# ETSI TR 103 823 V1.1.2 (2021-10)


**CYBER;**
**Quantum-Safe Public-Key Encryption and Key Encapsulation**

Reference
DTR/CYBER-QSC-0017rev

Keywords
algorithm, cybersecurity

***ETSI***

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - APE 7112B
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° w061004871

# Contents

# Intellectual Property Rights

## Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The declarations pertaining to these essential IPRs, if any, are publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (https://ipr.etsi.org/).

Pursuant to the ETSI Directives including the ETSI IPR Policy, no investigation regarding the essentiality of IPRs, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

## Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

**DECT™**, **PLUGTESTS™**, **UMTS™** and the ETSI logo are trademarks of ETSI registered for the benefit of its Members. **3GPP™** and **LTE™** are trademarks of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners. **oneM2M™** logo is a trademark of ETSI registered for the benefit of its Members and of the oneM2M Partners. **GSM**® and the GSM logo are trademarks registered and owned by the GSM Association.

# Foreword

This Technical Report (TR) has been produced by ETSI Technical Committee Cyber Security (CYBER).

# Modal verbs terminology

In the present document "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

# 1 Scope

The present document provides technical descriptions of the Public-Key Encryption (PKE) and Key Encapsulation Mechanisms (KEMs) submitted to the National Institute for Standards and Technology (NIST) for the third round of their Post-Quantum Cryptography (PQC) standardization process.

# 2 References

## 2.1 Normative references

Normative references are not applicable in the present document.

## 2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] NIST FIPS 197: "Advanced Encryption Standard (AES)".

[i.2] NIST FIPS 180-4: "Secure Hash Standard".

[i.3] NIST FIPS 202: "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions".

[i.4] NIST IR 8105: "Report on Post-Quantum Cryptography".

[i.5] NIST FIPS 186-4: "Digital Signature Standard (DSS)".

[i.6] NIST SP-56A: "Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography".

[i.7] NIST SP-56B: "Recommendation for Pair-Wise Key Establishment Schemes Using Integer Factorization Cryptography".

[i.8] NIST: "Submission Requirements and Evaluation Criteria for the Post-Quantum Cryptography Standardization Process", December 2016.

NOTE: Available at https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf.

[i.9] NIST Post-Quantum Cryptography Standardization: "Round 1 Submissions".

NOTE: Available at https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Round-1-Submissions.

[i.10] NIST IR 8240: "Status Report on the First Round of the NIST Post-Quantum Standardization Process".

[i.11] NIST Post-Quantum Cryptography Standardization: "Round 2 Submissions".

NOTE: Available at https://csrc.nist.gov/Projects/post-quantum-cryptography/round-2-submissions.

[i.12] NIST IR 8309: "Status Report on the Second Round of the NIST Post-Quantum Standardization Process".

[i.13] NIST Post-Quantum Cryptography Standardization: "Round 3 Submissions".

NOTE: Available at https://csrc.nist.gov/Projects/post-quantum-cryptography/round-3-submissions.

[i.14] ETSI TR 103 616: "CYBER; Quantum-Safe Signatures".

[i.15] E. Fujisaki and T. Okamoto: "Secure integration of asymmetric and symmetric encryption schemes", CRYPTO, 1999.

[i.16] D. Hofheinz, K. Hövelmanns and E. Kiltz: "A modular analysis of the Fujisaki-Okamoto transformation", TCC, 2017.

[i.17] N. Drucker, Shay Gueron and D. Kostic: "QC-MDPC decoders with several shades of gray", PQCrypto, 2020.

[i.18] R. Canto Torres and N. Sendrier: "Analysis of information set decoding for a sub-linear error weight", PQCrypto, 2016.

[i.19] N. Bindel, M. Hamburg, K. Hövelmanns, A. Hülsing, and E. Persichetti: "Tighter proofs of CCA security in the quantum random oracle model", TCC, 2019.

[i.20] M.R. Albrecht, V. Gheorghiu, E.W. Postlethwaite and J.M. Schanck: "Estimating quantum speedups for lattice sieves", Cryptology ePrint Archive, Report 2019/1161, 2019.

[i.21] E. Prange: "The use of information sets in decoding cyclic codes", IRE Transactions on Information Theory 8.5 (1962): 5-9.

[i.22] P.J. Lee and E.F. Brickell: "An observation on the security of McEliece's public-key cryptosystem", EUROCRYPT, 1988.

[i.23] J. Stern: "A method for finding codewords of small weight", International Colloquium on Coding Theory and Applications. Springer, Berlin, Heidelberg, 1988.

[i.24] A. May, A. Meurer and E. Thomae: "Decoding random linear codes in $\tilde{O}(2^{0.054n})$", ASIACRYPT, 2011.

[i.25] A. Becker, A. Joux, A. May and A. Meurer: "Decoding random binary linear codes in 2^(n/20): How 1 + 1 = 0 improves information set decoding", EUROCRYPT, 2012.

[i.26] N. Sendrier: "Decoding one out of many", PQCrypto, 2011.

[i.27] D.J. Bernstein: "Grover vs. McEliece", PQCrypto, 2010.

[i.28] G. Kachigar and J.-P. Tillich: "Quantum information set decoding algorithms", PQCrypto, 2017.

[i.29] M. Naehrig and J. Renes: "Dual isogenies and their application to public-key compression for isogeny-based cryptography", ASIACRYPT, 2019.

[i.30] G. Pereira, J. Doliskani and D. Jao: "x-only point addition formula and faster torsion basis generation in compressed SIKE", Cryptology ePrint Archive, Report 2020/431, 2020.

# 3 Definition of terms, symbols and abbreviations

## 3.1 Terms

For the purposes of the present document, the following terms apply:

**weight:** number of non-zero components of a vector or the number of non-zero coefficients of a polynomial

## 3.2 Symbols

For the purposes of the present document, the following symbols apply:

| | |
|---|---|
| $\boldsymbol{M}$ | Bold upper-case letters denote matrices (over some ring or field) |
| $\boldsymbol{M}^T$ | The transpose of the matrix $\boldsymbol{M}$ |
| $\boldsymbol{I}_k$ | The $k \times k$ identity matrix |
| $\boldsymbol{v}$ | Bold lower-case letters denote vectors (over some ring or field) |
| $\boldsymbol{v}^T$ | The transpose of the vector $\boldsymbol{v}$ |
| $\langle \mathbf{a}, \mathbf{b} \rangle$ | The inner product of vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ (defined over some common ring) |
| $0_k$ | The all-zero vector consisting of $k$ entries |
| $x \coloneqq y$ | $x$ is assigned the value of $y$ |
| $x = y$ | The values of $x$ and $y$ are equal |
| $x \neq y$ | The values of $x$ and $y$ are not equal |
| $x \parallel y$ | The concatenation of $x$ and $y$ |
| $\oplus$ | Bitwise exclusive or |
| $\perp$ | Failure |
| $\lceil x \rfloor$ | The value of $x$ when rounded to the nearest integer, with ties broken by rounding up |
| $\lfloor x \rceil_{q \to p}$ | Modulus switching of $x$ from modulus $q$ to modulus $p$ |
| $G(x)$ | A cryptographic hash function |
| $H(x)$ | A cryptographic hash function |
| $wt(f)$ | The weight of the polynomial $f$ |
| $\mathbb{F}$ | A finite field |
| $\mathbb{F}_q$ | A finite field modulo $q$ |
| $\mathbb{Z}$ | The ring of integers |
| $\mathbb{Z}_q$ | The ring of integers modulo $q$ |
| $R$ | A ring of polynomials |
| $R_q$ | A ring of polynomials modulo $q$ |
| $R_q^{k \times k}$ | The set of $k \times k$ matrices with coefficients in $R_q$ |
| $R_q^k$ | The set of $1 \times k$ matrices with coefficients in $R_q$ |
| $B_\eta$ | Centered binomial distribution of width $\eta$ |
| $\chi$ | Probability distribution over $\mathbb{Z}$ |

## 3.3 Abbreviations

For the purposes of the present document, the following abbreviations apply:

| | |
|---|---|
| AES | Advanced Encryption Standard |
| BIKE | Bit Flipping Key Exchange |
| BKZ | Blockwise Korkine-Zolotarev |
| CCA | Chosen-Ciphertext Attack |
| CPA | Chosen-Plaintext Attack |
| DEM | Data Encapsulation Mechanism |
| HQC | Hamming Quasi-Cyclic |
| KEM | Key Encapsulation Mechanism |
| KDF | Key Derivation Function |
| LWE | Learning With Errors |
| LWR | Learning With Rounding |
| MLWE | Module Learning With Errors |
| MLWR | Module Learning With Rounding |
| NIST | National Institute of Standards and Technology |
| NTT | Number Theoretic Transform |
| OW-CPA | One-Wayness against Chosen-Plaintext Attack |
| PKE | Public-Key Encryption |
| PQC | Post-Quantum Cryptography |
| PRF | Pseudorandom Function |
| QROM | Quantum Random Oracle Model |
| RLWR | Ring Learning With Rounding |
| ROM | Random Oracle Model |

| | |
|---|---|
| SHA | Secure Hash Algorithm |
| SIDH | Supersingular Isogeny Diffie-Hellman |
| SIKE | Supersingular Isogeny Key Encapsulation |
| SVP | Shortest Vector Problem |
| XOF | Extendable Output Function |

# 4 Introduction

The National Institute of Standards and Technology (NIST), an agency of the U.S. Department of Commerce, is responsible for producing cryptographic standards for the protection of sensitive U.S. Federal Government information. NIST standards, such as the Advanced Encryption Standard (AES) [i.1] and Secure Hash Algorithm (SHA) standards [i.2] [i.3], are used globally in many different protocols and products.

In April 2016 NIST announced [i.4] their intention to augment their existing portfolio of public-key cryptography standards [i.5], [i.6], [i.7] by developing new standards for post-quantum cryptography. In December 2016 they initiated a competition-like process with a call for proposals [i.8] for digital signatures, Public-Key Encryption (PKE) schemes, and Key Encapsulation Mechanisms (KEMs), that will remain secure even in the presence of a cryptographically relevant quantum computer. The goal of the process is to perform several rounds of public evaluation over a three to five-year period, and select one or more acceptable algorithms for standardization based on that evaluation.

NIST's deadline for submissions was November 2017. They received 69 candidates that met the minimum acceptance criteria and submission requirements: 20 digital signature schemes, and 49 PKE schemes and KEMs. Five submissions were quickly broken and formally withdrawn from the process by their designers. This left a total of 64 first round candidates [i.9]. In January 2019 NIST announced [i.10] that 26 candidate algorithms would progress to the second round of evaluation: nine digital signature schemes, and 17 PKE schemes and KEMs [i.11].

In July 2020 NIST announced [i.12] that 15 candidate algorithms would progress to the third round of evaluation. These were split into seven finalists and eight alternate candidates. NIST described the finalists as the algorithms they consider to be the most promising for the majority of use cases, and the most likely to be ready for standardization after the end of the third round. The seven finalists include three digital signature schemes, and four PKE schemes and KEMs. The alternate candidates were described as having potential for future standardization, but most likely after another round of evaluation. The eight alternate candidates include three digital signature schemes, and five PKE schemes and KEMs.

The purpose of the present document is to give concise descriptions of the nine PKE schemes and KEMs remaining in the third round of NIST's standardization process. ETSI TR 103 616 [i.14] provides similar descriptions of the six remaining digital signature schemes.

The four PKE and KEM finalists are:

- **Classic McEliece** (see clause 6.1)
- **KYBER** (see clause 6.2)
- **NTRU** (see clause 6.3)
- **SABER** (see clause 6.4)

The five PKE and KEM alternate candidates are:

- **Bit Flipping Key Exchange (BIKE)** (see clause 7.1)
- **FrodoKEM** (see clause 7.2)
- **Hamming Quasi-Cyclic (HQC)** (see clause 7.3)
- **NTRU Prime** (see clause 7.4)
- **Supersingular Isogeny Key Exchange (SIKE)** (see clause 7.5)

Each of these schemes has a different profile in terms of security properties and performance characteristics, so it is expected that some of these schemes will be more suited to specific deployment scenarios than others.

The descriptions provided in the present document are not intended to be substitutes for the detailed specifications submitted to NIST. Instead, the emphasis is on clear mathematical descriptions that facilitate easy comparison of the different schemes. Implementation details, such as how to encode polynomials as bit strings, have been omitted wherever possible. As such, some of the descriptions differ from the submissions in terms of level of abstraction, use of notation, and choice of variable names. It is expected that details of some of the schemes, such as specific parameter choices, will change during the third round of evaluation, so for consistency the descriptions are based on the official submission packages provided to NIST at the beginning of the third round [i.13].

# 5 Background

## 5.1 Terminology

A PKE scheme consists of a triple of algorithms:

- **Key Generation (PKE.KeyGen).** Returns a new public and private key pair.
- **Encryption (PKE.Enc).** Takes a public key and plaintext as input and returns a ciphertext.
- **Decryption (PKE.Dec).** Takes a private key and ciphertext as input and returns a plaintext.

NOTE 1: Some of the PKE schemes described in the present document use randomized encryption where the same public key and plaintext correspond to many different possible ciphertexts. In these schemes the randomness is derived from an additional input to the encryption process.

NOTE 2: Some of the PKE schemes described in the present document can have decryption failures where the plaintext returned by the decryption process does not match the original plaintext used in encryption. Decryption is assumed to always return a plaintext.

PKE schemes are usually unsuitable for bulk data encryption. Consequently, they are often converted into KEMs where one party encapsulates a session key for another party using the second party's public key. The session key, or a value derived from that key, is subsequently used by both parties to perform bulk data encryption using a (symmetric) Data Encapsulation Mechanism (DEM) such as AES. This approach is often referred to as the KEM/DEM paradigm.

A KEM consists of a triple of algorithms:

- **Key Generation (KEM.KeyGen).** Returns a new public and private key pair.
- **Encapsulation (KEM.Enc).** Takes a public key as input and returns a randomly selected session key and a ciphertext that is an encapsulation of the session key.
- **Decapsulation (KEM.Dec).** Takes as input a private key and a ciphertext and returns a session key.

NOTE 3: Some of the KEM schemes described in the present document can have decapsulation failures where the session key returned by the decapsulation process does not match the encapsulated session key.

In practice, PKE schemes and KEMs usually involve two parties: a sender and a recipient. The sender encrypts data or encapsulates a key for the recipient, using the recipient's public key.

## 5.2 Families of post-quantum algorithms

There are five prominent families of post-quantum algorithms:

- **Code-based schemes.** The security of code-based schemes depends on the difficulty of decoding vectors to find the closest codeword or the shortest error vector. Code-based schemes generally fall into two categories: McEliece-style schemes, which use error correcting codes that can be efficiently decoded given some private information; and noisy ElGamal-style schemes, which use random linear codes. Code-based cryptography lends itself more naturally to the construction of PKE schemes and KEMs than to digital signature algorithms.

- **Lattice-based schemes.** The security of lattice-based schemes depends on the difficulty of finding vectors in a lattice that are relatively short, or relatively close to some target vector. Lattice-based schemes generally fall into two categories: NTRU-style schemes, which use lattices that have been specifically constructed to contain private short vectors; and Learning With Errors (LWE) or Learning With Rounding (LWR) style schemes, which use particular classes of random lattices. Lattice-based cryptography can be used to construct PKE schemes, KEMs, and digital signature algorithms. In many cases lattice-based schemes admit worst-case to average-case security reductions, though these reductions are often not relevant to proposed parameter sets; see Annex A for more information.

- **Multivariate schemes.** The security of multivariate schemes depends on the difficulty of solving systems of quadratic or higher degree multivariate polynomials. Multivariate cryptography lends itself more naturally to the construction of digital signature algorithms than to PKE and KEM schemes.

- **Isogeny-based schemes.** The security of isogeny-based schemes depends on the difficulty of recovering a secret isogeny between a pair of elliptic curves. Isogeny-based cryptography seems to lend itself more naturally to the construction of PKE and KEM schemes than to digital signatures, though there has been some recent progress in this area.

- **Symmetric schemes.** The security of such schemes depends on the security of symmetric cryptographic primitives such as hash functions and block ciphers. Symmetric cryptography only lends itself to the construction of digital signature algorithms. Examples include hash-based signatures, such as SPHINCS+, and the PICNIC digital signature scheme.

Different post-quantum schemes utilize different algebraic structures. In code- and lattice-based cryptography, the introduction of more structure can lead to improved computational performance and reduced bandwidth requirements. However, there is a risk that additional structure could introduce new, more efficient attack possibilities. For example, the most efficient lattice-based schemes, which utilize rings of polynomials, have the most algebraic structure, but because it is unclear how to exploit this additional structure, security costings usually assume that it offers an attacker no extra advantage. Understanding whether additional algebraic structure introduces new attack possibilities, for both code- and lattice-based cryptography, remains an important research topic.

## 5.3 Security categories

NIST have provided guidance on the evaluation criteria they intend to apply to candidate submissions [i.8]. As part of this guidance, they have defined the following security categories in terms of the (classical or quantum) resources required to attack different NIST-approved symmetric primitives:

- **Category 1.** Resources equivalent to or greater than key recovery for AES-128.

- **Category 2.** Resources equivalent to or greater than collision search for SHA3-256.

- **Category 3.** Resources equivalent to or greater than key recovery for AES-192.

- **Category 4.** Resources equivalent to or greater than collision search for SHA3-384.

- **Category 5.** Resources equivalent to or greater than key recovery for AES-256.

NIST recommended that submissions include parameter sets that meet the requirements for categories 1, 2 and/or 3, as they believe that these categories will provide sufficient security for the foreseeable future. However, to demonstrate flexibility, and to protect against future cryptanalytic breakthroughs, NIST also recommended that submissions include at least one parameter set that provides a substantially higher level of security. Submitters were asked to include justifications for the security categories claimed for their proposed parameter sets.

## 5.4 Security properties

The two main security goals that are relevant for PKE schemes and KEMs are referred to as indistinguishability under chosen-plaintext, and indistinguishability under chosen-ciphertext, where the latter provides a stronger notion of security than the former. Both security goals are usually modelled as games:

- **Chosen-Plaintext Attack (CPA) security for PKE.** The attacker selects two plaintexts and is given the corresponding ciphertext for one of them. The attacker's goal is to determine which of the plaintexts was encrypted. The scheme is CPA-secure if the attacker cannot do significantly better than guessing.

