FrodoKEM
Learning With Errors Key Encapsulation

Algorithm Specifications And Supporting Documentation

Erdem Alkim Joppe W. Bos Léo Ducas Patrick Longa Ilya Mironov
Michael Naehrig Valeria Nikolaenko Chris Peikert Ananth Raghunathan
Douglas Stebila

September 30, 2020

Contents

1 Introduction and design rationale

This submission defines a family of key-encapsulation mechanisms (KEMs), collectively called FrodoKEM. The FrodoKEM schemes are designed to be *conservative* yet *practical* post-quantum constructions whose security derives from cautious parameterizations of the well-studied *learning with errors* problem, which in turn has close connections to conjectured-hard problems on *generic*, "algebraically unstructured" lattices.

Concretely, FrodoKEM is designed for IND-CCA security at three levels:

- FrodoKEM-640, which targets Level 1 in the NIST call for proposals (matching or exceeding the brute-force security of AES-128), and
- FrodoKEM-976, which targets Level 3 in the NIST call for proposals (matching or exceeding the brute-force security of AES-192).
- FrodoKEM-1344, which targets Level 5 in the NIST call for proposals (matching or exceeding the brute-force security of AES-256).

Two variants of each of the above schemes are provided:

- FrodoKEM-640-AES, FrodoKEM-976-AES, and FrodoKEM-1344-AES, which use AES128 to pseudorandomly generate a large public matrix (called $\mathbf{A}$).
- FrodoKEM-640-SHAKE, FrodoKEM-976-SHAKE, and FrodoKEM-1344-SHAKE, which use SHAKE128 to pseudorandomly generate the matrix.

The AES variants are particularly suitable for devices having AES hardware acceleration (such as AES-NI on Intel platforms), while the SHAKE variants generally provide competitive or better performance in comparison with the AES variants in the absence of hardware acceleration.

In the remainder of this section, we outline FrodoKEM's scientific lineage, briefly explain our design choices (with further details appearing in subsequent sections), and describe other features of our proposal beyond those explicitly requested by NIST.

Appendix A describes the changes/tweaks since the initial Round 1 submission to NIST.

1.1 Pedigree

The core of FrodoKEM is a public-key encryption scheme called FrodoPKE,[1] whose IND-CPA security is tightly related to the hardness of a corresponding *learning with errors* problem. Here we briefly recall the scientific lineage of these systems. See the surveys [90, 114, 101] for further details.

The seminal works of Ajtai [3] (published in 1996) and Ajtai–Dwork [4] (published in 1997) gave the first cryptographic constructions whose security properties followed from the conjectured *worst-case* hardness of various problems on point *lattices* in $\mathbb{R}^n$. In subsequent years, these works were substantially refined and improved, e.g., in [63, 32, 89, 112, 92]. Notably, in work published in 2005, Regev [113] defined the *learning with errors* (LWE) problem, proved the hardness of (certain parameterizations of) LWE assuming the hardness of various worst-case lattice problems against *quantum* algorithms, and defined a public-key encryption scheme whose IND-CPA security is tightly related to the hardness of LWE.[2]

Regev's initial work on LWE was followed by much more, which, among other things:

- provided additional theoretical support for the hardness of various LWE parameterizations (e.g., [97, 15, 30, 52, 91, 103]),
- extensively analyzed the concrete security of LWE and closely related lattice problems (e.g., [93, 42, 84, 9, 41, 6, 8, 77, 73, 12, 13, 26, 5, 11], among countless others), and
- constructed LWE-based cryptosystems with improved efficiency or additional functionality (e.g., [72, 105, 104, 61, 34, 31, 62, 24, 64]).

In particular, in work published in 2011, Lindner and Peikert [83] gave a more efficient LWE-based public-key encryption scheme that uses a square public matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times n}$ instead of an oblong rectangular one.

The FrodoPKE scheme from this submission is an instantiation and implementation of the Lindner–Peikert scheme [83] with some modifications, such as: pseudorandom generation of the public matrix $\mathbf{A}$ from a small seed, more balanced key and ciphertext sizes, and new LWE parameters.

[1] FrodoPKE is an intermediate building block used to create FrodoKEM, but is not a submission to the NIST competition.

[2] As pointed out in [98], Regev's encryption scheme implicitly contains an (unauthenticated) "approximate" key-exchange protocol analogous to the classic Diffie–Hellman protocol [50].

Frodo. FrodoPKE is closely related to an earlier work [26], called "Frodo," by a subset of the authors of this submission, which appeared at the 2016 ACM CCS conference. For clarity, we refer to the conference version as FrodoCCS, and the KEM defined in this submission as FrodoKEM. The main differences are as follows:

- FrodoCCS was described as an unauthenticated key-exchange protocol, which can equivalently be viewed as an IND-CPA-secure KEM, whereas FrodoKEM is designed to be an IND-CCA-secure KEM.
- FrodoCCS used a "reconciliation mechanism" to extract shared-key bits from approximately equal values (similarly to [51, 100, 27, 13]), whereas FrodoKEM uses simpler key transport via public-key encryption (as in [113, 83]).
- FrodoKEM uses significantly "wider" LWE error distributions than FrodoCCS does, which conform to certain worst-case hardness theorems (see below).
- FrodoKEM uses different symmetric-key primitives than FrodoCCS does.

Chosen-ciphertext security. FrodoKEM achieves IND-CCA security by way of a transformation of the IND-CPA-secure FrodoPKE. In work published in 1999, Fujisaki and Okamoto [57] gave a generic transform from an IND-CPA PKE to an IND-CCA PKE, in the random-oracle model. At a high level, the Fujisaki–Okamoto (FO) transform derives encryption coins pseudorandomly, and decryption regenerates these coins to re-encrypt and check that the ciphertext is well-formed. In 2016, Targhi and Unruh [122] gave a modification of the Fujisaki–Okamoto transform that achieves IND-CCA security in the *quantum* random-oracle model (QROM) by adding an extra hash. In 2017, Hofheinz, Hövelmanns, and Kiltz [68] gave several variants of the Fujisaki–Okamoto and Targhi–Unruh transforms that in particular convert an IND-CPA-secure PKE into an IND-CCA-secure KEM, and analyzed them in both the classical and quantum random-oracle models, even for PKEs with non-zero decryption error. Jiang et al. [70] show how to prove security of one of these variant FO transforms (specifically, $\mathsf{FO}^{\not\perp}$) in the QROM without requiring the extra hash from Targhi–Unruh. FrodoKEM is constructed from FrodoPKE using a slight variant of the $\mathsf{FO}^{\not\perp}$ transform that includes additional values in hash computations to avoid multi-target attacks.

1.2 Design overview and rationale

Given the high cost and slow deployment of entirely new cryptographic systems, the desired decades-long lifetime of such systems, and the unpredictable trajectory of quantum computing technology and quantum cryptanalysis over the coming years, we argue that any post-quantum standard should follow a *conservative* approach that errs comfortably on the side of *security* and *simplicity* over performance and (premature) optimization. This principle permeates the design choices behind FrodoKEM, as we now describe.

1.2.1 Generic, algebraically unstructured lattices

The security of every public-key cryptosystem depends on the presumed intractability of one or more computational problems. In lattice-based cryptography, the (plain) LWE problem [113] relates to solving a "noisy" linear system modulo a known integer; it can also be interpreted as the problem of decoding a random "unstructured" lattice from a certain class. There are also "algebraically structured" variants, called Ring-LWE [86, 103] and Module-LWE [29, 80], and problems associated with the classic NTRU cryptosystem [67], which are more compact and computationally efficient, but also have the potential for weaknesses due to the extra structure.

After a good deal of investigation, the state of the art for recommended parameterizations of algebraic LWE variants does not indicate any particular weaknesses in comparison to plain LWE. However, at present there appear to be some gaps between the (quantum) complexity of some *related*, seemingly weaker problems on certain kinds of algebraic lattices and their counterparts on general lattices. (See below for details.) Of course, this only represents our current understanding of these problems, which could potentially change with further cryptanalytic effort.

Given the unpredictable long-term outlook for algebraically structured lattices, and because any post-quantum standard should remain secure for decades into the future—including against new quantum attacks—we have based our proposal on the algebraically unstructured, plain LWE problem with conservative parameterizations (see Section 1.2.2). While this choice comes at some cost in efficiency versus algebraic lattice problems, our proposal is still eminently practical for the vast majority of today's devices, networks, and applications, and will become only more so in the coming years.

Algebraic lattices. Ring-LWE, Module-LWE, and NTRU-related problems can be viewed as decoding (or in the case of NTRU, shortest vector) problems on random "algebraically structured" lattices over certain polynomial rings. (Formally, the lattices are modules of a certain rank over the ring.) Similarly to LWE, various parameterizations of Ring-LWE and Module-LWE, and even some non-standard versions of NTRU [121], have been proven hard assuming the *worst-case* quantum hardness of certain problems on lattices corresponding to *ideals* or *modules* over the ring [86, 80, 103].

