST.IR.8240 Cited on 2.
2. Alkim, E., Avanzi, R., Bos, J.W., Ducas, L., la Piedra, A.d., Poppelmann, T., Schwabe, P., Stebila, D.: NewHope (Version 1.1): Algorithm Specifications And Supporting Documentation. https://newhopecrypto.org/resources.shtml (Apr 2020) Cited on 1, 2, and 18.
3. Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P.: Post-quantum key exchange – a new hope. In: 25th USENIX Security Symposium (USENIX Security 16). pp. 327–343 (2016) Cited on 2, 6, 7, 20, 22, and 28.
4. Aragon, N., Barreto, P., Bettaieb, S., Bidoux, L., Blazy, O., Deneuville, J.C., Gaborit, P., Ghosh, S., Gueron, S., Güneysu, T., et al.: BIKE: Bit flipping key encapsulation (2022) Cited on 6.
5. Avanzi, R., Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS-Kyber: Algorithm specifications and supporting documentation (2017), https://pq-crystals.org/kyber/data/kyber-specification.pdf. Cited on 24 and 26.
6. Avanzi, R., Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS-Kyber: Algorithm specifications and supporting documentation (version 2.0) (2019), https://pq-crystals.org/kyber/data/kyber-specification-round2.pdf. Cited on 7.
7. Avanzi, R., Bos, J.W., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS-Kyber: Algorithm specifications and supporting documentation (version 3.02) (2021), https://pq-crystals.org/kyber/data/kyber-specification-round3-20210804.pdf. Cited on 1, 2, 7, 12, 17, and 25.
8. Baan, H., Bhattacharya, S., Fluhrer, S., Garcia-Morchon, O., Laarhoven, T., Rietman, R., Saarinen, M.J.O., Tolhuizen, L., Zhang, Z.: Round5: Compact and fast post-quantum public-key encryption. In: Post-Quantum Cryptography: 10th International Conference, PQCrypto 2019, Chongqing, China, May 8–10, 2019 Revised Selected Papers 10. pp. 83–102. Springer (2019). https://doi.org/10.1007/978-3-030-25510-7_5 Cited on 2.
9. Bailey, D.H.: FFTs in external or hierarchical memory. The journal of Supercomputing **4**, 23–35 (1990) Cited on 23.
10. Barber, C., Dobkin, D., Huhdanpaa, H.: The Quickhull algorithm for convex hulls. ACM Trans. on Mathematical Software **22**(4), 469–483 (Dec 1996), http://www.qhull.org Cited on 22.
11. Barber, C.B., Dobkin, D.P., Huhdanpaa, H.: The Quickhull algorithm for convex hulls. ACM Transactions on Mathematical Software (TOMS) **22**(4), 469–483 (1996) Cited on 22.

12. Becker, H., Hwang, V., Kannwischer, M.J., Yang, B.Y., Yang, S.Y.: Neon NTT: Faster Dilithium, Kyber, and Saber on Cortex-A72 and Apple M1. IACR Transactions on Cryptographic Hardware and Embedded Systems **2022**(1), 221–244 (Nov 2021). https://doi.org/10.46586/tches.v2022.i1.221-244, artifact available at https://artifacts.iacr.org/tches/2022/a2 Cited on 17 and 25.
13. Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS-Kyber: a CCA-secure module-lattice-based KEM. In: 2018 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 353–367. IEEE (2018) Cited on 27.
14. Chen, C., Danba, O., Hoffstein, J., Hülsing, A., Rijneveld, J., Schanck, J.M., Schwabe, P., Whyte, W., Zhang, Z.: NTRU: Algorithm specifications and supporting documentation (March 20, 2019). Submission to the NIST post-quantum project (2019) Cited on 1.
15. D'Anvers, J.P., Karmakar, A., Sinha Roy, S., Vercauteren, F.: SABER: Mod-LWR based KEM (Round 3 Submission). Submission to the NIST post-quantum project (2020) Cited on 1, 2, 7, 12, 18, and 28.
16. D'Anvers, J.P., Vercauteren, F., Verbauwhede, I.: The impact of error dependencies on Ring/Mod-LWE/LWR based schemes. In: Ding, J., Steinwandt, R. (eds.) Post-Quantum Cryptography. pp. 103–115. Springer International Publishing, Cham (2019). https://doi.org/10.1007/978-3-030-25510-7_6 Cited on 2, 7, 18, and 28.
17. Ducas, L., Schanck, J.: Security estimation scripts for Kyber and Dilithium. Available: https://github.com/pq-crystals/security-estimates (2021) Cited on 5, 6, 21, 25, and 26.
18. D'Anvers, J.P., Guo, Q., Johansson, T., Nilsson, A., Vercauteren, F., Verbauwhede, I.: Decryption failure attacks on IND-CCA secure lattice-based schemes. In: Public-Key Cryptography–PKC 2019: 22nd IACR International Conference on Practice and Theory of Public-Key Cryptography, Beijing, China, April 14-17, 2019, Proceedings, Part II 22. pp. 565–598. Springer (2019). https://doi.org/10.1007/978-3-030-17259-6_19 Cited on 2 and 6.
19. Enge, A., Gastineau, M., Théveny, P., Zimmermann, P.: MPC – A library for multiprecision complex arithmetic with exact rounding (Dec 2022), http://www.multiprecision.org/mpc/ Cited on 21.