- **Chosen-Ciphertext Attack (CCA) security for PKE.** The attacker selects two plaintexts and is given the corresponding ciphertext for one of them. The attacker's goal is to determine which of the plaintexts was encrypted. The attacker is allowed to request the decryption of ciphertexts of their choice, except for the challenge ciphertext. The scheme is CCA-secure if the attacker cannot do significantly better than guessing, even with access to the decryption oracle.

- **Chosen-Plaintext Attack (CPA) security for KEMs.** The attacker is given a ciphertext and either a session key that is encapsulated by that ciphertext, or a uniformly random key. The attacker's goal is to determine whether they have been given the session key, or a random key. The scheme is CPA-secure if the attacker cannot do significantly better than guessing.

- **Chosen-Ciphertext Attack (CCA) security for KEMs.** The attacker is given a ciphertext and either a session key that is encapsulated by that ciphertext, or a uniformly random key. The attacker's goal is to determine whether they have been given the session key, or a random key. The attacker is allowed to request the decapsulation of ciphertexts of their choice, except for the challenge ciphertext. The scheme is CCA-secure if the attacker cannot do significantly better than guessing, even with access to the decryption oracle.

Expanded definitions of these properties and additional definitions are given in Annex B.

There are standard techniques available for converting a CPA-secure PKE scheme into a CCA-secure KEM. The most common approach is to use a variant of the Fujisaki-Okamoto transform [i.15]. Broadly speaking, this usually involves deriving the randomness required for encryption (or encapsulation) from the value to be encrypted (or encapsulated); note that this includes the randomness required for the sender to generate an ephemeral key pair. This allows the recipient to attempt to reconstruct the received ciphertext, and check that the protocol has been followed as expected.

As mentioned above, CCA security is stronger than CPA security: a recipient's public key that is used for encryption or encapsulation in a CPA-secure scheme can only be safely used once, or the security of the scheme could be compromised, but a recipient's public key that is used for encryption or encapsulation in a CCA-secure scheme can be safely reused. If an active adversary is able to reuse a recipient's public key in a CPA-secure scheme, they can send messages that consist of erroneous ciphertexts that will reveal information about the recipient's private key.

NIST have stated that they intend to standardize at least one CCA-secure PKE scheme or KEM for general use, and that they will consider standardizing a CPA-secure PKE scheme or KEM for applications where keys are never reused [i.8]. NIST have not mandated that submissions include proofs of CPA or CCA security, but they will consider proofs where they are made available.

## 5.5 Finalists and alternate candidates at a glance

Table 1 contains a summary of each of the NIST public-key encryption and key encapsulation finalists.

**Table 1: Summary of finalists**

| Scheme | Family | Type | Structure | Categories | | | | | Security | | Comments |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | 1 | 2 | 3 | 4 | 5 | CPA | CCA | |
| Classic McEliece | Codes | McEliece | None | Y | | Y | | Y | Y | Y | NOTE 1 |
| KYBER | Lattice | LWE | Module | Y | | Y | | Y | Y | Y | NOTE 2 |
| NTRU | Lattice | NTRU | Ring | Y | | Y | | | Y | Y | NOTE 3 |
| SABER | Lattice | LWR | Module | Y | | Y | | Y | Y | Y | |
| NOTE 1: Classic McEliece is a merger of the second round Classic McEliece and NTS-KEM submissions.<br>NOTE 2: The KYBER submission states that only the CCA version is to be used in practice.<br>NOTE 3: NTRU is a merger of the first round NTRUEncrypt and NTRU-HRSS-KEM submissions. | | | | | | | | | | | |

Table 2 contains a summary of each of the NIST public-key encryption and key encapsulation alternate candidates.

**Table 2: Summary of alternate candidates**

| Scheme | Family | Type | Structure | Categories | | | | | Security | | Comments |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | 1 | 2 | 3 | 4 | 5 | CPA | CCA | |
| BIKE | Codes | McEliece | Ring | Y | | Y | | Y | Y | | NOTE 1 |
| FrodoKEM | Lattice | LWE | None | Y | | Y | | Y | Y | Y | |
| HQC | Lattice | Random | Ring | Y | | Y | | Y | Y | Y | |
| NTRU Prime | Lattice | NTRU | Field | Y | Y | Y | Y | Y | Y | Y | NOTE 2 |
| SIKE | Other | Isogeny | None | Y | Y | Y | | Y | Y | Y | |
| NOTE 1: The BIKE submission does not formally claim that the proposed parameters are CCA-secure.<br>NOTE 2: The NTRU Prime submission states that only the CCA version is to be used in practice. | | | | | | | | | | | |

# 6 Finalists

## 6.1 Classic McEliece

### 6.1.1 Overview

Classic McEliece is a merger of the Classic McEliece and NTS-KEM submissions from the second round of the NIST standardization process. Classic McEliece consists of a CCA-secure KEM built from a OW-CPA-secure PKE scheme using a variant of the Fujisaki-Okamoto transform from [i.16]. The security of Classic McEliece is based on the difficulty of the syndrome decoding problem for general binary linear codes.

A binary Goppa code is defined by a monic irreducible polynomial $g(X) \in \mathbb{F}_{2^m}[X]$ of degree $t$, and a sequence of $n$ distinct elements $(\alpha_1, \ldots, \alpha_n)$ where $\alpha_i \in \mathbb{F}_{2^m}$. These define a parity-check matrix $\widetilde{\boldsymbol{H}} \in \mathbb{F}_2^{t \times n}$ by setting the $(i, j)$-th entry of $\widetilde{\boldsymbol{H}}$ to be $\alpha_j^{i-1}/g(\alpha_j)$. The matrix $\widetilde{\boldsymbol{H}}$ is associated with a parity-check matrix $\widehat{\boldsymbol{H}} \in \mathbb{F}_2^{mt \times n}$ to define a binary linear code $\mathcal{C} = \{ \boldsymbol{c} \in \mathbb{F}_2^n \mid \widehat{\boldsymbol{H}}\boldsymbol{c}^T = 0\}$ of length $n$ and dimension $k = n - mt$ with an efficient algorithm for decoding up to $t$ errors.

Given a public general parity-check matrix $\boldsymbol{H} \in \mathbb{F}_2^{(n-k) \times n}$ for the code, it is believed to be computationally hard to recover the private Goppa parity-check matrix $\widehat{\boldsymbol{H}} \in \mathbb{F}_2^{(n-k) \times n}$ that allows for fast decoding. It is also believed that without the private parity-check matrix, there are no decoding algorithms that are more efficient than generic information set decoding.

Classic McEliece is defined as a CCA-secure KEM only, as the underlying PKE scheme is a building block that is not intended as a separate submission to the NIST standardization process.

### 6.1.2 Parameters

The main parameters for Classic McEliece are:

- $n$, the code length;
- $t$, the error-correction capability;
- $m$, the degree of the field $\mathbb{F}_{2^m}$; and
- $k = n - mt$, the code dimension.

### 6.1.3 Auxiliary primitives

Classic McEliece makes use of two auxiliary, symmetric primitives:

- $H$, a 256-bit cryptographic hash function; and
- KDF, a key derivation function.

The submission describes how to use SHAKE-256 to instantiate these primitives.

## 6.1.4 Public-key encryption scheme

### 6.1.4.1 McEliece.PKE.KeyGen

Input: None

Output: Public key $pk$
Private key $sk$

1) Sample a uniformly random monic irreducible polynomial $g(X) \in \mathbb{F}_{2^m}[X]$ of degree $t$.

2) Sample $(\alpha_1, \ldots, \alpha_n)$, a uniformly random sequence of $n$ distinct elements of $\mathbb{F}_{2^m}$.

3) Construct the parity-check matrix $\widetilde{\boldsymbol{H}} \in \mathbb{F}_{2^m}^{t \times n}$.

4) Convert $\widetilde{\boldsymbol{H}}$ to a parity-check matrix $\widehat{\boldsymbol{H}} \in \mathbb{F}_2^{(n-k) \times n}$.

5) Row reduce $\widehat{\boldsymbol{H}}$ to systematic form $\boldsymbol{H} \coloneqq [\boldsymbol{I}_{n-k} \quad \boldsymbol{H}'] \in \mathbb{F}_2^{(n-k) \times n}$.

The public key is $pk \coloneqq \boldsymbol{H}' \in \mathbb{F}_2^{(n-k) \times k}$. The private key is $sk \coloneqq (g, \alpha_1, \ldots, \alpha_n)$.

NOTE 1: In Step 5), if $\widehat{\boldsymbol{H}}$ cannot be row reduced to systematic form then key generation is restarted.

NOTE 2: The submission describes a version of Classic McEliece that allows the parity-check matrix in Step 5) to be computed in semi-systematic form. This is designed to decrease the failure probability of Step 5).

### 6.1.4.2 McEliece.PKE.Enc

Input: Public key $pk$
Vector $\boldsymbol{e}$ (weight $t$)

Output: Ciphertext $\boldsymbol{c}$

1) Parse the public key as $pk = \boldsymbol{H}' \in \mathbb{F}_2^{(n-k) \times k}$.

2) Construct the parity-check matrix $\boldsymbol{H} \coloneqq [\boldsymbol{I}_{n-k} \quad \boldsymbol{H}'] \in \mathbb{F}_2^{(n-k) \times n}$.

3) Compute $\boldsymbol{c} \coloneqq \boldsymbol{H}\boldsymbol{e} \in \mathbb{F}_2^{n-k}$.

The ciphertext is $\boldsymbol{c}$.

### 6.1.4.3 McEliece.PKE.Dec

Input: Private key $sk$
Ciphertext $\boldsymbol{c}$

Output: Vector $\boldsymbol{e}$ (weight $t$),
or $\perp$

1) Construct the $n$-bit vector $\boldsymbol{v} \coloneqq \boldsymbol{c} \parallel 0_k \in \mathbb{F}_2^n$, where $0_k$ denotes the all-zero vector with $k$ entries.

2) Return the unique codeword $\boldsymbol{e}$ in the binary Goppa code defined by $sk$ that is distance $t$ from $\boldsymbol{v}$.

3) If there is no such codeword, return $\perp$.

NOTE: The Classic McEliece submission does not define a specific algorithm for use in Step 2), but it provides references for different approaches to finding the nearest codeword in a binary Goppa code.

### 6.1.5 Key encapsulation mechanism

#### 6.1.5.1 McEliece.KEM.KeyGen

Input: None

Output: Public key $pk$
Augmented private key $sk$

1) Sample a uniformly random $n$-bit string $z$.

2) Convert $z$ to a vector $\boldsymbol{z} \in \mathbb{F}_2^n$.

3) Call McEliece.PKE.KeyGen() to generate a public key $pk$ and private key $sk'$.

The public key is $pk$. The augmented private key is $sk \coloneqq (sk', pk, \boldsymbol{z})$.

NOTE: The Classic McEliece submission specifies how these elements, including $z$, can be generated deterministically from a seed.

#### 6.1.5.2 McEliece.KEM.Enc

Input: Public key $pk$

Output: Ciphertext $c$
Session key $K$ (length 256 bits)

1) Sample a uniformly random vector $\boldsymbol{e} \in \mathbb{F}_2^n$ of weight $t$.

2) Encrypt $\boldsymbol{c_0} \coloneqq \text{McEliece.PKE.Enc}(pk, \boldsymbol{e})$.

3) Compute $c_1 \coloneqq H(2 \parallel \boldsymbol{e})$.

4) Derive the session key $K \coloneqq \text{KDF}(1 \parallel \boldsymbol{e} \parallel \boldsymbol{c_0} \parallel c_1)$.

The ciphertext is $c \coloneqq (\boldsymbol{c_0}, c_1)$. The session key is $K$.

#### 6.1.5.3 McEliece.KEM.Dec

Input: Augmented private key $sk$
Ciphertext $c$

Output: Session key $K$ (length 256 bits)

1) Parse the private key as $sk = (sk', pk, \boldsymbol{z})$ and the ciphertext as $c = (\boldsymbol{c_0}, c_1)$.

2) Set $b \coloneqq 1$.

3) Call $\text{McEliece.PKE.Dec}(sk, \boldsymbol{c_0})$ to recover $\boldsymbol{e}$ or $\perp$.

4) If the output is $\perp$, set $b \coloneqq 0$ and $\boldsymbol{e} \coloneqq \boldsymbol{z}$.

5) Re-encrypt $\boldsymbol{c_0'} \coloneqq \text{McEliece.PKE.Enc}(pk, \boldsymbol{e})$.

6) If $\boldsymbol{c_0'} \neq \boldsymbol{c_0}$, set $b \coloneqq 0$ and $\boldsymbol{e} \coloneqq \boldsymbol{z}$.

7) Compute $c_1' \coloneqq H(2 \parallel \boldsymbol{e})$.

8) If $c_1' \neq c_1$, set $b \coloneqq 0$ and $\boldsymbol{e} \coloneqq \boldsymbol{z}$.

9) Derive the session key $K \coloneqq \text{KDF}(b \parallel \boldsymbol{e} \parallel \boldsymbol{c_0} \parallel c_1)$.

NOTE: In Step 9), $K$ will be a random key rather than a shared session key if $b = 0$ and $\boldsymbol{e} = \boldsymbol{z}$.

### 6.1.6 Parameter sets

The Classic McEliece submission includes the parameter sets shown in Table 3.

**Table 3: Proposed parameters for Classic McEliece**

| Set | $m$ | $n$ | $t$ | Claimed Security |
|---|---|---|---|---|
| mceliece348864 | 12 | 3 488 | 64 | Category 1 |
| mceliece460896 | 13 | 4 608 | 96 | Category 3 |
| mceliece6960119 | 13 | 6 960 | 119 | Category 5 |
| mceliece6688128 | 13 | 6 688 | 128 | Category 5 |
| mceliece8192128 | 13 | 8 192 | 128 | Category 5 |

These parameter sets lead to the public key, private key, and ciphertext sizes shown in Table 4.

**Table 4: Classic McEliece public key, private key, and ciphertext sizes**

| Set | Public Key (bytes) | Private Key (bytes) | Ciphertext (bytes) |
|---|---|---|---|
| mceliece348864 | 261 120 | 6 452 | 128 |
| mceliece460896 | 524 160 | 13 568 | 188 |
| mceliece6960119 | 1 044 992 | 13 892 | 240 |
| mceliece6688128 | 1 047 319 | 13 908 | 226 |
| mceliece8192128 | 1 357 824 | 14 080 | 240 |

### 6.1.7 Security

The main attacks considered are based on information set decoding, as described in Annex C. The Classic McEliece submission does not include explicit security costings, so Table 5 shows estimated costs for each parameter set for the classical security of message recovery, derived using the methodology described in Annex C. Costs are not included for key recovery, as key recovery is believed to be significantly more difficult than message recovery.

**Table 5: Classical security costings for Classic McEliece**

| Set | Message recovery (bits) |
|---|---|
| mceliece348864 | 140 |
| mceliece460896 | 181 |
| mceliece6688128 | 257 |
| mceliece6960119 | 258 |
| mceliece8192128 | 294 |

McEliece.KEM uses a variant of the Fujisaki-Okamoto transform from [i.16] to achieve tight CCA security. Further work has achieved similar tightness for QROM attacks [i.19].

### 6.1.8 Performance

The Classic McEliece submission includes performance figures for an AVX2-optimized implementation run on a single core of a 3,5 GHz Intel® Xeon® E3-1275 v3 processor. The performance figures for each parameter set are shown in Table 6. The semi-systematic parameter sets have different key generation algorithms, but the same encryption and decryption algorithms.

**Table 6: Classic McEliece performance figures**

| Set | Version | McEliece.KEM.KeyGen (cycles) | McEliece.KEM.Enc (cycles) | McEliece.KEM.Dec (cycles) |
|---|---|---|---|---|
| mceliece348864 | Systematic | 58 034 411 | 44 350 | 134 745 |
| | Semi-systematic | 36 641 040 | | |
| mceliece460896 | Systematic | 215 785 433 | 117 782 | 271 694 |
| | Semi-systematic | 117 067 765 | | |
| mceliece6960119 | Systematic | 556 495 649 | 151 721 | 323 957 |
| | Semi-systematic | 284 584 602 | | |
| mceliece6688128 | Systematic | 438 217 685 | 161 224 | 301 480 |
| | Semi-systematic | 246 508 730 | | |
| mceliece8192128 | Systematic | 514 489 441 | 178 093 | 326 531 |
| | Semi-Systematic | 316 202 817 | | |

# 6.2 KYBER

## 6.2.1 Overview

KYBER is part of the CRYSTALS package, along with the DILITHIUM digital signature scheme. KYBER consists of a CPA-secure PKE scheme that is converted into a CCA-secure KEM using a variant of the Fujisaki-Okamoto transform from [i.16]. The security of KYBER is based on the Module Learning With Errors (MLWE) problem.

Let $R_q$ denote the polynomial ring $(\mathbb{Z}_q[X])/(X^n + 1)$ for a prime $q$ and a power-of-two $n$. A MLWE sample is a pair of the form $(\boldsymbol{A}, \boldsymbol{As} + \boldsymbol{e})$, where $\boldsymbol{A} \in R_q^{k \times k}$ is a public matrix consisting of polynomials whose coefficients are sampled uniformly at random from $\mathbb{Z}_q$, and $\boldsymbol{s} \in R_q^k$ and $\boldsymbol{e} \in R_q^k$ are private vectors of polynomials whose coefficients are sampled from a small distribution over $\mathbb{Z}_q$. The MLWE problem asserts that it is computationally hard to distinguish MLWE samples of the form $(\boldsymbol{A}, \boldsymbol{As} + \boldsymbol{e}_i)$ from pairs of the form $(\boldsymbol{A}, \boldsymbol{u})$, where $\boldsymbol{u} \in R_q^k$ is a vector consisting of polynomials sampled uniformly at random from $R_q$.

The submission makes it clear that KYBER is defined as a CCA-secure KEM only, and that the underlying PKE scheme is a building block that is not intended as a separate submission to the NIST standardization process. Part of the provided rationale for this decision is that in addition to protecting against key reuse, the CCA transform also protects against some implementation mistakes, such as sampling small polynomials from the wrong distribution.

## 6.2.2 Parameters

The main parameters for KYBER are:

- $n$, the degree of the polynomial ring $R_q$;
- $q$, the modulus of the polynomial ring $R_q$;
- $k$, the rank of the matrices and vectors over $R_q$; and
- $\eta_1$ and $\eta_2$, the width of the zero-centred binomial distributions $B_{\eta_1}$ and $B_{\eta_2}$.

For all KYBER parameter sets $n \coloneqq 256$ and $q \coloneqq 3\,329$. The security level is varied primarily by changing $\eta_1$ and the rank $k$ of the module, which means that the underlying polynomial arithmetic operations remain fixed.