For recommended parameterizations of Ring- and Module-LWE, the current best attacks perform essentially the same as those for plain LWE, apart from some obvious linear-factor (in the ring dimension) savings in time and memory; the same goes for the underlying worst-case problems on ideal and module lattices, versus generic lattices [42, 118, 69, 28, 78].[3] However, some conventional NTRU parameterizations admit specialized attacks with significantly better asymptotic performance than on generic lattices with the same parameters [73, 74]. In addition, a series of recent works [33, 44, 45] has yielded a quantum polynomial-time algorithm for very large but *subexponential* $2^{\tilde{O}(\sqrt{n})}$ approximations to the worst-case Shortest Vector Problem on *ideal* lattices over a widely used class of rings (in contrast to just slightly subexponential $2^{O(n \log \log n / \log n)}$ factors obtainable for general lattices [82, 119]). Note that these subexponential approximation factors are still much larger than the small *polynomial* factors that are typically used in cryptography (so the reductions have not been made vacuous). In addition, for dimensions $n \leq 2000$ used in NIST proposals, the *concrete* approximation factors obtained by these algorithms are actually *worse* than what can be obtained from lattice attacks corresponding roughly to Level 1 security [54]. Finally, the algorithms from [33, 44, 45] do not apply to Ring- or Module-LWE themselves, only Ideal-SVP.

1.2.2 Parameters from worst-case reductions and conservative cryptanalysis

Like all cryptographic problems, LWE is an *average-case* problem, i.e., input instances are chosen *at random* from a prescribed probability distribution. As already mentioned, some parameterizations of LWE admit (quantum or classical) reductions from *worst-case* lattice problems. That is, any algorithm that solves n-dimensional LWE (with some non-negligible advantage) can be converted with some polynomial overhead into a (quantum) algorithm that solves certain short-vector problems on *any* n-dimensional lattice (with high probability). Therefore, if the latter problems have *some* (quantumly) hard instances, then *random* instances of LWE are also hard.

Worst-case/average-case reductions help guide the search for cryptographically hard problems in a large design space, and offer (at minimum) evidence that the particular distribution of inputs does not introduce any fundamental structural weaknesses. This is in contrast to several lattice-based proposals that lacked such reductions, and turned out to be insecure because their distributions made "planted" secrets easy to recover, e.g., [120, 94, 33, 44]. Indeed, Micciancio and Regev [93] argue that a reduction from a hard worst-case problem

> "... assures us that attacks on the cryptographic construction are likely to be effective only for small choices of parameters and not asymptotically. In other words, it assures us that there are no fundamental flaws in the design of our cryptographic construction... In principle the worst-case security guarantee can help us in choosing concrete parameters for the cryptosystem, although in practice this leads to what seems like overly conservative estimates, and ... one often sets the parameters based on the best known attacks."

Not all LWE parameterizations admit reductions from worst-case lattice problems. For example, the iterative quantum reductions from [113, 103] require the use of Gaussian error having standard deviation at least $c\sqrt{n}$ for an arbitrary constant $c > 1/(2\pi)$, where n is the dimension of the LWE secret. In practice, a drawback of using such "wide" error distributions for cryptography is the relatively large modulus required

[3] Some unconventional parameterizations of Ring-LWE were specifically devised to be breakable by certain algebraic attacks [56, 39, 35, 40]. However, it was later shown that their error distributions are insufficiently "wide" relative to the ring, so they reveal errorless (or nearly so) linear equations and can therefore be broken even more efficiently using elementary, non-algebraic means [35, 102].

to avoid decryption error, which leads to larger dimensions n and sizes of keys and ciphertexts for a desired level of concrete security. Subsequent works like [52, 91] provided weaker reductions for "narrower" error distributions, such as uniform over a small set (even $\{0, 1\}$), but only by restricting the number of LWE samples available to the attacker—to fewer than the number exposed by LWE-based cryptosystems [113, 83], in the case of moderately small errors. We emphasize that *some* limitation on the number of samples is necessary, because LWE with errors bounded by (say) a constant is solvable in polynomial time, given a large enough polynomial number of samples [16, 6]. Currently, there is still a sizeable gap between small-error LWE parameters that are known to be vulnerable, and those conforming to a worst-case reduction. Most proposed implementations use parameters that lie within this gap.

Error width and an alternative worst-case reduction. In keeping with our philosophy of using conservative choices of hard problems that still admit practical implementations, our proposal uses "moderately wide" Gaussian error of standard deviation $\sigma \geq 2.3$ for security Levels 1 and 3 (and $\sigma = 1.4$ for Level 5), and automatically limits the number of LWE samples available to the adversary to *far less* than the number required by known attacks on LWE with these moderate error widths [16, 6]. Although such parameters do not conform to the full quantum reductions from [113, 103] for our choices of n, we show that they *do* conform to an alternative, classical worst-case reduction that can be extracted from those works. (We note that the original version of Frodo from [26] used a smaller σ that is not compatible with any of these reductions.)

In a little more detail, the alternative reduction is from a worst-case lattice problem we call "Bounded Distance Decoding with Discrete Gaussian Samples" (BDDwDGS), which has been closely investigated (though not under that name) in several works [2, 85, 113, 48]. Along with being classical (non-quantum), a main advantage of the alternative reduction is that it works for LWE with (1) Gaussian error whose width only needs to exceed the "smoothing parameter" [92] of the integer lattice $\mathbb{Z}$ for tiny enough $\varepsilon > 0$, and (2) a correspondingly bounded number of samples. We view this reduction as evidence that the smoothing parameter of $\mathbb{Z}$ is an important qualitative threshold for LWE error, which is why we use a standard deviation σ which is comfortably above it. We also view the reduction as narrowing the gap between the known weakness of small-error LWE with a large number of samples, and its apparent hardness with a small number of samples. See Section 5.1.5 for full details.

We stress that we use the worst-case reduction only for guidance in choosing a narrow enough error distribution for practice that still has some theoretical support, and not for any concrete security claim. As alluded to in the above quote from [93] (see also [38]), the known worst-case reduction does not yield any meaningful "end-to-end" security guarantee for our concrete parameters based on the conjectured hardness of a worst-case problem, because the reduction is *non-tight*: it has some significant polynomial overhead in running time and number of discrete Gaussian samples used, versus the number of LWE samples it produces. (Improving the tightness of worst-case reductions is an interesting problem.) Instead, we choose concrete parameters using a conservative analysis of the best known cryptanalytic attacks, as described next.

Concrete cryptanalysis using core-SVP hardness. Our concrete security estimates are based on a conservative methodology, as previously used for NewHope [13] and Frodo [26] and detailed in Section 5.2.1, that estimates the "core-SVP hardness" of solving the underlying LWE problem. This methodology builds on the extensive prior cryptanalysis of LWE and related lattice problems, and was further validated by recent work [11], which concluded that its experimental results "confirm that lattice reduction largely follows the behavior expected from the 2016 estimate [13]." The core-SVP methodology counts only the *first-order* exponential cost of just *one* (quantum) shortest-vector computation on a lattice of appropriate dimension to solve the relevant LWE problem. Because it ignores lower-order terms like the significant subexponential factors in the runtime, as well as the large exponential memory requirements, it significantly underestimates the actual cost of known attacks, and allows for significant future improvement in these attacks.

1.2.3 Simplicity of design and implementation

Using plain LWE allows us to construct encryption and key-encapsulation schemes that are *simple* and *easy to implement*, reducing the potential for errors. Wherever possible, design decisions were made in favor of simplicity over more sophisticated mechanisms.

Modular arithmetic. Our LWE parameters use an integer modulus $q \leq 2^{16}$ that is always a power of two. This ensures that only single-precision arithmetic is needed, and that reduction modulo q can be computed almost for free by bit-masking. (Reduction modulo 2^{16} is even entirely free when 16-bit data types are used.) Modular arithmetic is thus easy to implement correctly and in a way that is resistant to cache and timing side-channel attacks.

Error sampling. Although our "ideal" LWE error distribution is a Gaussian with an appropriate standard deviation, our implementation actually uses a distribution that is very close to it. Sampling from the distribution is quite simple via a small lookup table and a few random bits, and is resistant to cache and timing side-channels. (See Section 2.2.4 for details.) Using this alternative error distribution comes at very little expense in the concrete security of FrodoKEM, which we show by analyzing the Rényi divergence between the two distributions, following [17]. See Section 5.1.1 for full details.

Matrix-vector operations. Apart from error sampling and calls to symmetric primitives like AES or SHAKE, the main operations in our schemes are simple matrix-vector products. Compared to systems like NewHope [13] or Kyber [25] that use algebraically structured LWE variants, our system has moderately larger running times and bandwidth requirements, but is also significantly simpler, because there is no need to implement fast polynomial multiplication algorithms (like the number-theoretic transform for a prime modulus) to exploit the algebraic structure.

