mathematics

MDPI

Review

Unfolding Post-Quantum Cryptosystems: CRYSTALS-Dilithium, McEliece, BIKE, and HQC

Vaghawan Prasad Ojha [1,*], Sumit Chauhan [2], Shantia Yarahmadian [3] and David Carvalho [2]

[1] Department of Mathematics and Statistics, Bagley College of Engineering, Mississippi State University, Starkville, MS 39762, USA
[2] Naoris Tech Inc., Wilmington, DE 19808, USA; sumit@naoris.com (S.C.); david@naoris.com (D.C.)
[3] Department of Mathematics and Statistics, Mississippi State University, Starkville, MS 39762, USA; syarahmadian@math.msstate.edu
* Correspondence: vpo4@msstate.edu

Abstract

The advent of quantum computers poses a significant threat to the security of classical cryptographic systems. To address this concern, researchers have been actively investigating the development of post-quantum cryptography, which aims to provide encryption schemes that remain secure even in the face of powerful quantum adversaries. To address this serious problem, the National Institute of Standards and Technology (NIST), a body of the US government, has been working on the selection and standardization of cryptographic algorithms through competitive and rigorous evaluation on different fronts. NIST has selected different candidate algorithms to standardize public-key encryption, including key establishment algorithms and digital signature algorithms. This paper reviews some selected cryptosystems, mainly based on lattice- and code-based cryptosystems. These include digital signature algorithms, such as CRYSTALS-Dilithium, code-based cryptosystems, such as McEliece, and key encapsulation methods, specifically, Classic McEliece, BIKE and HQC. We will review these algorithms and discuss their security aspects and the current state-of-the-art in the development of these algorithms post NIST 3rd finalized selection. We will also touch briefly on the differences and practical applications of each of these schema. This review is intended for engineers and practitioners alike.

Keywords: post-quantum; cryptosystem; cryptography; CRYSTALS-dilithium; McEliece; BIKE; HQC

MSC: 81P94

check for updates

Academic Editors: Chi Cheng, Jintai Ding and Yanbin Pan

Received: 26 June 2025
Revised: 16 August 2025
Accepted: 27 August 2025
Published: 3 September 2025

Citation: Ojha, V.P.; Chauhan, S.; Yarahmadian, S.; Carvalho, D. Unfolding Post-Quantum Cryptosystems: CRYSTALS-Dilithium, McEliece, BIKE, and HQC. *Mathematics* **2025**, *13*, 2841. https://doi.org/10.3390/math13172841

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Today the world has essentially become a complex mesh of digital devices through which we work, communicate, earn our living, do business and debate, with every aspect of both large and small events that are happening around the world being quickly disseminated as they happen. Essentially, digital communication is at the heart of almost every technology connected to the internet, be it directly or indirectly. When transferring goods from one place to another, safeguarding the goods throughout the journey from one point to another is crucial; similarly, in digital communication, safeguarding the information being transferred is equally important. The whole digital world basically hinges on the assurance that the information being passed and received is safeguarded throughout the communication channels through which it travels. Cryptographic algorithms are used to

generate the public and private keys which are used to encrypt and decrypt the message so that any risk of information leaks to an eavesdropper is avoided. The cryptographic algorithms that are used to safeguard the tranfer of messages are becoming increasingly vulnerable as more powerful quantum computers evolve.

The emergence of Shor's algorithm [1], which factors primes in $O(\log^3 N)$ time, in contrast to the $O\Big(\exp\Big((64/9)^{1/3}(\log N)^{1/3}(\log\log N)^{2/3}\Big)\Big)$ required by classical computers employing the fastest known General Number Field Sieve (GNFS) algorithm [2], has led to significant concerns regarding the security of traditional encryption systems [1,3]. Another major advancement is Grover's algorithm [4], which searches for a value within an unstructured collection of N items in $O(\sqrt{N})$ time, in contrast to $O(N)$ for classical search methods [5,6]. This quantum search technique can also be applied to locating the root of a function [7].

Quantum computers, although far from being realized in practice, pose a theoretical risk with respect to certain important problems such as prime-factoring, which is believed to be NP, co-NP, thus posing significant threats to cryptographic algorithms which are based on the hardness of this problem [8–10]. Public-key cryptography, which takes advantage of the computational difficulties of factoring large prime numbers, provides the foundation for security. Nonetheless, Shor's method can be used by quantum computers to efficiently factor large numbers, which puts public-key encryption at risk. The Internet's communication infrastructure depends on protocols such as HTTP and HTTPS, which are part of the SSL/TLS protocol stack, to maintain the security mantle. RSA, Diffie-Hellman (DH), Elliptic-curve Diffie-Hellman (ECDH), ECDSA, and Digital Signature Algorithm (DSA) are examples of non-quantum-safe encryption algorithms that TLS supports. However, if reliable quantum computers become more prevalent, these mechanisms will not be sufficient [7]. Table 1 shows the vulnerability of the popular classical cryptographic algorithms which are most widely used.

To deal with the impending threat, the National Institute of Standards and Technology has been synthesizing research and creating new standards for post-quantum cryptography (PQC) [11]. In contrast to Quantum Key Distribution (QDK), PQC concerns more complex mathematical issues rather than underlying processes [7,12]. To produce quantum-safe asymmetric key pairs, PQC attacks the problems from different fronts. These include lattice-based cryptography, which addresses well-known lattice problems like the shortest vector problem; code-based cryptography, which takes advantage of the complexity of decoding generic linear codes; and multivariate cryptography, which expands on multivariate polynomials over a finite field [7].

Seven out of the 15 third-round candidates of NIST are lattice-based cryptosystems. These cryptosystems are supported by a large body of academic research, which emphasizes asymptotic provable security based on worst-case hardness of lattice problems (via the worst-case-to-average case) reduction [13]. In Round 3 of NIST PQC, three of the final public key encryption (PKE) methods and key encapsulation mechanisms (KEMs) were latticed-based [14].

A seminal work by Regev [15] from 2005 revealed a crucial connection between the difficulty of solving complex lattice problems and possible public key encryption techniques. Regev [15] presented the idea of the learning with errors (LWE) problem and suggested that a public-key encryption architecture be built around it. Additionally, he presented a theoretical argument linking the intrinsic difficulty of lattice theory's Shortest Vector Problem (SVP) to the quantum resilience of this encryption system. Specifically, the SVP poses the problem of finding the shortest possible vectors within a lattice structure, which becomes NP-hard when attempting solutions for general lattices with small enough approximation factors. However, the lattice-theoretic practical encryption schemes usually

handle SVP forms outside of this NP-hard classification. Moreover, the NP-hardness considerations may not necessarily apply to lattices with particular algebraic structures; rather, they are mostly relevant to theoretical worst-case scenarios.

Similarly, the code-based cryptosystem is based on the problem of decoding linear codes with errors. Based on the error-correcting codes and the challenging task of decoding a message with random errors, the security of these cryptosystems is independent of the problem of prime factorization or use of discrete logarithms [16]. Code-based cryptography techniques are regarded as important in the era of quantum computing since these cryptosystems are trustworthy, while their difficulty originates from hard coding theory problems like learning parity with noise (LPN) and syndrome decoding (SN) [17]. In Round 3 of the NIST competition, one code-based cryptosystem reached the final stages whereas two more did so as alternative candidates [14]. These algorithms progressed as possible candidates for Round 4 evaluation [18]. These candidates' PKE/KEM algorithms are also listed in Table 2.

While CRYSTALS-Dilithium [13,19] has already been standardized in Round 3 for public key signatures, the code-based cryptosystems for PKE/KEM have been forwarded to Round 4. These include BIKE, Classic McEliece, and HQC. Hence, in this review, we explore the intricacies of some of these algorithms, namely, CRYSTALS-Dilithium, McEliece, BIKE, and HQC. The objective of this review is to make it easier for the user to read and understand these algorithms, to shed light on these algorithms from introductory as well as security standpoints, and to provide some critical observations from the NIST evaluation, with the purpose of helping developers and engineers make appropriate choices of algorithms.

Table 1. Impact of large-scale quantum computers on cryptographic algorithms. Table adapted from [20,21].

Algorithm	Type	Purpose	Pre-Quantum Security Level	Post-Quantum Security Level	Quantum Impact
AES-256	Symmetric	Encryption	256	128	Cracked by Grover's algorithm
SHA-256	Symmetric	Hash function	256	128	Cracked by Grover's algorithm
SHA-3	Symmetric	Hash function	256	128	Cracked by Grover's algorithm
RSA	Public key	Signatures, key establishment	128	Broken	Cracked by Shor's algorithm
ECDSA	Public key	Signatures	128	Broken	Cracked by Shor's algorithm
ECDH	Public key	Key exchange	128	Broken	Cracked by Shor's algorithm
DSA	Public key	Signatures	128	Broken	Cracked by Shor's algorithm

Table 2. NIST post-quantum cryptography standardization process (Round 4). Table source: [18].

	PKE/KEM	Signature
Candidates		
Code	3	0
Lattice	0	0

2. Available PQC

2.1. CRYSTALS-Dilithium

CRYSTALS-Dilithium is part of the cryptographic suite for the Algebraic Lattice digital signature schema. It operates on the hardness of the lattice problem over module lattices,

making it a candidate for post-quantum cryptography standards. The security concept is that an adversary having access to a signing oracle cannot produce a signature of a message whose signature they have not yet seen, nor produce a different signature of a message that they already saw signed [19]. It is a lattice-based digital signature based on the Fiat–Shamir paradigm [13,22,23]. In Dilithium, the public key can be viewed as a special kind of lattice-based sample constructed over a polynomial ring, where operations are performed modulo both a prime number q and a fixed polynomial. Conceptually, it consists of a matrix–vector relationship with two small secret vectors, sampled from a narrow, symmetric range around zero, which provides both efficiency and security [13,24]. Unlike other lattice-based cryptosystems, Dilithium uses a uniform distribution for generating error vectors, while most others rely on some form of truncated Gaussian distribution [13].

2.1.1. Preliminaries

Before discussing the algorithms implemented by CRYSTALS-Dilithium, we will first introduce a few important mathematical prerequisites.

Rings and Fields

A ring $(R, +, \cdot)$ is defined as a nonempty set R endowed with two binary operations, addition $+ : R \times R \to R$ and multiplication $\cdot : R \times R \to R$, satisfying the following properties. First, $(R, +)$ forms an abelian group, meaning that for all $a, b, c \in R$, the operation of addition is associative, satisfying $a + (b + c) = (a + b) + c$; there exists an additive identity $0 \in R$ such that $a + 0 = a$; each element $a \in R$ has an additive inverse $-a \in R$ such that $a + (-a) = 0$; and addition is commutative, so $a + b = b + a$. Second, multiplication is associative, ensuring that $a \cdot (b \cdot c) = (a \cdot b) \cdot c$ for all $a, b, c \in R$. Third, multiplication distributes over addition, satisfying $a \cdot (b + c) = a \cdot b + a \cdot c$ and $(a + b) \cdot c = a \cdot c + b \cdot c$ for all $a, b, c \in R$. Unless explicitly stated, rings are not assumed to be commutative with respect to multiplication nor to possess a multiplicative identity.

A ring R is classified as commutative if $a \cdot b = b \cdot a$ for all $a, b \in R$. It is unital, or a ring with unity, if there exists an element $1 \in R$, distinct from 0, such that $1 \cdot a = a \cdot 1 = a$ for all $a \in R$. A ring is an integral domain if it is commutative, unital, and contains no zero divisors, meaning that for all $a, b \in R$, if $a \cdot b = 0$, then either $a = 0$ or $b = 0$. A division ring is a unital ring in which every nonzero element $a \in R$ has a multiplicative inverse $a^{-1} \in R$ such that $a \cdot a^{-1} = a^{-1} \cdot a = 1$. A field is a commutative division ring, combining commutativity with the existence of multiplicative inverses for all nonzero elements. In a ring, multiplicative inverses are not necessarily required to exist. A nonzero commutative ring in which every nonzero element has a multiplicative inverse is called a field. Further mathematical definition of the field is also given in the McEliece Section 2.2.

Ideal of a Ring

An ideal of a ring is a distinguished subset that exhibits closure under specific algebraic operations. Formally, let R be a ring and I a subset of R. Then, I is an ideal of R if it satisfies the following conditions: for all $a, b \in I$, the elements $a + b$ and $a - b$ belong to I, ensuring closure under addition and subtraction; and for all $r \in R$ and $a \in I$, the products $r \cdot a$ and $a \cdot r$ are in I, guaranteeing absorption under multiplication by any ring element.

Quotient Ring

A quotient ring, also referred to as a factor ring, arises from partitioning a ring by an ideal and defining operations on the resulting cosets. Formally, let R be a ring and I an ideal of R. The quotient ring R/I is constructed as follows. The elements of R/I are the cosets of I in R, denoted $a + I$ for any $a \in R$, such that $R/I = \{a + I \mid a \in R\}$. The operations of addition and multiplication on R/I are defined by $(a + I) + (b + I) = (a + b) + I$ and

$(a + I) \cdot (b + I) = (a \cdot b) + I$, respectively, for all $a, b \in R$. These operations are well defined, as they are independent of the choice of coset representatives a and b for $a + I$ and $b + I$.

Ring of Polynomials over a Finite Field

The ring $\mathbb{Z}_q[x]$ denotes the ring of polynomials with coefficients in the finite field $\mathbb{Z}_q$, where q is a prime power and $\mathbb{Z}_q$ represents the finite field of order q. For a prime p, the finite field $\mathbb{Z}_q$ consists of residue classes modulo p, given by $\{0, 1, 2, \ldots, p-1\}$, with addition and multiplication defined modulo p.

The ring $\mathbb{Z}_q[x]$ comprises all polynomials of the form $f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$, where each coefficient $a_i \in \mathbb{Z}_q$ for $0 \leq i \leq n$. In this ring, the arithmetic operations of addition, subtraction, and multiplication are performed with coefficients reduced modulo q, ensuring that all resulting coefficients remain in $\mathbb{Z}_q$. This structure endows $\mathbb{Z}_q[x]$ with the properties of a ring, facilitating the study of polynomial algebra over finite fields.

Lattice in a Vector Space

A lattice in a vector space is a discrete subgroup that spans the space over the real numbers. Formally, let V be a real vector space, typically $\mathbb{R}^n$, and $\Lambda \subseteq V$ a subset. The set Λ is a lattice in V if it satisfies two conditions: first, Λ is an additive subgroup of V, meaning that for all $\mathbf{v}, \mathbf{w} \in \Lambda$, the sum $\mathbf{v} + \mathbf{w} \in \Lambda$; second, Λ spans V over $\mathbb{R}$, such that any vector $\mathbf{v} \in V$ can be expressed as a linear combination $\mathbf{v} = a_1 \mathbf{v}_1 + a_2 \mathbf{v}_2 + \cdots + a_n \mathbf{v}_n$, where $a_i \in \mathbb{R}$ and $\mathbf{v}_i \in \Lambda$ for $i = 1, 2, \ldots, n$.

Equivalently, a lattice L is a discrete set of points in $\mathbb{R}^n$ generated by a basis $\mathbf{B} = \{\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n\}$. The lattice induced by $\mathbf{B}$ is defined as $L(\mathbf{B}) = \{\sum_{i=1}^{n} z_i \mathbf{b}_i \mid z_i \in \mathbb{Z}\}$. Thus, a lattice forms a periodic, grid-like structure of vectors in the vector space, characterized by its regular arrangement of points spanning the entire space.

Modules and Module Lattices

A module extends the concept of a vector space by allowing scalars to be elements of an arbitrary ring rather than a field. Formally, let R be a ring, not necessarily commutative, and M an abelian group under addition. An R-module structure on M is defined by a scalar multiplication operation $\cdot : R \times M \to M$, mapping $(r, m) \mapsto r \cdot m$, which satisfies the following properties: for all $r, s \in R$ and $m, n \in M$, compatibility with ring multiplication holds, such that $(r \cdot s) \cdot m = r \cdot (s \cdot m)$, and distributivity is satisfied, such that $r \cdot (m + n) = r \cdot m + r \cdot n$ and $(r + s) \cdot m = r \cdot m + s \cdot m$. Thus, an R-module is an abelian group M equipped with an R-action generalizing the scalar multiplication of vector spaces. When R is a field, the module becomes a vector space over that field. The example of a two-dimensional module lattice is shown in Figure 1.

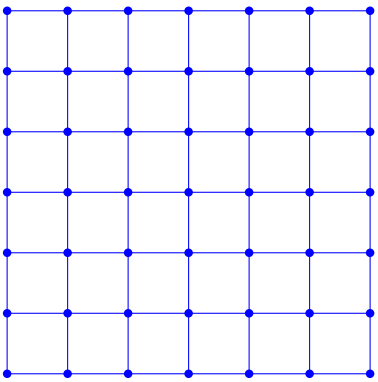


Figure 1. Example of a two-dimensional module lattice.

A module lattice is a submodule of an R-module that additionally possesses the structure of a lattice, integrating the algebraic properties of an R-module with the order-theoretic structure of a partially ordered set. Specifically, let M be an R-module and $\Lambda \subseteq M$ a submodule. The set Λ is a module lattice if it is an additive subgroup of M, discrete with respect to a suitable topology on M (when applicable, e.g., for $M \cong R^n$ with R a topological ring like $\mathbb{R}$ or $\mathbb{Z}$), and generates M as an R-module, such that $R\Lambda = M$. Additionally, Λ forms a lattice in the geometric sense, often represented as $\Lambda = \{\sum_{i=1}^{n} r_i \mathbf{b}_i \mid r_i \in R\}$ for a basis $\{\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n\}$ of Λ, when M is free of rank n. For instance, when $R = \mathbb{Z}$ and $M = \mathbb{Z}^n$, a module lattice is a free abelian group of rank n, forming a grid-like structure. This interplay of algebraic and geometric properties positions module lattices as a significant object of study, bridging module theory and lattice theory.

The Polynomial Ring $\mathbb{Z}[x]$ and the Quotient Ring $\mathbb{Z}[x]/(x^n + 1)$

The ring $\mathbb{Z}[x]$, consisting of all polynomials with integer coefficients, is equipped with the standard operations of polynomial addition and multiplication. This commutative ring with unity extends the integers $\mathbb{Z}$ by incorporating an indeterminate x, thereby serving as a fundamental structure in algebra with applications across various mathematical domains.

A key construction derived from $\mathbb{Z}[x]$ is the quotient ring $\mathbb{Z}[x]/(x^n + 1)$, formed by factoring out the principal ideal generated by the polynomial $x^n + 1$. Elements of $\mathbb{Z}[x]/(x^n + 1)$ are equivalence classes of polynomials in $\mathbb{Z}[x]$, where two polynomials $f(x), g(x) \in \mathbb{Z}[x]$ are equivalent if $f(x) \equiv g(x) \pmod{x^n + 1}$, meaning $f(x) - g(x) = (x^n + 1)h(x)$ for some $h(x) \in \mathbb{Z}[x]$. The equivalence class of a polynomial $f(x)$ is denoted $[f(x)]$.