The value of $q$ used by KYBER is chosen to keep the probability of decryption failures low, while allowing a variant of the Number Theoretic Transform (NTT) to be used to carry out fast multiplication of elements in $R_q$. For efficiency, some values are computed or transmitted in the NTT domain. Consequently, the specific NTT used by KYBER is part of the definition of the scheme. However, for ease of exposition, the present document does not include a description of the NTT, as its details are not integral to the overall design of KYBER, and do not affect any of the security arguments.

### 6.2.3 Auxiliary primitives

KYBER makes use of several auxiliary, symmetric primitives:

- $G$, a 512-bit cryptographic hash function;
- $H$, a 256-bit cryptographic hash function;
- KDF, a key derivation function;
- PRF, a pseudorandom function; and
- XOF, an extendable output function.

The submission describes two different approaches to instantiate these primitives, as shown in Table 7.

**Table 7: Auxiliary symmetric primitives for KYBER**

| Primitive | Version | |
|---|---|---|
| | FIPS-202 | 90s |
| $G$ | SHA3-512 | SHA-512 |
| $H$ | SHA3-256 | SHA-256 |
| KDF | SHAKE-256 | SHAKE-256 |
| PRF | SHAKE-256 | AES-256 in counter mode |
| XOF | SHAKE-128 | AES-256 in counter mode |

The "90s" variant is included to evaluate performance on platforms that provide hardware support for AES and SHA-2.

KYBER also makes use of the following two functions:

- $\text{Compress}(x, d) \coloneqq \lceil (2^d/q) \cdot x \rfloor \bmod 2^d$
- $\text{Decompress}(x, d) \coloneqq \lceil (q/2^d) \cdot x \rfloor$

Both functions are generalized to work with polynomials by operating coefficientwise.

### 6.2.4 Public-key encryption scheme

#### 6.2.4.1 KYBER.PKE.KeyGen

Input: None

Output: Public key $pk$
Private key $sk$

1) Sample a uniformly random 256-bit seed $d$.
2) Hash the seed $d$ using $G$ to produce two 256-bit seeds $d_0 \parallel d_1 \coloneqq G(d)$.
3) Expand the seed $d_0$ using XOF to produce the public matrix $\boldsymbol{A} \in R_q^{k \times k}$.
4) Sample $\boldsymbol{s_0}, \boldsymbol{e_0} \in R_q^k$ deterministically from $B_{\eta_1}$ using PRF with the seed $d_1$.
5) Compute $\boldsymbol{t} \coloneqq \boldsymbol{A s_0} + \boldsymbol{e_0} \in R_q^k$.

The public key is $pk \coloneqq (\boldsymbol{t}, d_0)$. The private key is $sk \coloneqq \boldsymbol{s}_0$.

NOTE: The KYBER submission uses the NTT for efficient polynomial multiplication. The public matrix $\boldsymbol{A}$ is expanded directly in the NTT domain, whereas the private values are sampled in the normal domain and then transformed to the NTT domain. The public value $\boldsymbol{t}$ is distributed in the NTT domain, and the private value $\boldsymbol{s}_0$ is stored in the NTT domain.

#### 6.2.4.2 KYBER.PKE.Enc

Input: Public key $pk$
Plaintext $m$ (length 256 bits)
Random seed $r$ (length 256 bits)

Output: Ciphertext $c$

1) Parse the public key as $pk = (\boldsymbol{t}, d_0)$.

2) Expand the seed $d_0$ using XOF to produce the public matrix $\boldsymbol{A} \in R_q^{k \times k}$.

3) 3) Sample $\boldsymbol{s_1}, \boldsymbol{e_1} \in R_q^k$ deterministically from $B_{\eta_1}$ and $B_{\eta_2}$ using PRF with the seed $r$.

4) Compute $\boldsymbol{u} \coloneqq \boldsymbol{A}^T \boldsymbol{s_1} + \boldsymbol{e_1} \in R_q^k$.

5) Sample $e_2 \in R_q$ deterministically from $B_{\eta_2}$ using PRF with the seed $r$.

6) Encode the plaintext as an element $\mu \in R_q$ by setting each coefficient $\mu_i$ to $\lceil m_i(q/2) \rfloor$.

7) Compute $v \coloneqq \boldsymbol{t}^T \boldsymbol{s_1} + e_2 + \mu \in R_q$.

8) Compute $\boldsymbol{u}' \coloneqq \mathrm{Compress}(\boldsymbol{u}, d_u)$ and $v' \coloneqq \mathrm{Compress}(v, d_v)$.

The ciphertext is $c \coloneqq (\boldsymbol{u}', v')$.

NOTE 1: The values $d_u$ and $d_v$ are specified as part of each parameter set.

NOTE 2: The public matrix $\boldsymbol{A}$ is expanded directly in the NTT domain. The public value $\boldsymbol{t}$ is transmitted in the NTT domain. The private vector $\boldsymbol{s}_1$ is sampled in the normal domain and transformed to the NTT domain to compute $\boldsymbol{A}^T \boldsymbol{s_1}$ and $\boldsymbol{t}^T \boldsymbol{s_1}$, but the resulting values are transformed back to the normal domain before adding $\boldsymbol{e_1}$ and $e_2 + \mu$, respectively. The final compression step is performed in the normal domain.

#### 6.2.4.3 KYBER.PKE.Dec

Input: Private key $sk$
Ciphertext $c$

Output: Plaintext $m$ (length 256 bits)

1) Parse the private key as $sk = \boldsymbol{s}_0$ and the ciphertext as $c = (\boldsymbol{u}', v')$.

2) Compute $\boldsymbol{u} \coloneqq \mathrm{Decompress}(\boldsymbol{u}', d_u)$ and $v \coloneqq \mathrm{Decompress}(v', d_v)$.

3) Compute $\mu' \coloneqq v - \boldsymbol{s_0}^T \boldsymbol{u} \in R_q$.

4) Recover the plaintext $m$ by setting each bit $m_i$ to $\lceil \mu_i'(2/q) \rfloor \bmod 2$.

NOTE: The private value $\boldsymbol{s}_0$ is stored in the NTT domain. The ciphertext value $\boldsymbol{u}$ is decompressed in the normal domain, transformed to the NTT domain to compute $\boldsymbol{s_0}^T \boldsymbol{u}$, and the result is transformed back to the normal domain before computing $v - \boldsymbol{s_0}^T \boldsymbol{u}$. The final decoding step is performed in the normal domain.

### 6.2.5 Key encapsulation mechanism

#### 6.2.5.1 KYBER.KEM.KeyGen

Input: None

Output: Public key $pk$
Augmented private key $sk$

1) Sample a uniformly random 256-bit value $z$.

2) Call KYBER.PKE.KeyGen() to generate a public key $pk$ and corresponding private key $sk'$.

The public key is $pk$. The augmented private key is $sk := (sk', pk, z)$.

#### 6.2.5.2 KYBER.KEM.Enc

Input: Public key $pk$

Output: Ciphertext $c$
Session key $K$ (length 256 bits)

1) Sample a uniformly random 256-bit message $m$.

2) Derive two 256-bit seeds $k$ and $r$ by computing $k \parallel r := G\big(H(m) \parallel H(pk)\big)$.

3) Encrypt $c :=$ KYBER.PKE.Enc$(pk, H(m), r)$.

4) Derive the session key $K :=$ KDF$\big(k \parallel H(c)\big)$.

The ciphertext is $c$. The session key is $K$.

#### 6.2.5.3 KYBER.KEM.Dec

Input: Augmented private key $sk$
Ciphertext $c$

Output: Session key $K$ (length 256 bits)

1) Parse the private key as $sk = (sk', pk, z)$.

2) Decrypt $m' :=$ KYBER.PKE.Dec$(sk', c)$.

3) Derive two 256-bit seeds $k'$ and $r'$ by computing $k' \parallel r' := G\big(m' \parallel H(pk)\big)$.

4) Re-encrypt $c' :=$ KYBER.PKE.Enc$(pk, m', r')$.

5) If $c' = c$, then derive the session key $K :=$ KDF$\big(k' \parallel H(c)\big)$.

6) Otherwise, derive a random key $K :=$ KDF$\big(z \parallel H(c)\big)$.

### 6.2.6 Parameter sets

The KYBER submission includes the parameter sets shown in Table 8.

**Table 8: Proposed parameters for KYBER**

| Set | $n$ | $q$ | $k$ | $\eta_1$ | $\eta_2$ | $d_v$ | $d_u$ | Failure probability | Claimed security |
|---|---|---|---|---|---|---|---|---|---|
| KYBER512 | 256 | 3 329 | 2 | 3 | 2 | 10 | 4 | $2^{-139}$ | Category 1 |
| KYBER768 | 256 | 3 329 | 3 | 2 | 2 | 10 | 4 | $2^{-164}$ | Category 3 |
| KYBER1024 | 256 | 3 329 | 4 | 2 | 2 | 11 | 5 | $2^{-174}$ | Category 5 |

These parameter sets lead to the public key, private key, and ciphertext sizes shown in Table 9.

**Table 9: KYBER public key, private key, and ciphertext sizes**

| Set | Public Key (bytes) | Private Key (bytes) | Ciphertext (bytes) |
|---|---|---|---|
| KYBER512 | 800 | 1 632 | 768 |
| KYBER768 | 1 184 | 2 400 | 1 088 |
| KYBER1024 | 1 568 | 3 168 | 1 568 |

### 6.2.7 Security

The main attacks considered are the primal and dual attacks described in Annex D. Because KYBER is based on the MLWE problem, its security depends on the difficulty of finding short vectors in a particular class of lattices, which are referred to as module lattices. However, because it is not known how to exploit KYBER's algebraic structure, the attacks are costed using the general-purpose core-SVP methodology described in Annex D. The costs for each parameter set are shown in Table 10.

**Table 10: Core-SVP costings for KYBER (primal attack only)**

| Set | Classical core-SVP (bits) | Quantum core-SVP (bits) |
|---|---|---|
| KYBER512 | 118 | 107 |
| KYBER768 | 182 | 165 |
| KYBER1024 | 256 | 232 |

The KYBER submission claims that KYBER.PKE has a tight proof of security in the ROM that it is CPA-secure based on the computational hardness of the MLWE problem, and that KYBER.KEM has a tight proof of security in the ROM that it is CCA-secure based on the CPA security of KYBER.PKE.

The KYBER submission claims that KYBER.KEM has a non-tight proof of security in the QROM that it is CCA-secure provided KYBER.PKE is CPA-secure in the QROM. A tight proof is possible, but relies on the assumption that a deterministic version of KYBER.PKE is pseudorandom in the QROM.

### 6.2.8 Performance

The KYBER submission includes performance figures for AVX2-optimized implementations run on a single core of a 3,5 GHz Intel® Core™ i7-4770K processor. The performance figures for each parameter set are shown in Table 11.

**Table 11: KYBER performance figures**

| Set | Version | KYBER.KEM.KeyGen (cycles) | KYBER.KEM.Enc (cycles) | KYBER.KEM.Dec (cycles) |
|---|---|---|---|---|
| KYBER512 | FIPS-202 | 33 856 | 45 200 | 59 088 |
| | 90s | 21 880 | 28 592 | 38 752 |
| KYBER768 | FIPS-202 | 52 732 | 67 624 | 82 220 |
| | 90s | 30 460 | 40 140 | 51 512 |
| KYBER1024 | FIPS-202 | 73 544 | 97 324 | 115 332 |
| | 90s | 43 212 | 56 556 | 71 180 |

## 6.3 NTRU

### 6.3.1 Overview

NTRU is a merger of the NTRUEncrypt and NTRU-HRSS-KEM submissions from the first round of the NIST standardization process. NTRU consists of a OW-CPA-secure PKE scheme that is converted into a CCA-secure KEM using a variant of the Fujisaki-Okamoto transform from [i.16]. The security of NTRU is based on the difficulty of finding short vectors in a particular class of structured lattices.

Let $R_q$ denote the polynomial ring $(\mathbb{Z}_q[X])/(X^n + 1)$ for a power-of-two $q$ and a prime $n$. The NTRU problem asserts that given a uniformly random polynomial $h \in R_q$, it is computationally hard to find polynomials $f$ and $g \in R_q$, such that $h = g/f$, and $f$ and $g$ are short when considered as vectors.

## 6.3.2 Parameters

The main parameters for NTRU are:

- $n$, the degree of the polynomial ring $R \coloneqq (\mathbb{Z}[X])/(X^n + 1)$;
- $q$, the modulus of the polynomial rings $R_q \coloneqq (\mathbb{Z}_q[X])/(X^n + 1)$ and $S_q \coloneqq (\mathbb{Z}_q[X])/(X^{n-1} + X^{n-2} + \cdots + 1)$;
- $p$, the auxiliary modulus; and
- $\mathcal{L}_f, \mathcal{L}_g, \mathcal{L}_r, \mathcal{L}_m$, sample spaces for polynomials $f, g, r, m$ respectively.

For all NTRU parameter sets $p \coloneqq 3$. The degree $n$ is chosen to be prime so that $X^{n-1} + X^{n-2} + \cdots + 1$ is irreducible. The modulus $q$ is chosen to be a power-of-two to make modular reductions trivial. The security level is varied by changing $n$ and $q$. The parameter sets are chosen to ensure that NTRU is not susceptible to decryption failures.

To describe the samples spaces, the following definitions are used:

- A polynomial in $\mathbb{Z}[x]$ is said to be ternary if its coefficients all lie in $\{-1,0,1\}$;
- $\mathcal{T}$ is the set of non-zero ternary polynomials in $\mathbb{Z}[x]$ of degree at most $n - 2$;
- $\mathcal{T}(d)$ is the set of polynomials in $\mathcal{T}$ with $d/2$ coefficients equal to $+1$ and $d/2$ coefficients equal to $-1$;
- $\mathcal{T}_+$ is the set of polynomials $v = v_{n-2}X^{n-2} + \cdots + v_1X + v_0 \in \mathcal{T}$ such that $v_0v_1 + \cdots + v_{n-3}v_{n-2} \geq 0$; and
- $\mathcal{T}'_+$ is the set of polynomials of the form $(X + 1)v$ for $v \in \mathcal{T}_+$.

## 6.3.3 Auxiliary primitives

NTRU makes use of the following auxiliary, symmetric primitives:

- $H$, a 256-bit cryptographic hash function; and
- KDF, a key derivation function.

The submission describes how to use SHA3-256 to instantiate $H$ and KDF.

The submission also makes use of an injection $\mathrm{Convert} : S_p \longrightarrow \mathbb{Z}[x]$ such that for any $m \in \mathcal{L}_m$ and $m' = m$ in $S_p$

$$\mathrm{Convert}(m') = m.$$

The Convert function depends on the choice of sample space $\mathcal{L}_m$.

## 6.3.4 Public-key encryption scheme

### 6.3.4.1 NTRU.PKE.KeyGen

Input: None

Output: Public key $pk$
Private key $sk$

1) Sample $f \in \mathcal{L}_f$ and $g \in \mathcal{L}_g$.
2) Compute the inverse $f_q$ of $f$ in $S_q$.
3) Compute $h \coloneqq pgf_q$ in $R_q$.

4) Compute the inverse $h_q$ of $h$ in $S_q$.

5) Compute the inverse $f_p$ of $f$ in $S_p$.

The public key is $pk \coloneqq h$. The private key is $sk \coloneqq (f, f_p, h_q)$.

NOTE: If the inverse does not exist in Step 2), 4), or 5) then key generation is restarted.

#### 6.3.4.2 NTRU.PKE.Enc

Input: Public key $pk$
Plaintext $(r, m)$ $(r \in \mathcal{L}_r, m \in \mathcal{L}_m)$

Output: Ciphertext $c$

1) Parse the public key as $pk = h$.

2) Compute $m' \coloneqq \mathrm{Convert}(m)$.

3) Compute $c \coloneqq rh + m'$ in $R_q$.

The ciphertext is $c$.

#### 6.3.4.3 NTRU.PKE.Dec

Input: Private key $sk$
Ciphertext $c$

Output: Plaintext $(r, m)$ $(r \in \mathcal{L}_r, m \in \mathcal{L}_m)$,
or $\perp$.

1) Parse the private key as $sk = (f, f_p, h_q)$.

2) If $c \neq 0$ in $S_q$ then return $\perp$.

3) Compute $a \coloneqq cf$ in $R_q$.

4) Compute $m \coloneqq af_p$ in $S_p$.

5) Compute $m' \coloneqq \mathrm{Convert}(m)$.

6) Compute $r \coloneqq (c - m')h_q$ in $S_q$.

7) If $r \notin \mathcal{L}_r$ or $m' \notin \mathcal{L}_m$ then return $\perp$.

The plaintext is $(r, m)$.

### 6.3.5 Key encapsulation mechanism

#### 6.3.5.1 NTRU.KEM.KeyGen

Input: None

Output: Public key $pk$
Augmented private key $sk$

1) Sample a uniformly random 256-bit value $z$.

2) Call NTRU.PKE.KeyGen() to generate a public key $pk$ and corresponding private key $sk'$.

The public key is $pk$. The augmented private key is $sk \coloneqq (sk', z)$.

#### 6.3.5.2 NTRU.KEM.Enc

Input: Public key $pk$

Output: Ciphertext $c$
Session key $K$ (length 256 bits)

1) Sample $r \in \mathcal{L}_r$ and $m \in \mathcal{L}_m$.

2) Encrypt $c \coloneqq \text{NTRU.PKE.Enc}\big(pk, (r, m)\big)$.

3) Derive the session key $K \coloneqq \text{KDF}(r \parallel m)$.

The ciphertext is $c$. The session key is $K$.

#### 6.3.5.3 NTRU.KEM.Dec

Input: Private key $sk$
Ciphertext $c$

Output: Session key $K$ (length 256 bits)

1) Parse the private key $sk = (sk', z)$.

2) Call NTRU.PKE.Dec$(sk', c)$ to recover $(r, m)$ or $\perp$.

3) If the output is $(r, m)$, derive the session key $K \coloneqq \text{KDF}(r \parallel m)$.

4) If the output is $\perp$, derive a random key $K \coloneqq \text{KDF}(z \parallel c)$.

### 6.3.6 Parameter sets

The NTRU submission includes the parameter sets shown in Table 12.