Encryption and key encapsulation without reconciliation. Our PKE and KEM follow the original method from Regev's encryption scheme [113] of transmitting secret bits by simply adding an encoding of them to pseudorandom values that the receiver can (approximately) subtract away. (Regev encoded single bits by multiplying by $\lfloor q/2 \rceil$; we encode B bits by multiplying by $\lfloor q/2^B \rceil$ as described by e.g. [72, 105, 104].) We do not need or use any of the more complicated reconciliation mechanisms that were developed in the context of key-exchange protocols (as mentioned above in Section 1.1).

In addition, unlike the Ring-LWE-based NewHope scheme [13], which transmits data using non-trivial lattice codes to make up for bandwidth losses arising from a sparse set of friendly ring dimensions, plain-LWE-based constructions do not have such bandwidth losses because the dimensions can be set freely. Therefore, we also have no need for complex bandwidth-saving optimizations.

Simple and compact code base. Our focus on simplicity is manifested in the FrodoKEM code base. For example, our x64 implementation of the full FrodoKEM scheme consists of only about 250 lines of plain C code (not including header files and code for symmetric primitives). Moreover, the exact same code can be used for other LWE parameters and security levels, solely by changing compile-time constants. Our Python 3 reference implementation containing all 6 FrodoKEM variants is about 465 lines of plain Python code (relying on other Python modules for symmetric primitives and bit manipulation).

1.3 Other features

Flexible, fine-grained choice of parameters. The plain LWE problem imposes very few requirements on its parameters, which makes it possible to rather tightly meet almost any desired security target in an automated way, using the methodology described in Section 5.2.1. Alternative parameters can be selected to reflect future advances in cryptanalysis, or to support other features beyond basic encryption and key encapsulation. For example, by using a larger LWE modulus (e.g., $q = 2^{32}$ or $q = 2^{64}$) and appropriate dimensions for a desired security level, FrodoPKE can easily support a large number of *homomorphic* additions, or multiplications by (small) public scalars, on ciphertexts. Using even larger moduli, it can even be made into a leveled or fully homomorphic encryption scheme, following [31].

Dynamically generated public matrices. To reduce the size of public keys and accelerate encryption, the public matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times n}$ could potentially be a fixed value that is chosen in a "nothing-up-my-sleeve" fashion [20] and used for all keys (see [27] for an example of this in a Ring-LWE-based system). However, to avoid the possibility of backdoors and all-for-the-price-of-one attacks [1], following prior work [13, 26] we dynamically and pseudorandomly generate a fresh matrix $\mathbf{A}$ for every generated key. The pseudorandom derivation is defined in a way that allows for fast generation of the entire matrix, or row-by-row generation on devices that cannot store the entire matrix in memory. See Section 2.2.5 for details.

2 Written specification

2.1 Background

This defines the cryptographic primitives and security notions that are relevant to FrodoPKE and FrodoKEM, as well as the mathematical background required to analyze their security.

2.1.1 Notation

We use the following notation throughout this document.

- Vectors are denoted with bold lower-case letters (e.g., $\mathbf{a}, \mathbf{b}, \mathbf{v}$), and matrices are denoted with bold upper-case letters (e.g., $\mathbf{A}, \mathbf{B}, \mathbf{S}$). For a set D, the set of m-dimensional vectors with entries in D is denoted by D^m, and the set of m-by-n matrices with entries in D is denoted by $D^{m\times n}$.
- For an n-dimensional vector $\mathbf{v}$, its ith entry for $0 \le i < n$ is denoted by $\mathbf{v}_i$.
- For an m-by-n matrix $\mathbf{A}$, its (i,j)th entry (i.e., the entry in the ith row and jth column) for $0 \le i < m$ and $0 \le j < n$ is denoted by $\mathbf{A}_{i,j}$, and its ith row is denoted by $\mathbf{A}_i = (\mathbf{A}_{i,0}, \mathbf{A}_{i,1}, \ldots, \mathbf{A}_{i,n-1})$.
- The transpose of a matrix $\mathbf{A}$ is denoted by $\mathbf{A}^{\mathrm{T}}$.
- An m-bit string $\mathbf{k} \in \{0,1\}^m$ is written as a vector over the set $\{0,1\}$ and its ith bit for $0 \le i < m$ is denoted by $\mathbf{k}_i$.
- The ring of integers is denoted by $\mathbb{Z}$, and, for a positive integer q, the quotient ring of integers modulo q is denoted by $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$.
- For a probability distribution χ, the notation $e \leftarrow\!\$\; \chi$ denotes drawing a value e according to χ. The n-fold product distribution of χ with itself is denoted by χ^n.
- For a finite set S, the uniform distribution on S is denoted by $U(S)$.
- The floor of a real number a, i.e., the largest integer less than or equal to a, is denoted by $\lfloor a \rfloor$.
- The closest integer to a real number a (with ties broken upward) is denoted by $\lfloor a \rceil = \lfloor a + 1/2 \rfloor$.
- For a real vector $\mathbf{v} \in \mathbb{R}^n$, its Euclidean (i.e., ℓ_2) norm is denoted by $\|\mathbf{v}\|$.
- For two n-dimensional vectors $\mathbf{a}, \mathbf{b}$ over a common ring R, their inner product is denoted by $\langle \mathbf{a}, \mathbf{b} \rangle = \sum_{i=0}^{n-1} \mathbf{a}_i \mathbf{b}_i \in R$.

2.1.2 Cryptographic definitions

This section states definitions of the cryptographic primitives that are specified in this document, along with their correctness and security notions. This document specifies a key encapsulation mechanism (KEM), formally defined by three algorithms as follows.

Definition 2.1 (Key encapsulation mechanism). A *key encapsulation mechanism* KEM is a tuple of algorithms (KeyGen, Encaps, Decaps) along with a finite keyspace $\mathcal{K}$:

- $\mathrm{KeyGen}() \;\$\!\!\rightarrow (pk, sk)$: A probabilistic *key generation algorithm* that outputs a public key pk and a secret key sk.
- $\mathrm{Encaps}(pk) \;\$\!\!\rightarrow (c, \mathbf{ss})$: A probabilistic *encapsulation algorithm* that takes as input a public key pk, and outputs an encapsulation c and a shared secret $\mathbf{ss} \in \mathcal{K}$. The encapsulation is sometimes called a ciphertext.
- $\mathrm{Decaps}(c, sk) \rightarrow \mathbf{ss}'$: A (usually deterministic) *decapsulation algorithm* that takes as input an encapsulation c and a secret key sk, and outputs a shared secret $\mathbf{ss}' \in \mathcal{K}$.

The notion of δ-correctness gives a bound on the probability of a legitimate protocol execution producing different keys in encapsulation and decapsulation.

Definition 2.2 (δ-correctness for KEMs). A key encapsulation mechanism KEM is *δ-correct* if

$$\Pr\left[\mathbf{ss}' \neq \mathbf{ss} : (pk, sk) \leftarrow\!\$\; \mathsf{KEM}.\mathrm{KeyGen}(); (c, \mathbf{ss}) \leftarrow\!\$\; \mathsf{KEM}.\mathrm{Encaps}(pk); \mathbf{ss}' \leftarrow \mathsf{KEM}.\mathrm{Decaps}(c, sk)\right] \le \delta \ .$$

The following defines IND-CCA security for a key encapsulation mechanism.

Definition 2.3 (IND-CCA **for KEMs).** Let KEM be a key encapsulation mechanism with keyspace $\mathcal{K}$, and let $\mathcal{A}$ be an algorithm. The security experiment for *indistinguishability under adaptive chosen ciphertext attack (*IND-CCA2*, or just* IND-CCA*)* of KEM is $\mathrm{Exp}^{\textsf{ind-cca}}_{\textsf{KEM}}(\mathcal{A})$ shown in Figure 1. The advantage of $\mathcal{A}$ in the experiment is

$$\mathrm{Adv}^{\textsf{ind-cca}}_{\textsf{KEM}}(\mathcal{A}) := \left|\Pr\left[\mathrm{Exp}^{\textsf{ind-cca}}_{\textsf{KEM}}(\mathcal{A}) \Rightarrow 1\right] - \frac{1}{2}\right| \ .$$

Note that $\mathcal{A}$ can be a classical or quantum algorithm. If $\mathcal{A}$ is a quantum algorithm, then we only consider the model in which the adversary makes classical queries to its $\mathcal{O}_{\mathrm{Decaps}}$ oracle.