Each element in $\mathbb{Z}[x]/(x^n + 1)$ has a unique representative polynomial of degree at most $n - 1$, expressible as $[a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}]$, where $a_i \in \mathbb{Z}$ for $0 \leq i \leq n - 1$. The defining relation $x^n = -1$ in the quotient ring allows higher powers of x to be reduced via $x^n \mapsto -1$. This algebraic structure, characterized by polynomial reduction modulo $x^n + 1$, renders $\mathbb{Z}[x]/(x^n + 1)$ particularly valuable in fields such as coding theory, lattice-based cryptography, and the arithmetic of cyclotomic fields.

2.1.2. Shortest Vector Problem (SVP) in Module Lattices

Let M be a module over a ring R, and let $\Lambda(M)$ denote the associated module lattice generated by a basis of M. The Shortest Vector Problem (SVP) in $\Lambda(M)$ is formulated as follows:

$$\text{Find } v \in \Lambda(M) \setminus \{0\} \text{ such that } \|v\| \leq \|w\| \quad \forall\, w \in \Lambda(M) \setminus \{0\}.$$

Here, $\|\cdot\|$ denotes a chosen norm on $\Lambda(M)$; the Euclidean norm is most commonly employed due to its favorable geometric properties and well-established role in lattice-based cryptography.

The SVP in module lattices (a simple illustration is given in Figure 2) generalizes the classical SVP in integer lattices by leveraging the additional algebraic structure of R-modules. This generalization is of significant interest in cryptographic constructions, as it enables more compact key sizes and efficient algorithms while preserving hardness guarantees under well-studied assumptions. The presumed intractability of approximating SVP within certain factors under the Euclidean norm forms the security foundation of numerous post-quantum cryptosystems, including those based on the Learning With Errors (LWE) and Ring-LWE problems. Consequently, precise formulations and complexity analyses of SVP in module lattices are critical both for theoretical investigations in computational number theory and for practical evaluations of cryptographic resilience against classical and quantum adversaries.

Although Dilithium does not directly use SVP or CVP, it relies on the hardness assumption of these problems indirectly. Dilithium is based on the hardness of Module Learning with Errors (MLWE) problems. MLWE, a variant of the Learning with Errors (LWE) problem adapted to module lattices, involves solving systems of linear equations with noise—a difficult task for both classical as well as quantum computers [24]. It involves solving a system of linear equations with noise, which is widely believed to be hard to solve efficiently for both classical as well as quantum computers.

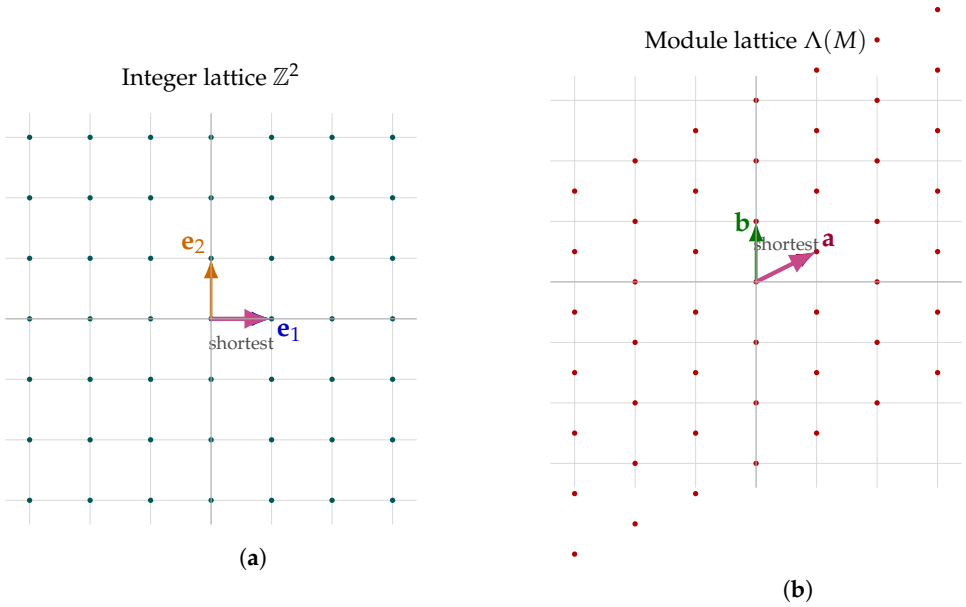


Figure 2. Illustration of the Shortest Vector Problem (SVP) in an integer lattice (**a**) and a sheared module lattice (**b**). Basis vectors use distinct colors; one shortest vector is highlighted. (**a**) Integer lattice with basis $\{\mathbf{e}_1, \mathbf{e}_2\}$. (**b**) Module lattice with basis $\{\mathbf{a}, \mathbf{b}\}$, $\mathbf{a} = (1, 0.5)$, $\mathbf{b} = (0, 1)$.

Since this process leverages MLWE, formally given a public matrix A, and a vector b,

$$b = A \times s + e \tag{1}$$

where e is an error vector, and the goal of MLWE is to solve s. The error vector e makes it difficult to solve for s.

2.1.3. NTT Domain Representation

Let q be a modulus admitting a primitive 512-th root of unity r mod q. In the cyclotomic ring $R_q = \mathbb{Z}_q[X]/(X^{256} + 1)$, the polynomial $X^{256} + 1$ factors as

$$X^{256} + 1 = \prod_{i \in \{1,3,\ldots,511\}} (X - r^i) \mod q,$$

which, by the Chinese Remainder Theorem [25], induces an isomorphism between R_q and a direct product of finite fields, allowing multiplication to be carried out component-wise. This structural property underpins the Number Theoretic Transform (NTT), a finite-field analog of the Fast Fourier Transform (FFT), which enables asymptotically fast polynomial multiplication. Following the classical Cooley–Tukey decomposition [26], the factorization

$$X^{256} + 1 = (X^{128} - r^{128})(X^{128} + r^{128})$$

is applied recursively until degree-one factors are reached, enabling efficient evaluation through butterfly operations. As in the FFT, the standard output order of the NTT is

bit-reversed. To fix a canonical ordering, we define the NTT domain representation of $a \in R_q$ as

$$\hat{a} = \mathrm{NTT}(a) = \big(a(r_0), a(-r_0), \ldots, a(r_{127}), a(-r_{127})\big),$$

where $r_i = r^{\mathrm{brv}(128+i)}$ and $\mathrm{brv}(\cdot)$ denotes the bit-reversal permutation of an 8-bit integer. Under this convention, multiplication in R_q is performed as

$$a \cdot b = \mathrm{NTT}^{-1}\big(\mathrm{NTT}(a) \odot \mathrm{NTT}(b)\big),$$

where $\odot$ denotes component-wise multiplication. For vectors or matrices over R_q, the NTT is applied element-wise, and the inverse transform maps the product back to coefficient form [24].

2.1.4. Components

The Dilithium suite contains sets of components that are responsible for generating the key, signature creation, and verification. Key generation handles the generation of public as well as private keys. Let S be a set, and $s \leftarrow S$ is a uniformly chosen random from S (in practical terms, S is a probability distribution).

A signature schema $SIG = (KeyGe, Sign, Verify)$ is a triplet of probabilistic polynomial time algorithms together with a message space M. The KeyGen algorithm generates two keys: public pk and private sk. The signing algorithm $Sign$ takes a secret key sk and a message $m \in M$ to generate a signature σ. Finally, the deterministic $Verify$ algorithm takes a public key pk, a message m, and a signature σ, and outputs binary 0 or 1; in other words, it accepts or rejects.

If we let R_q denote a ring [24],

$$R = \frac{Z[X]}{(X^n + 1)}, R_q = \frac{Z_q[X]}{(X^n + 1)}$$

We denote by

$$\rho : R \longrightarrow R_q$$

the natural reduction map sending each integer coefficient to its residue modulo q. We work with the cyclotomic polynomial $X^n + 1$ (with n a power of two) so that R has degree n and admits efficient Number-Theoretic Transform operations. Secrets and errors are drawn from the sparse ternary set

$$B_h = \big\{ f \in R : f \text{ has exactly } h \text{ nonzero coefficients, each } \pm 1 \big\},$$

whose cardinality is $|B_h| = 2^h \binom{n}{h}$. Finally, in key-generation and signing (or encryption), we sample $s, e \in B$, compute $t = \rho\big(a \cdot s + e\big)$, and proceed with the standard Dilithium-style routines.

The value of n originally used is 256 and q is the prime, $8380417 = 2^{23} - 2^{13} + 1$ [13,24].

For the CRYSTALS-Dilithium original version, we need a cryptographic hash function that hashes onto B_{60}, which has more than 2^{256} elements. The Algorithm 1 shows the process for generating random elements. The algorithm used is also called the "inside-out" version of the Fisher–Yates method.

Algorithm 1 Create a random 256-element array with 60 ± 1's and 196 0's [24]

Initialize $c = c_0 c_1 \dots c_{255} = 0 \dots 0$
for $i = 196$ to 255 **do**
    $j \leftarrow \{0, 1, \dots, i\}$
    $s \leftarrow \{0, 1\}$
    $c_i \leftarrow c_j$
    $c_j \leftarrow (-1)^s$
end for
return c

Expanding the Matrix **A**

The function ExpandMatrix shown in the Algorithm 2 maps a uniform seed $\rho \in \{0,1\}^{256}$ to a matrix $\mathbf{A} \in R_q^{k \times \ell}$ in NTT domain representation. For each entry (i, j), it uses ρ and the index $u = 256i + j$ as a domain separator to initialize either a `SHAKE-128` instance or an `AES256-CTR` stream, depending on the implementation variant. The output stream is interpreted as a sequence of 23-bit integers obtained by reading three consecutive bytes in little-endian order, masking the highest bit of the third byte to zero. This ensures each parsed integer lies in the range $[0, 2^{23} - 1]$. Integers greater than or equal to q are discarded via rejection sampling, and sampling continues until exactly n coefficients are obtained. Each polynomial $a_{i,j} \in R_q$ produced in this manner is then transformed to the NTT domain, yielding $\hat{a}_{i,j}$. This process is repeated for all (i, j) to obtain the full matrix $\hat{\mathbf{A}}$ in NTT form.

The choice of $q = 8380417$ in Dilithium satisfies $q < 2^{23}$, which makes this sampling efficient. The use of ρ and (i, j) as domain separators ensures that each $\hat{a}_{i,j}$ is generated independently and deterministically, enabling the matrix to be reconstructed from ρ without storing it explicitly. This greatly reduces the public key size while preserving the uniformity and independence of matrix entries, which is essential for the Module-LWE security of the scheme.

Algorithm 2 $\hat{\mathbf{A}} \leftarrow$ ExpandMatrix$(\rho, q, n, k, \ell, \mathsf{mode})$

Input: seed $\rho \in \{0,1\}^{256}$; modulus $q < 2^{23}$; degree n (e.g., 256); dimensions (k, ℓ); $\mathsf{mode} \in \{\texttt{SHAKE128}, \texttt{AES256CTR}\}$
Output: matrix $\hat{\mathbf{A}} \in R_q^{k \times \ell}$ in NTT domain
for $i = 0$ **to** $k - 1$ **do**
    for $j = 0$ **to** $\ell - 1$ **do**
        $u \leftarrow 256 \cdot i + j$ ▷ two-byte domain separator
        if $\mathsf{mode} = \texttt{SHAKE128}$ **then**
            $\mathsf{XOF} \leftarrow \texttt{SHAKE128.init}()$
            `absorb` the 32 bytes of ρ; `absorb` $\mathrm{LE16}(u)$
        else
            $\mathsf{XOF} \leftarrow \texttt{AES256CTR.init}(\text{key} = \rho,\ \text{nonce} = \mathrm{LE16}(u) \parallel \mathbf{0}_{10})$
        end if
        Initialize polynomial $a(X)$ with n coefficients (unset)
        $t \leftarrow 0$
        while $t < n$ **do**
            $(b_0, b_1, b_2) \leftarrow \mathsf{XOF.next3}()$ ▷ three consecutive bytes
            $b_2' \leftarrow b_2$ & 0x7F ▷ zero highest bit of every third byte
            $x \leftarrow b_0 + (b_1 \ll 8) + (b_2' \ll 16)$ ▷ little-endian 23-bit integer
            if $x < q$ **then**
                $a_t \leftarrow x; \quad t \leftarrow t + 1$
            end if
        end while
        $\hat{a}_{i,j} \leftarrow \texttt{NTT}(a)$ ▷ place in the scheme's reference NTT order
    end for
end for
return $\hat{\mathbf{A}}$

Module LWE

Let l be a positive integer. The hard problem underlying the security of the scheme is the Module-LWE problem. The Module-LWE is a distribution on $R_q^k \times R_q$ induced by pairs $(a; b;)$, where $a_i \leftarrow R_q^l$ is uniform and $b = a_i^T s + e_i$, with $s \leftarrow S_\eta^l$ common to all samples and $e_i \leftarrow S_n$ fresh for every sample.

Module LWE consists in recovering from polynomially many samples chosen from the Module-LWE distribution such that,

$$\Pr\left[\begin{array}{l} A \leftarrow \mathbf{R}_q^{k\times l}; (s,e) \leftarrow \mathbf{S}_q^k \times \mathbf{S}_p^l; \\ b \leftarrow As + e; x \leftarrow A(A,b); \end{array} \middle| x = s \right].$$

We say that (t, ϵ), the $Module - LWE_{k,l,\eta}$ hardness assumption holds if no algorithm A running in at most t has an advantage greater than ϵ.

Key Generation

The first step of the overall suite key generation process works as below,

The key generation described in Algorithm 3 procedure produces a public/secret key pair (pk, sk) based on the Module-LWE problem over the polynomial ring $R_q = \mathbb{Z}_q[X]/(X^n + 1)$. It begins by sampling two independent uniform bitstrings $(\mathtt{seed}_A, \mathtt{seed}_s) \in \{0,1\}^\lambda$ using a cryptographically secure pseudorandom number generator, where λ is the security parameter. The value $\mathtt{seed}_A$ is expanded via ExpandMatrix into a matrix $\mathbf{A} \in R_q^{k\times\ell}$, ensuring that $\mathbf{A}$ can be reconstructed by anyone possessing $\mathtt{seed}_A$, thus reducing the public key size.

Algorithm 3 KeyGen

Requires: Security parameter λ, modulus q, dimension parameters (k, ℓ, n), decomposition parameter d, hash function Hash, seed expansion function ExpandMatrix
Returns: Public key pk, secret key sk

1: $(\mathsf{seed}_A,\ \mathsf{seed}_s) \xleftarrow{\$} \{0,1\}^\lambda$ ▷ Uniform seeds for matrix and secret sampling
2: $A \leftarrow \mathsf{ExpandMatrix}(\mathsf{seed}_A)$ ▷ $A \in R_q^{k\times\ell}$
3: $(s_1, s_2) \xleftarrow{\$} B_\kappa \times B_\eta$ ▷ Sample $s_1 \in B_\kappa,\ s_2 \in B_\eta$ in R^ℓ and R^k
4: $t \leftarrow A \cdot s_1 + s_2$ ▷ $t \in R_q^k$
5: $(t_0, t_1) \leftarrow \mathsf{Power2Round}_q(t, d)$ ▷ Decompose t into low- and high-bits
6: $pk \leftarrow (\mathsf{seed}_A,\ t_1)$
7: $sk \leftarrow (\mathsf{seed}_A,\ \mathsf{seed}_s,\ s_1,\ s_2,\ t_0)$
8: **return** $(pk,\ sk)$

The secret key consists of two short vectors $\mathbf{s}_1 \in S_\eta^\ell$ and $\mathbf{s}_2 \in S_\eta^k$ sampled uniformly from a bounded discrete distribution S_η, where η is small (e.g., $\eta \in \{2, 4\}$ for recommended parameter sets). These distributions produce coefficients in $[-\eta, \eta]$, ensuring both efficiency and security against lattice-reduction attacks.

The vector $\mathbf{t} \in R_q^k$ is computed as

$$\mathbf{t} \ = \ \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2 \pmod q,$$

which is essentially a Module-LWE sample with secret $(\mathbf{s}_1, \mathbf{s}_2)$. To enable compression and reduce public key size, $\mathbf{t}$ is decomposed into a high-order part $\mathbf{t}_1$ and a low-order part $\mathbf{t}_0$ using the Power2Round function:

$$(\mathbf{t}_0, \mathbf{t}_1) \ = \ \mathsf{Power2Round}_q(\mathbf{t}, d),$$

where d controls the number of bits truncated from each coefficient. The high-order bits $\mathbf{t}_1$ are included in the public key, while $\mathbf{t}_0$ is stored in the secret key to allow lossless reconstruction of $\mathbf{t}$ during verification. The final keys are:

$$pk = (\texttt{seed}_A, \mathbf{t}_1), \quad sk = (\texttt{seed}_A, \texttt{seed}_s, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0).$$

The use of seeds instead of explicit matrices and randomness vectors significantly reduces storage requirements while maintaining reproducibility and security [24].

Signature Creation

The signature algorithm (also described in Algorithm 4) progresses by having the signer generate a vector y with entries from S_{y-1}, using the extendable output function Sam seeded by r. The vector w is computed as $2\gamma_2 \cdot w_1 + w_0$, where w_0 lies within $[-\gamma_2, \gamma_2]$. The signer then calculates $c = H(\rho, t_1, w_1, \mu)$ and subsequently derives $z = y + cs$. The algorithm restarts if any z component is at least $\gamma_1 - \beta$, or if any coefficient of $r_0 = \text{LowBits}_q(w - cs_2, 2\gamma_2)$ exceeds $\gamma_2 - \beta$. This is to ensure the security of the signing process and to prevent any information leakage regarding s_1, s_2. The verification $r_1 = w_1$ is crucial for the correctness of the protocol. It is noted that if $||cs_2||_\infty < \beta$, then $||r_0||_\infty < \gamma_2 - \beta$, and thus $r_1 = w_1$. The probability of $||cs_2||_\infty$ being below β is designed to be high, ensuring the protocol's security by making the chances of violation negligible. Nevertheless, this check is included to raise the probability of a verifier accepting a genuine signature to unity.