**Table 12: Proposed parameters for NTRU**

| Set | $n$ | $q$ | $\mathcal{L}_f$ | $\mathcal{L}_g$ | $\mathcal{L}_r$ | $\mathcal{L}_m$ | Failure probability | Claimed security |
|---|---|---|---|---|---|---|---|---|
| ntruhps2048509 | 509 | 2 048 | $\mathcal{T}$ | $\mathcal{T}(254)$ | $\mathcal{T}$ | $\mathcal{T}(254)$ | $2^{-214,3}$ | - |
| ntruhps2048677 | 677 | 2 048 | $\mathcal{T}$ | $\mathcal{T}(254)$ | $\mathcal{T}$ | $\mathcal{T}(254)$ | $2^{-213,9}$ | Category 1 |
| ntruhps4096821 | 821 | 4 096 | $\mathcal{T}_+$ | $\mathcal{T}(510)$ | $\mathcal{T}$ | $\mathcal{T}(254)$ | $2^{-433,2}$ | Category 3 |
| ntruhrss701 | 701 | 8 192 | $\mathcal{T}_+$ | $\mathcal{T}'_+$ | $\mathcal{T}$ | $\mathcal{T}$ | $2^{-796,6}$ | Category 1 |

These parameter sets lead to the public key, private key, and ciphertext sizes shown in Table 13.

**Table 13: NTRU public key, private key, and ciphertext sizes**

| Set | Public Key (bytes) | Private Key (bytes) | Ciphertext (bytes) |
|---|---|---|---|
| ntruhps2048509 | 699 | 935 | 699 |
| ntruhps2048677 | 930 | 1 234 | 930 |
| ntruhps4096821 | 1 230 | 1 590 | 1 230 |
| ntruhrss701 | 1 138 | 1 450 | 1 138 |

### 6.3.7 Security

The main attacks considered are the primal and hybrid attacks described in Annex D. Because NTRU is based on the NTRU assumption, its security depends on the difficulty of finding short vectors in a particular class of lattices, which are referred to as NTRU lattices. However, because it is not known how to exploit NTRU's algebraic structure, the attacks are costed using the general-purpose core-SVP methodology described in Annex D. The costs for both attacks are shown in Table 14.

**Table 14: Core-SVP costings for NTRU**

| Set | Primal attack classical core-SVP (bits) | Hybrid attack classical core-SVP (bits) |
|---|---|---|
| ntruhps2048509 | 106 | 105 |
| ntruhps2048677 | 145 | 144 |
| ntruhps4096821 | 179 | 178 |
| ntruhrss701 | 136 | 134 |

The NTRU submission claims that NTRU.KEM is CCA-secure in the ROM and the QROM provided NTRU.PKE is OW-CPA-secure. The submission claims that the proof in the ROM is tight, but the proof in the QROM is non-tight. A tight proof in the QROM is possible, but relies on the additional assumption that NTRU.PKE is pseudorandom.

### 6.3.8 Performance

The NTRU submission includes performance figures for an AVX2-optimized implementation run on a single core of a 3,5 GHz Intel® Core™ i7-4770K processor. The performance figures for each parameter set are shown in Table 15.

**Table 15: NTRU performance figures**

| Set | NTRU.KEM.KeyGen (cycles) | NTRU.KEM.Enc (cycles) | NTRU.KEM.Dec (cycles) |
|---|---|---|---|
| ntruhps2048509 | 191 279 | 61 331 | 40 026 |
| ntruhps2048677 | 309 216 | 83 519 | 59 729 |
| ntruhps4096821 | 431 667 | 98 809 | 75 384 |
| ntruhrss701 | 340 823 | 50 441 | 62 267 |

## 6.4 SABER

### 6.4.1 Overview

SABER consists of a CPA-secure PKE scheme that is converted into a CCA-secure KEM using a variant of the Fujisaki-Okamoto transform from [i.16]. The security of SABER is based on the Module Learning With Rounding (MLWR) problem.

Let $R_p$ and $R_q$ denote the polynomial rings $(\mathbb{Z}_p[X])/(X^n + 1)$ and $(\mathbb{Z}_q[X])/(X^n + 1)$, where $p$ and $q$ are chosen so that $p$ divides $q$, and $n$ is a power-of-two. A MLWR sample is a pair of the form $(\boldsymbol{A}, \lfloor \boldsymbol{As} \rceil_{q \to p})$, where $\boldsymbol{A} \in R_q^{k \times k}$ is a public matrix consisting of polynomials with coefficients sampled uniformly at random from $\mathbb{Z}_q$, $\boldsymbol{s} \in R_q^k$ is a private vector of polynomials with coefficients sampled from a small distribution over $\mathbb{Z}_q$, and $\lfloor \boldsymbol{As} \rceil_{q \to p}$ denotes a modulus switching operation that deterministically rounds the components of $\boldsymbol{As}$ from elements in $R_q$ to elements in $R_p$. The MLWR problem asserts that it is computationally hard to distinguish MLWR samples of the form $(\boldsymbol{A}, \lfloor \boldsymbol{As} \rceil_{q \to p})$ from pairs of the form $(\boldsymbol{A}, \lfloor \boldsymbol{u} \rceil_{q \to p})$, where $\boldsymbol{u}$ is a uniformly random element of $R_q^k$.

The standard way to instantiate the modulus switching operation $\lfloor x \rceil_{q \to p}$ is as the function $\lfloor (p/q)\, x \rceil \bmod p$, where $x \in \mathbb{Z}_q$ and $\lfloor x \rceil_{q \to p} \in \mathbb{Z}_p$, which can be generalized to work over vectors and polynomials by operating on each component. This function can be efficiently implemented by adding a constant and shifting.

NOTE: The SABER submission does not instantiate $\lfloor x \rceil_{q \to p}$ in this way due to the choice of constants used in the adding and shifting operations.

### 6.4.2 Parameters

The main parameters for SABER are:

- $n$, the degree of the polynomial rings $R_q$, $R_p$, and $R_T$;
- $q$, the modulus of the base polynomial ring $R_q$;

- $p$, the modulus of the polynomial ring $R_p$, used when modulus switching from $q$ to $p$;
- $T$, the modulus of the polynomial ring $R_T$, used when modulus switching from $p$ to $T$;
- $k$, the rank of the matrices and vectors over $R_q$; and
- $\eta$, the width of the zero-centred binomial distribution $B_\eta$.

For all SABER parameter sets $n := 256$, $q := 2^{13}$, and $p := 2^{10}$. Note that $p$, $q$, and $T$, are chosen as powers of 2 so that $p$ divides $q$, and $T$ divides $p$, and to make modular reductions trivial. Modulus switching from $p$ to $T$ is used to compress the ciphertext. The security level is varied by changing $\eta$ and $k$, which means that the underlying polynomial arithmetic operations remain fixed.

## 6.4.3 Auxiliary primitives

SABER makes use of several auxiliary, symmetric primitives:

- $G$, a 512-bit cryptographic hash function;
- $H$, a 256-bit cryptographic hash function;
- KDF, a key derivation function;
- PRF, a pseudorandom function; and
- XOF, an extendable output function.

The submission describes two different approaches to instantiate these primitives, as shown in Table 16.

**Table 16: Auxiliary symmetric primitives for SABER**

| Primitive | Version | |
|---|---|---|
| | Saber | Saber-90s |
| $G$ | SHA3-512 | SHA2-512 |
| $H$ | SHA3-256 | SHA2-256 |
| KDF | SHA3-256 | SHA2-256 |
| PRF | SHAKE-128 | AES-256 in counter mode |
| XOF | SHAKE-128 | AES-256 in counter mode |

## 6.4.4 Public-key encryption scheme

### 6.4.4.1 SABER.PKE.KeyGen

Input: None

Output: Public key $pk$
Private key $sk$

1) Sample a uniformly random 256-bit seed $d$.
2) Expand the seed $d$ using XOF to produce the public matrix $\boldsymbol{A} \in R_q^{k \times k}$.
3) Sample a uniformly random 256-bit seed $r$.
4) Sample $\boldsymbol{s_0} \in R_q^k$ deterministically from $B_\eta$ using PRF with the seed $r$.
5) Compute $\boldsymbol{t} := \lfloor \boldsymbol{A}^T \boldsymbol{s_0} \rceil_{q \to p} \in R_p^k$.

The public key is $pk := (\boldsymbol{t}, d)$. The private key is $sk := \boldsymbol{s}_0$.

#### 6.4.4.2 SABER.PKE.Enc

Input: Public key $pk$
Plaintext $m$ (length 256 bits)
Random seed $r$ (length 256 bits)

Output: Ciphertext $c$

1) Parse the public key as $pk = (\boldsymbol{t}, d)$.
2) Expand the seed $d$ using XOF to produce the public matrix $\boldsymbol{A} \in R_q^{k \times k}$.
3) Sample $\boldsymbol{s_1} \in R_q^k$ deterministically from $B_\eta$ using PRF with the seed $r$.
4) Compute $\boldsymbol{u} \coloneqq \lfloor \boldsymbol{A s_1} \rceil_{q \to p} \in R_p^k$.
5) Compute $v \coloneqq \boldsymbol{t}^T \boldsymbol{s_1} \in R_p$.
6) Encode the plaintext as an element $\mu \in R_p$ by setting each coefficient $\mu_i$ to $m_i(p/2)$.
7) Compress $v' \coloneqq \lfloor v - \mu \rceil_{p \to T} \in R_T$.

The ciphertext is $c \coloneqq (\boldsymbol{u}, v')$.

#### 6.4.4.3 SABER.PKE.Dec

Input: Private key $sk$
Ciphertext $c$

Output: Plaintext $m$ (length 256 bits)

1) Parse the private key as $sk = \boldsymbol{s}_0$ and the ciphertext as $c = (\boldsymbol{u}, v')$.
2) Compute $v \coloneqq \boldsymbol{u}^T \boldsymbol{s_0} \in R_p$.
3) Recover the plaintext by computing $m \coloneqq \left\lfloor v - \frac{p}{T} v' \right\rceil_{p \to 2} \in R_2$.

### 6.4.5 Key encapsulation mechanism

#### 6.4.5.1 SABER.KEM.KeyGen

Input: None

Output: Public key $pk$
Augmented private key $sk$

1) Sample a uniformly random 256-bit value $z$.
2) Call SABER.PKE.KeyGen() to generate a public key $pk$ and corresponding private key $sk'$.

The public key is $pk$. The augmented private key is $sk \coloneqq (sk', pk, z)$.

#### 6.4.5.2 SABER.KEM.Enc

Input: Public key $pk$

Output: Ciphertext $c$
Session key $K$ (length 256 bits)

1) Sample a uniformly random 256-bit message $m$.
2) Derive two 256-bit seeds $k$ and $r$ by computing $k \parallel r \coloneqq G(H(pk) \parallel m)$.

3) Encrypt $c \coloneqq \text{SABER.PKE.Enc}(pk, m, r)$.

4) Derive the session key $K \coloneqq \text{KDF}(k \parallel c)$.

The ciphertext is $c$. The session key is $K$.

#### 6.4.5.3 SABER.KEM.Dec

Input: Ciphertext $c$
Augmented private key $sk$

Output: Session key $K$ (length 256 bits)

1) Parse the private key as $sk = (sk', pk, z)$.

2) Decrypt $m' \coloneqq \text{SABER.PKE.Dec}(c, sk')$.

3) Derive two 256-bit seeds $k'$ and $r'$ by computing $k' \parallel r' \coloneqq G(H(pk) \parallel m')$.

4) Re-encrypt $c' \coloneqq \text{SABER.PKE.Enc}(pk, m', r')$.

5) If $c' = c$, then derive the session key $K \coloneqq \text{KDF}(k' \parallel c)$.

6) Otherwise, derive a random key $K \coloneqq \text{KDF}(z \parallel c)$.

### 6.4.6 Parameter sets

The SABER submission includes the parameter sets shown in Table 17.

**Table 17: Proposed parameters for SABER**

| Set | $n$ | $q$ | $p$ | $T$ | $k$ | $\eta$ | Failure probability | Claimed security |
|---|---|---|---|---|---|---|---|---|
| LightSaber | 256 | $2^{13}$ | $2^{10}$ | $2^{3}$ | 2 | 10 | $2^{-120}$ | Category 1 |
| Saber | 256 | $2^{13}$ | $2^{10}$ | $2^{4}$ | 3 | 8 | $2^{-136}$ | Category 3 |
| FireSaber | 256 | $2^{13}$ | $2^{10}$ | $2^{6}$ | 4 | 6 | $2^{-165}$ | Category 5 |

These parameter sets lead to the public key, private key, and ciphertext sizes shown in Table 18.

**Table 18: SABER public key, private key, and ciphertext sizes**

| Set | Public Key (bytes) | Private Key (bytes) | Ciphertext (bytes) |
|---|---|---|---|
| LightSaber | 672 | 1 568 | 736 |
| Saber | 992 | 2 304 | 1 088 |
| FireSaber | 1 312 | 3 040 | 1 472 |

### 6.4.7 Security

The main attacks considered are the primal and dual attacks described in Annex D. Because SABER is based on the MLWR problem, its security depends on the difficulty of finding short vectors in a particular class of lattices, which are referred to as module lattices. However, because it is not known how to exploit SABER's algebraic structure, or its use of rounding, the attacks are costed using the general-purpose core-SVP methodology described in Annex D. The costs for each parameter set are shown in Table 19.

**Table 19: Core-SVP costings for SABER (primal attack only)**

| Set | Classical core-SVP (bits) | Quantum core-SVP (bits) |
|---|---|---|
| LightSaber | 118 | 107 |
| Saber | 189 | 172 |
| FireSaber | 260 | 236 |

The SABER submission claims that SABER.PKE has a tight proof of security in the ROM that it is CPA-secure based on the computational hardness of the MLWR problem, and that SABER.KEM has a tight proof of security in the ROM that it is CCA-secure based on the CPA security of SABER.PKE.

The SABER submission claims that SABER.KEM has a non-tight proof of security in the QROM that it is CCA-secure provided SABER.PKE is OW-CPA-secure, which holds by virtue of SABER.PKE being CPA-secure.

### 6.4.8 Performance

The SABER submission includes performance figures for AVX2-optimized implementations run on a 2,0 GHz Intel® Core™ i7-4510U processor. The performance figures for each parameter set are shown in Table 20.

**Table 20: SABER performance figures**

| Set | Version | SABER.KEM.Keygen (cycles) | SABER.KEM.Enc (cycles) | SABER.KEM.Dec (cycles) |
|---|---|---|---|---|
| LightSaber | Saber | 45 152 | 49 948 | 47 852 |
| | Saber-90s | 28 928 | 35 491 | 35 123 |
| Saber | Saber | 66 727 | 79 064 | 76 612 |
| | Saber-90s | 36 315 | 45 575 | 46 380 |
| FireSaber | Saber | 100 959 | 117 151 | 116 095 |
| | Saber-90s | 57 144 | 70 335 | 72 797 |

# 7 Alternate candidates

## 7.1 BIKE

### 7.1.1 Overview

BIKE is a CPA-secure KEM that is based on Quasi-Cyclic Moderate Density Parity Check (QC-MDPC) codes.

Let $R_2$ denote the polynomial ring $R_2 = (\mathbb{Z}_2[X])/(X^n + 1)$ for a prime $n$. A QC-MDPC code of length $2n$ and rank $n$ is a linear code $\mathcal{C} = \{ \boldsymbol{c} \in R_2^{1\times 2} \mid \boldsymbol{H}\boldsymbol{c}^T = 0\}$ defined by a quasi-cyclic parity-check matrix $\boldsymbol{H} = [h_0 \quad h_1] \in R_2^{1\times 2}$ which has moderate Hamming weight $w = O(\sqrt{n})$. Recovering the private moderate-density parity-check matrix $\boldsymbol{H}$ from a public description of the QC-MDPC code $\mathcal{C}$ is equivalent to finding a codeword of weight $w$ in the dual code generated by $\boldsymbol{H}$. This is believed to be computationally hard when the parameters are chosen appropriately.

Let $\boldsymbol{e} = [e_0 \quad e_1] \in R_2^{1\times 2}$ be an error vector with moderate Hamming weight $t = O(\sqrt{n})$. The syndrome corresponding to $\boldsymbol{e}$ is the element $s = \boldsymbol{H}\boldsymbol{e}^T \in R_2$. There are efficient algorithms for recovering the weight $t$ error vector $\boldsymbol{e}$ from the syndrome $s$ when the moderate-density parity-check matrix $\boldsymbol{H}$ is known. If the moderate-density parity-check matrix $\boldsymbol{H}$ is not known, then syndrome decoding for a QC-MDPC code is believed to be as hard as syndrome decoding for a random quasi-cyclic code.

BIKE uses a variant of the Fujisaki-Okamoto transform from [i.16] to convert a CPA-secure PKE into a CCA-secure KEM. However, CCA security relies on the probability of a decoding failure being sufficiently low. The parameters proposed for BIKE are estimated to have a low enough decoding failure probability, but there is no formal proof of this, so the BIKE submission only claims CPA-security for the KEM.

### 7.1.2 Parameters

The main parameters for BIKE are:

- $n$, the degree of the polynomial ring $R_2$;
- $w$, the combined weight of the private polynomials in key generation; and
- $t$, the combined weight of the private polynomials in encryption and encapsulation.

The degree $n$ is chosen to be a prime where the only irreducible factors of $X^n + 1$ over the field $\mathbb{Z}_2$ are $X + 1$ and $X^{n-1} + \cdots + X + 1$.

The weight $w$ is chosen so that $w \equiv 2 \pmod 4$. This guarantees that the polynomials of weight $w/2$ sampled in key generation will always be invertible in $R_2$.

NOTE: Where necessary, a weight function, denoted $wt(f)$, which returns the weight of the input polynomial $f$, is used to clarify certain requirements in the algorithm descriptions.

### 7.1.3 Decoding

BIKE uses a decoder that iteratively flips bits in a candidate error vector based on the number of incorrect parity-check equations that use those bits. The third round submission recommends the Black-Gray-Flip decoder by Drucker, Gueron, and Kostic [i.17]. Previous versions of BIKE used different decoders.

Bit flipping decoders can fail to recover the correct error vector; such decoding failures leak information about the private parity-check matrix. The probability of a decoding failure is sensitive to the specific choice of decoder and decoding parameters.

NOTE: The description of BIKE given in the present clause assumes that the decoding algorithm always returns an error vector, even if it is incorrect.

### 7.1.4 Auxiliary primitives

BIKE makes use of several auxiliary, symmetric primitives:

- $H$, a 256-bit cryptographic hash function;
- KDF, a key derivation function; and
- PRF, a pseudorandom function.

The submission describes how to use SHA-2 and AES to instantiate these primitives, as shown in Table 21.