Experiment $\mathrm{Exp}^{\textsf{ind-cca}}_{\textsf{KEM}}(\mathcal{A})$:

$(pk, sk) \leftarrow\!\!\$\ \textsf{KEM}.\mathrm{KeyGen}()$
$b \leftarrow\!\!\$\ \{0,1\}$
$(c^*, \mathbf{ss}_0) \leftarrow\!\!\$\ \textsf{KEM}.\mathrm{Encaps}(pk)$
$\mathbf{ss}_1 \leftarrow\!\!\$\ U(\mathcal{K})$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\mathcal{O}_{\mathrm{Decaps}}(\cdot)}(pk, \mathbf{ss}_b, c^*)$
if $b' = b$ **then**
  return 1
else
  return 0

Oracle $\mathcal{O}_{\mathrm{Decaps}}(c)$:

if $c = c^*$ **then**
  return $\perp$
else
  return $\textsf{KEM}.\mathrm{Decaps}(c, sk)$

Figure 1: Security experiment for indistinguishability under adaptive chosen ciphertext attack (IND-CCA2, or just IND-CCA) of a key encapsulation mechanism KEM for an adversary $\mathcal{A}$.

The key encapsulation mechanism specified in this document is obtained by a transformation from a public-key encryption (PKE) scheme; a PKE scheme is formally defined as follows.

Definition 2.4 (Public-key encryption scheme). A *public-key encryption scheme* PKE is a tuple of algorithms (KeyGen, Enc, Dec) along with a message space $\mathcal{M}$:

- $\mathrm{KeyGen}() \ \$\!\!\rightarrow (pk, sk)$: A probabilistic *key generation algorithm* that outputs a public key pk and a secret key sk.
- $\mathrm{Enc}(m, pk) \ \$\!\!\rightarrow c$: A probabilistic *encryption algorithm* that takes as input a message $m \in \mathcal{M}$ and public key pk, and outputs a ciphertext c. The deterministic form is denoted $\mathrm{Enc}(m, pk; r) \rightarrow c$, where the randomness $r \in \mathcal{R}$ is passed as an explicit input; $\mathcal{R}$ is called the *randomness space* of the encryption algorithm.
- $\mathrm{Dec}(c, sk) \rightarrow m'$ or $\perp$: A deterministic *decryption algorithm* that takes as input a ciphertext c and secret key sk, and outputs a message $m' \in \mathcal{M}$ or a special error symbol $\perp \notin \mathcal{M}$.

The notion of δ-correctness captures an upper bound on the probability of decryption failure in a legitimate execution of the scheme.

Definition 2.5 (δ-correctness for PKEs [68]). A public-key encryption scheme PKE with message space $\mathcal{M}$ is *δ-correct* if

$$\mathbb{E}\left[\max_{m \in \mathcal{M}} \Pr\left[\textsf{PKE}.\mathrm{Dec}(c, sk) \neq m : c \leftarrow\!\!\$\ \textsf{PKE}.\mathrm{Enc}(m, pk)\right]\right] \leq \delta \ , \tag{1}$$

where the expectation is taken over $(pk, sk) \leftarrow\!\!\$\ \textsf{PKE}.\mathrm{KeyGen}()$.

In our PKE, the probability expression in Equation (1) has no dependence on m, so the condition simplifies to

$$\Pr\left[\textsf{PKE}.\mathrm{Dec}(c, sk) \neq m : (pk, sk) \leftarrow \textsf{PKE}.\mathrm{KeyGen}(); c \leftarrow\!\!\$\ \textsf{PKE}.\mathrm{Enc}(m, pk)\right] \leq \delta \ , \tag{2}$$

which is what we analyze when calculating the probability of decryption failure (see Section 2.2.7).

The PKE scheme we use as the basis for the KEM transformation in Section 2.2.8 is required to satisfy the notion of IND-CPA security, which is defined as follows.

Definition 2.6 (IND-CPA **for PKE).** Let PKE be a public-key encryption scheme, and let $\mathcal{A}$ be an algorithm. The security experiment for *indistinguishability under chosen plaintext attack (*IND-CPA*)* of PKE is $\mathrm{Exp}^{\mathsf{ind\text{-}cpa}}_{\mathsf{PKE}}(\mathcal{A})$ shown in Figure 2. The advantage of $\mathcal{A}$ in the experiment is

$$\mathrm{Adv}^{\mathsf{ind\text{-}cpa}}_{\mathsf{PKE}}(\mathcal{A}) := \left| \Pr\left[\mathrm{Exp}^{\mathsf{ind\text{-}cpa}}_{\mathsf{PKE}}(\mathcal{A}) \Rightarrow 1\right] - \frac{1}{2} \right| .$$

Note that $\mathcal{A}$ can be a classical or quantum algorithm.

Experiment $\mathrm{Exp}^{\mathsf{ind\text{-}cpa}}_{\mathsf{PKE}}(\mathcal{A})$:

1: $(pk, sk) \leftarrow_\$ \mathsf{PKE}.\mathrm{KeyGen}()$
2: $(m_0, m_1, st) \leftarrow_\$ \mathcal{A}(pk)$
3: $b \leftarrow_\$ \{0,1\}$
4: $c^* \leftarrow_\$ \mathsf{PKE}.\mathrm{Enc}(m_b, pk)$
5: $b' \leftarrow_\$ \mathcal{A}(pk, c^*, st)$
6: **if** $b' = b$ **then**
7:     **return** 1
8: **else**
9:     **return** 0

Figure 2: Security experiment for indistinguishability under chosen plaintext attack (IND-CPA) of a public-key encryption scheme PKE against an adversary $\mathcal{A}$.

2.1.3 Learning With Errors

The security of our proposed PKE and KEM relies on the hardness of the *Learning With Errors* (LWE) problem, a generalization of the classic Learning Parities with Noise problem (see, e.g., [22]) first defined by Regev [113]. This section defines the LWE probability distributions and computational problems.

Definition 2.7 (LWE distribution). Let n, q be positive integers, and let χ be a distribution over $\mathbb{Z}$. For an $\mathbf{s} \in \mathbb{Z}_q^n$, the *LWE distribution* $A_{\mathbf{s},\chi}$ is the distribution over $\mathbb{Z}_q^n \times \mathbb{Z}_q$ obtained by choosing $\mathbf{a} \in \mathbb{Z}_q^n$ uniformly at random and an integer error $e \in \mathbb{Z}$ from χ, and outputting the pair $(\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + e \bmod q) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$.

There are two main kinds of computational LWE problem: *search*, which is to recover the secret $\mathbf{s} \in \mathbb{Z}_q^n$ given a certain number of samples drawn from the LWE distribution $A_{\mathbf{s},\chi}$; and *decision*, which is to distinguish a certain number of samples drawn from the LWE distribution from uniformly random samples. For both variants, one often considers two distributions of the secret $\mathbf{s} \in \mathbb{Z}_q^n$: the uniform distribution, and the distribution $\chi^n \bmod q$ where each coordinate is drawn from the error distribution χ and reduced modulo q. The latter is often called the "normal form" of LWE.

Definition 2.8 (LWE Search Problem). Let n, m, q be positive integers, and let χ be a distribution over $\mathbb{Z}$. The *uniform-secret* (respectively, *normal-form*) learning with errors *search* problem with parameters (n, m, q, χ), denoted by $\mathsf{SLWE}_{n,m,q,\chi}$ (respectively, $\mathsf{nf\text{-}SLWE}_{n,m,q,\chi}$), is as follows: given m samples from the LWE distribution $A_{\mathbf{s},\chi}$ for uniformly random $\mathbf{s}$ (resp, $\mathbf{s} \leftarrow_\$ \chi^n \bmod q$), find $\mathbf{s}$. More formally, for an adversary $\mathcal{A}$, define (for the uniform-secret case)

$$\mathrm{Adv}^{\mathsf{slwe}}_{n,m,q,\chi}(\mathcal{A}) = \Pr\left[\mathcal{A}(((\mathbf{a}_i, b_i))_{i=1,\ldots,m}) \Rightarrow \mathbf{s} : \mathbf{s} \leftarrow_\$ U(\mathbb{Z}_q^n), (\mathbf{a}_i, b_i) \leftarrow_\$ A_{\mathbf{s},\chi} \text{ for } i = 1, \ldots, m\right] .$$

Similarly, define (for the normal-form case) $\mathrm{Adv}^{\mathsf{nf\text{-}slwe}}_{n,m,q,\chi}(\mathcal{A})$, where $\mathbf{s} \leftarrow_\$ \chi^n \bmod q$ instead of $\mathbf{s} \leftarrow_\$ U(\mathbb{Z}_q^n)$.