Algorithm 4 Sign$(sk = (\rho, s_1, s_2, t), \mu \in \mathcal{M})$

1: $A \leftarrow \mathbf{R}_q^{k \times l}$; $Sam(\rho)$
2: $t_1 \leftarrow \text{Power2Round}_q(t, d)$
3: $t_0 \leftarrow t - t_1 \cdot 2^d$
4: $r \leftarrow \{0,1\}^{256}$
5: $y \leftarrow \mathbf{S}_q^l\ Sam(r)$
6: $w \leftarrow Ay$
7: $w_1 \leftarrow \text{HighBits}_q(w, 2\gamma_2)$
8: $c \leftarrow H(\rho, t_1, w_1, \mu)$
9: $z \leftarrow y + cs_1$
10: $(r_1, r_0) \leftarrow \text{Decompose}_q(w - cs_2, 2\gamma_2)$
11: **if** $||z||_\infty > \gamma_1 - \beta$ **or** $||r_0||_\infty > \gamma_2 - \beta$ **or** $r_1 \neq w_1$ **then**
12: **goto** 4
13: **end if**
14: $h \leftarrow \text{MakeHint}_q(-ct_0, w - cs_2 + ct_0, 2\gamma_2)$
15: **if** $||ct||_\infty > \gamma_2$ **or** the number of 1's in h is greater than α **then**
16: **goto** 4
17: **end if**
18: **return** $\sigma \leftarrow (z, h, c)$

Provided all conditions are met and a restart is deemed unnecessary, one can assert that $\text{HighBits}_q(Az - ct, 2\gamma_2) = w_1$. If the verifier had access to the complete t and (z, c), w_1 could be independently ascertained after confirming that $||z||_\infty < \gamma_1 - \beta$ and $c = H(\rho, t_1, w_1, \mu)$. Yet, to reduce the size of the public key, the verifier is only acquainted with t_1. Consequently, the signer must provide a "hint" to enable the verifier to deduce $\text{HighBits}_q(Az - ct, 2\gamma_2)$. This is the essence of Step 12. During Step 13, the verifier conducts a series of checks that infrequently result in failure (under 1% chance), which negligibly impacts the overall execution duration. Notably, Step 12's primary function is to compress and it does not compromise the security of the scheme, assuming the verifier has full knowledge of t.

This assumption potentially renders the actual scheme even more robust in practical applications [24].

Verification

Given a public key $pk = (p, t_1)$, message $\mu \in \mathcal{M}$, and signature $\sigma = (z, h, c)$, the verifier reconstructs the NTT-domain matrix $\mathbf{A} \leftarrow \mathsf{Sam}(\rho)$ from the seed ρ in the public key. Using the hint h, the function $\mathsf{UseHint}_q$ recovers the high-order bits w_1 from $\mathbf{A}z - ct_1$ modulo q, where $c \in \mathbb{Z}_q$ is the challenge polynomial. The hint encodes which coefficients require rounding adjustments so that the verifier can deterministically recover the same high-order bits as computed during signing without transmitting the full w_1 vector. This significantly reduces the signature size while preserving correctness. The Algorithm 5 shows the verfication process.

Algorithm 5 Verify$(pk = (p, t_1), \mu \in \mathcal{M}, \sigma = (z, h, c))$

$A \leftarrow Sam(\rho)$
$w_1 \leftarrow \text{UseHint}_q(h, Az - ct_1, 2'd, 2\gamma_2)$
if $c = H(\rho, t_1, w_1, \mu)$ **and** $||z||_\infty < \gamma_1 - \beta$ **and** the number of 1's in h is $\leq \omega$ **then**
    return 1
else
    return 0
end if

Next, the verifier recomputes the challenge polynomial

$$c' \leftarrow H(\rho, t_1, w_1, \mu)$$

and checks equality $c' = c$. This step binds the signature to both the message and the public key, preventing forgery by ensuring that w_1 corresponds to the actual $\mathbf{A}z - ct_1$ value for the claimed signer.

A crucial norm bound check $\|z\|_\infty < \gamma_1 - \beta$ ensures that the signing vector z remains within a narrow range, mitigating leakage of the secret key via lattice-reduction attacks. Additionally, the verifier enforces that the Hamming weight of h does not exceed ω, where ω is derived from the masking strategy used in the signing process. This bound limits the number of coefficients in w_1 that could have been adjusted, constraining an adversary's ability to manipulate hints. Due to the structure of $\mathbf{A}$ and parameter selection in Dilithium, the probability of verification failure when a valid signature is presented is negligible, dominated by the rare carry-propagation events involving ct_0 coefficients. For the parameter sets recommended in [24], the carry count exceeds ω with probability below 1%, giving an empirical verification success rate above 99%.

2.1.5. Efficiency Trade-Offs

To optimize key size and efficiency, Dilithium's signing and verification procedures reconstruct the matrix $\mathbf{A}$ (or its NTT-domain version $\hat{\mathbf{A}}$) from a short seed ρ [24]. When memory constraints are relaxed, $\hat{\mathbf{A}}$ can be precomputed and stored as part of the public or secret key, along with precomputed NTT forms of $\mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0$ to accelerate signing. Conversely, for minimal secret key size, only a compact seed ζ is retained, from which all necessary randomness for $\rho, K, \mathbf{s}_1, \mathbf{s}_2$ is generated. Memory use during computations can also be reduced by retaining only the NTT components currently in use. Furthermore, the scheme supports both deterministic and randomized signatures: in the deterministic variant, the signing randomness seed ρ' is derived from the message and key, producing identical signatures for the same message; in the randomized variant, ρ' is chosen uniformly at random. While

deterministic signing avoids randomness costs, randomized signing may be preferable when mitigating side-channel attacks or concealing the exact message being signed.

2.1.6. Parameter Sets for CRYSTALS-Dilithium

Tables 3–5 summarize the output sizes, core parameters, and estimated hardness levels for CRYSTALS-Dilithium across multiple security targets. The first two tables correspond to the parameter sets proposed for NIST security levels 2, 3, and 5, which align with ≈128, ≈192, and ≈256 bits of post-quantum security, respectively, under the Module-LWE and Module-SIS hardness assumptions. The third table extends this to include "challenge" parameter sets (1− − , 1−, 5+, 5++), designed to explore the margins of the scheme's security.

Table 3. Output sizes and security for Dilithium at various NIST security levels. Table source: [24].

	NIST Security Level		
	2	**3**	**5**
Output Size			
Public key size (bytes)	1312	1952	2592
Signature size (bytes)	2420	3293	4595
LWE Hardness (Core-SVP and refined)			
BKZ block-size b (GSA)	423	624	863
Classical Core-SVP	123	182	252
Quantum Core-SVP	112	165	229
BKZ block-size b (simulation)	433	638	883
$\log_2$ Classical Gates	159	217	285
$\log_2$ Classical Memory	98	139	187
SIS Hardness (Core-SVP)			
BKZ block-size b	423 (417)	638 (602)	909 (868)
Classical Core-SVP	123 (121)	186 (176)	265 (253)
Quantum Core-SVP	112 (110)	169 (159)	241 (230)

Table 4. Scheme parameters for Dilithium at NIST security levels 2, 3, and 5. Table source: [24].

Parameter	**Level 2**	**Level 3**	**Level 5**
q [modulus]	8380417	8380417	8380417
d [dropped bits from t]	13	13	13
τ [# of ± 1's in c]	39	49	60
Challenge entropy [bits]	192	225	257
γ_1 [y coefficient bound]	2^{17}	2^{19}	2^{19}
γ_2 [low-order rounding range]	$(q-1)/88$	$(q-1)/32$	$(q-1)/32$
(k, ℓ) [matrix dimensions]	$(4, 4)$	$(6, 5)$	$(8, 7)$
η [secret key range]	2	4	2
$\beta = \tau \cdot \eta$	78	196	120
ω [max. # of 1's in h]	80	55	75
Repetitions	4.25	5.1	3.85

Parameter Details

- q: The modulus used for all polynomial coefficient arithmetic. For Dilithium, $q = 8380417$ is a 23-bit prime supporting efficient NTTs.
- d: The number of least significant bits dropped from each coefficient of $\mathbf{t}$ during Power2Round, used to compress the public key.
- τ: The Hamming weight of the challenge polynomial c, i.e., the number of coefficients equal to ± 1.
- Challenge entropy: $\log \binom{n}{\tau} + \tau$ bits; reflects the search space for valid challenges.

- γ_1: Bound on the infinity norm of the vector $\mathbf{y}$ in signing; higher values increase rejection probability but enlarge the range of signatures.
- γ_2: Range parameter for the HighBits/LowBits decomposition; impacts correctness and compression efficiency.
- (k, ℓ): Dimensions of the public matrix $\mathbf{A}$ in $R_q^{k \times \ell}$; jointly determine the Module-LWE dimension.
- η: Bound for secret key coefficients; small η yields more efficient signing but affects hardness.
- β: Verification bound parameter, typically $\beta = \tau \cdot \eta$.
- ω: Maximum allowed number of 1's in the hint vector h; constrains information leakage.
- Repetitions: Expected number of signing attempts (restarts) due to bound checks.

Table 5. Extended challenge parameter sets for Dilithium. These cover lower-than-NIST ($1--$, $1-$) and higher-than-NIST (5+, 5++) levels. Table source: [24].

Parameter	**1−−**	**1−**	**5+**	**5++**
q [modulus]	8380417	8380417	8380417	8380417
d [dropped bits from t]	10	13	13	13
Weight of c [± 1's]	24	30	60	60
Challenge entropy [bits]	135	160	257	257
γ_1 [y coefficient bound]	2^{17}	2^{17}	2^{19}	2^{19}
γ_2 [low-order rounding range]	$(q-1)/128$	$(q-1)/128$	$(q-1)/32$	$(q-1)/32$
(k, ℓ) [matrix dimensions]	$(2, 2)$	$(3, 3)$	$(9, 8)$	$(10, 9)$
η [secret key range]	6	3	2	2
$\beta = \tau \cdot \eta$	144	90	120	120
ω [max. # of 1's in h]	10	80	85	90
Public key size (bytes)	864	992	2912	3232
Signature size (bytes)	1196	1843	5246	5892
Expected repetitions (Equation (5))	5.2	4.87	4.59	5.48
SIS Hardness (Core-SVP)				
BKZ block-size b	190 (165)	305 (305)	1055 (1005)	1200 (1145)
Core-SVP Classical	55 (49)	89 (89)	308 (293)	360 (334)
LWE Hardness (Core-SVP)				
BKZ block-size b	200	305	1020	1175
Core-SVP Classical	58	89	298	343

For the extended challenge parameter sets in Table 5, the "1−−" and "1−" levels correspond to parameter choices below the NIST level 1 target. The 1− set targets significantly reduced security (less than 60 bits of Core-SVP hardness) to act as a "canary" for improvements in lattice cryptanalysis: if such parameters can be broken in practice, it signals a need to reassess all higher levels. The 1− set retains roughly 90 bits of Core-SVP hardness and a BKZ block-size of ≈300, serving as a conservative low-end benchmark.

The "5+" and "5++" sets represent *above* NIST level 5 security, anticipating moderate advances in lattice algorithms. The 5+ set achieves slightly more than NIST level 5 hardness, while the 5++ set has roughly twice the Core-SVP security of level 3 (and well above level 5), making it a forward-looking choice if future cryptanalytic results begin to threaten current level 3 or level 5 parameters.

These extended sets help bound the safe operating range for Dilithium [24]. On the low-security side, they offer an early warning indicator for practical breaks; on the high-security side, they show how the parameters would scale to preserve security in the face of improved attacks. Because the underlying lattice hardness dominates the overall security, these adjustments are made without altering the security of symmetric primitives, such as the hash functions or PRFs used internally.

2.1.7. Known Attacks and Security

The security of Dilithium depends on the standard hard problems: Module Learning with Errors (MLWE), Module Short Integer Solution (MSIS), and Self Target MSIS. The standard definition of these three problems has been taken from [27]. This means that it suffices to show that the secret key can not be forged using the public key [13]. Under another assumption, a variant of the Module-SIS, called SelfTargetMSIS [27], Dilithium is strongly unforgeable [13,24]. The most well-known attacks against Dilithium, which do not take advantage of side-channels, are similar to other lattice-based approaches in that they involve using general algorithms to locate short vectors within lattices. For Dilithium, the core SVP security is roughly 124, 186, and 265 for NIST levels 2, 3, and 5, respectively [13]. Ref. [28] shows the single-trace side-channel attacks by exploiting some information leakage in the secret key. The authors of [29] introduce two novel attacks on randomized/hedged Dilithium that include key recovery. The concept of fixing incorrect signatures after they have been signed underlies both attacks. The value of a secret intermediate carrying key information is obtained upon successful rectification. Once a large number of incorrect signatures and their matching rectification values have been gathered, the signing key can be found using either basic linear algebra or lattice-reduction methods. The common form of attacks on Dilithium are as follows: (i) Side-Channel attack which focuses on the strategy to recover the secret key by exploiting the underlying polynomial multiplication [27,29–31]; (ii) Fault Attack which makes the deterministic version of Dilithium vulnerable. In this type of attack, an adversary can inject a single random fault during the signature generation to a message m, then let the m be signed again without any breach. By doing this a fault-induced non-reuse scenario is achieved, making the key recovery trivial using a single faulty signature [29,32]. And [33] extended this attack to the randomized version of Dilithum as well. Ref. [29] introduced another variant of the skipping fault attack introduced by [34]. Readers are suggested to read [24,27,28,30] for a more comprehensive security analysis.

Definition 1 (**Module Learning with Errors (MLWE)**). *Let $m, k, \eta \in \mathbb{N}$. The advantage of an algorithm $\mathcal{A}$ for solving $MLWE_{m,k,\eta}$ is defined as:*

$$\begin{aligned} Adv_{MLWE_{m,k,\eta}}(\mathcal{A}) := \Big| & \Pr\Big[b = 0 \mid A \leftarrow R_q^{m \times k}, t \leftarrow R_q^m, b \leftarrow \mathcal{A}(A, t)\Big] \\ & - \Pr\Big[b = 0 \mid A \leftarrow R_q^{m \times k}, (s_1, s_2) \leftarrow S_\eta^k \times S_\eta^m, t := As_1 + s_2, b \leftarrow \mathcal{A}(A, t)\Big]\Big|. \end{aligned} \tag{2}$$

Here, the notation $\mathcal{A}(x)$ denotes $\mathcal{A}$ as a function of x. We note that the MLWE problem is often phrased in other contexts with the short vectors s_1 and s_2 coming from a Gaussian, rather than a uniform, distribution. The use of a uniform distribution is one of the particular features of CRYSTALS-Dilithium [27].

The second problem, MSIS, is concerned with finding short solutions to randomly chosen linear systems over R_q.

Definition 2 (**Module Short Integer Solution (MSIS)**). *Let $m, k, \gamma \in \mathbb{N}$. The advantage of an algorithm $\mathcal{A}$ for solving $MSIS_{m,k,\gamma}$ is defined as:*

$$Adv_{MSIS_{m,k,\gamma}}(\mathcal{A}) := \Pr\Big[[I_m \mid A] \cdot y = 0 \wedge 0 < \|y\|_\infty \leq \gamma \mid A \leftarrow R_q^{m \times k}, y \leftarrow \mathcal{A}(A)\Big]. \tag{3}$$

The third problem is a more complex variant of MSIS that incorporates a hash function H.

Definition 3 (SelfTargetMSIS). *Let $\tau, m, k, \gamma \in \mathbb{N}$ and $H : \{0,1\}^* \to B_\tau$, where $B_\tau \subseteq R_q$ is the set of polynomials with exactly τ coefficients in $\{-1, 1\}$ and all remaining coefficients are zero. The advantage of an algorithm $\mathcal{A}$ for solving $SelfTargetMSIS_{H,\tau,m,k,\gamma}$ is defined as:*

$$Adv_{SelfTargetMSIS_{H,\tau,m,k,\gamma}}(\mathcal{A}) := \Pr\left[H([I_m \mid A] \cdot y \mid M) = y_{m+k} \wedge \|y\|_\infty \leq \gamma \mid A \leftarrow R_q^{m \times k}, (y, M) \leftarrow A^H(\mathcal{A})\right]. \quad (4)$$

From [27,35],

$$\mathrm{Adv}^{\mathrm{Dilithium}}_{\mathrm{sEUF\text{-}CMA}}(\mathcal{A}) \leq \mathrm{Adv}_{\mathrm{MLWE}_{k,l,\eta}}(B) + \mathrm{Adv}_{\mathrm{SelfTargetMSIS}_{H,\tau,k,l+1,\zeta}}(C) + \mathrm{Adv}_{\mathrm{MSIS}_{k,l,\zeta}}(D), \quad (5)$$

where all terms on the right-hand side of the inequality depend on parameters that specify Dilithium, and sEUF-CMA stands for strong unforgeability under chosen message attacks. The interpretation of Equation (5) is: if there exists a quantum algorithm $\mathcal{A}$ that attacks the sEUF-CMA-security of Dilithium, then there exist quantum algorithms B, C, D for MLWE, SelfTargetMSIS, and MSIS that have advantages satisfying the inequality comparable to $\mathcal{A}$. Equation (5) implies that breaking the sEUF-CMA security of Dilithium is at least as hard as solving one of the MLWE, MSIS, or SelfTargetMSIS problems. MLWE and MSIS are known to be no harder than LWE and SIS, respectively [27,35].

2.1.8. Evolution of Dilithium on NIST Submission

Based on the submission details provided by CRYSTALS-Dilithium [24] and the NIST submission report [13], we note the major changes and updates that occurred in the core algorithms and related sub-processes in Dilithium suites in the Figure 3.

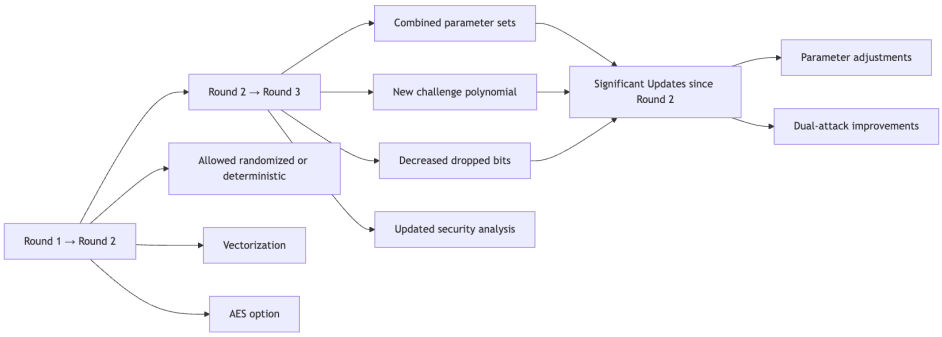


Figure 3. Evolution of Dilithium over multiple rounds.

Important Updates (Round 1 → Round 2)

- The primary conceptual design change was to allow the scheme to be either randomized or deterministic.
- For the deterministic and randomized version, the function ExpandMask uses a 48-byte seed instead of a key and the message.
- The nodes of various expansion functions for matrix A, masking vector y, and secret vectors s_1, s_2 are all 16-bit integers.
- Implementation changes included vectorization and a simpler assembler NTT implementation using macros.
- An optimized version using AES instead of SHAKE was introduced.

Important Updates (Round 2 → Round 3)

- Combined the first two parameter sets into one NIST level 2 set and introduced a level 5 parameter set.
- Now outputs challenges with 39 and 49 coefficients for security levels 2 and 3, respectively, instead of always outputting with 60 nonzero coefficients.

- Decreased the number of dropped bits from the public key from 14 to 13, increasing SIS hardness.
- The masking polynomial is now sampled from a range with power-of-2 possibilities.
- Changed the challenge sampling and transmission method to use a 32-byte challenge seed.
- Updated concrete security analysis based on recent progress and lattice algorithms.