**Table 21: Auxiliary symmetric primitives for BIKE**

| Primitive | Instantiation |
|---|---|
| $H$ | SHA-384, output truncated to 256 bits |
| KDF | SHA-384, output truncated to 256 bits |
| PRF | AES-256 in counter mode |

### 7.1.5 Public-key encryption scheme

#### 7.1.5.1 BIKE.PKE.KeyGen

Input: None

Output: Public key $pk$
Private key $sk$

1) Sample uniformly random $f, g \in R_2$ so that $wt(f) = wt(g) = w/2$.

2) Compute $h \coloneqq g/f \in R_2$.

The public key is $pk \coloneqq h$. The private key is $sk \coloneqq (f, g)$.

#### 7.1.5.2 BIKE.PKE.Enc

Input: Public key $pk$
Plaintext $m$ (length 256 bits)
Random seed $r$ (length 256 bits)

Output: Ciphertext $c$

1) Parse the public key as $pk = h \in R_2$.

2) Sample $e_0, e_1 \in R_2$ deterministically using PRF with the seed $r$, so that $wt(e_0) + wt(e_1) = t$.

3) Compute $c_0 \coloneqq he_0 + e_1 \in R_2$.

4) Compute $u \coloneqq H(e_0 \parallel e_1)$.

5) Compute $c_1 \coloneqq m \oplus u$.

The ciphertext is $c \coloneqq (c_0, c_1)$.

#### 7.1.5.3 BIKE.PKE.Dec

Input: Private key $sk$
Ciphertext $c$

Output: Message $m'$ (length 256 bits)

1) Parse the private key as $sk = (f, g)$ and the ciphertext as $c = (c_0, c_1)$.

2) Compute $s \coloneqq c_0 f \in R_2$.

3) Decode $s$ using $f$ and $g$ to recover $e_0'$ and $e_1'$.

4) Compute $u' \coloneqq H(e_0' \parallel e_1')$.

5) Recover the plaintext by computing $m' \coloneqq c_1 \oplus u'$.

NOTE: Step 3) computes the syndrome $s = ge_0 + fe_1$ corresponding to the error vector $\boldsymbol{e} = [e_0 \quad e_1]$ and parity-check matrix $\boldsymbol{H} = [g \quad f]$. Step 4) recovers a candidate error vector $\boldsymbol{e'} = [e_0' \quad e_1']$ from the syndrome $s$ using the Black-Gray-Flip decoder.

### 7.1.6 Key encapsulation mechanism

#### 7.1.6.1 BIKE.KEM.KeyGen

Input: None

Output: Public key $pk$
Augmented private key $sk$

1) Sample a uniformly random 256-bit private value $z$.

2) Call BIKE.PKE.KeyGen() to generate a public key $pk$ and corresponding private key $sk'$.

The public key is $pk$. The augmented private key is $sk \coloneqq (sk', pk, z)$.

#### 7.1.6.2 BIKE.KEM.Enc

Input: Public key $pk$

Output: Ciphertext c
Session key $K$ (length 256 bits)

1) Sample a uniformly random 256-bit plaintext $m$.

2) Encrypt $c \coloneqq \text{BIKE.PKE.Enc}(pk, m, m)$.

3) Derive the session key $K \coloneqq \text{KDF}(m \parallel c)$.

The ciphertext is $c$. The session key is $K$.

NOTE: In Step 2), in the call to BIKE.PKE.Enc, the message is used as the random seed.

#### 7.1.6.3 BIKE.KEM.Dec

Input: Augmented private key $sk$
Ciphertext $c$

Output: Session key $K$ (length 256 bits)

1) Parse the private key as $sk = (sk', pk, z)$.

2) Decrypt $m' \coloneqq \text{BIKE.PKE.Dec}(sk', c)$.

3) Re-encrypt $c' \coloneqq \text{BIKE.PKE.Enc}(pk, m', m')$.

4) If $c' = c$, then derive the session key $K \coloneqq \text{KDF}(m' \parallel c)$.

5) Otherwise, derive a random key $K \coloneqq \text{KDF}(z \parallel c)$.

NOTE: The BIKE submission does not perform a full re-encyption check. Instead, it checks that the candidate error vector $\boldsymbol{e}' = [e'_0 \quad e'_1]$ recovered in Step 4) of BIKE.PKE.Dec matches the error vector $\boldsymbol{e} = [e_0 \quad e_1]$ derived from $m'$ in Step 2) of BIKE.PKE.Enc. This is sufficient to guarantee that the ciphertexts $c$ and $c'$ will match with overwhelming probability.

### 7.1.7 Parameter sets

The BIKE submission includes the parameter sets shown in Table 22.

**Table 22: Proposed parameters for BIKE**

| Set | $n$ | $w$ | $t$ | Failure probability | Claimed security |
|---|---|---|---|---|---|
| Level 1 | 12 323 | 142 | 134 | $2^{-128}$ | Category 1 |
| Level 3 | 24 659 | 206 | 199 | $2^{-192}$ | Category 3 |
| Level 5 | 40 973 | 274 | 264 | $2^{-256}$ | Category 5 |

These parameter sets lead to the public key, private key and ciphertext sizes shown in Table 23.

**Table 23: BIKE public key, private key, and ciphertext sizes**

| Set | Public key (bytes) | Private key (bytes) | Ciphertext (bytes) |
|---|---|---|---|
| Level 1 | 1 541 | 281 | 1 573 |
| Level 3 | 3 083 | 419 | 3 115 |
| Level 5 | 5 122 | 580 | 5 154 |

### 7.1.8 Security

The security of BIKE depends on the difficulty of finding a moderate-weight codeword or decoding a syndrome with a moderate-weight error. In both cases, the best attacks involve information set decoding, and target multiple codewords or errors of the same weight obtained from the quasi-cyclic structure; see Annex C for more information. The BIKE submission does not include explicit security costings, so Table 24 shows estimated costs for each parameter set for the classical security of key and message recovery, derived using the methodology described in Annex C.

**Table 24: Classical security estimates for BIKE**

| Set | Key recovery (bits) | Message recovery (bits) |
|---|---|---|
| Level 1 | 128 | 127 |
| Level 3 | 191 | 191 |
| Level 5 | 258 | 256 |

BIKE.KEM has a tight proof of security in the ROM that it is CCA-secure, based on the hardness of the quasi-cyclic syndrome decoding and quasi-cyclic codeword finding problems, and provided that the decoding failure rate is sufficiently low. The tight proof of security remains valid in the QROM. However, the decoding failure rates for the BIKE parameters shown in Table 22 are estimates based on simulation and extrapolation rather than rigorous upper bounds. Consequently, the BIKE submission only claims CPA security for BIKE.KEM.

### 7.1.9 Performance

The BIKE submission includes performance figures for an AVX2-optimized implementation run on a single core of a 1,3 GHz Intel® Core™ i7-1065G7 processor. The performance figures for the Level 1 and Level 3 parameter sets are shown in Table 25. The submission does not include performance figures for the Level 5 parameter set.

**Table 25: BIKE performance figures**

| Set | BIKE.KEM.KeyGen (cycles) | BIKE.KEM.Enc (cycles) | BIKE.KEM.Dec (cycles) |
|---|---|---|---|
| Level 1 | 600 000 | 220 000 | 2 220 000 |
| Level 3 | 1 780 000 | 465 000 | 6 610 000 |

## 7.2 FrodoKEM

### 7.2.1 Overview

FrodoKEM consists of a CPA-secure PKE scheme that is converted into a CCA-secure KEM using a variant of the Fujisaki-Okamoto transform from [i.16]. FrodoKEM's security is based on the Learning With Errors (LWE) problem.

Let $n$ and $q$ be positive integers, and $\chi$ be a distribution over $\mathbb{Z}$. For a fixed, private $\boldsymbol{s} \in \mathbb{Z}_q^n$, a sample from the LWE distribution $A_{s,\chi}$ is obtained by sampling $\boldsymbol{a} \in \mathbb{Z}_q^n$ uniformly at random, sampling an integer error $e \in \mathbb{Z}$ from the distribution $\chi$, and outputting the pair $(\boldsymbol{a}, \langle \boldsymbol{a}, \boldsymbol{s} \rangle + e \bmod q) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$. In FrodoKEM, the coefficients for all vectors are sampled from a symmetric distribution on $\mathbb{Z}$, centred at 0, with small support, which approximates a rounded continuous Gaussian distribution. The LWE problem asserts that it is computationally hard to distinguish LWE samples of the form $(\boldsymbol{a}, \langle \boldsymbol{a}, \boldsymbol{s} \rangle + e)$ from pairs of the form $(\boldsymbol{a}, u)$, where $u$ is a uniformly random element of $\mathbb{Z}_q$.

The submission makes it clear that FrodoKEM is defined as a CCA-secure KEM only, and that the underlying PKE scheme is a building block that is not intended as a separate submission to the NIST standardization process.

### 7.2.2 Parameters

The main parameters for FrodoKEM are:

- $n, m,$ integer matrix dimensions;

- $q = 2^D$, a power-of-two modulus with exponent $D \leq 16$;
- $\chi$, a probability distribution over $\mathbb{Z}$;
- $B \leq D$, the number of bits encoded in each matrix entry; and
- $\ell = B \cdot m^2$, the length of seeds, messages, and session keys.

For all FrodoKEM parameter sets $m := 8$.

### 7.2.3 Auxiliary primitives

FrodoKEM makes use of several auxiliary, symmetric primitives:

- G, a $2\ell$-bit cryptographic hash function;
- $H$, an $\ell$-bit cryptographic hash function;
- KDF, a key derivation function;
- PRF, a pseudorandom function; and
- XOF, an extendable output function.

The submission describes how to use SHAKE-128 or SHAKE-256 to instantiate $G$, $H$, KDF, and PRF, depending on the parameter set being used. XOF can be instantiated using either AES-128 or SHAKE-128.

FrodoKEM also makes use of the following two functions:

- Encode maps a bit string of length $\ell$ to a matrix in $\mathbb{Z}_q^{m \times m}$; and
- Decode extracts a bit string of length $\ell$ from a matrix in $\mathbb{Z}_q^{m \times m}$.

### 7.2.4 Public-key encryption scheme

#### 7.2.4.1 Frodo.PKE.KeyGen

Input: None

Output: Public key $pk$
Private key $sk$

1) Sample a uniformly random 128-bit seed $d_0$.
2) Expand the seed $d_0$ using XOF to produce the public matrix $\boldsymbol{A} \in \mathbb{Z}_q^{n \times n}$.
3) Sample a uniformly random $\ell$-bit seed $d_1$.
4) Sample $\boldsymbol{S_0}, \boldsymbol{E_0} \in \mathbb{Z}_q^{n \times m}$ deterministically from $\chi$ using PRF with the seed $d_1$.
5) Compute $\boldsymbol{T} := \boldsymbol{AS_0} + \boldsymbol{E_0} \in \mathbb{Z}_q^{n \times m}$.

The public key is $pk := (\boldsymbol{T}, d_0)$. The private key is $sk := \boldsymbol{S}_0$.

#### 7.2.4.2 Frodo.PKE.Enc

Input: Public key $pk$
Plaintext $m$ (length $\ell$ bits)
Random seed $r$ (length $\ell$ bits)

Output: Ciphertext $c$

1) Parse the public key as $pk = (\boldsymbol{T}, d_0)$.

2) Expand the seed $d_0$ using XOF to produce the public matrix $\boldsymbol{A} \in \mathbb{Z}_q^{n \times n}$.

3) Sample $\boldsymbol{S_1}, \boldsymbol{E_1} \in \mathbb{Z}_q^{m \times n}$ and $\boldsymbol{E_2} \in \mathbb{Z}_q^{m \times m}$ from $\chi$ deterministically using PRF with the seed $r$.

4) Compute $\boldsymbol{U} \coloneqq \boldsymbol{S_1 A} + \boldsymbol{E_1} \in \mathbb{Z}_q^{m \times n}$.

5) Compute $\boldsymbol{V} \coloneqq \boldsymbol{S_1 T} + \boldsymbol{E_2} + \mathrm{Encode}(m) \in \mathbb{Z}_q^{m \times m}$.

The ciphertext is $c \coloneqq (\boldsymbol{U}, \boldsymbol{V})$.

#### 7.2.4.3 Frodo.PKE.Dec

Input: Private key $sk$
Ciphertext $c$

Output: Plaintext $m$ (length $\ell$ bits)

1) Parse the private key as $sk = \boldsymbol{S_0}$ and the ciphertext as $c = (\boldsymbol{U}, \boldsymbol{V})$.

2) Compute $\boldsymbol{V'} \coloneqq \boldsymbol{V} - \boldsymbol{U S_0} \in \mathbb{Z}_q^{m \times m}$.

3) Recover the plaintext by computing $m \coloneqq \mathrm{Decode}(\boldsymbol{V'})$.

### 7.2.5 Key encapsulation mechanism

#### 7.2.5.1 Frodo.KEM.KeyGen

Input: None

Output: Public key $pk$
Augmented private key $sk$

1) Sample a uniformly random $\ell$-bit value $z$.

2) Call FrodoPKE.KeyGen() to generate a public key $pk$ and corresponding private key $sk'$.

The public key is $pk$. The augmented private key is $sk \coloneqq (sk', pk, z)$.

#### 7.2.5.2 Frodo.KEM.Enc

Input: Public key $pk$

Output: Ciphertext $c$
Session key $K$ (length $\ell$ bits)

1) Sample a uniformly random $\ell$-bit message $m$.

2) Derive two $\ell$-bit seeds $k$ and $r$ by computing $k \parallel r \coloneqq G(m \parallel H(pk))$.

3) Encrypt $c \coloneqq \mathrm{Frodo.PKE.Enc}(pk, m, r)$.

4) Derive the session key $K \coloneqq \mathrm{KDF}(k \parallel c)$.

The ciphertext is $c$. The session key is $K$.

#### 7.2.5.3 Frodo.KEM.Dec

Input: Augmented private key $sk$
Ciphertext $c$

Output: Session key $K$ (length $\ell$ bits)

1) Parse the private key as $sk = (sk', pk, z)$.

2) Decrypt $m' \coloneqq \text{Frodo.PKE.Dec}(sk', c)$.

3) Derive two $\ell$-bit seeds $k'$ and $r'$ by computing $k' \parallel r' \coloneqq G(m' \parallel H(pk))$.

4) Re-encrypt $c' \coloneqq \text{Frodo.PKE.Enc}(pk, m', r')$.

5) If $c' = c$, then derive the session key $K \coloneqq \text{KDF}(k' \parallel c)$.

6) Otherwise, derive a random key $K \coloneqq \text{KDF}(z \parallel c)$.

### 7.2.6 Parameter sets

The FrodoKEM submission includes the parameter sets shown in Table 26.

**Table 26: Proposed parameters for FrodoKEM**

| Set | $n$ | $m$ | $q$ | Support of $\chi$ | $B$ | Failure probability | Claimed security |
|---|---|---|---|---|---|---|---|
| Frodo-640 | 640 | 8 | $2^{15}$ | $[-12 \dots 12]$ | 2 | $2^{-138,7}$ | Category 1 |
| Frodo-976 | 976 | 8 | $2^{16}$ | $[-10 \dots 10]$ | 3 | $2^{-199,6}$ | Category 3 |
| Frodo-1344 | 1 344 | 8 | $2^{16}$ | $[-6 \dots 6]$ | 4 | $2^{-252,5}$ | Category 5 |

These parameter sets lead to the public key, private key, and ciphertext sizes shown in Table 27.

**Table 27: FrodoKEM public key, private key, and ciphertext sizes**

| Set | Public Key (bytes) | Private Key (bytes) | Ciphertext (bytes) |
|---|---|---|---|
| Frodo-640 | 9 616 | 19 888 | 9 720 |
| Frodo-976 | 15 632 | 31 296 | 15 744 |
| Frodo-1344 | 21 520 | 43 088 | 21 632 |

### 7.2.7 Security

The main attacks considered are the primal and dual attacks described in Annex D. The attacks are costed using the general-purpose core-SVP methodology described in Annex D. The costs for each parameter set are shown in Table 28.

**Table 28: Core-SVP costings for FrodoKEM (primal attack only)**

| Set | Classical core-SVP (bits) | Quantum core-SVP (bits) |
|---|---|---|
| Frodo-640 | 151 | 138 |
| Frodo-976 | 216 | 197 |
| Frodo-1344 | 282 | 256 |

The FrodoKEM submission claims that for a uniformly random public matrix $\boldsymbol{A}$, Frodo.PKE has a tight proof of security in the standard model that it is CPA-secure against classical and quantum adversaries, based on the computational hardness of the LWE problem.

The submission claims that this result holds when $\boldsymbol{A}$ is generated from a seed, provided the pseudorandom generator is modelled as an ideal cipher (when using AES-128) or as a random oracle (when using SHAKE-128).

The submission also claims that Frodo.KEM has:

- a tight proof of security in the ROM that it is CCA-secure based on the CPA security of Frodo.PKE; and
- a non-tight proof of security in the QROM that it is CCA-secure provided Frodo.PKE is OW-CPA-secure.

### 7.2.8 Performance

The FrodoKEM submission includes performance figures for AVX2-optimized implementations run on a 3,4 GHz Intel® Core™ i7-6700 processor. The implementations use either AES-128 or SHAKE-128 to instantiate XOF to generate the public matrix $\boldsymbol{A}$. The performance figures for each parameter set are shown in Table 29.

**Table 29: FrodoKEM performance figures**

| Set | Version | Frodo.KEM.Keygen (cycles) | Frodo.KEM.Enc (cycles) | Frodo.KEM.Dec (cycles) |
|---|---|---|---|---|
| Frodo-640 | AES | 1 384 000 | 1 861 000 | 1 751 000 |
| | SHAKE | 4 022 000 | 4 440 000 | 4 325 00 |
| Frodo-976 | AES | 2 896 000 | 3 563 000 | 3 399 000 |
| | SHAKE | 8 579 000 | 9 302 000 | 9 143 000 |
| Frodo-1344 | AES | 4 732 000 | 5 965 000 | 5 738 000 |
| | SHAKE | 15 191 000 | 16 357 000 | 16 148 000 |

## 7.3 HQC

### 7.3.1 Overview

HQC is a CCA-secure KEM based on the difficulty of syndrome decoding for random quasi-cyclic codes.