Definition 2.9 (LWE Decision Problem). Let n, m, q be positive integers, and let χ be a distribution over $\mathbb{Z}$. The *uniform-secret* (respectively, *normal-form*) *learning with errors decision problem* with parameters (n, m, q, χ), denoted $\mathsf{DLWE}_{n,m,q,\chi}$ (respectively, $\mathsf{nf\text{-}DLWE}_{n,m,q,\chi}$), is as follows: distinguish m samples drawn from the LWE distribution $A_{\mathbf{s},\chi}$ from m samples drawn from the uniform distribution $U(\mathbb{Z}_q^n \times \mathbb{Z}_q)$. More formally, for an adversary $\mathcal{A}$, define (for the uniform-secret case)

$$\begin{aligned}\mathrm{Adv}^{\mathsf{dlwe}}_{n,m,q,\chi}(\mathcal{A}) = \Big| &\Pr\big[\mathcal{A}((\mathbf{a}_i, b_i)_{i=1,\dots,m}) \Rightarrow 1 : \mathbf{s} \leftarrow\!\!\$\; U(\mathbb{Z}_q^n), (\mathbf{a}_i, b_i) \leftarrow\!\!\$\; A_{\mathbf{s},\chi} \text{ for } i = 1,\dots,m\big] \\ &- \Pr\big[\mathcal{A}((\mathbf{a}_i, b_i)_{i=1,\dots,m}) \Rightarrow 1 : (\mathbf{a}_i, b_i) \leftarrow\!\!\$\; U(\mathbb{Z}_q^n \times \mathbb{Z}_q) \text{ for } i = 1,\dots,m\big]\Big| \ .\end{aligned}$$

Similarly, define (for the normal-form case) $\mathrm{Adv}^{\mathsf{nf\text{-}dlwe}}_{n,m,q,\chi}(\mathcal{A})$, where $\mathbf{s} \leftarrow\!\!\$\; \chi^n \bmod q$ instead of $\mathbf{s} \leftarrow\!\!\$\; U(\mathbb{Z}_q^n)$.

For all of the above problems, when $\chi = \Psi_{\alpha q}$ is the continuous Gaussian of parameter αq, rounded to the nearest integer (see Definition 2.11 below), we sometimes replace the subscript χ by α.

2.1.4 Gaussians

For any real $s > 0$, the (one-dimensional) *Gaussian function* with parameter (or width) s is the function $\rho_s \colon \mathbb{R} \to \mathbb{R}^+$, defined as

$$\rho_s(\mathbf{x}) := \exp(-\pi\|\mathbf{x}\|^2/s^2) \ .$$

Definition 2.10 (Gaussian distribution). For any real $s > 0$, the (one-dimensional) *Gaussian distribution* with parameter (or width) s, denoted D_s, is the distribution over $\mathbb{R}$ having probability density function $D_s(x) = \rho_s(x)/s$.

Note that D_s has standard deviation $\sigma = s/\sqrt{2\pi}$.

Definition 2.11 (Rounded Gaussian distribution). For any real $s > 0$, the *rounded Gaussian distribution* with parameter (or width) s, denoted Ψ_s, is the distribution over $\mathbb{Z}$ obtained by rounding a sample from D_s to the nearest integer:

$$\Psi_s(x) = \int_{\{z \colon \lfloor z \rceil = x\}} D_s(z)\, dz \ .$$

2.1.5 Lattices

Here we recall some background on lattices that will be used when relating LWE to lattice problems.

Definition 2.12 (Lattice). A (full-rank) *n-dimensional lattice* $\mathcal{L}$ is a discrete additive subset of $\mathbb{R}^n$ for which $\mathrm{span}_{\mathbb{R}}(\mathcal{L}) = \mathbb{R}^n$. Any such lattice can be generated by a (non-unique) *basis* $\mathbf{B} = \{\mathbf{b}_1, \dots, \mathbf{b}_n\} \subset \mathbb{R}^n$ of linearly independent vectors, as

$$\mathcal{L} = \mathcal{L}(\mathbf{B}) := \mathbf{B} \cdot \mathbb{Z}^n = \Big\{ \sum_{i=1}^{n} z_i \cdot \mathbf{b}_i \colon z_i \in \mathbb{Z} \Big\} \ .$$

The *volume*, or *determinant*, of $\mathcal{L}$ is defined as $\mathrm{vol}(\mathcal{L}) := |\det(\mathbf{B})|$. An *integer lattice* is a lattice that is a subset of $\mathbb{Z}^n$. For an integer q, a *q-ary lattice* is an integer lattice that contains $q\mathbb{Z}^n$.

Definition 2.13 (Minimum distance). For a lattice $\mathcal{L}$, its *minimum distance* is the length (in the Euclidean norm) of a shortest non-zero lattice vector:

$$\lambda_1(\mathcal{L}) = \min_{\mathbf{v} \in \mathcal{L} \setminus \{\mathbf{0}\}} \|\mathbf{v}\| \ .$$

More generally, its *ith successive minimum* $\lambda_i(\mathcal{L})$ is the smallest real $r > 0$ such that $\mathcal{L}$ has i linearly independent vectors of length at most r.

Definition 2.14 (Discrete Gaussian). For a lattice $\mathcal{L} \subset \mathbb{R}^n$, the *discrete Gaussian distribution* over $\mathcal{L}$ with parameter s, denoted $D_{\mathcal{L},s}$, is defined as $D_s(\mathbf{x}) = \rho_s(\mathbf{x})/\rho_s(\mathcal{L})$ for $\mathbf{x} \in \mathcal{L}$ (and $D_s(\mathbf{x}) = 0$ otherwise), where $\rho_s(\mathcal{L}) = \sum_{\mathbf{v} \in \mathcal{L}} \rho_s(\mathbf{v})$ is a normalization factor.

We now recall various computational problems on lattices. We stress that these are *worst-case* problems, i.e., to solve such a problem an algorithm must succeed on *every* input. The following two problems are parameterized by an *approximation factor* $\gamma = \gamma(n)$, which is a function of the lattice dimension n.

Definition 2.15 (Decisional approximate shortest vector problem (GapSVP_γ)). Given a basis $\mathbf{B}$ of an n-dimensional lattice $\mathcal{L} = \mathcal{L}(\mathbf{B})$, where $\lambda_1(\mathcal{L}) \leq 1$ or $\lambda_1(\mathcal{L}) > \gamma(n)$, determine which is the case.

Definition 2.16 (Approximate shortest independent vectors problem (SIVP_γ)). Given a basis $\mathbf{B}$ of an n-dimensional lattice $\mathcal{L} = \mathcal{L}(\mathbf{B})$, output a set $\{\mathbf{v}_1, \ldots, \mathbf{v}_n\} \subset \mathcal{L}$ of n linearly independent lattice vectors where $\|\mathbf{v}_i\| \leq \gamma(n) \cdot \lambda_n(\mathcal{L})$ for all i.

The following problem is parameterized by a function φ from lattices to positive real numbers.

Definition 2.17 (Discrete Gaussian Sampling (DGS_φ)). Given a basis $\mathbf{B}$ of an n-dimensional lattice $\mathcal{L} = \mathcal{L}(\mathbf{B})$ and a real number $s \geq \varphi(L)$, output a sample from the discrete Gaussian distribution $D_{L,s}$.

2.2 Algorithm description

This section specifies the algorithms comprising the FrodoKEM key encapsulation mechanism. FrodoKEM is built from a public-key encryption scheme, FrodoPKE, as well as several other components.

Notation. The algorithms in this document are described in terms of the following parameters:

- χ, a probability distribution on $\mathbb{Z}$;
- $q = 2^D$, a power-of-two integer modulus with exponent $D \leq 16$;
- $n, \overline{m}, \overline{n}$, integer matrix dimensions with $n \equiv 0 \pmod 8$;
- $B \leq D$, the number of bits encoded in each matrix entry;
- $\ell = B \cdot \overline{m} \cdot \overline{n}$, the length of bit strings that are encoded as $\overline{m}$-by-$\overline{n}$ matrices;
- $\mathsf{len}_{\mathsf{seed}_\mathbf{A}}$, the bit length of seeds used for pseudorandom matrix generation;
- $\mathsf{len}_{\mathsf{seed}_\mathbf{SE}}$, the bit length of seeds used for pseudorandom bit generation for error sampling.

Additional parameters for specific algorithms accompany the algorithm description.

2.2.1 Matrix encoding of bit strings

This subsection describes how bit strings are encoded as mod-q integer matrices. Recall that $2^B \leq q$. The encoding function $\mathrm{ec}(\cdot)$ encodes an integer $0 \leq k < 2^B$ as an element in $\mathbb{Z}_q$ by multiplying it by $q/2^B = 2^{D-B}$:

$$\mathrm{ec}(k) := k \cdot q/2^B \ .$$

This encoding function can be found in early works on LWE-based encryption, for example [72, 105, 104]. Using this function, the function Frodo.Encode encodes bit strings of length $\ell = B \cdot \overline{m} \cdot \overline{n}$ as $\overline{m}$-by-$\overline{n}$-matrices with entries in $\mathbb{Z}_q$ by applying $\mathrm{ec}(\cdot)$ to B-bit sub-strings sequentially and filling the matrix row by row entry-wise. The function Frodo.Encode is shown in Algorithm 1. Each B-bit sub-string is interpreted as an integer $0 \leq k < 2^B$ and then encoded by $\mathrm{ec}(k)$, which means that B-bit values are placed into the B most significant bits of the corresponding entry modulo q.