20. Fousse, L., Hanrot, G., Lefèvre, V., Pélissier, P., Zimmermann, P.: MPFR: A multiple-precision binary floating-point library with correct rounding. ACM Transactions on Mathematical Software (TOMS) **33**(2), 13–es (2007). https://doi.org/10.1145/1236463.1236468 Cited on 21.
21. Fujisaki, E., Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. In: Annual International Cryptology Conference. pp. 537–554. Springer (1999) Cited on 4.
22. Guo, Q., Johansson, T., Yang, J.: A novel CCA attack using decryption errors against LAC. In: Galbraith, S.D., Moriai, S. (eds.) Advances in Cryptology – ASIACRYPT 2019. pp. 82–111. Springer International Publishing, Cham (2019). https://doi.org/10.1007/978-3-030-34578-5_4 Cited on 2 and 6.
23. Hofheinz, D., Hövelmanns, K., Kiltz, E.: A modular analysis of the Fujisaki-Okamoto transformation. In: Kalai, Y., Reyzin, L. (eds.) Theory of Cryptography. pp. 341–371. Springer International Publishing, Cham (2017). https://doi.org/10.1007/978-3-319-70500-2_12 Cited on 4.
24. Johnson, D., Menezes, A., Vanstone, S.: The elliptic curve digital signature algorithm (ECDSA). International journal of information security **1**(1), 36–63 (2001). https://doi.org/10.1007/s102070100002 Cited on 1.

25. Kannwischer, M., Schwabe, P., Stebila, D., Wiggers, T.: Improving software quality in cryptography standardization projects. In: IEEE European Symposium on Security and Privacy, EuroS&P - Workshops. pp. 19–30. IEEE Computer Society, Los Alamitos, CA, USA (Jun 2022). https://doi.org/10.1109/EuroSPW55150.2022.00010, https://eprint.iacr.org/2022/337 Cited on 17.
26. Kannwischer, M.J., Rijneveld, J., Schwabe, P., Stoffelen, K.: pqm4: Testing and benchmarking NIST PQC on ARM Cortex-M4. In: Second PQC Standardization Conference: University of California, Santa Barbara and co-located with Crypto 2019. pp. 1–22 (2019) Cited on 17 and 25.
27. Liu, S., Sakzad, A.: Lattice codes for CRYSTALS-Kyber (Sep 2023), http://arxiv.org/abs/2308.13981 Cited on 2 and 7.
28. Lu, X., Liu, Y., Zhang, Z., Jia, D., Xue, H., He, J., Li, B., Wang, K.: LAC: Practical Ring-LWE based public-key encryption with byte-level modulus. Cryptology ePrint Archive, Paper 2018/1009 (2018), https://eprint.iacr.org/2018/1009 Cited on 2.
29. Melchor, C.A., Aragon, N., Bettaieb, S., Bidoux, L., Blazy, O., Bos, J., Deneuville, J.C., Dion, A., Gaborit, P., Lacan, J., Persichetti, E., Robert, J.M., Véron, P., Zémor, G.: Hamming Quasi-Cyclic: HQC (2021), https://pqc-hqc.org/doc/hqc-specification_2021-06-06.pdf Cited on 6 and 14.
30. National Institute of Standards and Technology: FIPS203: Module-lattice-based key-encapsulation mechanism standard. Federal Inf. Process. Stds. (NIST FIPS), National Institute of Standards and Technology (Aug 2024), https://doi.org/10.6028/NIST.FIPS.203 Cited on 1 and 5.
31. Plantard, T., Sipasseuth, A., Susilo, W., Zucca, V.: Tight bound on NewHope failure probability. IEEE Transactions on Emerging Topics in Computing **10**(4), 1955–1965 (2022). https://doi.org/10.1109/TETC.2021.3138951 Cited on 7.
32. Raspberry Pi: Raspberry Pi Zero 2 W. Available: https://www.raspberrypi.com/products/raspberry-pi-zero-2-w/ (2024) Cited on 17.
33. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. Journal of the ACM (JACM) **56**(6), 1–40 (2009). https://doi.org/10.1145/1568318.1568324 Cited on 2.
34. Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM **21**(2), 120–126 (1978). https://doi.org/10.1145/359340.359342 Cited on 1.
35. Saarinen, M.J.O.: Hila5: On reliability, reconciliation, and error correction for ring-lwe encryption. In: Selected Areas in Cryptography–SAC 2017: 24th International Conference, Ottawa, ON, Canada, August 16-18, 2017, Revised Selected Papers 24. pp. 192–212. Springer (2018) Cited on 2.
36. Saliba, C.: Error correction and reconciliation techniques for lattice-based key generation protocols. Ph.D. thesis, CY Cergy Paris Université (2022), https://theses.hal.science/tel-03718212v1/document Cited on 2, 7, 8, and 26.
37. Saliba, C., Luzzi, L., Ling, C.: A reconciliation approach to key generation based on Module-LWE. In: 2021 IEEE International Symposium on Information Theory (ISIT). pp. 1636–1641 (2021). https://doi.org/10.1109/ISIT45174.2021.9517882 Cited on 2, 7, 8, and 26.
38. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM Journal on Computing **26**(5), 1484–1509 (1997). https://doi.org/10.1137/S0097539795293172 Cited on 1.
39. STMicroelectronics: STM32 Nucleo-144 development board with STM32F439ZI MCU, supports Arduino, ST Zio and morpho connectivity. Available: https://www.st.com/en/evaluation-tools/nucleo-f439zi.html (2024) Cited on 17.