Significant Updates Since Round 2

- Changes and parameter set adjustments to better match NIST security levels.
- Improvements to the dual attack proposed during the third round suggested lower estimated security in the RAM model than claimed, indicating two of the three-parameter sets fell slightly below the claimed security levels when memory access costs are considered.

2.2. McEliece Cryptosystem

In 1978, in the early history of public key cryptography, McEliece proposed using a generator matrix as a public key, and encrypting a code word (an element of code) by adding a specified number of errors to it [16]. The McEliece cryptosystem [36] is based on the concept of error correcting codes. The error-correcting codes are designed to detect and correct errors in transmitted messages. These codes can be in the form of vectors. The general approach is to add errors during the encoding and also add some extra information to the message, such that during decoding, some of the errors may be corrected using the supplied extra information. For example, let us assume we have a message ciphertext c, and we add errors e, which makes the final message $M = c + e$. Then, when the person on the receiving end receives this message, he/she has to work his/her way to recover from the error and reconstruct the original message that was intended to be received.

Right now, the McEliece cryptosystem is being considered as one of the candidates for the Round 4 standardization for KEM [18]. To begin to understand this code-based cryptosystem, we need to describe some of the core mathematical concepts.

Fields and Their Extensions

A field is a set F equipped with two binary operations, addition and multiplication, denoted $+ : F \times F \to F$ and $\cdot : F \times F \to F$, satisfying the following properties for all $a, b, c \in F$. First, addition and multiplication are associative, such that $(a + b) + c = a + (b + c)$ and $(a \cdot b) \cdot c = a \cdot (b \cdot c)$. Second, both operations are commutative, so $a + b = b + a$ and $a \cdot b = b \cdot a$. Third, there exist distinct identity elements: an additive identity $0 \in F$ such that $a + 0 = a$, and a multiplicative identity $1 \in F$ such that $a \cdot 1 = a$ and $1 \neq 0$. Fourth, for each $a \in F$, there exists an additive inverse $-a \in F$ such that $a + (-a) = 0$, and for each $a \in F$ with $a \neq 0$, there exists a multiplicative inverse $a^{-1} \in F$ such that $a \cdot a^{-1} = 1$. Finally, multiplication distributes over addition, so $a \cdot (b + c) = a \cdot b + a \cdot c$.

A finite field $\mathbb{F}_q$, also known as a Galois field and denoted $GF(q)$, is a field with q elements, where $q = p^n$ is a prime power for some prime p and positive integer n. The prime p is the characteristic of the field.

An extension of a finite field $GF(p)$ is a Galois field $GF(p^k)$ of order $q = p^k$, where p is a prime and k is a positive integer. The field $GF(p^k)$ is a field extension of degree k over $GF(p)$, constructed as a vector space of dimension k over $GF(p)$ with operations defined modulo an irreducible polynomial of degree k in $GF(p)[x]$. This structure underpins many applications in algebra, number theory, and related fields.

Hamming Distance and Weight

The Hamming distance between two binary strings c_i and c_j of equal length n is defined as the number of positions at which their corresponding bits differ. Formally, for binary strings $c_i, c_j \in \{0,1\}^n$, the Hamming distance $d(c_i, c_j)$ is given by

$$d(c_i, c_j) = \sum_{k=1}^{n} (c_i[k] \oplus c_j[k]),$$

where $\oplus$ denotes the exclusive OR operation, and $c_i[k], c_j[k] \in \{0,1\}$ represent the k-th bits of c_i and c_j, respectively.

For an error-correcting code $C \subseteq \{0,1\}^n$, the minimum Hamming distance of C, denoted $d(C)$, is the smallest Hamming distance between any pair of distinct codewords in C. It is defined as

$$d(C) = \min_{\substack{c_i, c_j \in C \\ c_i \neq c_j}} d(c_i, c_j).$$

This quantity determines the error-correcting capability of the code, as a larger $d(C)$ enables the detection and correction of more errors.

The Hamming weight of a codeword $c_i \in \{0,1\}^n$, denoted $\mathrm{wt}(c_i)$, is the number of nonzero entries (i.e., ones) in c_i. Mathematically,

$$\mathrm{wt}(c_i) = \sum_{k=1}^{n} c_i[k].$$

To optimize error correction, codewords in C are designed to be maximally separated in terms of the Hamming distance. This is achieved by ensuring that legal codewords, those belonging to C and adhering to its structure, are sufficiently distinct from one another. Illegal codewords, which lie outside C and do not conform to its structure, are distributed to maximize the separation, thereby enhancing the minimum Hamming distance $d(C)$. This construction underpins the robustness of error-correcting codes in applications such as coding theory and data transmission.

Linear Codes

A linear code of dimension k and length n over a field F is a k-dimensional subspace of the vector space F^n, which consists of all n-dimensional vectors over F. This code is referred to as an $[n, k]$ code. If the minimum Hamming distance of the code is d, then the code is called an $[n, k, d]$ code. A linear code C of dimension k and length n over a field F is a k-dimensional subspace of the vector space F^n. The elements of C are called codewords. The parameters of the code are defined as follows:

- Length n: The number of components in each codeword.
- Dimension k: The number of linearly independent codewords in the code.
- Minimum Hamming Distance d: The smallest Hamming distance between any two distinct codewords in the code.

The code C can be described as an $[n, k]$ code. If the minimum Hamming distance is d, the code is described as an $[n, k, d]$ code. Since a linear code is a vector space, each codeword within it can be expressed as a linear combination of basis vectors. This means that knowing a basis for the linear code is essential for describing all the codewords explicitly [16].

Goppa Codes

A Goppa code is a linear, error-correcting code for message encryption and decryption. Let a Goppa polynomial be defined as a polynomial over $GF(p^m)$, that is,

$$g(x) = g_0 + g_1 x + \cdots + g_t x_t = \sum_{i=0}^{t} g_i x_i$$

with each $g_i \in GF(p^m)$. Let L be a finite subset of extension field $GF(p^m)$, p being a prime number, say

$$L = \{\alpha_1, \ldots, \alpha_n\} \subset GF(p^m)$$

such that $g(\alpha_i) \neq 0, \forall \alpha_i \in L$.

Now, given a codeword vector $c = (c_1, \ldots, c_n)$ over $GF(q)$, we have a function,

$$R_c(z) = \sum_{i=1}^{n} \frac{c_i}{x - \alpha_i}$$

where $\frac{1}{x-\alpha_i}$ is the unique polynomial with $(x - \alpha_i) \times \frac{1}{x-\alpha_i} 1 (\mathrm{mod} g(x))$ with a degree less than or equal to $t - 1$. Then, a Goppa code $\Gamma(L, g(x))$ is made up of all code vectors C such that $R_C(x) = 0 (\mathrm{mod} g(x))$. This means that the polynomial $g(x)$ divides $R_c(x)$.

A generator matrix G for a Goppa code $\Gamma(L, G(x))$ is a $k \times n$ matrix whose rows form a basis for the Goppa code. Here, k is the dimension of the code and n is the length of the code.

Given a Goppa code $\Gamma(L, G(x))$ with $L = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ and a Goppa polynomial $G(x)$, the generator matrix G is defined as follows:

$$G = \begin{bmatrix} \mathbf{g}_1 \\ \mathbf{g}_2 \\ \vdots \\ \mathbf{g}_k \end{bmatrix}$$

where each row $\mathbf{g}_i$ (for $i = 1, 2, \ldots, k$) is a basis of the Goppa code $\Gamma(L, G(x))$.

Encoding and Decoding Using Goppa Codes

To encode a message using Goppa codes, the message is multiplied by the generator matrix of the Goppa code, G. Specifically, the message is written as a block of k symbols. Each block is then multiplied by the generator matrix G, resulting in a set of codewords C.

1. Generator Matrix: Let G be the $k \times n$ generator matrix for the Goppa code $\Gamma(L, G(x))$:

$$G = \begin{bmatrix} g_{11} & g_{12} & \cdots & g_{1n} \\ g_{21} & g_{22} & \cdots & g_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ g_{k1} & g_{k2} & \cdots & g_{kn} \end{bmatrix}$$

2. Message Block: Let $\mathbf{m}$ be a message block of k symbols:

$$\mathbf{m} = \begin{bmatrix} m_1 & m_2 & \cdots & m_k \end{bmatrix}$$

3. Encoded Codeword: The encoded codeword $\mathbf{c}$ is obtained by multiplying the message block $\mathbf{m}$ by the generator matrix G:

$$\mathbf{c} = \mathbf{m} \cdot G$$

4. Resulting Codeword: The resulting codeword $\mathbf{c}$ is a row vector of length n:

$$\mathbf{c} = \begin{bmatrix} c_1 & c_2 & \cdots & c_n \end{bmatrix}$$

Decoding is performed by correcting the code using Patterson's Algorithm [37] (also presented in Algorithm 6, which essentially involves solving the system of n equations with

k unknowns. For the message vector $r = (r_1, r_2, \ldots, r_n) = (c_1, c_2, \ldots, c_n) + (e_1 + e_2 + \ldots e_n)$, where $e_i \neq 0$ exactly at r places. The process starts by locating the position of the error, $\{B = \{i : 1 \leq i \leq n\}$ and $e_i \neq 0\}$. After locating the position of the error, we then retrieve the value of the error $e_i : i \in B$. To achieve this, we need two polynomials: a locator polynomial $\sigma(x)$ and an error evaluator polynomial $\omega(z)$. These polynomials have j and $j-1$ degree, respectively. We need to compute syndrome $S(r) = \sum_{i=1}^{n} \frac{r_i}{x-\alpha_i}$, solve the equation $\sigma(x)S(r) \equiv \omega(x)(\mod g(z))$, then determine the error location $\{B = \{i : 1 \leq i \leq n\}$ and $\sigma(\alpha_i) \neq 0\}$, then compute the error value using the position, and finally, the codeword sent is calculated by $c_i = r_i - e_i$ or in vector form, $c = r - e$.

Algorithm 6 Patterson's Algorithm for decoding Goppa codes

1: **Input**: Received word $\mathbf{r} \in \mathbb{F}_q^n$
2: **Output**: Decoded codeword $\mathbf{c} \in \mathbb{F}_q^n$
3: **Initialization**: Let $\mathbf{r} = (r_1, r_2, \ldots, r_n)$
4: **Compute the Syndrome**
5: Compute the syndrome $\mathbf{s}(\mathbf{r})$:

$$\mathbf{s}(\mathbf{r}) = \sum_{i=1}^{n} \frac{r_i}{x - \alpha_i}$$

6: **Forming the Key Equation**
7: Construct the syndrome polynomial $S(x)$ from $\mathbf{s}$
8: Form the key equation:

$$\sigma(x)S(r) \equiv \omega(x) \pmod{G(x)}$$

9: **Solving the Key Equation**
10: Use the Euclidean algorithm to solve for the error locator polynomial $\sigma(x)$ and the error evaluator polynomial $\omega(x)$
11: **Finding Error Locations**
12: Determine the error locations by finding the roots of $\sigma(x)$:

$$\sigma(x) = \prod_{i \in \mathcal{B}} (x - \alpha_i)$$

where $\mathcal{B}$ is the set of error positions
13: **Step 5: Correcting Errors**
14: Correct the errors at the identified positions in $\mathbf{r}$ to obtain the decoded codeword $\mathbf{c}$
15: **return c**

2.3. Classic McEliece KEM Algorithms

The Classic McEliece Key encapsulation mechanism presented in the NIST submission [38] consists of algorithms as follows: irreducible-polynomial generation, field-ordering algorithm, key generation, fixed-weight vector generation, and finally, encapsulation and decapsulation. These algorithms are presented below:

Encoding Subroutine (`Encode`)

The `Encode` function (also given in the Algorithm 7) maps a weight-t error vector $e \in \mathbb{F}_2^n$ to a corresponding codeword $C \in \mathbb{F}_2^{mt}$ using the public key T, an $mt \times k$ matrix over $\mathbb{F}_2$. The algorithm constructs the systematic form of the parity-check matrix $H = (I_{mt} \mid T)$ and computes the syndrome $C = He$ in $\mathbb{F}_2^{mt}$. This transformation is linear and deterministic, enabling direct verification in the decoding stage.

Algorithm 7 `Encode`(e, T)

Requires: $e \in \mathbb{F}_2^n$ with $\mathrm{wt}(e) = t$, public key $T \in \mathbb{F}_2^{mt \times k}$
Returns: Codeword $C \in \mathbb{F}_2^{mt}$
1: Define $H \leftarrow (I_{mt} \mid T)$
2: $C \leftarrow He \pmod 2$
3: **return** C

Decoding Subroutine (`Decode`)

The `Decode` function (also presented in the Algorithm 8) attempts to recover the weight-t error vector e from a given codeword $C \in \mathbb{F}_2^{mt}$ using the private key information Γ'. The private key includes (g, α', S) where g is the Goppa polynomial and α' are field elements used in the code definition. Decoding succeeds if, and only if, there exists a unique e of weight t such that $C = He$ with $H = (I_{mt} \mid T)$; otherwise the function returns $\perp$. The uniqueness property of e follows from the error-correcting capability of the Goppa code.

Algorithm 8 `Decode`(C, Γ')

Requires: $C \in \mathbb{F}_2^{mt}$, secret key component $\Gamma' = (g, \alpha', S)$
Returns: Error vector e or $\perp$ if decoding fails
1: Extend C to $v \leftarrow (C, 0, \ldots, 0) \in \mathbb{F}_2^n$ by appending k zeros
2: Find the unique $c \in \mathbb{F}_2^n$ such that:
   1. $Hc = 0$ (syndrome check)
   2. c has Hamming distance $\leq t$ from v

   If no such c exists, return $\perp$
3: $e \leftarrow v + c \pmod 2$
4: **if** $\mathrm{wt}(e) = t$ and $C = He$ **then return** e
5: **elsereturn** $\perp$
6: **end if**

Irreducible Polynomial Generation (`Irreducible`)

The `Irreducible` Algorithm 9 generates a monic irreducible polynomial $g \in \mathbb{F}_q[x]$ of degree t, where $q = 2^m$. The procedure begins by parsing the input bitstring into coefficients $\beta_j \in \mathbb{F}_q$, thereby constructing an element β in the extension field. The minimal polynomial g of β over $\mathbb{F}_q$ is computed; this ensures that g is the smallest-degree polynomial having β as a root. If the resulting degree is not exactly t, the candidate is discarded and a new one is generated. This polynomial defines the Goppa code parameters and has a direct impact on the security of the scheme.

Algorithm 9 `Irreducible`(σ)

Requires: Random seed σ, parameters m, t
Returns: Monic irreducible polynomial $g \in \mathbb{F}_q[x]$ of degree t
1: Parse σ into $(\beta_0, \beta_1, \ldots, \beta_{m-1}) \in \mathbb{F}_q^m$
2: $\beta \leftarrow \sum_{j=0}^{m-1} \beta_j x^j$
3: $g \leftarrow$ minimal polynomial of β over $\mathbb{F}_q$
4: **if** $\deg(g) \neq t$ **then return** failure
5: **end if**
6: **return** g

Field Ordering (`FieldOrdering`)

The `FieldOrdering` Algorithm 10 deterministically maps a bitstring to an ordered sequence $(\alpha_0, \ldots, \alpha_{q-1})$ of distinct elements in $\mathbb{F}_q$. The input is interpreted as integers $(a_0, \ldots, a_{q-1})$, and uniqueness is verified. The elements are then sorted to produce a fixed

ordering used in the evaluation of the Goppa code. Duplicate entries invalidate the ordering and trigger a restart.

Algorithm 10 `FieldOrdering`(σ)

Requires: Random seed σ, parameter m
Returns: Ordered list $(\alpha_0, \ldots, \alpha_{q-1}) \in \mathbb{F}_q$
1: Parse σ into $(a_0, \ldots, a_{q-1})$
2: **if** $\{a_i\}$ are not all distinct **then return** failure
3: **end if**
4: Sort $(a_0, \ldots, a_{q-1})$ lexicographically
5: Map each a_i to $\alpha_i \in \mathbb{F}_q$
6: **return** $(\alpha_0, \ldots, \alpha_{q-1})$

Key Generation (`KeyGen` and `SeededKeyGen`)

`KeyGen` (Algorithm 11) produces a public/secret key pair by first generating a master seed δ and delegating to `SeededKeyGen` (Algorithm 12). The seeded variant expands δ into seeds for field ordering, Goppa polynomial generation, and matrix generation. The public key contains the systematic form of the parity-check matrix, while the private key stores $(g, (\alpha_i), S)$ are needed for decoding. Failures in irreducibility checks, ordering uniqueness, or matrix formation trigger regeneration.

Algorithm 11 `KeyGen`()

1: $\delta \leftarrow$ random seed
2: **return** `SeededKeyGen`(δ)

Algorithm 12 `SeededKeyGen`(δ)

Requires: Seed δ
Returns: Public key pk, secret key sk
1: Expand δ into $(\sigma_1, \sigma_2, \sigma_3)$
2: $(\alpha_0, \ldots, \alpha_{q-1}) \leftarrow$ `FieldOrdering`(σ_1)
3: $g \leftarrow$ `Irreducible`(σ_2)
4: $(T, S) \leftarrow$ `MatGen`(g, α, σ_3)
5: **if** any step fails **then return** failure
6: **end if**
7: $pk \leftarrow T$
8: $sk \leftarrow (g, \alpha, S)$
9: **return** (pk, sk)

Matrix Generation (`MatGen`)

The `MatGen` function (Algorithm 13) constructs the public parity-check matrix from the Goppa polynomial and ordered field elements. A Gaussian elimination procedure attempts to reduce the matrix to systematic form. In semi-systematic cases, partial reduction with controlled column swaps is applied. The procedure may fail if the matrix is not full rank.

Algorithm 13 `MatGen`(g, α, σ)

Requires: $g \in \mathbb{F}_q[x]$, $(\alpha_0, \ldots, \alpha_{q-1}) \in \mathbb{F}_q$, seed σ
Returns: Public matrix T, secret permutation S
1: Build H from g and α
2: Attempt to reduce H to systematic form via Gaussian elimination
3: **if** reduction fails **then return** failure
4: **end if**
5: Output reduced T and permutation S

Fixed-Weight Vector Generation (`FixedWeight`)

This Algorithm 14 samples a binary vector $e \in \{0,1\}^n$ with exactly t ones, uniformly over all such vectors. It is implemented by selecting t distinct indices without replacement and setting the corresponding positions to 1.

Algorithm 14 `FixedWeight`(t, n)

Requires: Weight t, length n
Returns: Vector $e \in \{0,1\}^n$ with $\mathrm{wt}(e) = t$
1: $e \leftarrow 0^n$
2: Choose t distinct positions uniformly at random
3: Set those positions in e to 1
4: **return** e

Encapsulation (`Encap`)

`Encap` (Algorithm 15) creates a shared secret and corresponding ciphertext. A random error vector of weight t is generated, encoded under the public key to produce the ciphertext, and hashed along with fixed domain separation to yield the session key.