The corresponding decoding function Frodo.Decode is defined as shown in Algorithm 2. It decodes the $\overline{m}$-by-$\overline{n}$ matrix $\mathbf{K}$ into a bit string of length $\ell = B \cdot \overline{m} \cdot \overline{n}$. It extracts B bits from each entry by applying the function $\mathrm{dc}(\cdot)$:

$$\mathrm{dc}(c) = \lfloor c \cdot 2^B/q \rceil \bmod 2^B \ .$$

That is, the $\mathbb{Z}_q$-entry is interpreted as an integer, then divided by $q/2^B$ and rounded. This amounts to rounding to the B most significant bits of each entry. With these definitions, it is the case that $\mathrm{dc}(\mathrm{ec}(k)) = k$ for all $0 \leq k < 2^B$.

Algorithm 1 Frodo.Encode

Input: Bit string $\mathbf{k} \in \{0,1\}^{\ell}$, $\ell = B \cdot \overline{m} \cdot \overline{n}$.
Output: Matrix $\mathbf{K} \in \mathbb{Z}_q^{\overline{m} \times \overline{n}}$.

for $(i = 0;\ i < \overline{m};\ i \leftarrow i+1)$ **do**
  for $(j = 0;\ j < \overline{n};\ j \leftarrow j+1)$ **do**
    $k \leftarrow \sum_{l=0}^{B-1} \mathbf{k}_{(i \cdot \overline{n} + j)B + l} \cdot 2^l$
    $\mathbf{K}_{i,j} \leftarrow \mathrm{ec}(k) = k \cdot q / 2^B$
return $\mathbf{K} = (\mathbf{K}_{i,j})_{0 \le i < \overline{m}, 0 \le j < \overline{n}}$

Algorithm 2 Frodo.Decode

Input: Matrix $\mathbf{K} \in \mathbb{Z}_q^{\overline{m} \times \overline{n}}$.
Output: Bit string $\mathbf{k} \in \{0,1\}^{\ell}$, $\ell = B \cdot \overline{m} \cdot \overline{n}$.

for $(i = 0;\ i < \overline{m};\ i \leftarrow i+1)$ **do**
  for $(j = 0;\ j < \overline{n};\ j \leftarrow j+1)$ **do**
    $k \leftarrow \mathrm{dc}(\mathbf{K}_{i,j}) = \lfloor \mathbf{K}_{i,j} \cdot 2^B / q \rceil \bmod 2^B$
    $k = \sum_{l=0}^{B-1} k_l \cdot 2^l$ where $k_l \in \{0,1\}$
    for $(l = 0;\ l < B;\ l \leftarrow l+1)$ **do**
      $\mathbf{k}_{(i \cdot \overline{n} + j)B + l} \leftarrow k_l$
return $\mathbf{k}$

2.2.2 Packing matrices modulo q

This section specifies packing and unpacking algorithms to transform matrices with entries in $\mathbb{Z}_q$ to bit strings and vice versa. The algorithm Frodo.Pack packs a matrix into a bit string by simply concatenating the D-bit matrix coefficients, as shown in Algorithm 3. Note that in the software implementation, the resulting bit string is stored as a byte array, padding with zero bits to make the length a multiple of 8. The reverse operation Frodo.Unpack is shown in Algorithm 4.

Algorithm 3 Frodo.Pack

Input: Matrix $\mathbf{C} \in \mathbb{Z}_q^{n_1 \times n_2}$.
Output: Bit string $\mathbf{b} \in \{0,1\}^{D \cdot n_1 \cdot n_2}$.

for $(i = 0;\ i < n_1;\ i \leftarrow i+1)$ **do**
  for $(j = 0;\ j < n_2;\ j \leftarrow j+1)$ **do**
    $\mathbf{C}_{i,j} = \sum_{l=0}^{D-1} c_l \cdot 2^l$ where $c_l \in \{0,1\}$
    for $(l = 0;\ l < D;\ l \leftarrow l+1)$ **do**
      $\mathbf{b}_{(i \cdot n_2 + j)D + l} \leftarrow c_{D-1-l}$
return $\mathbf{b}$

Algorithm 4 Frodo.Unpack

Input: Bit string $\mathbf{b} \in \{0,1\}^{D \cdot n_1 \cdot n_2}$, n_1, n_2.
Output: Matrix $\mathbf{C} \in \mathbb{Z}_q^{n_1 \times n_2}$.

for $(i = 0;\ i < n_1;\ i \leftarrow i+1)$ **do**
  for $(j = 0;\ j < n_2;\ j \leftarrow j+1)$ **do**
    $\mathbf{C}_{i,j} \leftarrow \sum_{l=0}^{D-1} \mathbf{b}_{(i \cdot n_2 + j)D + l} \cdot 2^{D-1-l}$
return $\mathbf{C}$

2.2.3 Deterministic random bit generation

FrodoKEM requires the deterministic generation of random bit sequences from a random seed value. This is done using the SHA-3-derived extendable output function SHAKE [55]. The function SHAKE is taken as either SHAKE128 or SHAKE256 (indicated below for each parameter set of FrodoKEM), and takes as input a bit string X and a requested output bit length L.

2.2.4 Sampling from the error distribution

The error distribution χ used in FrodoKEM is a discrete, symmetric distribution on $\mathbb{Z}$, centered at zero and with small support, which approximates a rounded continuous Gaussian distribution.

The support of χ is $S_\chi = \{-s, -s+1, \ldots, -1, 0, 1, \ldots, s-1, s\}$ for a positive integer s. The probabilities $\chi(z) = \chi(-z)$ for $z \in S_\chi$ are given by a discrete probability density function, which is described by a table

$$T_\chi = (T_\chi(0), T_\chi(1), \ldots, T_\chi(s))$$

of $s+1$ positive integers related to the cumulative distribution function. For a certain positive integer len_χ, the table entries satisfy the following conditions:

$$T_\chi(0) = 2^{\mathsf{len}_\chi - 1} \cdot \chi(0) - 1 \qquad \text{and} \qquad T_\chi(z) = T_\chi(0) + 2^{\mathsf{len}_\chi} \sum_{i=1}^{z} \chi(i) \quad \text{for } 1 \le z \le s.$$

Since the distribution χ is symmetric and centered at zero, it is easy to verify that $T_\chi(s) = 2^{\mathsf{len}_\chi - 1} - 1$.

Sampling from χ via inversion sampling is done as shown in Algorithm 5. Given a string of len_χ uniformly random bits $\mathbf{r} \in \{0,1\}^{\mathsf{len}_\chi}$ and a distribution table T_χ, the algorithm Frodo.Sample returns a sample e from the distribution χ. (Note that $T_\chi(s)$ is never accessed.) We emphasize that it is important to perform this sampling in constant time to avoid exposing timing side-channels, which is why Step 3 of the algorithm does a complete loop through the entire table T_χ. The comparison in Step 4 needs to be implemented in a constant-time manner.

Algorithm 5 Frodo.Sample

Input: A (random) bit string $\mathbf{r} = (\mathbf{r}_0, \mathbf{r}_1, \ldots, \mathbf{r}_{\mathsf{len}_\chi - 1}) \in \{0,1\}^{\mathsf{len}_\chi}$, the table $T_\chi = (T_\chi(0), T_\chi(1), \ldots, T_\chi(s))$.
Output: A sample $e \in \mathbb{Z}$.

```
1: t ← Σ_{i=1}^{len_χ−1} r_i · 2^{i−1}
2: e ← 0
3: for (z = 0; z < s; z ← z + 1) do
4:    if t > T_χ(z) then
5:       e ← e + 1
6: e ← (−1)^{r_0} · e
7: return e
```

An n_1-by-n_2 matrix of $n_1 n_2$ samples from the error distribution is sampled on input of a $(n_1 n_2 \cdot \mathsf{len}_\chi)$-bit string, here written as a sequence $(\mathbf{r}^{(0)}, \mathbf{r}^{(1)}, \ldots, \mathbf{r}^{(n_1 n_2 - 1)})$ of $n_1 n_2$ bit vectors of length len_χ each, by sampling $n_1 n_2$ error terms through calls to Frodo.Sample on a corresponding len_χ-bit substring $\mathbf{r}^{(i \cdot n_2 + j)}$ and the distribution table T_χ to sample the matrix entry $\mathbf{E}_{i,j}$. The algorithm Frodo.SampleMatrix is shown in Algorithm 6.