Let $R_2$ denote the polynomial ring $R_2 = \mathbb{Z}_2[X]/(X^n + 1)$ for a prime $n$. A quasi-cyclic parity-check matrix $\boldsymbol{H} \in R_2^{1\times\alpha}$ defines quasi-cyclic code $\mathcal{C} = \{\boldsymbol{c} \in R_2^{1\times\alpha} \mid \boldsymbol{Hc}^T = 0\}$ of length $\alpha n$ and rank $n$. The syndrome corresponding to an error vector $\boldsymbol{e} \in R_2^{1\times\alpha}$ of weight $d$ is the element $s = \boldsymbol{He}^T \in R_2$. Recovering the weight $d$ error from a syndrome $s$ and parity-check matrix $\boldsymbol{H}$ is believed to be computationally hard for random quasi-cyclic codes. Similarly, distinguishing a syndrome $s$ corresponding to a weight $d$ error from a uniformly random element of $R_2$ is also believed to be computationally hard.

HQC uses a variant of the Fujisaki-Okamoto transform from [i.16] to convert a CPA-secure PKE into a CCA-secure KEM. CCA security relies on the probability of a decoding failure being sufficiently low. HQC plaintexts are encoded using an auxiliary error correcting code that reduces the decoding failure rate and allows a theoretical bound to be calculated. Consequently, the HQC submission claims that HQC.KEM is CCA-secure for the proposed parameter sets.

### 7.3.2 Parameters

The main parameters for HQC are:

- $n$, the degree of the polynomial ring $R_2$;
- $w$, the weight of the private polynomials in key generation;
- $t$, the weight of the private polynomials in encryption and encapsulation;
- $n'$, the length of the auxiliary error correcting code; and
- $k$, the rank of the auxiliary error correction code.

The degree $n$ is chosen to be a prime where the only irreducible factors of $X^n + 1$ over the field $\mathbb{Z}_2$ are $X + 1$ and $X^{n-1} + \cdots + X + 1$.

NOTE: Where necessary, a weight function, denoted $wt(f)$, which returns the weight of the input polynomial $f$, is used to clarify certain requirements in the algorithm descriptions.

### 7.3.3 Auxiliary error correction

HQC encodes plaintexts using an auxiliary error correcting code of length $n'$ and rank $k$ that can efficiently correct at least $\delta$ errors. The auxiliary code is fixed for each parameter set and forms part of the public parameters for the scheme.

The third round submission specifies a concatenated code with a duplicated Reed-Muller code for the internal code, and a shortened Reed-Solomon code for the external code. Previous versions of HQC used different auxiliary codes.

The function Encode takes a $k$-bit plaintext as input and returns the corresponding codeword of length $n'$. The function Decode takes a codeword of length $n'$ as input and returns a candidate $k$-bit plaintext $m'$. Decoding will recover the correct plaintext provided the input contains at most $\delta$ errors, otherwise it will fail.

NOTE: The description of HQC given in the present clause assumes that the decoding algorithm always returns a plaintext, even if it is incorrect.

## 7.3.4 Auxiliary primitives

HQC makes use of several auxiliary, symmetric primitives:

- $G$, a 512-bit cryptographic hash function;
- $H$, a 512-bit cryptographic hash function;
- KDF, a key derivation function; and
- PRF, a pseudorandom function.

The submission describes how to instantiate $G$ using SHA3-512, and $H$ using SHA-512. It does not give explicit instantiations for KDF or PRF, but the reference implementation included with the submission uses SHA-512 for KDF and AES-256 in counter mode for PRF.

## 7.3.5 Public-key encryption scheme

### 7.3.5.1 HQC.PKE.KeyGen

Input: None

Output: Public key $pk$
Private key $sk$

1) Sample a uniformly random public element $a \in R_2$.
2) Sample uniformly random $s_0, e_0 \in R_2$ so that $wt(s_0) = wt(e_0) = w$.
3) Compute $b \coloneqq as_0 + e_0 \in R_2$.

The public key is $pk \coloneqq (a, b)$. The private key is $sk \coloneqq s_0$.

NOTE 1: The submission suggests that the public key can be compressed by sampling $a$ deterministically using PRF with a 320-bit seed.

NOTE 2: The submission suggests that the private key can be compressed by sampling $s_0$ and $e_0$ deterministically using PRF with a 320-bit seed.

### 7.3.5.2 HQC.PKE.Enc

Input: Public key $pk$
Plaintext $m$ (length $k$ bits)
Random seed $r$ (length 512 bits)

Output: Ciphertext $c$

1) Parse the public key as $pk = (a, b)$.
2) Sample $s_1, e_1, e_2 \in R_2$ deterministically using PRF with the seed $r$, so that $wt(s_1) = wt(e_1) = wt(e_2) = t$.
3) Compute the public element $c_0 \coloneqq as_1 + e_1 \in R_2$.

4) Compute $c_1 \coloneqq bs_1 + e_2 + \text{Encode}(m) \in R_2$.

The ciphertext is $c \coloneqq (c_0, c_1)$.

NOTE: The plaintext $m$ is encoded as a vector of length $n'$ where $n' < n$. The submission suggests compressing the second component $c_1$ of the ciphertext slightly by dropping the final $n - n'$ entries.

#### 7.3.5.3 HQC.PKE.Dec

Input: Private key $sk$
Ciphertext $c$

Output: Plaintext $m$ (length $k$ bits)

1) Parse the private key as $sk = s_0$ and the ciphertext as $c = (c_0, c_1)$.

2) Compute $v \coloneqq c_1 - c_0 s_0 \in R_2$.

3) Recover the plaintext by computing $m \coloneqq \text{Decode}(v)$.

### 7.3.6 Key encapsulation mechanism

#### 7.3.6.1 HQC.KEM.KeyGen

This is identical to HQC.PKE.KeyGen.

#### 7.3.6.2 HQC.KEM.Enc

Input: Public key $pk$

Output: Ciphertext $ct$
Session key $K$ (length 512 bits)

1) Sample a uniformly random $k$-bit plaintext $m$.

2) Encrypt $(c_0, c_1) \coloneqq \text{HQC.PKE.Enc}(pk, m, G(m))$.

3) Compute $c_2 \coloneqq H(m)$.

4) Derive the session key $K \coloneqq \text{KDF}(m \parallel c_0 \parallel c_1)$.

The ciphertext is $c \coloneqq (c_0, c_1, c_2)$. The session key is $K$.

#### 7.3.6.3 HQC.KEM.Dec

Input: Ciphertext $c$
Private key $sk$

Output: Session key $K$ (length 512 bits),
or $\perp$

1) Parse the ciphertext as $c = (c_0, c_1, c_2)$.

2) Decrypt $m' \coloneqq \text{HQC.PKE.Dec}(sk, (c_0, c_1))$.

3) Re-encrypt $(c_0', c_1') \coloneqq \text{HQC.PKE.Enc}(pk, m', G(m'))$.

4) Recompute $c_2' \coloneqq H(m')$.

5) If $(c_0', c_1', c_2') = (c_0, c_1, c_2)$, then derive the session key $K \coloneqq \text{KDF}(m' \parallel c_0 \parallel c_1)$.

6) Otherwise, return $\perp$.

### 7.3.7 Parameter sets

The HQC submission includes the parameter sets shown in Table 30.

**Table 30: Proposed parameters for HQC**

| Set | $n$ | $w$ | $t$ | $n'$ | $k$ | Failure probability | Claimed security |
|---|---|---|---|---|---|---|---|
| hqc-128 | 17 669 | 66 | 75 | 17 664 | 128 | $2^{-128}$ | Category 1 |
| hqc-192 | 35 851 | 100 | 114 | 35 840 | 192 | $2^{-192}$ | Category 3 |
| hqc-256 | 57 637 | 131 | 149 | 57 600 | 256 | $2^{-256}$ | Category 5 |

These parameter sets lead to the public key, private key and ciphertext sizes shown in Table 31.

**Table 31: HQC public key, private key, and ciphertext sizes**

| Set | Public key (bytes) | Private key (bytes) | Ciphertext (bytes) | Comments |
|---|---|---|---|---|
| hqc-128 | 4 418 | 2 209 | 4 481 | NOTE 1 |
| hqc-192 | 8 964 | 4 482 | 9 026 | NOTE 2 |
| hqc-256 | 14 410 | 7 205 | 14 469 | NOTE 3 |
| NOTE 1: The compressed private key is 40 bytes and compressed public key is 2 249 bytes.<br>NOTE 2: The compressed private key is 40 bytes and compressed public key is 4 522 bytes.<br>NOTE 3: The compressed private key is 40 bytes and compressed public key is 7 245 bytes. | | | | |

### 7.3.8 Security

The security of HQC depends on the difficulty of syndrome decoding in random quasi-cyclic codes. The best attacks involve information set decoding, and target multiple codewords or errors of the same weight obtained from the quasi-cyclic structure; see Annex C for more information. The HQC submission does not include explicit security costings, so Table 32 shows estimated costs for each parameter set for the classical security of key and message recovery, derived using the methodology described in Annex C.

**Table 32: Classical security estimates for HQC**

| Set | Key recovery (bits) | Message recovery (bits) |
|---|---|---|
| hqc-128 | 132 | 132 |
| hqc-192 | 200 | 200 |
| hqc-256 | 262 | 261 |

The provable security of HQC depends on variants of the decisional quasi-cyclic syndrome decoding problem. Public keys of the form $(a, b)$ can be distinguished from random as the parity of $b$ is completely determined by the parity of $a$ and the weight $w$ of the private key elements. Further, the component $c_1$ of the ciphertext does not correspond to a full syndrome as the final entries are dropped. Consequently, HQC.PKE is CPA-secure in the ROM based on the computational hardness of the decisional quasi-cyclic syndrome decoding problem with fixed parity and erasures.

The submission claims that the conversion of the CPA-secure HQC.PKE into the CCA-secure HQC.KEM is tight, but only holds in the ROM. The choice of auxiliary code allows a theoretical upper bound for the decoding failure rate to be calculated. Consequently, the submission claims that HQC.KEM is CCA-secure for the proposed parameter sets.

### 7.3.9 Performance

The HQC submission includes performance figures for an AVX2-optimized implementation run on a 3,6 GHz Intel® Core™ i7-7820X processor. The performance figures for each parameter set are shown in Table 33.

**Table 33: HQC performance figures**

| Set | HQC.KEM.KeyGen (cycles) | HQC.KEM.Enc (cycles) | HQC.KEM.Dec (cycles) |
|---|---|---|---|
| hqc-128 | 136 000 | 220 000 | 384 000 |
| hqc-192 | 305 000 | 501 000 | 821 000 |
| hqc-256 | 545 000 | 918 000 | 1 538 000 |

## 7.4 NTRU Prime

### 7.4.1 Overview

Let $R$ denote the polynomial ring $(\mathbb{Z}[X])/(X^n - X - 1)$, and $R_q$ denote the polynomial ring $(\mathbb{Z}_q[X])/(X^n - X - 1)$, where $n$ and $q$ are primes chosen so that $X^n - X - 1$ is irreducible modulo $q$. This means that $R_q$ is a prime degree extension field of $\mathbb{F}_q$.

The NTRU Prime submission describes two related but separate schemes:

- Streamlined NTRU Prime, which consists of a CPA-secure PKE scheme based on NTRU with rounding over the field $R_q$ that is converted into a CCA-secure KEM; and
- NTRU LPRime, which consists of a CPA-secure PKE scheme based on Ring Learning With Rounding (RLWR) over the field $R_q$ that is converted into a CCA-secure KEM.

Both Streamlined NTRU Prime and NTRU LPRime use a variant of the Fujisaki-Okamoto transform from [i.16] that includes an additional confirmation hash.

The submission makes it clear that Streamlined NTRU Prime and NTRU LPRime are defined as CCA-secure KEMs only, and that the underlying PKE schemes are building blocks that are not intended as separate submissions to the NIST standardization process.

NOTE: Streamlined NTRU Prime is abbreviated to SNTRUP, and NTRU LPRime is abbreviated to NTRULPR.

### 7.4.2 Parameters

The main parameters for NTRU Prime are:

- $n$, the degree of the polynomial ring $R_q$;
- $q$, the modulus of the polynomial ring $R_q$; and
- $w$, the weight of the ternary polynomials in key generation and encryption.

A polynomial in $R$ is said to be ternary if its coefficients all lie in $\{-1,0,1\}$.

The modulus $q$ and degree $n$ are primes chosen so that $X^n - X - 1$ is irreducible modulo $q$. The weight $w$ is chosen to ensure that NTRU Prime is not susceptible to decryption failures.

### 7.4.3 Auxiliary primitives

NTRU Prime makes use of the following auxiliary, symmetric primitives:

- $H$, a 256-bit cryptographic hash function;
- KDF, a key derivation function; and

- XOF, an extendable output function.

The submission describes how to use SHA-2 and AES to instantiate these primitives, as shown in Table 34.

**Table 34: Auxiliary symmetric primitives for NTRU Prime**

| Primitive | Instantiation |
|---|---|
| $H$ | SHA-512, output truncated to 256 bits |
| KDF | SHA-512, output truncated to 256 bits |
| XOF | AES-256 in counter mode |

Streamlined NTRU Prime and NTRU LPRime use a function $\mathrm{Round}: R_q \longrightarrow 3R$ which converts each coefficient of the input polynomial to $\{-(q-1)/2, \ldots, (q-1)/2\}$ and then rounds it to the nearest multiple of 3.

NTRU LPRime uses functions $\mathrm{Compress}: R_q \longrightarrow R_{16}$ and $\mathrm{Decompress}: R_{16} \longrightarrow R_q$ related to modulus switching in order to reduce the size of the ciphertext.

NTRU LPRime also uses a hash function $G: \{0,1\}^{256} \longrightarrow R$ that maps 256-bit values to ternary polynomials in $R$ of weight $w$. This is built from $H$ and XOF.

## 7.4.4 Streamlined NTRU Prime public-key encryption scheme

### 7.4.4.1 SNTRUP.PKE.KeyGen

Input: None

Output: Public key $pk$
Private key $sk$

1) Sample a uniformly random ternary polynomial $g \in R$ that is invertible modulo 3.
2) Compute $g'$, the inverse of $g$ modulo 3.
3) Sample a uniformly random ternary polynomial $f \in R$ of weight $w$.
4) Compute $h \coloneqq g/3f$ in $R_q$.

The public key is $pk \coloneqq h$. The private key is $sk \coloneqq (f, g')$.

NOTE: The polynomial $f$ in Step 4) is a non-zero element of the field $R_q$ so will always be invertible.

### 7.4.4.2 SNTRUP.PKE.Enc

Input: Public key $pk$
Plaintext $m$ (a ternary polynomial in $R$ of weight $w$)

Output: Ciphertext $c$

1) Parse the public key as $pk = h$.
2) Compute $hm$ in $R_q$.
3) Compute $c \coloneqq \mathrm{Round}(hm)$.

The ciphertext is $c$.

#### 7.4.4.3 SNTRUP.PKE.Dec

Input: Private key $sk$
Ciphertext $c$

Output: Plaintext $m$ (a ternary polynomial in $R$ of weight $w$)

1) Parse the private key as $sk = (f, g')$.
2) Compute $e \coloneqq 3fc$ in $R_q$ and convert to a polynomial in $R$ with coefficients in $\{-(q-1)/2, \ldots, (q-1)/2\}$.
3) Compute $m \coloneqq eg' \bmod 3$.
4) Convert $m$ to a ternary polynomial in $R$.

### 7.4.5 Streamlined NTRU Prime key encapsulation mechanism

#### 7.4.5.1 SNTRUP.KEM.KeyGen

Input: None

Output: Public key $pk$
Augmented private key $sk$

1) Sample a uniformly random ternary polynomial $z \in R$ of weight $w$.
2) Call SNTRUP.PKE.KeyGen() to generate a public key $pk$ and corresponding private key $sk'$.

The public key is $pk$. The augmented private key is $sk \coloneqq (sk', pk, z)$.

#### 7.4.5.2 SNTRUP.KEM.Enc

Input: Public key $pk$

Output: Ciphertext $c$
Session key $K$ (length 256 bits)

1) Sample a uniformly random ternary polynomial $m \in R$ of weight $w$.
2) Encrypt $c_0 \coloneqq \text{SNTRUP.PKE.Enc}(pk, m)$.
3) Compute $c_1 \coloneqq H(m \parallel pk)$.
4) Derive the session key $K \coloneqq \text{KDF}(1 \parallel m \parallel c_0 \parallel c_1)$.

The ciphertext is $c \coloneqq (c_0, c_1)$. The session key is $K$.

#### 7.4.5.3 SNTRUP.KEM.Dec

Input: Ciphertext $c$
Augmented private key $sk$

Output: Session key $K$ (length 256 bits)

1) Parse the ciphertext as $c = (c_0, c_1)$ and the private key as $sk = (sk', pk, z)$.
2) Decrypt $m' \coloneqq \text{SNTRUP.PKE.Dec}(sk', c_0)$.
3) Re-encrypt $c_0' \coloneqq \text{SNTRUP.PKE.Enc}(pk, m')$.
4) Compute $c_1' \coloneqq H(m' \parallel pk)$.
5) If $c_0' = c_0$ and $c_1' = c_1$, then derive the session key $K \coloneqq \text{KDF}(1 \parallel m' \parallel c_0 \parallel c_1)$.

6) Otherwise, derive a random key $K \coloneqq \mathrm{KDF}(0 \parallel z \parallel c_0 \parallel c_1)$.

## 7.4.6 NTRU LPRime public-key encryption scheme

### 7.4.6.1 NTRULPR.PKE.KeyGen

Input: None

Output: Public key $pk$
Private key $sk$

1) Sample a uniformly random 256-bit seed $d$.

2) Expand the seed $d$ using XOF to produce the public polynomial $a \in R_q$.

3) Sample a uniformly random ternary polynomial $s_0 \in R$ of weight $w$.

4) Compute $as_0$ in $R_q$.

5) Compute $t \coloneqq \mathrm{Round}(as_0)$.

The public key is $pk \coloneqq (t, d)$. The private key is $sk \coloneqq s_0$.

### 7.4.6.2 NTRULPR.PKE.Enc

Input: Public key $pk$
Plaintext $m$ (length 256 bits)

Output: Ciphertext $c$

1) Parse the public key as $pk = (t, d)$.

2) Expand the seed $d$ using XOF to produce the public polynomial $a \in R_q$.

3) Compute $s_1 \coloneqq G(m)$.

4) Compute $as_1$ in $R_q$.

5) Compute the public polynomial $u \coloneqq \mathrm{Round}(as_1)$.

6) Compute $v \coloneqq ts_1$ in $R_q$.

7) Encode the plaintext as an element $\mu \in R_q$ by setting each coefficient $\mu_i$ to $m_i(q-1)/2$.

8) Compute $v \coloneqq ts_1 + \mu \in R_q$.