Algorithm 15 `Encap`(pk)

Requires: Public key pk
Returns: Ciphertext C, shared key K
1: $e \leftarrow$ `FixedWeight`(t, n)
2: $C \leftarrow \mathrm{Encode}(pk, e)$
3: $K \leftarrow \mathrm{Hash}(1 \parallel e \parallel C)$
4: **return** (C, K)

Decapsulation (`Decap`)

`Decap` (Algorithm 16) recovers the session key from the ciphertext. Using the private key, the ciphertext is decoded to retrieve e. If decoding fails, a fallback secret is used to preserve CCA security.

Algorithm 16 `Decap`(sk, C)

Requires: Secret key sk, ciphertext C
Returns: Shared key K
1: $e \leftarrow \mathrm{Decode}(sk, C)$
2: **if** decoding fails **then**
3: $\quad e \leftarrow$ default secret
4: **end if**
5: $K \leftarrow \mathrm{Hash}(b \parallel e \parallel C)$
6: **return** K

The overall process consists of three core stages: KeyGen (and its variant Seeded-KeyGen), Encap, and Decap, which together enable secure key establishment under the chosen code-based cryptosystem. During the KeyGen stage, a uniform random seed ρ is generated, from which the public matrix A (or T in the McEliece context) and associated secret vectors are derived via deterministic expansion functions such as `ExpandMatrix`. The public key pk encapsulates the matrix seed and auxiliary high-bit components, while the private key sk stores the matrix seed, secret polynomials or vectors, and decomposition artifacts necessary for reconstruction and decoding. In the Encap stage, the sender selects a fixed-weight error vector $e \in \mathbb{F}_2^n$ (with $\mathrm{wt}(e) = t$) uniformly at random, then computes the syndrome $C = He$ using the public parity-check matrix $H = (I_{mt} \mid T)$. This syndrome C forms the ciphertext (optionally concatenated with auxiliary components) and is used as input to a cryptographic hash function Hash to derive the shared session key K. This ensures

that the session key is computationally indistinguishable from random under the hardness of the underlying syndrome decoding problem. The DECAP stage uses the private key to perform error recovery via the DECODE subroutine, which applies the secret structure (g, α', S) to correct the received codeword and retrieve the original error vector e. Integrity checks are applied by recomputing the syndrome $C' = He$ and comparing it against the received ciphertext. If the verification passes, the same Hash function is applied to C to reconstruct the session key K; otherwise, a pseudorandom fallback key is returned to preserve CCA-security. This design ensures that only holders of the private key can recover the legitimate session key, while adversaries without decoding capability cannot distinguish it from random.

2.3.1. Parameter Sets

In Table 6 Coefficients of $F(y)$ lie in $\mathbb{F}_q = \mathbb{F}_2[z]/\langle f(z)\rangle$; hence, terms like "$+z$" denote field coefficients in $\mathbb{F}_q$. A dash "—" indicates the public key is in full systematic form. The semi–systematic choice $(\mu, \nu) = (32, 64)$ follows the submission and adjusts the reduction shape of the parity-check matrix to improve robustness of key generation for those sets.

Table 6. Classic McEliece parameter sets from the NIST submission. Table source: [38]. Here, $q = 2^m$, n is the code length, and t is the designed error weight. The polynomial $f(z) \in \mathbb{F}_2[z]$ defines $\mathbb{F}_q = \mathbb{F}_2[z]/\langle f(z)\rangle$, and $F(y) \in \mathbb{F}_q[y]$ is the Goppa polynomial. Rows with "f" use the semi–systematic public key form with the listed (μ, ν).

Set Name	m	n	t	$f(z)$	$F(y)$	Semi–Systematic (μ, ν)
mceliece348864	12	3488	64	$z^{12} + z^3 + 1$	$y^{64} + y^3 + y + z$	—
mceliece348864**f**	12	3488	64	$z^{12} + z^3 + 1$	$y^{64} + y^3 + y + z$	(32, 64)
mceliece460896	13	4608	96	$z^{13} + z^4 + z^3 + z + 1$	$y^{96} + y^{10} + y^9 + y^6 + 1$	—
mceliece460896**f**	13	4608	96	$z^{13} + z^4 + z^3 + z + 1$	$y^{96} + y^{10} + y^9 + y^6 + 1$	(32, 64)
mceliece6688128	13	6688	128	$z^{13} + z^4 + z^3 + z + 1$	$y^{128} + y^7 + y^2 + y + 1$	—
mceliece6688128**f**	13	6688	128	$z^{13} + z^4 + z^3 + z + 1$	$y^{128} + y^7 + y^2 + y + 1$	(32, 64)
mceliece6960119	13	6960	119	$z^{13} + z^4 + z^3 + z + 1$	$y^{119} + y^8 + 1$	—
mceliece6960119**f**	13	6960	119	$z^{13} + z^4 + z^3 + z + 1$	$y^{119} + y^8 + 1$	(32, 64)
mceliece8192128	13	8192	128	$z^{13} + z^4 + z^3 + z + 1$	$y^{128} + y^7 + y^2 + y + 1$	—
mceliece8192128**f**	13	8192	128	$z^{13} + z^4 + z^3 + z + 1$	$y^{128} + y^7 + y^2 + y + 1$	(32, 64)

2.3.2. Known Attacks and Security

The security of McEliece like other code-based KEMs relies on the inherent difficulty of general and syndrome decoding problems [13]. For any vector $v \in \mathbb{F}_2^m, m \in \mathbb{N}$, let $|v|$ denote the Hamming weight of v. These hard problems are defined as below [13,16,39].

(Decisional Syndrome Decoding problem) Given an $(n-k) \times n$ parity-check matrix H for C, a vector $y \in \mathbb{F}_2^{n-k}$, and a target $t \in \mathbb{N}$, determine whether there exists $x \in \mathbb{F}_2^n$ that satisfies $Hx^T = y$ and $|x| \leq t$.

(Decisional Codeword Finding problem) Given an $(n-k) \times n$ parity-check matrix H for C and a target $w \in \mathbb{N}$, determine whether there exists $x \in \mathbb{F}_2^n$ that satisfies $Hx^T = 0$ and $|x| = w$.

The most common form of attacks on the code-based cryptographic algorithm are the Information-Set Decoding (ISD) attack originally introduced by Prange [40] and many variants that have been studied afterward [39,41]. This approach ignores the structure of the binary code and seeks to recover the error vector based on its low Hamming weight [13]. Given an $n \times k$ matrix G over $\mathbb{F}_2$, vector $a \in \mathbb{F}_2^k$, and error vector $e \in \mathbb{F}_2^n$ of weight t, we consider the ciphertext $C = Ga + e$. Select k positions in C, ensuring these positions in e are all zero with probability $\binom{n-k}{t}/\binom{n}{t}$. If this set is error-free, then the selected positions in C match those in Ga. Recover a from these positions using linear algebra. Verify success by checking if $C - Ga$ has weight t; otherwise, choose a new set. The $k \times k$ matrix formed by

selecting these positions in G may not be invertible. To handle this, choose an "information set" where the resulting matrix is invertible, retrying as needed. This is the original form of information-set decoding [39,40]. Moving forward after Prange, the ISD attacks were improved by Lee-Brikell [39,42] by assuming multiple errors in the information set rather than the original 0 case. Later the attack was further improved by Leon's work [43], which instead of computing all positions of $C - Ga$, just proposed checking if $C - Ga$ has weight t, from the information set a [39]. Later on, Stern, and more lately Becker et al., showed a more sophisticated version of this attack by leveraging the combinatorial searches for errors in the given information set [39,44,45].

Alternatively, the scheme's security can be based on the assumptions that row-reduced parity check matrices for binary Goppa codes in Classic McEliece are indistinguishable from those of random linear codes of the same dimensions and that the syndrome decoding problem is hard for such random codes. Current cryptanalysis supports these assumptions [13]. It has also been shown as IND-CCA2 secure against all ROM attacks [13,39,46,47].

2.3.3. Major Updates in McEliece KEM NIST Submission [13,47]

1. Security Stability: The security level of the McEliece system has remained stable over 40 years despite numerous attack attempts; it was originally designed for 2^{64} security but is scalable to counter advanced computing technologies, including quantum computing.
2. Efficiency Improvements: Significant follow-up work has improved the system's efficiency while preserving security, including a "dual" PKE proposed by Niederreiter, and various software and hardware speedups.
3. KEM Conversion: The method to convert an OW-CPA PKE into a KEM that is IND-CCA2 secure against all ROM attacks is well known and tight, preserving security level with no decryption failures for valid ciphertexts.
4. Handling QROM Attacks: Recent advances have extended security to handle a broader class of attacks, including QROM attacks, by using a high-security, unstructured hash function.
5. Classic McEliece (CM) Design: The NIST submission for Classic McEliece (CM) aims for IND-CCA2 security at a very high level, even against quantum computers, leveraging Niederreiter's dual version using binary Goppa codes.

2.4. BIKE: Bit Flipping Key Encapsulation

A bit flipping key encapsulation (BIKE) is a public key encapsulation technique [48] also known widely as a KEM technique, which is also based on coding theory. It is also one of the finalized candidate post-quantum algorithms in the NIST PQC standardization competition and is currently being considered for the Round 4 competition [49]. The design of BIKE is based on the Niederreiter-based KEM instantiated with QC-MDPC codes and leverages the Black-Gray-Flip Decoder implemented in constant times [48,49]. In other words, it is the McEliece schema instantiated with QC-MDPC and relies on the hardness of quasi-cyclic variants of the hard problems from coding theory [49].

2.4.1. Preliminaries

$\mathbb{F}_2$: The binary finite field, consisting of two elements {0, 1} with addition and multiplication defined modulo 2. *Usage*: Used for defining elements in binary operations and polynomial arithmetic.

$\mathbb{R}$: A cyclic polynomial ring, specifically, $\mathbb{F}_2[X]/(X^r - 1)$. Here, $X^r - 1$ is a polynomial of degree r, and the operations are performed modulo this polynomial. *Usage*: Represents the set of polynomials used in encoding and decoding processes.

$\mathcal{H}_w$: The private key space consisting of pairs of sparse polynomials (h_0, h_1) in $\mathbb{R}^2$ such that the Hamming weights of h_0 and h_1 are each $w/2$. *Usage*: Used to generate private keys in the BIKE scheme.

$\mathcal{E}_t$: The error space, consisting of pairs of polynomials (e_0, e_1) in $\mathbb{R}^2$ where the sum of their Hamming weights is t. *Usage*: Represents the error vectors used in the encoding process.

$|g|$: The Hamming weight of a binary polynomial g in $\mathbb{R}$, defined as the number of nonzero coefficients in g. *Usage*: Used to measure the sparsity of polynomials.

$u \xleftarrow{R} U$: Denotes that the variable u is sampled uniformly at random from the set U. *Usage*: Used in key generation and error vector selection processes.

$\oplus$: The exclusive OR (XOR) operation, performed component-wise when applied to vectors. *Usage*: Used in bitwise operations within the encoding and decoding algorithms.

2.4.2. QC-MDPC Code

BIKE is based on the Quasi-Cyclic Moderate Density Parity Check (QC-MDPC) codes, which are also related to the Low Density Parity Check (LDPC) codes [50]. A Quasi-Cyclic Moderate Density Parity Check code of index 2, length n, and row weight w is defined as a pair of sparse parity polynomials $(h_0, h_1) \in \mathcal{H}_w$. These polynomials form the private key used in the encoding and decoding processes. Decryption in BIKE is performed by decoding a QC-MDPC code, which is usually performed using a variant of the bit-flipping algorithm [50].

Mathematically, a QC-MDPC code is defined by its parity-check matrix [51] H, which has the following properties:

- H is composed of two circulant blocks, each of size $r \times r$.
- The matrix can be written as:

$$H = \begin{pmatrix} H_0 & H_1 \end{pmatrix}$$

  where H_0 and H_1 are circulant matrices derived from the polynomials h_0 and h_1, respectively.
- Each row of H_0 and H_1 contains exactly $w/2$ ones, reflecting the sparsity condition.

A codeword c in the QC-MDPC code satisfies the equation:

$$H \cdot c^T = 0 \mod 2$$

where c is a binary vector of length n.

2.4.3. Decoder

Let $\mathbb{R} = \mathbb{F}_2[x]/\langle x^r - 1\rangle$ be the ring of binary circulant polynomials of degree $< r$. The private key consists of two sparse parity polynomials $(h_0, h_1) \in \mathcal{H}_w \subset \mathbb{R}^2$, each of Hamming weight w, and the public key is $h = h_1 h_0^{-1} \in \mathbb{R}$. The decoder is required to satisfy the correctness condition that for any error pair $(e_0, e_1) \in \mathbb{R}^2$ with $\|e_0\| + \|e_1\| \leq t$,

$$(e_0, e_1) = \text{decoder}(e_0 h_0 + e_1 h_1,\ h_0,\ h_1).$$

Equivalently, given a syndrome $s = e_0 h_0 + e_1 h_1 \in \mathbb{R}$ and the secret support (h_0, h_1), the decoder recovers the unique (e_0, e_1) of weight at most t. In practice, BIKE instantiates the decoder with an iterative bit-flipping procedure on the QC-MDPC code defined by (h_0, h_1).

2.4.4. Algorithms for BIKE-KEM

Key Generation

The key generator (Algorithm 17) samples two weight-w polynomials (h_0, h_1) uniformly from $\mathcal{H}_w$ and computes the public key $h = h_1 h_0^{-1}$ in $\mathbb{R}$ (the inverse exists with overwhelming probability for BIKE parameters). A random seed $\sigma \in \mathcal{M}$ is stored for FO-style fallback during decapsulation.

Algorithm 17 Key Generation (KeyGen)

Input: None
Output: $((h_0, h_1), \sigma) \in \mathcal{H}_w \times \mathcal{M}, h \in \mathbb{R}$
$(h_0, h_1) \xleftarrow{R} \mathcal{H}_w$
$h \leftarrow h_1 h_0^{-1}$
$\sigma \xleftarrow{R} \mathcal{M}$

Encapsulation

To encapsulate (Algorithm 18, the sender draws $m \in \mathcal{M}$, hashes it to a sparse error pair $(e_0, e_1) = \mathcal{H}(m)$ of total weight t, and forms the ciphertext $c = (c_0, c_1)$ with $c_0 = e_0 + e_1 h \in \mathbb{R}$ and $c_1 = m \oplus \mathcal{L}(e_0, e_1)$, where $\mathcal{L}$ is a leakage-resistant linearization of the error (typically a compression of its support positions). The session key is $K = \mathcal{K}(m, c)$ via a KDF.

Algorithm 18 Encapsulation (Encaps)

Input: $h \in \mathbb{R}$
Output: $K \in \mathcal{K},\ c = (c_0, c_1) \in \mathbb{R} \times \mathcal{M}$
$m \xleftarrow{R} \mathcal{M}$
$(e_0, e_1) \leftarrow \mathcal{H}(m)$
$c \leftarrow (\, e_0 + e_1 h,\ m \oplus \mathcal{L}(e_0, e_1)\,)$
$K \leftarrow \mathcal{K}(m, c)$

Decapsulation

The receiver uses the secret parity pair (h_0, h_1) to decode the first ciphertext component and recover an error estimate e'. The message is then reconstructed from c_1 by inverting $\mathcal{L}$. If $e' = \mathcal{H}(m')$ holds (consistency check tying the recovered message to its error), the legitimate key $K = \mathcal{K}(m', c)$ is returned; otherwise the FO-fallback $K = \mathcal{K}(\sigma, c)$ is derived to preserve CCA security. The algorithm is decribed in Algorithm 19.

Algorithm 19 Decapsulation (Decaps)

Input: $((h_0, h_1), \sigma) \in \mathcal{H}_w \times \mathcal{M},\ c = (c_0, c_1) \in \mathbb{R} \times \mathcal{M}$
Output: $K \in \mathcal{K}$
$e' \leftarrow \text{decoder}(c_0 h_0,\ h_0,\ h_1)$
$m' \leftarrow c_1 \oplus \mathcal{L}(e')$ ▷ with the convention $\perp = (0, 0)$
if $e' = \mathcal{H}(m')$ **then**
    $K \leftarrow \mathcal{K}(m', c)$
else
    $K \leftarrow \mathcal{K}(\sigma, c)$
end if

Iterative Bit-Flipping Decoder

BIKE instantiates decoder$(\cdot)$ (Algorithm 20) with a hard-decision, majority-logic bit-flipping algorithm over the QC-MDPC code defined by the $r \times 2r$ parity-check matrix $H = [H_0\ H_1]$ induced by (h_0, h_1). Starting from the syndrome $s \in \mathbb{F}_2^r$, the algorithm repeatedly (i) counts unsatisfied checks per code bit, (ii) flips bits whose counters exceed

an iteration-dependent threshold, and (iii) updates the running syndrome. The number of iterations (`NbIter`) and threshold schedule are selected to balance decoding success and constant-time behavior.

Algorithm 20 BIKE Decoder

Input: $s \in \mathbb{F}_2^r$, $H \in \mathbb{F}_2^{r \times n}$ $(n = 2r)$
$\tilde{e} \leftarrow 0^n$, $\tilde{s} \leftarrow s$
for $i = 1, \ldots,$ `NbIter` **do**
  $T \leftarrow$ `THRESHOLD`$(i, s, \tilde{s})$
  for $j = 0, \ldots, n-1$ **do**
    $\sigma_j \leftarrow \mathrm{ctr}(H, \tilde{s}, j)$ ▷ unsatisfied-check counter for bit j
  end for
  for $j = 0, \ldots, n-1$ **do**
    if $\sigma_j \geq T$ **then**
      $\tilde{e}_j \leftarrow \tilde{e}_j \oplus 1$
      $\tilde{s} \leftarrow \tilde{s} - \mathrm{col}(H, j)$
    end if
  end for
end for
return $\tilde{e}$

Adaptive Threshold Schedule

The function `THRESHOLD` (Algorithm 21) chooses the flipping threshold as a convex combination of (i) an "optimal" value $T' = f_t(\|s\|)$ predicted from the syndrome weight, and (ii) the majority value $M = (d+1)/2$, where d is the row weight of H (approximately the MDPC density). The additive margin δ enforces robustness and constant-time behavior. The final threshold is lower-bounded by $f_t(\|\tilde{s}\|)$ to avoid over-flipping late in the decoding.