Algorithm 6 Frodo.SampleMatrix

Input: A (random) bit string $(\mathbf{r}^{(0)}, \mathbf{r}^{(1)}, \ldots, \mathbf{r}^{(n_1 n_2 - 1)}) \in \{0,1\}^{n_1 n_2 \cdot \mathsf{len}_\chi}$ (here, each $\mathbf{r}^{(i)}$ is a vector of len_χ bits), the table T_χ.
Output: A sample $\mathbf{E} \in \mathbb{Z}^{n_1 \times n_2}$.

```
for (i = 0; i < n_1; i ← i + 1) do
   for (j = 0; j < n_2; j ← j + 1) do
      E_{i,j} ← Frodo.Sample(r^{(i·n_2+j)}, T_χ)
return E
```

2.2.5 Pseudorandom matrix generation

The algorithm Frodo.Gen takes as input a seed $\mathsf{seed}_\mathbf{A} \in \{0,1\}^{\mathsf{len}_{\mathsf{seed}_\mathbf{A}}}$ and a dimension $n \in \mathbb{Z}$, and outputs a pseudorandom matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times n}$. There are two options for instantiating Frodo.Gen. The first one uses AES128 and is shown in Algorithm 7; the second uses SHAKE128 and is shown in Algorithm 8.

Using AES128**.** Algorithm 7 generates a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times n}$ as follows. For each row index $i = 0, 1, \ldots, n-1$ and column index $j = 0, 8, \ldots, n-8$, the algorithm generates a 128-bit block, which it uses to set the matrix entries $\mathbf{A}_{i,j}, \mathbf{A}_{i,j+1}, \ldots, \mathbf{A}_{i,j+7}$ as follows. It applies AES128 with key $\mathsf{seed}_\mathbf{A}$ to the input block $\langle i \rangle \| \langle j \rangle \| 0 \cdots 0 \in \{0,1\}^{128}$, where i, j are encoded as 16-bit strings, represented in little-endian byte order. It then splits the 128-bit AES output block into eight 16-bit strings, which it interprets as nonnegative integers $c_{i,j+k}$ for $k = 0, 1, \ldots, 7$ in little-endian byte order. Finally, it sets $\mathbf{A}_{i,j+k} = c_{i,j+k} \bmod q$ for all k.

Algorithm 7 Frodo.Gen using AES128

Input: Seed $\mathsf{seed}_{\mathbf{A}} \in \{0,1\}^{\mathsf{len}_{\mathsf{seed}_{\mathbf{A}}}}$.
Output: Matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times n}$.

for $(i = 0;\ i < n;\ i \leftarrow i+1)$ **do**
  for $(j = 0;\ j < n;\ j \leftarrow j+8)$ **do**
    $\mathbf{b} \leftarrow \langle i \rangle \| \langle j \rangle \| 0 \cdots 0 \in \{0,1\}^{128}$ where $\langle i \rangle, \langle j \rangle \in \{0,1\}^{16}$
    $\langle c_{i,j} \rangle \| \langle c_{i,j+1} \rangle \| \cdots \| \langle c_{i,j+7} \rangle \leftarrow \mathrm{AES128}_{\mathsf{seed}_{\mathbf{A}}}(\mathbf{b})$ where each $\langle c_{i,k} \rangle \in \{0,1\}^{16}$
    for $(k = 0;\ k < 8;\ k \leftarrow k+1)$ **do**
      $\mathbf{A}_{i,j+k} \leftarrow c_{i,j+k} \bmod q$
return $\mathbf{A}$

Using SHAKE128**.** Algorithm 8 generates a matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times n}$ as follows. For each row index $i = 0, 1, \ldots, n-1$, it calls SHAKE128 with a main input of $\mathsf{seed}_{\mathbf{A}}$, prefixed with a counter (represented as a 16-bit integer in little-endian byte order), to produce a $16n$-bit output string. It splits this output into 16-bit integers (in little-endian byte order) $c_{i,j}$ for $j = 0, 1, \ldots, n-1$, and sets $\mathbf{A}_{i,j} = c_{i,j} \bmod q$ for all j.

Algorithm 8 Frodo.Gen using SHAKE128

Input: Seed $\mathsf{seed}_{\mathbf{A}} \in \{0,1\}^{\mathsf{len}_{\mathsf{seed}_{\mathbf{A}}}}$.
Output: Pseudorandom matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times n}$.

for $(i = 0;\ i < n;\ i \leftarrow i+1)$ **do**
  $\mathbf{b} \leftarrow \langle i \rangle \| \mathsf{seed}_{\mathbf{A}} \in \{0,1\}^{16+\mathsf{len}_{\mathsf{seed}_{\mathbf{A}}}}$ where $\langle i \rangle \in \{0,1\}^{16}$
  $\langle c_{i,0} \rangle \| \langle c_{i,1} \rangle \| \cdots \| \langle c_{i,n-1} \rangle \leftarrow \mathrm{SHAKE128}(\mathbf{b}, 16n)$ where each $\langle c_{i,j} \rangle \in \{0,1\}^{16}$
  for $(j = 0;\ j < n;\ j \leftarrow j+1)$ **do**
    $\mathbf{A}_{i,j} \leftarrow c_{i,j} \bmod q$
return $\mathbf{A}$

Using other functions. In principle, other functions could be used to pseudorandomly generate the matrix $\mathbf{A}$, such as a lightweight stream cipher for platforms without the hardware instructions that make fast AES and SHAKE implementations possible. As NIST currently does not standardize such a primitive, and the call for proposals indicated that submissions should use NIST primitives, we do not currently propose such an alternate instantiation.

2.2.6 FrodoPKE: IND-CPA-secure public-key encryption scheme

This section describes FrodoPKE, a public-key encryption scheme with fixed-length message space, targeting IND-CPA security, that will be used as a building block for FrodoKEM. FrodoPKE is based on the public-key encryption scheme presented by Lindner and Peikert in [83], with the following adaptations and specializations:

- The matrix $\mathbf{A}$ is generated from a seed using the function Frodo.Gen specified in Section 2.2.5.
- Several $(\overline{m})$ ciphertexts are generated at once.
- The same Gaussian-derived error distribution is used for both key generation and encryption.

The PKE scheme is given by three algorithms (FrodoPKE.KeyGen, FrodoPKE.Enc, FrodoPKE.Dec), defined respectively in Algorithm 9, Algorithm 10, and Algorithm 11. FrodoPKE is parameterized by the following parameters:

- $q = 2^D$, a power-of-two integer modulus with exponent $D \le 16$;
- $n, \overline{m}, \overline{n}$, integer matrix dimensions with $n \equiv 0 \pmod 8$;
- $B \le D$, the number of bits encoded in each matrix entry;
- $\ell = B \cdot \overline{m} \cdot \overline{n}$, the length of bit strings that are encoded as $\overline{m}$-by-$\overline{n}$ matrices;
- $\mathsf{len}_\mu = \ell$, the bit length of messages;
- $\mathcal{M} = \{0,1\}^{\mathsf{len}_\mu}$, the message space;
- $\mathsf{len}_{\mathsf{seed}_{\mathbf{A}}}$, the bit length of seeds used for pseudorandom matrix generation;
- $\mathsf{len}_{\mathsf{seed}_{\mathbf{SE}}}$, the bit length of seeds used for pseudorandom bit generation for error sampling;

- $\mathsf{Frodo.Gen}$, the matrix-generation algorithm, either Algorithm 7 or Algorithm 8;
- T_χ, the distribution table for sampling.

In the notation of [83], their n_1 and n_2 both equal n here, and their dimension ℓ is $\overline{n}$ here.

Algorithm 9 $\mathsf{FrodoPKE}$.KeyGen.

Input: None.
Output: Key pair $(pk, sk) \in (\{0,1\}^{\mathsf{len}_{\mathsf{seed}_\mathbf{A}}} \times \mathbb{Z}_q^{n\times\overline{n}}) \times \mathbb{Z}_q^{\overline{n}\times n}$.

Choose a uniformly random seed $\mathsf{seed}_\mathbf{A} \leftarrow\!\!\$\; U(\{0,1\}^{\mathsf{len}_{\mathsf{seed}_\mathbf{A}}})$
Generate the matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times n}$ via $\mathbf{A} \leftarrow \mathsf{Frodo.Gen}(\mathsf{seed}_\mathbf{A})$
Choose a uniformly random seed $\mathsf{seed}_\mathbf{SE} \leftarrow\!\!\$\; U(\{0,1\}^{\mathsf{len}_{\mathsf{seed}_\mathbf{SE}}})$
Generate pseudorandom bit string $(\mathbf{r}^{(0)}, \mathbf{r}^{(1)}, \ldots, \mathbf{r}^{(2n\overline{n}-1)}) \leftarrow \mathrm{SHAKE}(\mathtt{0x5F} \| \mathsf{seed}_\mathbf{SE}, 2n\overline{n} \cdot \mathsf{len}_\chi)$
Sample error matrix $\mathbf{S}^\mathrm{T} \leftarrow \mathsf{Frodo.SampleMatrix}((\mathbf{r}^{(0)}, \mathbf{r}^{(1)}, \ldots, \mathbf{r}^{(n\overline{n}-1)}), \overline{n}, n, T_\chi)$
Sample error matrix $\mathbf{E} \leftarrow \mathsf{Frodo.SampleMatrix}((\mathbf{r}^{(n\overline{n})}, \mathbf{r}^{(n\overline{n}+1)}, \ldots, \mathbf{r}^{(2n\overline{n}-1)}), n, \overline{n}, T_\chi)$
Compute $\mathbf{B} = \mathbf{AS} + \mathbf{E}$
return public key $pk \leftarrow (\mathsf{seed}_\mathbf{A}, \mathbf{B})$ and secret key $sk \leftarrow \mathbf{S}^\mathrm{T}$

Algorithm 10 $\mathsf{FrodoPKE}$.Enc.