9) Compute $v' \coloneqq \mathrm{Compress}(v)$.

The ciphertext is $c \coloneqq (u, v')$.

### 7.4.6.3 NTRULPR.PKE.Dec

Input: Private key $sk$
Ciphertext $c$

Output: Plaintext $m$ (length 256 bits)

1) Parse the private key as $sk = s_0$ and the ciphertext as $c = (u, v')$.

2) Compute $v \coloneqq \mathrm{Decompress}(v')$.

3) Compute $us_0 \in R_q$.

4) View each $v_i - (us_0)_i + 4w + 1 \in \mathbb{Z}_q$ as an integer in $\{-(q-1)/2, \ldots, (q-1)/2\}$.

5) If $v_i$ is negative, then set $m_i \coloneqq 1$, otherwise set $m_i \coloneqq 0$.

## 7.4.7 NTRU LPRime key encapsulation mechanism

### 7.4.7.1 NTRULPR.KEM.KeyGen

Input: None

Output: Public key $pk$
Augmented private key $sk$

1) Sample a uniformly random 256-bit value $z$.

2) Call NTRULPR.PKE.KeyGen() to generate a public key $pk$ and corresponding private key $sk'$.

The public key is $pk$. The augmented private key is $sk \coloneqq (sk', pk, z)$.

### 7.4.7.2 NTRULPR.KEM.Enc

Input: Public key $pk$

Output: Ciphertext $c$
Session key $K$ (length 256 bits)

1) Sample a uniformly random 256-bit message $m$.

2) Encrypt $c_0 \coloneqq \text{NTRULPR.PKE.Enc}(pk, m)$.

3) Compute $c_1 \coloneqq H(m \parallel pk)$.

4) Derive the session key $K \coloneqq \text{KDF}(1 \parallel m \parallel c_0 \parallel c_1)$.

The ciphertext is $c \coloneqq (c_0, c_1)$. The session key is $K$.

### 7.4.7.3 NTRULPR.KEM.Dec

Input: Ciphertext $c$
Augmented private key $sk$

Output: Session key $K$ (length 256 bits)

1) Parse the ciphertext as $c = (c_0, c_1)$ and the private key as $sk = (sk', pk, z)$.

2) Decrypt $m' \coloneqq \text{NTRULPR.PKE.Dec}(sk', c_0)$.

3) Re-encrypt $c_0' \coloneqq \text{NTRULPR.PKE.Enc}(pk, m')$.

4) Compute $c_1' \coloneqq H(m' \parallel pk)$.

5) If $c_0' = c_0$ and $c_1' = c_1$, then derive the session key $K \coloneqq \text{KDF}(1 \parallel m' \parallel c_0 \parallel c_1)$.

6) Otherwise, derive a random key $K \coloneqq \text{KDF}(0 \parallel z \parallel c_0 \parallel c_1)$.

## 7.4.8 Parameter sets

The NTRU Prime submission includes the parameter sets for Streamlined NTRU Prime shown in Table 35 and for NTRU LPRime shown in Table 36.

**Table 35: Proposed parameters for Streamlined NTRU Prime**

| Set | $n$ | $q$ | $w$ | Claimed security |
|---|---|---|---|---|
| sntrup653 | 653 | 4 621 | 288 | Category 1 |
| sntrup761 | 761 | 4 591 | 286 | Category 2 |
| sntrup857 | 857 | 5 167 | 322 | Category 3 |
| sntrup953 | 953 | 6 343 | 396 | Category 4 |
| sntrup1013 | 1 013 | 7 177 | 448 | Category 4 |
| sntrup1277 | 1 277 | 7 879 | 492 | Category 5 |

**Table 36: Proposed parameters for NTRU LPRime**

| Set | $n$ | $q$ | $w$ | Claimed security |
|---|---|---|---|---|
| ntrulpr653 | 653 | 4 621 | 252 | Category 1 |
| ntrulpr761 | 761 | 4 591 | 250 | Category 2 |
| ntrulpr857 | 857 | 5 167 | 281 | Category 3 |
| ntrulpr953 | 953 | 6 343 | 345 | Category 4 |
| ntrulpr1013 | 1 013 | 7 177 | 392 | Category 4 |
| ntrulpr1277 | 1 277 | 7 879 | 429 | Category 5 |

These parameter sets lead to the public key, private key, and ciphertext sizes for Streamlined NTRU Prime shown in Table 37 and for NTRU LPRime shown in Table 38.

**Table 37: Streamlined NTRU Prime public key, private key, and ciphertext sizes**

| Set | Public Key (bytes) | Private Key (bytes) | Ciphertext (bytes) |
|---|---|---|---|
| sntrup653 | 994 | 1 518 | 897 |
| sntrup761 | 1 158 | 1 763 | 1 039 |
| sntrup857 | 1 322 | 1 999 | 1 184 |
| sntrup953 | 1 505 | 2 254 | 1 349 |
| sntrup1013 | 1 623 | 2 417 | 1 455 |
| sntrup1277 | 2 067 | 3 059 | 1 847 |

**Table 38: NTRU LPRime public key, private key, and ciphertext sizes**

| Set | Public Key (bytes) | Private Key (bytes) | Ciphertext (bytes) |
|---|---|---|---|
| ntrulpr653 | 897 | 1 125 | 1 025 |
| ntrulpr761 | 1 039 | 1 294 | 1 167 |
| ntrulpr857 | 1 184 | 1 463 | 1 312 |
| ntrulpr953 | 1 349 | 1 652 | 1 477 |
| ntrulpr1013 | 1 455 | 1 773 | 1 583 |
| ntrulpr1277 | 1 847 | 2 231 | 1 975 |

### 7.4.9 Security

The main attacks considered are the hybrid and meet-in-the-middle attacks mentioned in Annex D. The security of NTRU Prime depends on the difficulty of finding lattice vectors that are short or close to a particular target vector in certain algebraically structured lattices. However, because it is not known how to exploit this algebraic structure, the attacks are costed using the general-purpose core-SVP methodology described in Annex D. The costs for each parameter set for Streamlined NTRU Prime are shown in Table 39. The costs for each parameter set for NTRU LPRime are shown in Table 40.

**Table 39: Core-SVP costings for Streamlined NTRU Prime (hybrid attack)**

| Set | Classical core-SVP (bits) | Quantum core-SVP (bits) |
|---|---|---|
| sntrup653 | 129 | 117 |
| sntrup761 | 153 | 139 |
| sntrup857 | 175 | 159 |
| sntrup953 | 196 | 178 |
| sntrup1013 | 209 | 190 |
| sntrup1277 | 270 | 245 |

**Table 40: Core-SVP costings for NTRU LPRime (hybrid attack)**

| Set | Classical core-SVP (bits) | Quantum core-SVP (bits) |
|---|---|---|
| ntrulpr653 | 130 | 118 |
| ntrulpr761 | 155 | 140 |
| ntrulpr857 | 176 | 160 |
| ntrulpr953 | 197 | 178 |
| ntrulpr1013 | 210 | 190 |
| ntrulpr1277 | 271 | 245 |

### 7.4.10 Performance

The NTRU Prime submission includes performance figures for optimized implementations of Streamlined NTRU Prime and NTRU LPRime run on a single core of a 3,5 GHz Intel® Xeon® E3-1275 processor. Table 41 shows the performance figures for Streamlined NTRU Prime with the sntrup761 parameter set. Table 42 shows the performance figures for NTRU LPRime with the ntrulpr761 parameter set.

**Table 41: Streamlined NTRU Prime performance figures**

| Set | NTRULPR.KEM.Keygen (cycles) | NTRULPR.KEM.Enc (cycles) | NTRULPR.KEM.Dec (cycles) |
|---|---|---|---|
| sntrup761 | 809 348 | 48 780 | 59 288 |

**Table 42: NTRU LPRime performance figures**

| Set | SNTRUP.KEM.Keygen (cycles) | SNTRUP.KEM.Enc (cycles) | SNTRUP.KEM.Dec (cycles) |
|---|---|---|---|
| ntrulpr761 | 44 540 | 72 388 | 86 976 |

## 7.5 SIKE

### 7.5.1 Overview

SIKE consists of a CPA-secure PKE scheme that is converted into a CCA-secure KEM using a variant of the Fujisaki-Okamoto transform from [i.16]. The security of SIKE is based on the Supersingular Isogeny Diffie-Hellman (SIDH) problem.

Let $E_0$ be a supersingular elliptic curve over $\mathbb{F}_{p^2}$ for a prime $p$ of the form $p = 2^{e_2}3^{e_3} - 1$ where:

- $\{P_2, Q_2\}$ is a pair of points of order $2^{e_2}$ on $E_0$ that generate the $2^{e_2}$-torsion subgroup; and
- $\{P_3, Q_3\}$ is a pair of points of order $3^{e_3}$ on $E_0$ that generate the $3^{e_3}$-torsion subgroup.

Each $a \in \{0, \ldots, 2^{e_2} - 1\}$ gives a degree $2^{e_2}$ isogeny $\phi_2 \colon E_0 \longrightarrow E_2$ with kernel $P_2 + [a]Q_2$ and each $b \in \{0, \ldots, 3^{e_3} - 1\}$ gives a degree $3^{e_3}$ isogeny $\phi_3 \colon E_0 \longrightarrow E_3$ with kernel $P_3 + [b]Q_3$. The corresponding isogenies $\phi_2' \colon E_3 \longrightarrow E$ with kernel $\phi_3(P_2) + [a]\phi_3(Q_2)$ and $\phi_3' \colon E_2 \longrightarrow E'$ with kernel $\phi_2(P_3) + [b]\phi_2(Q_3)$ produce isomorphic curves $E \cong E'$. This means that their $j$-invariants will be the same. The SIDH problem is to recover the $j$-invariant for $E$ given the curves $E_0, E_2, E_3$, the points $P_2, Q_2, P_3, Q_3$ on $E_0$, the points $\phi_2(P_3), \phi_2(Q_3)$ on $E_2$, and the points $\phi_3(P_2), \phi_3(Q_2)$ on $E_3$.

## 7.5.2 Parameters

The main parameters for SIKE are:

- $e_2$ and $e_3$, positive integers specifying a prime $p = 2^{e_2}3^{e_3} - 1$; and
- $\ell$, the length of messages and session keys in bits.

The parameters for SIKE also include a starting curve $E_0$ over $\mathbb{F}_{p^2}$, a pair of points $\{P_2, Q_2\}$ of order $2^{e_2}$ on $E_0$, and a pair of points $\{P_3, Q_3\}$ of order $3^{e_3}$ on $E_0$. The starting curve is chosen to be:

$$E_0 \colon y^2 = x^3 + 6x^2 + x$$

for all SIKE parameter sets.

## 7.5.3 Auxiliary primitives

SIKE makes use of several auxiliary, symmetric primitives:

- $H$, an $\ell$-bit cryptographic hash function;
- KDF, a key derivation function; and
- XOF, an extendable output function.

The submission describes how to use SHAKE-256 to instantiate these primitives.

## 7.5.4 Public-key encryption scheme

### 7.5.4.1 SIKE.PKE.KeyGen

Input: None

Output: Public key $pk$
Private key $sk$

1) Sample a uniformly random value $x$ in the range $\{0, \ldots, 2^k - 1\}$ where $k = \lfloor \log_2 3^{e_3} \rfloor$.
2) Let $\phi_3 \colon E_0 \longrightarrow E_3$ be the isogeny corresponding to the point $P_3 + [x]Q_3$ on the curve $E_0$.
3) Compute the points $P_2' \coloneqq \phi_3(P_2)$ and $Q_2' \coloneqq \phi_3(Q_2)$ on the curve $E_3$.

The public key is $pk \coloneqq (E_3, P_2', Q_2')$. The private key is $sk \coloneqq x$.

NOTE 1: The SIKE submission represents the public key by the x-coordinates of the points $P_2'$, $Q_2'$ and $P_2' - Q_2'$.

NOTE 2: The submission also includes a compressed version of SIKE that reduces the size of the public key using techniques from [i.29] and [i.30].

### 7.5.4.2 SIKE.PKE.Enc

Input: Public key $pk$
Plaintext $m$ (length $\ell$ bits)
Random seed $r$ (length $e_2$ bits)

Output: Ciphertext $c$

1) Parse the public key as $pk = (E_3, P'_2, Q'_2)$.

2) Let $\phi_2 : E_0 \longrightarrow E_2$ be the isogeny corresponding to the point $P_2 + [r]Q_2$ on the curve $E_0$.

3) Compute the points $P'_3 \coloneqq \phi_2(P_3)$ and $Q'_3 \coloneqq \phi_2(Q_3)$ on the curve $E_2$.

4) Set $c_0 \coloneqq (E_2, P'_3, Q'_3)$.

5) Let $\phi'_2 : E_3 \longrightarrow E$ be the isogeny corresponding to the point $P'_2 + [r]Q'_2$ on the curve $E_3$.

6) Compute the $j$-invariant $j$ of the curve $E$.

7) Compute $h \coloneqq H(j)$.

8) Compute $c_1 = h \oplus m$.

The ciphertext is $c \coloneqq (c_0, c_1)$.

NOTE 1: The SIKE submission represents the first component $c_0$ of the ciphertext by the $x$-coordinates of the points $P'_3$, $Q'_3$ and $P'_3 - Q'_3$.

NOTE 2: The submission also includes a compressed version of SIKE that reduces the size of the ciphertext using techniques from [i.29] and [i.30].

#### 7.5.4.3 SIKE.PKE.Dec

Input: Private key $sk$
Ciphertext $c$

Output: Plaintext $m$ (length $\ell$ bits)

1) Parse the private key as $sk = x$ and the ciphertext as $c = (c_0, c_1)$.

2) Parse $c_0 = (E_2, P'_3, Q'_3)$.

3) Let $\phi'_3 : E_2 \longrightarrow E'$ be the isogeny corresponding to the point $P'_3 + [x]Q'_3$ on the curve $E_2$.

4) Compute the $j$-invariant $j$ of the curve $E'$.

5) Compute $h \coloneqq H(j)$.

6) Recover the plaintext by computing $m \coloneqq h \oplus c_1$.

### 7.5.5 Key encapsulation mechanism

#### 7.5.5.1 SIKE.KEM.KeyGen

Input: None

Output: Public key $pk$
Augmented private key $sk$

1) Sample a uniformly random $\ell$-bit value $z$.

2) Call SIKE.PKE.KeyGen() to generate a public key $pk$ and corresponding private key $sk'$.

The public key is $pk$. The augmented private key is $sk \coloneqq (sk', pk, z)$.

#### 7.5.5.2 SIKE.KEM.Enc

Input: Public key $pk$

Output: Ciphertext $c$
Session key $K$ (length $\ell$ bits)

1) Sample a uniformly random $\ell$-bit message $m$.

2) Derive an $e_2$-bit value $r$ from $m$ and $pk$ using XOF.

3) Encrypt $c \coloneqq \text{SIKE.PKE.Enc}(pk, m, r)$.

4) Derive the session key $K \coloneqq \text{KDF}(m \parallel c)$.

The ciphertext is $c$. The session key is $K$.

#### 7.5.5.3 SIKE.KEM.Dec

Input: Augmented private key $sk$
Ciphertext $c$

Output: Session key $K$ (length $\ell$ bits)

1) Parse the private key as $sk = (sk', pk, z)$.

2) Decrypt $m' \coloneqq \text{SIKE.PKE.Dec}(sk', c)$.

3) Derive an $e_2$-bit value $r'$ from $m'$ and $pk$ using XOF.

4) Re-encrypt $c' \coloneqq \text{SIKE.PKE.Enc}(pk, m', r')$.

5) If $c' = c$ then derive the session key $K \coloneqq \text{KDF}(m' \parallel c)$.

6) Otherwise, derive a random key $K \coloneqq \text{KDF}(z \parallel c)$.

NOTE: The SIKE submission does not perform a full re-encryption check. Instead, it checks that the first component $c_0'$ of the ciphertext recomputed in Steps 1) to 4) of SIKE.PKE.Enc matches the first component $c_0$ of the provided ciphertext $c$. This is sufficient to guarantee that the full ciphertexts $c'$ and $c$ will match.

### 7.5.6 Parameter sets

The SIKE submission includes the parameter sets shown in Table 43.

**Table 43: Proposed parameters for SIKE**

| Set | $e_2$ | $e_3$ | $\ell$ | Claimed security |
|---|---|---|---|---|
| SIKEp434 | 216 | 137 | 128 | Category 1 |
| SIKEp503 | 250 | 159 | 192 | Category 2 |
| SIKEp610 | 305 | 192 | 192 | Category 3 |
| SIKEp751 | 372 | 239 | 256 | Category 5 |

Each parameter set is given in a regular form and in a compressed form. These parameter sets lead to the public key, private key, and ciphertext sizes as shown in Table 44.

**Table 44: SIKE public key, private key, and ciphertext sizes**

| Set | Version | Public key (bytes) | Private key (bytes) | Ciphertext (bytes) |
|---|---|---|---|---|
| SIKEp434 | Regular | 330 | 374 | 346 |
| | Compressed | 197 | 350 | 236 |
| SIKEp503 | Regular | 378 | 434 | 402 |
| | Compressed | 225 | 407 | 280 |
| SIKEp610 | Regular | 462 | 524 | 486 |
| | Compressed | 274 | 491 | 336 |
| SIKEp751 | Regular | 564 | 644 | 596 |
| | Compressed | 335 | 602 | 410 |

### 7.5.7 Security

The security of SIKE depends on the difficulty of recovering a private isogeny. Generic meet-in-the-middle attacks require $O(p^{1/4})$ work and $O(p^{1/4})$ memory. The SIKE submission argues that the most relevant classical attack is the parallel collision-finding algorithm of van Oorschot and Wiener. Table 45 gives estimates for the classical cost of parallel collision-finding (in gates) when the memory is restricted to $2^{96}$ bits.

**Table 45: Classical security estimates for SIKE**

| Set | Classical gates |
|---|---|
| SIKEp434 | $2^{142}$ |
| SIKEp503 | $2^{169}$ |
| SIKEp610 | $2^{209}$ |
| SIKEp751 | $2^{263}$ |

Tani's quantum claw-finding algorithm recovers the private isogenies with a query complexity of $O(q^{1/6})$. However, the SIKE submission argues that classical parallel collision-finding requires fewer resources than quantum claw-finding when the circuit depth is restricted and quantum memory costs are taken into account. Table 46 gives estimates for the quantum cost of claw-finding with a maximum circuit depth of $2^{64}$ or $2^{96}$.