Algorithm 21 Function `THRESHOLD`

function THRESHOLD$(i, s, \tilde{s})$
  $T' \leftarrow f_t(\|s\|)$ ▷ syndrome-weight heuristic
  $M \leftarrow (d+1)/2$ ▷ majority threshold
  if $i = 1$ **then**
    $T \leftarrow T' + \delta$
  else if $i = 2$ **then**
    $T \leftarrow (2T' + M)/3 + \delta$
  else if $i = 3$ **then**
    $T \leftarrow (T' + 2M)/3 + \delta$
  else
    $T \leftarrow M + \delta$
  end if
  return $\max\left(f_t(\|\tilde{s}\|),\ T\right)$
end function

Key generation samples $(h_0, h_1) \in \mathcal{H}_w$ and publishes $h = h_1 h_0^{-1}$. Encapsulation hashes a fresh m into a weight-t error pair (e_0, e_1), forms $c_0 = e_0 + e_1 h$, protects m as $c_1 = m \oplus \mathcal{L}(e_0, e_1)$, and derives $K = \mathcal{K}(m, c)$. Decapsulation computes $e' = \mathrm{decoder}(c_0 h_0, h_0, h_1)$ and $m' = c_1 \oplus \mathcal{L}(e')$; if $e' = \mathcal{H}(m')$ the session key $K = \mathcal{K}(m', c)$ is output, else the fallback $K = \mathcal{K}(\sigma, c)$ is returned. Under the decoder correctness condition and for $\|e_0\| + \|e_1\| \leq t$, we have $c_0 = e_0 + e_1 h$ and $e' = (e_0, e_1)$, implying $m' = m$ and consistent key derivation on both sides; the FO-style fallback ensures CCA security for invalid ciphertexts.

2.4.5. BIKE Parameters

The BIKE Key Encapsulation Mechanism (BIKE-KEM) is parameterized to meet the NIST security categories corresponding to the classical security levels of AES-128, AES-192, and AES-256, referred to as Levels 1, 3, and 5, respectively. The parameter set for BIKE is defined as a triple (r, w, t), where r denotes the code length parameter (half the code length in bits), w is the Hamming weight of each secret key polynomial, and t is the maximum number of errors that can be corrected during decapsulation. For all security levels, the key length is fixed to $\ell = 256$, and the target Decoding Failure Rate (DFR) is chosen to be negligible relative to the claimed security level—2^{-128}, 2^{-192}, and 2^{-256} for Levels 1, 3, and 5, respectively, as shown in Table 7. The values of (r, w, t) are carefully selected to balance decoding performance, security against information-set decoding (ISD) attacks, and resistance to structural attacks on quasi-cyclic codes. Additionally, the decoding process employs the Bit-Guess-and-Flip (BGF) algorithm, whose performance is tuned through parameters such as the number of decoding iterations (`NbIter`), the threshold adaptation parameter τ, and a level-specific linear threshold function $\texttt{threshold}(S)$ based on the syndrome weight S. These parameters ensure that decoding remains both efficient and secure, while maintaining a decoding failure probability well below the security margin mandated by NIST. The configuration for Decoder is given in Table 8.

Table 7. Suggested BIKE parameter sets (all levels use key length $\ell = 256$). Each set is a triple (r, w, t) over the ring $\mathbb{F}_2[x]/\langle x^r - 1\rangle$. The target decoding-failure rate (DFR) is shown per NIST level. Table referenced from [49].

NIST Level	r	w	t	Target DFR
Level 1	12,323	142	134	2^{-128}
Level 3	24,659	206	199	2^{-192}
Level 5	40,973	274	264	2^{-256}

Table 8. Decoder configuration (BGF bit-flipping) used for the DFR estimates in Table 7. Here, $S = \|s\|$ is the syndrome weight and the threshold is independent of the iteration index i. Table source: [49].

NIST Level	NbIter	τ	Threshold Rule $\texttt{threshold}(S) = \max(\alpha S + \beta,\ C)$
Level 1	5	3	$\alpha = 0.0069722,\ \beta = 13.530,\ C = 36$
Level 3	5	3	$\alpha = 0.0052650,\ \beta = 15.2588,\ C = 52$
Level 5	5	3	$\alpha = 0.00402312,\ \beta = 17.8785,\ C = 69$

2.4.6. Known Attacks and Security

The security of BIKE relies on the hardness of two hard coding theory problems: the Hardness of QCSD (Quasi-Cyclic Syndrome Decoding) and hardness of QCCF (Quasi-Cyclic Codeword Finding). The complete definitions of these problems are given below. It also has another security assumption which is the Correctness of Decoder ($\lambda - corr$: a KEM is λ correct if the decapsulation fails with probability at most λ on average over all keys and messages [49]).

Let the element of $\mathcal{R}$ is odd and even weights are denoted as $\mathcal{R}_{\text{odd}}$ and $\mathcal{R}_{\text{even}}$. For any integer t, the parity is denoted as $p(t) \in \{odd, even\}$, then the standard hard problems of QCSD and QCCF are [13,49].

QC Syndrome Decoding-QCSD)

Given $(h, s) \in \mathcal{R}_{\text{odd}} \times \mathcal{R}_{p(t)}$ and an integer $t > 0$, determine if there exists $(e_0, e_1) \in \mathcal{E}_t$ such that $e_0 + e_1 h = s$.

Codeword Finding-QCCF)

Given $h \in \mathcal{R}_{\text{odd}}$ and an even integer $w > 0$ with $w/2$ odd, determine if there exists $(h_0, h_1) \in \mathcal{H}_w$ such that $h_1 + h_0 h = 0$.

While there is a known search-to-decision reduction for the general syndrome decoding problem, no such reduction exists for the quasi-cyclic case, and the best solvers for quasi-cyclic problems, which are based on Information Set Decoding (ISD), perform similarly for both search and decision problems [49]. The best-known attacks against BIKE's KEM are Information Set Decoding and its variants, as in the case of any code-based KEM, such as McEliece [13]. Further the Quasi-Cyclic structure of the quasi-cyclic code can be exploited; it makes both codeword finding and decoding relatively easier. The details of these are given in [49]. But different sets of parameters can be chosen to adapt to these challenges and BIKE still is IND-CAP and IND-CCA proven [13,49]. In the purposed cryptosystem, the key-pair should not be reused [49], but if they are ever, then it becomes vulnerable to decoding failure attacks such as GJS rejection attacks [52].

BIKE targets IND-CCA security by requiring decoders with extremely low decryption failure rates (DFRs)—specifically, of the order of 2^{-128}, 2^{-192}, and 2^{-256} for NIST Security Levels 1, 3, and 5, respectively [49]. However, more recent empirical analysis indicates that the real-world average DFR for Level 1 may be closer to $2^{-116.6}$, falling short of the design goal [53]. This discrepancy has implications for security because multi-target key recovery attacks become more feasible when decryption failures exceed expected thresholds. Consequently, BIKE's specification has evolved to adjust decoder thresholds and incorporate mitigations, such as weak-key filtering, to reduce DFR and preserve IND-CCA security under more realistic operational conditions [49,54].

2.4.7. Major Updates in BIKE KEM NIST Submission [13,49]

1. Single Variant: The BIKE team narrowed down various versions of the algorithm to a single variant using the Black-Grey-Flip (BGF) decoder for enhanced security and efficiency.
2. Security Enhancements: Security category 5 parameters were added, and all random oracles were updated to SHA-3-based constructions to improve hardware performance and avoid IP issues.
3. Specification Simplification: The document structure was significantly simplified, with most mathematical background moved to the appendix.
4. IND-CCA Security Claim: BIKE now claims IND-CCA security with additional analysis supporting this claim.
5. Data-Oblivious Sampling: Introduced a data-oblivious sampling technique to mitigate side-channel attacks, generating new Known Answer Tests (KATs).

Figure 4 shows the major updates in BIKE algorithms throughout the multiple rounds of submissions.

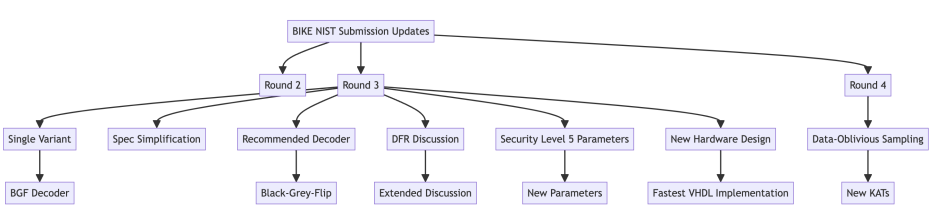


Figure 4. Evolution of BIKE over multiple rounds.

2.5. Hamming Quasi-Cycle Cryptosystem

The Hamming Quasi-Cycle Cryptosystem is another KEM candidate currently under consideration for the 4th round of the NIST post-quantum standardization effort. HQC was motivated by the schema introduced by Alexhnovich [55]. The trapdoor, also known as the secret key, in the [55] schema is a random error vector that has been combined with a random codeword of a random code. Hence, finding the secret key is similar to finding out how to decode a random code that has no hidden structure [56]. The HQC cryptosystem uses two codes: a known efficient decoding code $C[n,k]$ and a $[2n],[n]$ random double-circulant code H in systematic form to generate noise. The public key consists of the generator matrix $\mathbf{G}$ and the syndrome $s = H(\mathbf{x},\mathbf{y})^\top$, where $\mathbf{x},\mathbf{y}$ form the secret key. To encrypt a message m, it is encoded with $\mathbf{G}$, combined with s and a short vector $\mathbf{e}$, resulting in the ciphertext $(\mathbf{u} = \mathbf{r}H^\top, \mathbf{v} = m\mathbf{G} + s \cdot r_2 + \mathbf{e})$. The recipient uses the secret key to decode $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$ and retrieve the plaintext [56].

This system does not rely on the error term being within the decoding capability of the code. In traditional McEliece approaches, the error term added to the encoding of the message must be less than or equal to the decoding capability of the code to ensure correctness. However, in this construction, this assumption is no longer required. The correctness of this cryptosystem is guaranteed as the legitimate recipient can remove enough errors from the noisy encoding $\mathbf{v}$ of the message using the secret key $\mathbf{sk}$ [13,56].

Most of the preliminaries of HQC have been covered in the earlier sections of McEliece and BIKE; hence, here we list only the aspects that were not already covered.

Circulant Matrix [56] Let $x = (x_1, \ldots, x_n) \in \mathbf{F^n}$, the circulant matrix induced by x is defined by,

$$\mathbf{rot}(\mathbf{x}) = \begin{pmatrix} x_1 & x_n & \ldots & x_2 \\ x_2 & x_1 & \ldots & x_3 \\ \vdots & \vdots & \ddots & \vdots \\ x_n & x_{n-1} & \ldots & x_1 \end{pmatrix} \in \mathbb{F}^{n \times n} \tag{6}$$

Quasi-Cyclic Codes [56] View a vector $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_s)$ of $\mathbb{F}_2^{sn}$ as s successive blocks (n-tuples). An $[sn,k,d]$ linear code $\mathcal{C}$ is Quasi-Cyclic (QC) of index s if, for any $\mathbf{c} = (\mathbf{c}_1, \ldots, \mathbf{c}_s) \in \mathcal{C}$, the vector obtained after applying a simultaneous circular shift to every block $\mathbf{c}_1, \ldots, \mathbf{c}_s$ is also a codeword.

More formally, by considering each block $\mathbf{c}_i$ as a polynomial in $\mathbb{R} = \mathbb{F}[X]/(X^n - 1)$, the code $\mathcal{C}$ is QC of index s if for any $\mathbf{c} = (\mathbf{c}_1, \ldots, \mathbf{c}_s) \in \mathcal{C}$ it holds that $(X \cdot \mathbf{c}_1, \ldots, X \cdot \mathbf{c}_s) \in \mathcal{C}$.

Systematic Quasi-Cyclic Codes [56] A systematic Quasi-Cyclic $[sn],[n]$ code of index s and rate $1/s$ is a quasi-cyclic code with an $(s-1)n \times sn$ parity-check matrix of the form:

$$\mathbf{H} = \begin{bmatrix} \mathbf{I}_n & 0 & \cdots & 0 & \mathbf{A}_1 \\ 0 & \mathbf{I}_n & & & \mathbf{A}_2 \\ & & \ddots & & \vdots \\ 0 & & \cdots & \mathbf{I}_n & \mathbf{A}_{s-1} \end{bmatrix} \tag{7}$$

The HQC schema consists of our polynomial time algorithms. Setup: generates the global parameters needed for the algorithms; KeyGen(params): generates the public and private key (pk, sk). Then, Encrypt(pk, m): takes the public key and message and generates the chipertext c. Finally, Decode(sk, c): uses the private key of the receiving party and ciphertext to reconstruct the original message.

HQC uses two types of codes: a decodable $[n,k]$ code C, generated by $\mathbf{G} \in \mathbb{F}_2^{k \times n}$ and which can correct at least Δ errors via an efficient algorithm $C.\mathrm{Decode}(\cdot)$; and a random double-circulant $[2n,n]$ code, of the parity-check matrix $(\mathbf{1},\mathbf{h})$ [57].

Setup

The Setup (Algorithm 22) procedure derives the global coding and masking parameters from the security parameter 1^λ. The tuple $\text{param} = (n, k, \Delta, w, w_r, w_e, \ell)$ fixes the ambient ring $\mathcal{R} = \mathbb{F}_2[x]/\langle x^n - 1\rangle$, the public generator matrix $\mathbf{G} \in \mathbb{F}_2^{k\times n}$ of the underlying code C, the masking/truncation length ℓ, and the Hamming–weight constraints for secrets (w), ephemeral randomness (w_r), and encryption noise (w_e). The decoding radius Δ captures the designed error-correction capability of C used in Decrypt.

Algorithm 22 Setup

1: **Input**: Security parameter 1^λ
2: **Output**: Global parameters $\text{param} = (n, k, \Delta, w, w_r, w_e, \ell)$

Key Generation for the PKE

Given param, KeyGen (Algorithm 23) samples a public ring element $h \in \mathcal{R}$ and secret vectors $(\mathbf{x}, \mathbf{y})$ of prescribed weights w and w_r, respectively. It computes $\mathbf{s} = \mathbf{x} + h \cdot \mathbf{y} \in \mathcal{R}$ and publishes $\text{pk} = (h, \mathbf{s})$ together with the public generator $\mathbf{G}$ of C. The secret key stores $(\mathbf{x}, \mathbf{y})$. Correctness follows since, under the designed weights, decryption will subtract the structured interference $\mathbf{u} \cdot \mathbf{y}$ and leave a codeword plus small error within the decoding radius.

Algorithm 23 KeyGen

1: **Input**: param
2: Sample $h \xleftarrow{\$} \mathcal{R}$
3: Generate the matrix $\mathbf{G} \in \mathbb{F}_2^{k\times n}$ of C
4: Sample $(\mathbf{x}, \mathbf{y}) \xleftarrow{\$} \mathcal{R}_w \times \mathcal{R}_{w_r}$
5: Set $\text{sk} = (\mathbf{x}, \mathbf{y})$
6: Set $\text{pk} = (\mathbf{h}, \mathbf{s} = \mathbf{x} + \mathbf{h} \cdot \mathbf{y})$
7: **Output**: (pk, sk)

Encryption (PKE)

Encrypt (Algorithm 24) samples an encryption error $\mathbf{e}$ of weight w_e and an ephemeral pair $\mathbf{r} = (r_1, r_2)$ of weight w_r each. It masks a message $m \in \mathbb{F}_2^k$ by forming $\mathbf{u} = r_1 + h \cdot r_2 \in \mathcal{R}$ and $\mathbf{v} = \text{truncate}\big(m\mathbf{G} + \mathbf{s} \cdot r_2 + \mathbf{e},\ \ell\big)$, where the truncation operator reduces bandwidth while retaining sufficient redundancy for reliable decoding. The ciphertext is $\mathbf{c} = (\mathbf{u}, \mathbf{v})$.

Algorithm 24 Encrypt

1: **Input**: pk, m
2: Generate $\mathbf{e} \xleftarrow{\$} \mathcal{R}_{w_e}$
3: Sample $\mathbf{r} = (r_1, r_2) \xleftarrow{\$} \mathcal{R}_{w_r} \times \mathcal{R}_{w_r}$
4: Set $\mathbf{u} = r_1 + \mathbf{h} \cdot r_2$
5: Set $\mathbf{v} = \text{truncate}(m\mathbf{G} + \mathbf{s} \cdot r_2 + \mathbf{e}, \ell)$
6: **Output**: $\mathbf{c} = (\mathbf{u}, \mathbf{v})$

Decryption (PKE)

Decrypt (Algorithm 25) cancels the structured mask by computing $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$, which ideally equals $m\mathbf{G} + \mathbf{e}'$ for a noise $\mathbf{e}'$ within the decoding radius Δ. Applying the decoder C.Decode recovers m (or outputs $\perp$ on failure). Correctness hinges on the weight design (w, w_r, w_e) and the code's guaranteed decoding capability.

Algorithm 25 Decrypt

1: **Input**: sk, $\mathbf{c}$
2: **Output**: $C.\text{Decode}(\mathbf{v} - \mathbf{u} \cdot \mathbf{y})$

From PKE to KEM. HQC-KEM wraps the above PKE with a Fujisaki–Okamoto-style transform. A fresh salt ensures domain separation and nonces the KDF/PRG, enabling deterministic re-encryption during decapsulation for robust CCA checks.

KEM Setup

The KEM's Setup (Algorithm 26) simply exposes the global tuple param for all parties and implementations, ensuring consistent code, weights, and truncation length across Encapsulate/Decapsulate

Algorithm 26 Setup(1^λ)

1: Outputs the global parameters $\text{param} = (n, k, \Delta, w, w_r, w_e, \ell)$.

KEM Key generation

KEM KeyGen (Algorithm 27) augments the PKE key pair with a uniformly random commitment string $\sigma \in \mathbb{F}_2^k$, used as FO fallback material in case re-encryption checks fail during decapsulation. The public key remains $(h, \mathbf{s})$.

Algorithm 27 KeyGen(param)

1: Samples $h \xleftarrow{\$} \mathcal{R}$.
2: Samples $\sigma \xleftarrow{\$} \mathbb{F}_2^k$.
3: Generates the matrix $\mathbf{G} \in \mathbb{F}_2^{k \times n}$ of C.
4: Samples $(\mathbf{x}, \mathbf{y}, \sigma) \xleftarrow{\$} \mathcal{R}_w \times \mathcal{R}_{w_r} \times \mathbb{F}_2^k$.
5: Sets $\text{sk} = (\mathbf{x}, \mathbf{y}, \sigma)$.
6: Sets $\text{pk} = (\mathbf{h}, \mathbf{s} = \mathbf{x} + \mathbf{h} \cdot \mathbf{y})$.
7: Returns (pk, sk).

Encapsulation (KEM)

Encapsulate (Algorithm 28) samples a session seed $m \in \mathbb{F}_2^k$ and a 128-bit salt, derives PRG output $\theta = \mathcal{G}(m \| \text{pk} \| \text{salt})$, and deterministically generates $(\mathbf{e}, r_1, r_2)$ with target weights (w_e, w_r, w_r). It then constructs $\mathbf{u}, \mathbf{v}$ exactly as in PKE and outputs the ciphertext $\mathbf{c} = (\mathbf{u}, \mathbf{v})$ together with the session key $K = \mathcal{K}(m, \mathbf{c})$. Binding K to both m and $\mathbf{c}$ prevents key-mismatch and contributes to CCA resilience.

Algorithm 28 Encapsulate(pk)

1: Generates $m \xleftarrow{\$} \mathbb{F}_2^k$.
2: Generates salt $\xleftarrow{\$} \mathbb{F}_{2^{128}}$.
3: Derives randomness $\theta \leftarrow \mathcal{G}(m || \text{pk} || \text{salt})$.
4: Uses θ to generate $(\mathbf{e}, r_1, r_2)$ such that $\omega(\mathbf{e}) = w_e$ and $\omega(r_1) = \omega(r_2) = w_r$.
5: Sets $\mathbf{u} = r_1 + \mathbf{h} \cdot r_2$.
6: Sets $\mathbf{v} = \text{truncate}(m\mathbf{G} + \mathbf{s} \cdot r_2 + \mathbf{e}, \ell)$.
7: Sets $\mathbf{c} = (\mathbf{u}, \mathbf{v})$.
8: Computes $K \leftarrow \mathcal{K}(m, \mathbf{c})$.
9: Returns (K, salt).

Decapsulation (KEM)

Decapsulate (Algorithm 29) first decrypts to m'. It derives $\theta' = \mathcal{G}(m' \| \text{pk} \| \text{salt})$, re-encrypts deterministically to $\mathbf{c}'$, and checks $\mathbf{c}' = \mathbf{c}$. On success, it outputs $K = \mathcal{K}(m', \mathbf{c})$; otherwise, it returns the FO fallback $K = \mathcal{K}(\sigma, \mathbf{c})$. This re-encryption check thwarts invalid-ciphertext and reaction attacks and ensures that only ciphertexts consistent with (m', θ') lead to the "real" key.

Algorithm 29 Decapsulate(sk, $\mathbf{c}$, salt)

1: Decrypts $m' \leftarrow \text{Decrypt}(\text{sk}, \mathbf{c})$.
2: Computes randomness $\theta' \leftarrow \mathcal{G}(m' || \text{pk} || \text{salt})$.
3: Re-encrypts m' by using θ' to generate $(\mathbf{e}', r_1', r_2')$ such that $\omega'(\mathbf{e}') = w_e$ and $\omega'(r_1') = \omega'(r_2') = w_r$.
4: Sets $\mathbf{u}' = r_1' + \mathbf{h} \cdot r_2'$.
5: Sets $\mathbf{v}' = \text{truncate}(m'\mathbf{G} + \mathbf{s} \cdot r_2' + \mathbf{e}', \ell)$.
6: Sets $\mathbf{c}' = (\mathbf{u}', \mathbf{v}')$.
7: **if** $m' = \perp$ or $\mathbf{c} \neq \mathbf{c}'$ **then**
8: $\quad K \leftarrow \mathcal{K}(\sigma, \mathbf{c})$.
9: **else**
10: $\quad K \leftarrow \mathcal{K}(m, \mathbf{c})$.
11: **end if**

The HQC-KEM scheme operates through a structured sequence of steps involving setup, key generation, encapsulation, and decapsulation. In the SETUP phase, global parameters $(n, k, \Delta, w, w_r, w_e, \ell)$ are established, defining the code length, dimension, decoding threshold, and weight parameters for error vectors and random vectors. The KEYGEN algorithm samples a random public vector h from $\mathcal{R}$, generates the generator matrix $\mathbf{G}$ of the underlying linear code C, and selects secret vectors $\mathbf{x}$ and $\mathbf{y}$ of the prescribed Hamming weights, forming the secret key sk and public key pk. The procedure ENCAPSULATE begins by sampling a random message m and a salt value, from which the randomness θ is derived using a hash-based function $\mathcal{G}$. Using θ, the sender generates an error vector $\mathbf{e}$ and two random vectors (r_1, r_2), encodes them into a pair $(\mathbf{u}, \mathbf{v})$ using the public key, and outputs the ciphertext $\mathbf{c}$ along with the session key $K = \mathcal{K}(m, \mathbf{c})$. In the DECAPSULATE phase, the receiver uses the secret key to decode m' from the ciphertext, regenerates the randomness θ' and corresponding vectors, and verifies the ciphertext by recomputation. If the integrity check passes, the original session key is recomputed; otherwise, a fallback key derived from the secret seed σ is used.

2.5.1. HQC Parameters

Table 9 summarizes the main parameters of HQC across its three recommended NIST security levels 1, 3, 5 (128, 192, and 256 bits). The table presents public key sizes, ciphertext sizes, and computational costs for key generation, encapsulation, and decapsulation. Notably, the DFR is also included, which represents the probability that a legitimate ciphertext fails to decrypt correctly under normal conditions.

Table 9. HQC parameter sets and corresponding decryption failure rates. Table source: [57].

	Public Key Size	Ciphertext Size	KeyGen	Encaps	Decaps	DFR
hqc-128	2249	4497	87	204	362	$<2^{-128}$
hqc-192	4522	9042	204	465	755	$<2^{-192}$
hqc-256	7245	14,485	409	904	1505	$<2^{-256}$

Although decryption failures can occur due to the probabilistic nature of code-based decoding, HQC ensures that the DFR remains negligibly small, well below the classical brute-force attack thresholds at each security level (2^{-128}, 2^{-192}, 2^{-256}, respectively). This makes HQC highly reliable in practical deployments, where such rare failures are statistically insignificant over the expected system lifetime.

2.5.2. Known Attacks and Security

The security of HQC, as in other code-based cryptosystems, relies on the decoding problem. Specifically, HQC relies on Quasi-Cycle Syndrome Decoding (QCSD) with parity problem. The Fujsaki–Okamoto transform has been applied to CPS-secure public key to achieve an IND-CCA KEM [13]. These problems are called 2-QCSD-P and 3-QCSD-P problems; we borrow the definition from [57]

2-DQCSD-P Problem Let n, w, b_1 be positive integers and $b_2 = w + b_1 \times w \mod 2$. Given $(\mathbf{H}, \mathbf{y}) \in \mathbb{F}_2^{n \times 2n} \times \mathbb{F}_2^{n,b_2}$, the Decisional 2-Quasi-Cyclic Syndrome Decoding with Parity Problem 2-DQCSD-P(n, w, b_1, b_2) asks to decide with non-negligible advantage whether $(\mathbf{H}, \mathbf{y})$ came from the 2-QCSD-P(n, w, b_1, b_2) distribution or the uniform distribution over $\mathbb{F}_2^{n \times 2n} \times \mathbb{F}_2^{n,b_2}$.

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

Both of these problems are believed to be hard [57] and as in other code-base KEM, the best-known attacks are ISD attacks and their variants [13]. Although Combinatorial and Algebraic attacks are studied [56] still the most promising attack, if there are any, remains the ISD.

2.5.3. Major Updates in HQC KEM NIST Submission [13,57]

- Parameter Set Reduction:
  - Initially included three parameter sets for security category 5 (HQC-256-1, HQC-256-2, HQC-256-3) targeting different decryption failure rates.
  - HQC-256-1 was broken during the second round.
  - Now only contains one parameter set per security category with low decryption failure rates.
- Side-Channel Attack Mitigation:
  - Implementations now run in constant time and avoid secret-dependent memory access to counter side-channel attacks.
- Removal of BCH-Repetition Decoder:
  - Removed due to overall improvements from the RMRS decoder.
- Security Proof and Implementation Updates:
  - Updated IND-CPA and IND-CCA2 security proofs and definitions.
  - Incorporated HHK transform with implicit rejection, enhancing IND-CCA2 security.
  - Replaced modulo operator with Barrett reduction to counter timing attacks.
- Handling Multi-Ciphertext Attacks:
  - Modified HQC-128 to include a public salt value in the ciphertext to counter multi-ciphertext attacks.
  - Added counter-measures for key-recovery timing attacks [58] based on Nicolas Sendrier's approach [59].

– Implemented a constant-time C implementation to improve security.

3. Discussion

NIST conducted a workshop on Cybersecurity in the Post-Quantum world in April 2015; then, around February 2016, the call for PQC standardization submission was announced. Fast-forward to December 2017, the first round candidates were announced. In July 2020, the third-round finalists and the alternative candidate algorithms were announced. In the third-round finalization, CRYSTALS-Dilithium was finalized as the PQC standard algorithm for signature creation and CRYSTALS-Kyber as KEM. CRYSTALS-Dilithium and CRYSTALS-Kyber are both lattice-based post-quantum cryptosystems. The algorithm relies on computationally hard problems, such as the shortest vector problem, and module learning with error problems. The classic McEliece, BIKE, and HQQC were suggested as alternative candidates for KEM, and taken to the fourth round for the standardization. These algorithms tackle the hard problems from coding theory, namely, syndrome decoding and code-word finding and their variants, such as quasi-cyclic codes. All the PQC algorithms which are under consideration and have been finalized rely on computationally hard problems, be it either hard problems in lattice or coding theory.

In each round, these algorithms go through significant updates in terms of security, their design, and various routines, which can have possible vulnerabilities. NIST has defined multiple levels of security categories, and based on these, algorithms set different parameters to meet the security level standardized by NIST. Security is the primary criterion NIST uses to evaluate candidate post-quantum algorithms, similarly to the AES and SHA-3 competitions. NIST's public-key standards are employed in various applications, including internet protocols (TLS, SSH, IKE, IPsec, DNSSEC), certificates, code signing, and secure bootloaders. The new standards will ensure post-quantum security for all these applications [13]; hence, the standarization process takes years. In fact, each algorithm has been analyzed comprehensively from all possible angles to declare them either as standard or usable under certain conditions.

NIST requires encryption and key-establishment schemes to provide IND-CCA2 security, with IND-CPA security accepted for ephemeral use cases, and digital signatures to offer EUF-CMA security. The five defined security strength categories are based on the computational resources needed for brute-force attacks against NIST standards for AES and SHA, considering both classical and quantum models [13]. Details about IND-CPA and IND-CCA2 are provided in [13] and each of the algorithm's submissions [24,46,49,57].

NIST has chosen CRYSTALS-Dilithium as one of the primary algorithms for digital signatures in its post-quantum cryptography (PQC) standards. Alongside FALCON, Dilithium stands out for its high efficiency and straightforward implementation, leveraging pseudo-randomness and truncated storage techniques for enhanced performance. Notably, Dilithium avoids the need for floating-point arithmetic, setting it apart from FALCON, which generally has shorter keys and signatures. Despite minor adjustments to parameter sets and addressing dual-attack vulnerabilities during the third round of evaluations, Dilithium maintains a robust security profile. Its strong theoretical foundation, combined with a broad range of cryptographic applications, positions Dilithium as a key component in ensuring long-term security against quantum attacks. Even though Dilithium has been standardized, as per [13], NIST looks forward to standardizing FALCON and SPHINCH++ as signature schema as well.

NIST faced a difficult choice between Kyber, NTRU, and Saber due to their comparable design, security, and performance, ultimately favoring Kyber for its reliance on the more convincing MLWE problem and detailed security analysis, despite patent-related challenges. NIST will continue evaluating KEM candidates (BIKE, Classic McEliece, HQC, SIKE) in

the fourth round, considering BIKE and HQC as general-purpose options, while SIKE is attractive for its small key and ciphertext sizes (but in August 2022, it was shown that SIKE is no longer secure [60]) and Classic McEliece is not being prioritized due to its large public key size.

A comparative summary of the considered post-quantum cryptographic schemes is presented in Table 10. The table shows each scheme's underlying security basis, key and ciphertext sizes, performance characteristics, and practical use cases. From the comparison, it is evident that BIKE and HQC, both code-based KEMs, offer robust security with differing efficiency profiles, making them suitable for constrained devices and high-security communications, respectively. Classic McEliece, while featuring exceptionally fast encapsulation and decapsulation, suffers from significantly large public keys, which limits its applicability to scenarios such as archival encryption. In contrast, Dilithium is a lattice-based digital signature scheme rather than a KEM, offering a balance between key size and performance, with particular strength in verification speed, making it ideal for blockchain and large-scale authentication systems. This comparative view facilitates the selection of an appropriate scheme based on application-specific constraints, such as key size limitations, computational resources, and long-term security requirements.

Table 10. Comparison of selected post-quantum cryptographic schema.

Scheme	Details
BIKE	Security Basis: QC–MDPC decoding Key Size: Medium / Large Ciphertext Size: Medium Performance: Fast key generation with moderate encapsulation/decapsulation speeds; efficient in constrained environments. Use Case: IoT devices; constrained networks.
HQC	Security Basis: QC–LDPC decoding Key Size: Large / Large Ciphertext Size: Large Performance: Robust, constant-time operations with balanced key generation and encapsulation performance. Use Case: High-security communications; long-term confidentiality.
Classic McEliece	Security Basis: Binary Goppa decoding Key Size: Very Large / Small Ciphertext Size: Small Performance: Extremely fast encapsulation/decapsulation but slow key generation due to large public keys. Use Case: Archival encryption; secure data storage.
Dilithium	Security Basis: Lattice (Module-LWE) Key Size: Medium / Medium Ciphertext Size: N/A (signature scheme) Performance: Very fast verification with moderate signing times; scalable for large deployments. Use Case: Digital signatures; blockchain; software authentication.

3.1. The Way Forward

NIST continues to look forward to evaluating the KEM candidates in the 4th round and standardising the best candidates in terms of security, performance, and stability. The evaluation of the HQC, BIKE, and McEliece algorithms reveals distinct strengths and targeted updates to address vulnerabilities and improve performance. HQC has made significant strides in parameter set reduction, side-channel attack mitigation, and multi-ciphertext

attack-handling, enhancing security and efficiency. BIKE has streamlined its implementation by focusing on a single variant and enhancing security with updated parameters and SHA-3-based constructions. Its simplification of the specification and introduction of data-oblivious sampling underscore its adaptability and focus on practical security measures. McEliece stands out for its enduring security stability and efficiency improvements, with a strong theoretical foundation and practical advancements to counter modern threats, including QROM attacks. Its conversion to a KEM and the Classic McEliece design further reinforce its high-security profile. Collectively, these updates and characteristics highlight the algorithms' potential for post-quantum cryptographic applications, ensuring protection against emerging quantum threats.

3.2. Industrial Adoption of PQC

The adoption of Post-Quantum Cryptography (PQC) in industry can be approached through two primary integration models: fresh integration and hybrid integration [61].

In the fresh integration model, PQC algorithms are introduced into newly developed systems without dependence on existing classical cryptographic schemes. This approach allows for a clean-slate design, where system architectures can be fully optimized for the performance, resource, and security characteristics of PQC algorithms. Fresh integration is particularly suited for emerging industries or greenfield deployments such as new blockchain infrastructures, IoT deployments, or next-generation secure communications where there is no requirement to maintain backward compatibility. Within this model, several strategies can be employed [61]: (i) Direct Integration of Quantum-Safe KEMs such as CRYSTALS-Kyber to replace RSA and ECC in key exchange protocols like TLS; (ii) Redesign of Protocols (e.g., SSL, TLS) to rely exclusively on PQC standards, streamlining operations and removing hybrid complexities; and (iii) Simplification of Validation Processes by replacing traditional validation algorithms with PQC-optimized mechanisms for improved security assurance. Higher communication layer strategies in this context include adopting PQC-based Signatures (e.g., CRYSTALS-Dilithium) as the primary authentication method, using PQC Encryption protocols to secure sensitive data long-term, and implementing Quantum-Safe Authentication Systems for verification and certificate issuance [61].

In contrast, the hybrid integration model combines PQC algorithms with classical cryptographic primitives (e.g., RSA, ECC) to form hybrid protocols that simultaneously offer protection against both classical and quantum adversaries. This model facilitates gradual migration and minimizes operational disruption, making it highly practical for industries with large legacy infrastructures, such as banking networks, critical infrastructure control systems, and government communication channels. Hybrid approaches also allow systems to benefit from the maturity and established trust of classical algorithms while incorporating quantum-safe protections. Common methods within this model include the integration of PQC algorithms into existing protocol stacks via `liboqs` or the Open Quantum Safe (OQS) framework [62], which enables systematic benchmarking [62] of PQC schemes for key generation, encryption/decryption, signature verification, handshake latency, and protocol compliance under realistic network conditions [61,62]. Additionally, Manual Integration approaches allow industry-specific modification of cryptographic libraries OpenSSL [63] to meet unique security policies and performance targets, offering maximum control over algorithm selection and protocol parameters [61].

Frameworks such as OPENQS (Open Quantum-Safe) [62] further support both models by offering a flexible and open-source reference architecture for the implementation, testing, and evaluation of PQC algorithms [62]. OPENQS enables developers to experiment with different PQC schemes, test performance under various workloads, and integrate hybrid

cryptographic constructs with minimal friction. Using OPENQS, industries can develop migration roadmaps that are both technically robust and economically viable.

In practice, the choice between fresh and hybrid integration depends on multiple factors, including regulatory requirements, performance budgets, interoperability constraints, and long-term infrastructure goals [61,64,65]. Table 10 provides a comparative overview of major NIST-selected PQC schemes, helping decision-makers align technical capabilities with their operational priorities.

4. Conclusions

In this review, we examined several major post-quantum cryptographic algorithms that have either been standardized or are under active consideration in the NIST PQC process. These include CRYSTALS-Dilithium for digital signatures, and Classic McEliece, BIKE, and HQC for key encapsulation mechanisms, all of which remain strong candidates in the ongoing fourth-round evaluations. While these schemes continue to evolve in response to emerging cryptanalytic techniques, their standardization trajectory signals a maturing landscape in which viable, quantum-resistant solutions are becoming operationally ready. In practical deployments, the choice among these schemes often relies on distinct trade-offs. CRYSTALS-Dilithium offers balanced performance and compact key/signature sizes, making it suitable for general-purpose digital signatures in constrained environments. In contrast, Classic McEliece provides exceptional resilience against quantum attacks but at the cost of very large public keys, making it better suited for applications with ample storage and bandwidth. BIKE and HQC, with their moderate key sizes and performance profiles, present viable alternatives for use cases where both security and efficiency must be balanced, particularly in resource-sensitive communication systems.

The inevitability of large-scale quantum computing demands proactive preparation. For industries, this means not only monitoring algorithmic developments but also beginning structured adoption of PQC technologies—either through fresh integration in greenfield systems or hybrid deployment within existing infrastructures. Frameworks such as OPENQS and implementations like `liboqs` provide the practical tooling necessary to benchmark performance, validate interoperability, and design migration strategies tailored to specific operational and regulatory contexts.

As quantum threats move from theoretical to practical, the protection of sensitive communications, data at rest, and long-term digital assets will depend on timely and strategic PQC adoption. The transition requires a balanced approach that accounts for performance, interoperability, and security longevity, but delay carries the risk of retrospective vulnerability exposure. By initiating migration planning now, organizations can ensure that their infrastructures are equipped to withstand both current and future adversaries, safeguarding the confidentiality, integrity, and authenticity of digital communications in the post-quantum era.