**Table 46: Quantum security estimates for SIKE**

| Set | Quantum gates | |
|---|---|---|
| | ***Maximum circuit depth of $2^{64}$*** | ***Maximum circuit depth of $2^{96}$*** |
| SIKEp434 | $2^{175}$ | $2^{143}$ |
| SIKEp503 | $2^{210}$ | $2^{178}$ |
| SIKEp610 | $2^{264}$ | $2^{232}$ |
| SIKEp751 | $2^{336}$ | $2^{304}$ |

The SIKE submission claims that SIKE.PKE is CPA-secure in the ROM based on the computational hardness of the SIDH problem, and that SIKE.KEM is CCA-secure in the ROM based on the CPA-security of SIKE.PKE. The submission notes that SIKE.PKE is CPA-secure in the standard model based on the hardness of a decisional variant of the SIDH problem, though this does not extend to CCA security in the standard model for SIKE.KEM.

### 7.5.8 Performance

The SIKE submission includes performance figures for an optimized implementation run on a 3,4 GHz Intel® Core™ i7-6700 processor. The performance figures for each parameter set are shown in Table 47.

**Table 47: SIKE performance figures**

| Set | Version | SIKE.KEM.KeyGen (cycles) | SIKE.KEM.Enc (cycles) | SIKE.KEM.Dec (cycles) |
|---|---|---|---|---|
| SIKEp434 | Regular | 5 927 000 | 9 681 000 | 10 343 000 |
| | Compressed | 10 158 000 | 15 120 000 | 11 077 000 |
| SIKEp503 | Regular | 8 243 000 | 13 544 000 | 14 415 000 |
| | Compressed | 14 452 000 | 21 190 000 | 15 733 000 |
| SIKEp610 | Regular | 14 890 000 | 27 254 000 | 27 445 000 |
| | Compressed | 26 360 000 | 37 470 000 | 29 216 000 |
| SIKEp751 | Regular | 25 197 000 | 40 703 000 | 43 851 000 |
| | Compressed | 40 935 000 | 63 254 000 | 46 606 000 |

# Annex A:
# Proofs of security

## A.1 Introduction

Proofs of security, also referred to as security reductions, usually involve showing that the ability of an adversary to break the security of a cryptographic scheme would necessarily imply their ability to solve a related problem that is believed to be computationally hard. Proofs of security can provide reassurance that cryptographic schemes are secure, but they require careful interpretation.

## A.2 Security models

To construct a proof of security it is necessary to specify what it means for the scheme of interest to be secure (and hence what it would mean to break that scheme) and the values and computational resources assumed to be available to an adversary. Examples of definitions of security include CPA and CCA security, as described in Annex B. The definitions for PKE schemes are subtly different to the definitions for KEMs, and indeed to the definitions for symmetric encryption schemes and modes of operation.

## A.3 Computational resources

With respect to computational resources, it is expected than an adversary with access to a quantum computer will be able to efficiently perform operations that are not available to an adversary that only has access to classical computers. To model such an adversary it is important to account for properties of quantum computation such as the no-cloning theorem; consequently, it is often necessary to construct different proofs for different models of computation. Proofs of security usually allow adversaries to make educated guesses, so classical adversaries are restricted to probabilistic polynomial-time algorithms, and quantum adversaries are restricted to quantum polynomial-time algorithms.

Proofs of security are usually quantified with respect to a security parameter, and usually involve asymptotic results. More specifically, proofs of security are usually constructed to hold for large enough security parameters, which can mean they do not hold for small parameters. Because asymptotic results tend to be non-constructive, it can be difficult to determine how large the security parameter needs to be for a proof to hold, and it can be difficult to interpret what a proof of security means if it only holds for parameters that are significantly larger than those used in practice.

## A.4 Tightness

Using the terminology of complexity theory, a proof of security consists of a reduction from a target problem to the problem of breaking the cryptosystem. This usually involves constructing an algorithm that uses an oracle that breaks the cryptographic scheme as a subroutine to solve the target problem. The goal is to show that because the target problem is computationally hard, it cannot be possible to instantiate the oracle with an efficient algorithm, and hence that the cryptographic scheme is secure.

The efficiency and effectiveness of the reduction is important when interpreting a proof of security. If the algorithm that uses the oracle has a similar running time and probability of success to the oracle, the proof is said to be tight. If the algorithm requires significantly more time to run than the oracle, or has a significantly lower probability of success, or both, the proof is said to have a tightness gap. A proof with a large tightness gap tells us relatively little about the security of the cryptographic scheme of interest; depending on the size of the gap, in some cases it can be possible to attack the scheme without solving the related hard problem.

## A.5 Worst-case to average-case reductions

It is also important to consider what assumptions are made about the intractability of the target problem. A security proof that assumes that the target problem is hard on average provides a weaker result than a proof that assumes the target problem is hard only in the worst case. A security proof that a cryptosystem is hard to attack on average, provided the target problem is computationally hard in the worst case, is referred to as a worst-case to average-case reduction.

## A.6 Random oracles

To construct some proofs of security it is necessary to make assumptions about or use idealized versions of certain cryptographic primitives, such as ciphers and hash functions; this can mean the proof does not apply to a concrete implementation. Proofs of security that avoid such assumptions are said to be constructed in the standard model.

In the Random Oracle Model (ROM) hash functions are modelled as ideal entities, referred to as random oracles, which respond to new queries with responses selected uniformly at random from the output domain, and respond to previously seen queries with whatever answer was given the first time the query was received.

In the ROM it is assumed that adversaries interact classically with random oracles, but in the Quantum Random Oracle Model (QROM) it is assumed that adversaries can query a random oracle in a quantum superposition of states. Although the QROM affords an adversary more computational power, it can be difficult to compare proofs in the ROM to proofs in the QROM, particularly if it is possible to construct a tight proof in the ROM but not the QROM.

# Annex B:
# Security properties

## B.1 Introduction

The two main security goals that are relevant for PKE and KEMs are indistinguishability under chosen-plaintext, and indistinguishability under chosen-ciphertext. Both security goals are usually modelled as games that take place between an attacker and a challenger; the games are slightly different for public-key encryption than for key encapsulation.

## B.2 Public-key encryption

The two security definitions for PKE schemes are:

- **Chosen-Plaintext Attack (CPA) security for PKE.** The challenger generates a key pair for some security parameter and provides the public values to the attacker. The attacker can perform a polynomial (in the size of the security parameter) number of operations, then submit a pair of plaintext messages of its choice to the challenger. The challenger selects one of the messages uniformly at random, encrypts it using fresh random values and returns the resulting ciphertext to the attacker. The goal of the attacker is to determine which of the two messages the challenger encrypted. A PKE scheme is said to be indistinguishable under chosen-plaintext attack, or CPA-secure, if every probabilistic polynomial time attacker has only a negligible (in the size of the security parameter) advantage over random guessing.

- **Chosen-Ciphertext Attack (CCA) security for PKE.** The CCA game for public-key encryption is the same as the CPA game described above, except the attacker is given access to a decryption oracle that it can query with values of its choice. In the basic version, often referred to as CCA1 security, the attacker is only allowed to use the decryption oracle prior to submitting its choice of messages to the challenger. In the adaptive case, often referred to as CCA2 security, the attacker is allowed to use the decryption oracle before and after it submits its choice of messages to the challenger, but it is not allowed to query the decryption oracle with the ciphertext received from the challenger.

## B.3 Key encapsulation

The two security definitions for KEMs are:

- **Chosen-Plaintext Attack (CPA) security for KEMs.** The challenger generates a key pair for some security parameter and provides the public values to the attacker. The attacker can perform a polynomial (in the size of the security parameter) number of operations, then request a challenge from the challenger. The challenger calls the encapsulation routine, which returns a uniformly random key, $K$, and a ciphertext, which represents the encapsulation of $K$. The challenger provides the attacker with the ciphertext, and either $K$ or a uniformly random value, $K'$. The goal of the attacker is to determine whether it has been given $K$ or $K'$. A KEM is said to be indistinguishable under chosen-plaintext attack, or CPA-secure, if every (probabilistic) polynomial time attacker has only a negligible (in the size of the security parameter) advantage over random guessing.

- **Chosen-Ciphertext Attack (CCA) security for KEMs.** The CCA game for key encapsulation is the same as the CPA game described above, except the attacker is given access to a decapsulation oracle that it can query with values of its choice. In the basic version, often referred to as CCA1 security, the attacker is only allowed to use the decapsulation oracle prior to requesting a challenge. In the adaptive case, often referred to as CCA2 security, the attacker is allowed to use the decapsulation oracle before and after it requests a challenge, but it is not allowed to query the decapsulation oracle with the ciphertext received from the challenger.

## B.4 One-wayness

One-Wayness against Chosen-Plaintext Attack (OW-CPA) security captures a weaker notion of CPA security for PKE schemes. The challenger provides the attacker with a ciphertext corresponding to the encryption of a message selected uniformly at random from the space of all possible messages. The goal of the attacker is to recover the message from the ciphertext. If a PKE scheme is CPA-secure then it is OW-CPA-secure, but the converse need not be true.

## B.5 CPA to CCA transforms

CPA security, including OW-CPA security, is used to model passive attackers, which are usually thought of as third parties that can observe messages exchanged between the sender and the recipient, who are both assumed to behave honestly. CCA security is used to model active attackers, such as a malicious sender that constructs malformed ciphertexts to learn information about the recipient's private key. Consequently, when using a CPA-secure PKE scheme or KEM in the presence of active attackers, a recipient's private key can only be used securely once.

There are standard techniques available for converting a CPA-secure PKE scheme into a CCA-secure KEM, where recipients can reuse their private keys even in the presence of active attackers. The most common approach is to use a variant of the Fujisaki-Okamoto transform [i.15]. This usually involves the sender deriving the randomness required for encryption from the value to be encrypted; note that this includes the randomness required for the sender to generate an ephemeral key pair. The recipient can decrypt the received ciphertext, then use the resulting message to attempt to rederive the randomness and reconstruct the ciphertext to check that the sender followed the protocol honestly.

# Annex C: Code-based costing methodology

## C.1 Introduction

Message recovery attacks against code-based PKE schemes and KEMs involve finding the codeword that is closest to a target vector derived from the ciphertext. Key recovery attacks for QC-MDPC schemes involve finding non-zero codewords of moderate weight. Estimates for the costs of both of these attacks usually assume that a variant of information set decoding is used.

NOTE: Key recovery attacks for schemes that use binary Goppa codes rely on fundamentally different techniques and are typically much more expensive than message recovery.

## C.2 Information set decoding

Information set decoding algorithms take a parity-check matrix $H \in \mathbb{F}_2^{(n-k)\times n}$ and syndrome $s \in \mathbb{F}_2^{n-k}$, and recovers an error vector $e \in \mathbb{F}_2^n$ of target weight $d$ such that $s = He^T$. The simplest version of information set decoding by Prange [i.21] performs the following loop:

1) Randomly permute the columns of $H$ and row reduce to echelon form $[I_{n-k} \quad H']$ where $H' \in \mathbb{F}_2^{(n-k)\times k}$.

2) Apply the same row operations to $s$ to give a vector $e' \in \mathbb{F}_2^{n-k}$, then set $e = [e' \quad 0_k] \in \mathbb{F}_2^n$ and reverse the column permutation.

3) Start again if $e$ does not have weight $d$.

The algorithm succeeds when the column permutation moves all $d$ errors into the first $n-k$ positions. More efficient variants of information set decoding reduce the number of iterations by allowing some of the errors to occur in the final $k$ positions. However, each iteration is more expensive and can involve significant amounts of memory.

## C.3 Asymptotic complexity

The asymptotic complexity of information set decoding depends on the rate of the code and the weight of the codeword or error being recovered. Table C.1 is adapted from [i.18]. It gives worst-case complexities for full decoding (that is, finding the closest codeword to an arbitrary vector) in a random code of length $n$.

**Table C.1: Asymptotic complexity of full decoding for information set decoding variants**

| Algorithm | Work | Memory | Reference |
|---|---|---|---|
| Lee-Brickell | $2^{0,1208n+o(n)}$ | $O(n^2)$ | [i.22] |
| Stern | $2^{0,1167n+o(n)}$ | $2^{0,0318n+o(n)}$ | [i.23] |
| May-Meurer-Thomae | $2^{0,1116n+o(n)}$ | $2^{0,0374n+o(n)}$ | [i.24] |
| Becker-Joux-May-Meurer | $2^{0,1019n+o(n)}$ | $2^{0,0769n+o(n)}$ | [i.25] |

In code-based PKE schemes and KEMs there is a guarantee on the weight of the codeword or error vector that is much smaller than needed for full decoding: binary Goppa codes use weight $d = O(n/\log(n))$ errors and QC-MDPC codes use weight $d = O(\sqrt{n})$. Canto Torres and Sendrier [i.18] show that decoding an error or recovering a codeword of sub-linear weight $d = o(n)$ has asymptotic complexity:

$$C_{\mathrm{ISD}}(n,k,d) = 2^{cd+o(d)}$$

where $c = \log_2(n/(n-k))$. This is independent of the variant of information set decoding being used.

## C.4 Decoding one out of many

If an attacker has a collection of syndromes and only needs to decode one of them, then collision decoding techniques can be used to reduce the cost of recovery compared to standard information set decoding. Sendrier [i.26] showed that the cost to decode one out of $T$ syndromes is:

$$\frac{C_{\mathrm{ISD}}(n,k,d)}{\sqrt{T}}$$

under mild assumptions on $T$.

This is directly relevant to code-based schemes that use quasi-cyclic codes. Message recovery corresponds to decoding one of the syndromes obtained by taking quasi-cyclic shifts of the ciphertext.

NOTE: Key recovery for schemes that use QC-MDPC codes corresponds to finding one of the quasi-cyclic shifts of the minimum weight codeword in the dual code. In this case, if there are $T$ quasi-cyclic shifts then the cost to recover one of them is $C_{\mathrm{ISD}}(n, k, d)/T$; that is, the cost is reduced by a factor of $T$ rather than $\sqrt{T}$.

## C.5 Quantum information set decoding

Quantum information set decoding algorithms apply Grover search [i.27] or quantum walks [i.28] to classical information set decoding algorithms to reduce the number of iterations. The quantum speed-up is similar to the quantum speed-up for AES key recovery, and has similar trade-offs between maximum circuit depth and total gate count.

## C.6 Costing metrics

There is a lack of consensus among the submissions on which metric to use when costing information set decoding:

- The BIKE submission suggests using the asymptotic complexity from [i.18] and ignoring the sub-exponential terms. This is the closest analogue to the core-SVP methodology for lattice-based schemes.
- The HQC submission suggests using the asymptotic complexity from [i.18] but proposes specific expressions for sub-exponential terms rather than ignoring them.
- The Classic McEliece submission argues that for the variants of information set decoding that involve significant amounts of memory, the practical cost will be dominated by the memory accesses.

# Annex D:
# Lattice costing methodology

## D.1 Introduction

There are two main attacks against lattice-based cryptosystems: the primal attack, and the dual attack. Both attacks involve finding short vectors in lattices. Primal attacks are considered for both LWE and NTRU-like schemes, but dual attacks only apply to LWE schemes. The primal attack constructs a lattice (referred to as the primal lattice) associated with a given public key, which contains the corresponding private key as a unique shortest vector. The dual attack distinguishes public keys from random by finding short vectors in a related lattice, referred to as the dual lattice. Estimates for the costs of both attacks are derived in terms of lattice reduction algorithms.

## D.2 Lattice reduction

Lattice reduction algorithms convert a given basis for a lattice into another basis for the same lattice that consists of vectors that are shorter and more orthogonal to one another. In general, producing an optimal basis that contains a shortest non-zero vector in the lattice is NP-hard, but the primal and dual attacks each only require a basis that consists of short enough vectors.

Block-Korkine-Zolotarev (BKZ) is a family of algorithms that reduce a basis for any $n$-dimensional lattice by using an oracle that solves the exact Shortest Vector Problem (SVP) in a smaller dimension, $\beta$. Efficiency is balanced against the quality of the basis produced by varying $\beta$, which is referred to as the block size, and using different SVP oracles. The larger the block size the better the basis returned, in the sense that it will consist of shorter vectors, but at the cost of more computation: clearly if $\beta = n$ the SVP oracle computes a shortest nonzero vector in the full $n$-dimensional lattice. This trade-off is used to determine how to parameterize lattice-based cryptosystems so that the quality of the basis required to carry out an attack is prohibitively expensive.

## D.3 Enumeration and sieving

There are two main approaches to instantiating the SVP oracle when using BKZ: enumeration, and sieving. Enumeration algorithms run in super-exponential time but require relatively little memory. Sieving algorithms run in exponential time, but also require exponential memory; however, current approaches to costing tend to focus on running time by assuming (optimistically) that memory and memory accesses are free. In practice, enumeration algorithms are usually more efficient than sieving algorithms for small dimensional lattices, with a cross-over at around dimension 80. Consequently, it is usually assumed that sieving algorithms are more efficient (in terms of running time) than enumeration algorithms when considering the security of lattice-based cryptosystems.

The heuristic complexity of the best sieving algorithms is approximately $2^{0,292\beta+o(\beta)}$ in the classical model, and $2^{0,265\beta+o(\beta)}$ in the quantum model, which makes use of Grover's search algorithm. The sub-exponential factors are usually ignored to give respective costs of $2^{0,292\beta}$ and $2^{0,265\beta}$. Despite the slightly better bound in the quantum model, it is not clear whether quantum computation will lead to improved running times in practice [i.20].

## D.4 Core-SVP

Improvements continue to be made in terms of amortizing the cost of calling the SVP oracle when running BKZ. Consequently, the core-SVP methodology makes the simplifying assumption that just a single call is made. For example, for KYBER512 the required block size for the primal attack is estimated to be 403. Therefore, KYBER512 has a classical core-SVP cost of approximately 118 bits, and a quantum core-SVP cost of approximately 107 bits.

## D.5 Alternative metrics

The core-SVP methodology provides a simple but conservative approach to costing attacks against lattice-based cryptosystems. An alternative metric is to consider the number of gates required to implement an attack, but this approach is also not well understood. For some schemes it is possible to combine lattice reduction techniques with meet-in-the-middle ideas to produce hybrid attacks that are more efficient than just relying on lattice reduction; however, this is a relatively new area of research. Consequently, understanding how to produce more accurate costings of attacks against lattice-based cryptosystems remains an important and active area of research.

# History

| Document history | | |
|---|---|---|
| V1.1.1 | September 2021 | Publication |
| V1.1.2 | October 2021 | Publication |
| | | |
| | | |
| | | |