Author Contributions: Conceptualization, V.P.O., S.C. and S.Y.; methodology, V.P.O., S.C. and S.Y.; software, V.P.O.; validation, S.C. and S.Y.; formal analysis, V.P.O., S.C. and S.Y.; investigation, V.P.O., S.C., S.Y. and D.C.; resources, S.Y. and D.C.; data curation, V.P.O.; writing—original draft preparation, V.P.O., S.C. and S.Y.; writing—review and editing, V.P.O., S.C., S.Y. and D.C.; supervision, S.C., S.Y. and D.C.; project administration, V.P.O., S.C., S.Y. and D.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: No new data were created or analyzed in this study.

Acknowledgments: The authors thank the Naoris technical team for their time and effort in examining the paper and their constructive comments. We also thank the reviewers for meticulously reviewing

the manuscript and providing feedback. The first author thanks Sanjay Rijal for painstakingly going through all spelling, grammatical errors, and every tiny detail hidden in the labyrinth of this manuscript.

Conflicts of Interest: Author Sumit Chauhan and David Carvalho were employed by the company Naoris Tech Inc. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Glossary

Glossary of Terminologies and Abbreviations

Abbreviation/Term	**Full Form / Description**
PQC	Post-Quantum Cryptography
NIST	National Institute of Standards and Technology
PKE	Public-Key Encryption
KEM	Key Encapsulation Mechanism
CRYSTALS	Cryptographic Suite for Algebraic Lattices
Dilithium	Lattice-based digital signature scheme under CRYSTALS
McEliece	Code-based cryptosystem using Goppa codes
BIKE	Bit Flipping Key Encapsulation
HQC	Hamming Quasi-Cyclic scheme
SVP	Shortest Vector Problem
CVP	Closest Vector Problem
LWE	Learning With Errors
MLWE	Module Learning With Errors
SIS	Short Integer Solution problem
MSIS	Module Short Integer Solution problem
SelfTargetMSIS	Self-Target Module-SIS problem
NTT	Number Theoretic Transform
FFT	Fast Fourier Transform
AES	Advanced Encryption Standard
SHA	Secure Hash Algorithm
SHAKE	Secure Hash Algorithm Keccak
RSA	Rivest–Shamir–Adleman algorithm
DH	Diffie–Hellman key exchange
ECDH	Elliptic Curve Diffie–Hellman
ECDSA	Elliptic Curve Digital Signature Algorithm
DSA	Digital Signature Algorithm
GNFS	General Number Field Sieve
QKD	Quantum Key Distribution
APICTA	Asia Pacific ICT Alliance
GF(q)	Galois Field of order q
Zq	Finite field with q elements (integers modulo q)
Zq[x]	Polynomial ring over the finite field Z_q
R	Generic Ring
Field	Algebraic structure where every nonzero element has a multiplicative inverse
Ideal	Subset of a ring closed under addition and absorption by multiplication
Quotient Ring	Ring formed by partitioning a ring by an ideal
Lattice	Discrete subgroup of a vector space spanning the space
Module	Generalization of vector space with scalars from a ring
Module Lattice	Submodule of an R-module that forms a lattice
Goppa Code	Linear error-correcting code defined using a Goppa polynomial

Hamming Distance	Number of differing positions between two codewords
Hamming Weight	Number of nonzero entries in a codeword
Generator Matrix (G)	Matrix whose rows form a basis of a linear/code subspace
Syndrome	Vector used in decoding error-correcting codes
Patterson's Algorithm	Decoding algorithm for Goppa codes
Cyclotomic Polynomial	Polynomial used in quotient rings such as $X^n + 1$
Power2Round	Function to decompose coefficients into high/low bits
Module-SIS	Hardness assumption based on module short integer solutions
Fisher–Yates Shuffle	Algorithm for generating random permutations
Regev's LWE	Public-key encryption scheme based on Learning With Errors
BKZ	Block Korkine–Zolotarev lattice reduction algorithm
GSA	Geometric Series Assumption (used in lattice cryptanalysis)
Core-SVP	Security estimation metric for lattice problems
Bit-reversal Permutation	Reordering used in NTT/FFT computations
DFR	Decryption Failure Rate

References

1. Shor, P. The Early Days of Quantum Computation. *arXiv* **2022**, arXiv:2208.09964. [CrossRef]
2. Integer Factorization Algorithms: A Comparative Analysis. Available online: https://softwaredominos.com/home/science-technology-and-other-fascinating-topics/integer-factorization-algorithms-a-comparative-analysis/ (accessed on 26 July 2025).
3. Blunt, N.; Camps, J.; Crawford, O.; Izsák, R.; Leontica, S.; Mirani, A.; Moylett, A.; Scivier, S.; Sünderhauf, C.; Schopf, P.; et al. Perspective on the Current State-of-the-Art of Quantum Computing for Drug Discovery Applications. *J. Chem. Theory Comput.* **2022**, *18*, 7001–7023. [CrossRef] [PubMed]
4. Wittek, P. 4—Quantum Computing. In *Quantum Machine Learning*; Wittek, P., Ed.; Academic Press: Boston, MA, USA, 2014; pp. 41–53. [CrossRef]
5. Bernstein, D.; Lange, T. *Post-Quantum Cryptography: Dealing with the Fallout of Physics Success*; Cryptology ePrint Archive; IACR: Bellevue, WA, USA, 2017.
6. Kearney, J.J.; Perez-Delgado, C.A. Vulnerability of blockchain technologies to quantum attacks. *Array* **2021**, *10*, 100065. [CrossRef]
7. Chauhan, S.; Ojha, V.P.; Yarahmadian, S.; Carvalho, D. Towards Building Quantum Resistant Blockchain. In Proceedings of the 2023 International Conference on Electrical, Computer and Energy Technologies (ICECET), Cape Town, South Africa, 16–17 November 2023; pp. 1–9. [CrossRef]
8. Brooks, M. Quantum Computers: What Are They Good For? *Nature* **2023,5**, *617*, S1–S3. [CrossRef] [PubMed]
9. Nature Research Custom Media. Fast Tracking Quantum-Computing Tech. 2023. Available online: https://www.nature.com/articles/d42473-023-00091-y (accessed on 7 January 2023).
10. Beverland, M.; Murali, P.; Troyer, M.; Svore, K.; Hoefler, T.; Kliuchnikov, V.; Low, G.; Soeken, M.; Sundaram, A.; Vaschillo, A. Assessing Requirements to Scale to Practical Quantum Advantage. *arXiv* **2022**, arXiv:2211.07629. [CrossRef]
11. Post-Quantum Cryptography | CSRC. Available online: https://csrc.nist.gov/projects/post-quantum-cryptography (accessed on 6 October 2024).
12. Allende, M.; León, D.L.; Cerón, S.; Leal, A.; Pareja, A.; Silva, M.D.; Pardo, A.; Jones, D.; Worrall, D.; Merriman, B.; et al. Quantum-resistance in blockchain networks. *Sci. Rep.* **2023**, *13*, 5664. [CrossRef]
13. Alagic, G.; Apon, D.; Cooper, D.; Dang, Q.; Dang, T.; Kelsey, J.; Lichtinger, J.; Liu, Y.K.; Miller, C.; Moody, D.; et al. *Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process*; NIST: Gaithersburg, MD, USA, 2022. [CrossRef]
14. Post-Quantum Cryptography | CSRC. Available online: https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions (accessed on 6 October 2024).
15. Regev, O. The Learning with Errors Problem. *Invit. Surv. CCC* **2010**, *7*, 11.
16. Singh, H. Code based Cryptography: Classic McEliece. *arXiv* **2019**, arXiv:1907.12754.
17. Sabani, M.E.; Savvas, I.K.; Poulakis, D.; Garani, G.; Makris, G.C. Evaluation and Comparison of Lattice-Based Cryptosystems for a Secure Quantum Computing Era. *Electronics* **2023**, *12*, 2643. [CrossRef]
18. Post-Quantum Cryptography | CSRC. Available online: https://csrc.nist.gov/Projects/post-quantum-cryptography/round-4-submissions (accessed on 6 October 2024).
19. Dilithium Official Website. Available online: https://pq-crystals.org/dilithium/index.shtml (accessed on 3 December 2024).
20. Chen, L.; Jordan, S.; Liu, Y.K.; Moody, D.; Peralta, R.; Perlner, R.; Smith-Tone, D. *Report on Post-Quantum Cryptography*; Technical Report; US Department of Commerce, National Institute of Standards and Technology: Gaithersburg, MD, USA, 2016. [CrossRef]

21. Li, S.; Chen, Y.; Chen, L.; Liao, J.; Kuang, C.; Li, K.; Liang, W.; Xiong, N. Post-Quantum Security: Opportunities and Challenges. *Sensors* **2023**, *23*, 8744. [CrossRef]
22. Gentry, C.; Peikert, C.; Vaikuntanathan, V. Trapdoors for hard lattices and new cryptographic constructions. In Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing, Victoria, BC, Canada, 17–20 May 2008; STOC '08; pp. 197–206. [CrossRef]
23. Lyubashevsky, V. Fiat-Shamir with Aborts: Applications to Lattice and Factoring-Based Signatures. In *Advances in Cryptology—ASIACRYPT 2009*; Matsui, M., Ed.; Springer: Berlin/Heidelberg, Germany, 2009; pp. 598–616.
24. CRYSTALS-Dilithium Specification Round 3. Available online: https://pq-crystals.org/dilithium/data/dilithium-specification-round3-20210208.pdf (accessed on 6 December 2024).
25. Number Theory-The Chinese Remainder Theorem—Crypto.Stanford.Edu. Available online: https://crypto.stanford.edu/pbc/notes/numbertheory/crt.html (accessed on 9 August 2025).
26. Cooley, J.W.; Tukey, J.W. An algorithm for the machine calculation of complex Fourier series. *Math. Comput.* **1965**, *19*, 297–301. [CrossRef]
27. Jackson, K.A.; Miller, C.A.; Wang, D. Evaluating the security of CRYSTALS-Dilithium in the quantum random oracle model. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*; Springer Nature: Cham, Switzerland, 2024.
28. Wang, R.; Ngo, K.; Gärtner, J.; Dubrova, E. Single-Trace Side-Channel Attacks on CRYSTALS-Dilithium: Myth or Reality? Cryptology ePrint Archive, Paper 2023/1931. 2023. Available online: https://eprint.iacr.org/2023/1931 (accessed on 26 August 2025).
29. Krahmer, E.; Pessl, P.; Land, G.; Güneysu, T. Correction Fault Attacks on Randomized CRYSTALS-Dilithium. Cryptology ePrint Archive, Paper 2024/138. 2024. Available online: https://eprint.iacr.org/2024/138 (accessed on 26 August 2025).
30. Ravi, P.; Chattopadhyay, A.; D'Anvers, J.P.; Baksi, A. Side-channel and Fault-injection attacks over Lattice-based Post-quantum Schemes (Kyber, Dilithium): Survey and New Results. *ACM Trans. Embed. Comput. Syst.* **2024**, *23*, 1–54. [CrossRef]
31. Chen, Z.; Karabulut, E.; Aysu, A.; Ma, Y.; Jing, J. An Efficient Non-Profiled Side-Channel Attack on the CRYSTALS-Dilithium Post-Quantum Signature. In Proceedings of the 2021 IEEE 39th International Conference on Computer Design (ICCD), Virtual, 24–27 October 2021; pp. 583–590. [CrossRef]
32. Groot Bruinderink, L.; Pessl, P. Differential Fault Attacks on Deterministic Lattice Signatures. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2018**, *2018*, 21–43. [CrossRef]
33. Islam, S.; Mus, K.; Singh, R.; Schaumont, P.; Sunar, B. Signature Correction Attack on Dilithium Signature Scheme. In Proceedings of the 2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P), Genoa, Italy, 6–10 June 2022.
34. Ravi, P.; Jhanwar, M.P.; Howe, J.; Chattopadhyay, A.; Bhasin, S. Exploiting Determinism in Lattice-Based Signatures-Practical Fault Attacks on pqm4 Implementations of NIST Candidates. Cryptology ePrint Archive, Paper 2019/769. 2019. Available online: https://eprint.iacr.org/2019/769 (accessed on 5 June 2024).
35. Kiltz, E.; Lyubashevsky, V.; Schaffner, C. A Concrete Treatment of Fiat-Shamir Signatures in the Quantum Random-Oracle Model. In *Advances in Cryptology—EUROCRYPT 2018*; Nielsen, J.B., Rijmen, V., Eds.; Springer: Cham, Switzerland, 2018; pp. 552–586.
36. Classic McEliece: Intro. Available online: https://classic.mceliece.org/index.html (accessed on 17 May 2024).
37. Patterson, N.J. The algebraic decoding of Goppa codes. *IEEE Trans. Inf. Theory* **1975**, *21*, 203–207. [CrossRef]
38. McElice Specification Round 4. Available online: https://classic.mceliece.org/mceliece-spec-20221023.pdf (accessed on 6 December 2024).
39. Gnan, N. Overview of the Mceliece Cryptosystem and Its Security. Available online: https://mysite.science.uottawa.ca/mnevins/papers/NoliveGnan2021McEliece.pdf (accessed on 13 July 2024).
40. Prange, E. The use of information sets in decoding cyclic codes. *IRE Trans. Inf. Theory* **1962**, *8*, 5–9. [CrossRef]
41. Chou, T. Code-Based Cryptography. Available online: https://troll.iis.sinica.edu.tw/school20/slides/Codes.pdf (accessed on 13 July 2024).
42. Lee, P.J.; Brickell, E.F. An Observation on the Security of McEliece's Public-Key Cryptosystem. In *Advances in Cryptology—EUROCRYPT '88*; Barstow, D., Brauer, W., Brinch Hansen, P., Gries, D., Luckham, D., Moler, C., Pnueli, A., Seegmüller, G., Stoer, J., Wirth, N., et al., Eds.; Springer: Berlin/Heidelberg, Germany, 1988; pp. 275–280.
43. Leon, J. A probabilistic algorithm for computing minimum weights of large error-correcting codes. *IEEE Trans. Inf. Theory* **1988**, *34*, 1354–1359. [CrossRef]
44. May, A.; Meurer, A.; Thomae, E. Decoding random linear codes in $\tilde{O}(20.054n)$. In Proceedings of the 17th International Conference on The Theory and Application of Cryptology and Information Security, Seoul, Republic of Korea, 4–8 December 2011; ASIACRYPT'11; pp. 107–124. [CrossRef]
45. Stern, J. A method for finding codewords of small weight. In *Coding Theory and Applications*; Cohen, G., Wolfmann, J., Eds.; Springer: Berlin/Heidelberg, Germany, 1989; pp. 106–113.
46. Repka, M.; Zajac, P. Overview of the McEliece cryptosystem and its security. *Tatra Mt. Math. Publ.* **2014**, *60*, 57–83. [CrossRef]

47. McElice Submission Details- Round 4. Available online: https://classic.mceliece.org/nist/mceliece-submission-20221023.pdf (accessed on 13 July 2024).
48. BIKE—Bit Flipping Key Encapsulation. Available online: https://bikesuite.org/#spec (accessed on 6 March 2024).
49. BIKE—Bit Flipping Key Encapsulation Round 4 Specification. Available online: https://bikesuite.org/files/v5.2/BIKE_Spec.2024.10.10.1.pdf (accessed on 6 March 2024).
50. Vasseur, V. QC-MDPC Codes DFR and the IND-CCA Security of Bike. 2022. Available online: https://eprint.iacr.org/2021/1458 (accessed on 26 August 2025).
51. Vasseur, V. Post-Quantum Cryptography: A Study of the Decoding of QC-MDPC Codes. Ph.D. Thesis, Université de Paris, Paris, France, 2021.
52. Guo, Q.; Johansson, T.; Stankovski, P. A key recovery attack on MDPC with CCA security using decoding errors. In *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Proceedings of the Advances in Cryptology—ASIACRYPT 2016—22nd International Conference on the Theory and Application of Cryptology and Information Security, Proceedings, Hanoi, Vietnam, 4–8 December* 2016; Springer: Berlin/Heidelberg, Germany, 2016; Volume 10031 LNCS, pp. 789–815. [CrossRef]
53. Gómez, J.D.F. HQC: A New Post-Quantum Cryptography Standard Based on Codes—Cyte.co. Available online: https://www.cyte.co/en/post/hqc-a-new-post-quantum-cryptography (accessed on 16 August 2025).
54. Bombar, M.; Resch, N.; Wiedijk, E. On the Independence Assumption in Quasi-Cyclic Code-Based Cryptography. *arXiv* **2025**, arXiv:2501.02626.
55. Alekhnovich, M. More on Average Case vs Approximation Complexity. *Comput. Complex.* **2011**, *20*, 755–786. [CrossRef]
56. Aguilar-Melchor, C.; Blazy, O.; Deneuville, J.C.; Gaborit, P.; Zémor, G. Efficient Encryption From Random Quasi-Cyclic Codes. *IEEE Trans. Inf. Theory* **2018**, *64*, 3927–3943. [CrossRef]
57. Melchor, C.; Blazy, O.; Deneuville, J.C.; Gaborit, P.; Zémor, G. Hamming Quasi-Cyclic (HQC) Fourth Round Version Updated Version 23/02/2024. Available online: https://pqc-hqc.org/doc/hqc_specifications_2025_08_22.pdf (accessed on 20 June 2024).
58. Guo, Q.; Hlauschek, C.; Johansson, T.; Lahr, N.; Nilsson, A.; Schröder, R.L. Don't Reject This: Key-Recovery Timing Attacks Due to Rejection-Sampling in HQC and BIKE. Cryptology ePrint Archive, Paper 2021/1485. 2021. Available online: https://eprint.iacr.org/2021/1485 (accessed on 26 August 2025).
59. Sendrier, N. Secure Sampling of Constant-Weight Words—Application to BIKE. Cryptology ePrint Archive, Paper 2021/1631. 2021. Available online: https://eprint.iacr.org/2021/1631 (accessed on 25 June 2024).
60. SIKE Foreword and Postscript. Available online: https://csrc.nist.gov/csrc/media/Projects/post-quantum-cryptography/documents/round-4/submissions/sike-team-note-insecure.pdf (accessed on 16 June 2024).
61. Ojha, V.P.; Chauhan, S.; Yarahmadhian, S.; Carvalho, D. Adoption of Post-Quantum Cryptography in Communication Technologies. In Proceedings of the 5th IFSA Winter Conference on Automation, Robotics & Communications for Industry 4.0/5.0 (ARCI' 2025), Granada, Spain, 19–21 February 2025.
62. Safe, O.Q. Open Quantum Safe. Available online: https://openquantumsafe.org/ (accessed 12 August 2025).
63. Foundation, O. OpenSSL. Available online: https://www.openssl.org/ (accessed 12 August 2025).
64. Joseph, D.; Misoczki, R.; Manzano, M.; Tricot, J.; Pinuaga, F.D.; Lacombe, O.; Leichenauer, S.; Hidary, J.; Venables, P.; Hansen, R. Transitioning organizations to post-quantum cryptography. *Nature* **2022**, *605*, 237–243. [CrossRef] [PubMed]
65. Mamatha, G.; Dimri, N.; Sinha, R. Post-quantum cryptography: Securing digital communication in the quantum era. *arXiv* **2024**, arXiv:2403.11741. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.