Input: Message $\mu \in \mathcal{M}$ and public key $pk = (\mathsf{seed}_\mathbf{A}, \mathbf{B}) \in \{0,1\}^{\mathsf{len}_{\mathsf{seed}_\mathbf{A}}} \times \mathbb{Z}_q^{n\times\overline{n}}$.
Output: Ciphertext $c = (\mathbf{C}_1, \mathbf{C}_2) \in \mathbb{Z}_q^{\overline{m}\times n} \times \mathbb{Z}_q^{\overline{m}\times\overline{n}}$.

Generate $\mathbf{A} \leftarrow \mathsf{Frodo.Gen}(\mathsf{seed}_\mathbf{A})$
Choose a uniformly random seed $\mathsf{seed}_\mathbf{SE} \leftarrow\!\!\$\; U(\{0,1\}^{\mathsf{len}_{\mathsf{seed}_\mathbf{SE}}})$
Generate pseudorandom bit string $(\mathbf{r}^{(0)}, \mathbf{r}^{(1)}, \ldots, \mathbf{r}^{(2\overline{m}n+\overline{mn}-1)}) \leftarrow \mathrm{SHAKE}(\mathtt{0x96} \| \mathsf{seed}_\mathbf{SE}, (2\overline{m}n + \overline{mn}) \cdot \mathsf{len}_\chi)$
Sample error matrix $\mathbf{S}' \leftarrow \mathsf{Frodo.SampleMatrix}((\mathbf{r}^{(0)}, \mathbf{r}^{(1)}, \ldots, \mathbf{r}^{(\overline{m}n-1)}), \overline{m}, n, T_\chi)$
Sample error matrix $\mathbf{E}' \leftarrow \mathsf{Frodo.SampleMatrix}((\mathbf{r}^{(\overline{m}n)}, \mathbf{r}^{(\overline{m}n+1)}, \ldots, \mathbf{r}^{(2\overline{m}n-1)}), \overline{m}, n, T_\chi)$
Sample error matrix $\mathbf{E}'' \leftarrow \mathsf{Frodo.SampleMatrix}((\mathbf{r}^{(2\overline{m}n)}, \mathbf{r}^{(2\overline{m}n+1)}, \ldots, \mathbf{r}^{(2\overline{m}n+\overline{mn}-1)}), \overline{m}, \overline{n}, T_\chi)$
Compute $\mathbf{B}' = \mathbf{S}'\mathbf{A} + \mathbf{E}'$ and $\mathbf{V} = \mathbf{S}'\mathbf{B} + \mathbf{E}''$
return ciphertext $c \leftarrow (\mathbf{C}_1, \mathbf{C}_2) = (\mathbf{B}', \mathbf{V} + \mathsf{Frodo.Encode}(\mu))$

Algorithm 11 $\mathsf{FrodoPKE}$.Dec.

Input: Ciphertext $c = (\mathbf{C}_1, \mathbf{C}_2) \in \mathbb{Z}_q^{\overline{m}\times n} \times \mathbb{Z}_q^{\overline{m}\times\overline{n}}$ and secret key $sk = \mathbf{S}^\mathrm{T} \in \mathbb{Z}_q^{\overline{n}\times n}$.
Output: Decrypted message $\mu' \in \mathcal{M}$.

Compute $\mathbf{M} = \mathbf{C}_2 - \mathbf{C}_1\mathbf{S}$
return message $\mu' \leftarrow \mathsf{Frodo.Decode}(\mathbf{M})$

2.2.7 Correctness of IND-CPA PKE

The next lemma states bounds on the size of errors that can be handled by the decoding algorithm.

Lemma 2.18. *Let $q = 2^D$, $B \leq D$. Then $\mathrm{dc}(\mathrm{ec}(k) + e) = k$ for any $k, e \in \mathbb{Z}$ such that $0 \leq k < 2^B$ and $-q/2^{B+1} \leq e < q/2^{B+1}$.*

Proof. This follows directly from the fact that $\mathrm{dc}(\mathrm{ec}(k) + e) = \lfloor k + e2^B/q \rceil \bmod 2^B$. $\square$

Correctness of decryption: The decryption algorithm $\mathsf{FrodoPKE}$.Dec computes

$$\begin{aligned}\mathbf{M} &= \mathbf{C}_2 - \mathbf{C}_1\mathbf{S} \\ &= \mathbf{V} + \mathsf{Frodo}.\mathrm{Encode}(\mu) - (\mathbf{S}'\mathbf{A} + \mathbf{E}')\mathbf{S} \\ &= \mathsf{Frodo}.\mathrm{Encode}(\mu) + \mathbf{S}'\mathbf{B} + \mathbf{E}'' - \mathbf{S}'\mathbf{A}\mathbf{S} - \mathbf{E}'\mathbf{S} \\ &= \mathsf{Frodo}.\mathrm{Encode}(\mu) + \mathbf{S}'\mathbf{A}\mathbf{S} + \mathbf{S}'\mathbf{E} + \mathbf{E}'' - \mathbf{S}'\mathbf{A}\mathbf{S} - \mathbf{E}'\mathbf{S} \\ &= \mathsf{Frodo}.\mathrm{Encode}(\mu) + \mathbf{S}'\mathbf{E} + \mathbf{E}'' - \mathbf{E}'\mathbf{S} \\ &= \mathsf{Frodo}.\mathrm{Encode}(\mu) + \mathbf{E}'''\end{aligned}$$

for some error matrix $\mathbf{E}''' = \mathbf{S}'\mathbf{E} + \mathbf{E}'' - \mathbf{E}'\mathbf{S}$. Therefore, any B-bit substring of the message μ corresponding to an entry of $\mathbf{M}$ will be decrypted correctly if the condition in Lemma 2.18 is satisfied for the corresponding entry of $\mathbf{E}'''$.

Failure probability. Each entry in the matrix $\mathbf{E}'''$ is the sum of $2n$ products of two independent samples from χ, and one more independent sample from χ. Denote the distribution of this sum by χ'. In the case of a power-of-2 modulus q, the probability of decryption failure for any single symbol is therefore the sum

$$p = \sum_{e \notin [-q/2^{B+1}, q/2^{B+1})} \chi'(e) \ .$$

The probability of decryption failure for the entire message can then be obtained using the union bound.

For the distributions χ we use, which have rather small support S_χ, the distribution χ' can be efficiently computed exactly. The probability that a product of two independent samples from χ equals e (modulo q) is simply

$$\sum_{(a,b) \in S_\chi \times S_\chi \,:\, ab = e \bmod q} \chi(a) \cdot \chi(b) \ .$$

Similarly, the probability that the sum of two entries assumes a certain value is given by the standard convolution sum. Section 2.4.3 reports the failure probability for each of the selected parameter sets.

2.2.8 Transform from IND-CPA PKE to IND-CCA KEM

The Fujisaki–Okamoto transform [57] constructs an IND-CCA2-secure public-key encryption scheme, in the classical random oracle model, from a one-way-secure public-key encryption scheme (assuming the distribution of ciphertexts for each plaintext is sufficiently close to uniform). Targhi and Unruh [122] gave a variant of the Fujisaki–Okamoto transform and showed its IND-CCA2 security against a quantum adversary in the quantum random oracle model under similar assumptions. The results of both FO and TU proceed under the assumption that the public-key encryption scheme has perfect correctness, which is often not the case for lattice-based schemes (including ours). Hofheinz, Hövelmanns, and Kiltz (HHK) [68] gave a variety of constructions in a modular fashion. We apply their $\mathsf{FO}^{\not\perp}$ ("FO with implicit rejection") transform, which constructs an IND-CCA-secure key encapsulation mechanism from an IND-CPA public-key encryption scheme and three hash functions; following [25], we make the following modifications (see Figure 3 for notation), denoting the resulting transform $\mathsf{FO}^{\not\perp\prime}$:

- A single hash function (with longer output) is used to compute $\mathbf{r}$ and $\mathbf{k}$.
- The computation of $\mathbf{r}$ and $\mathbf{k}$ also takes the public key pk as input.
- The computation of the shared secret $\mathbf{ss}$ also takes the encapsulation c as input.

Definition 2.19 ($\mathsf{FO}^{\not\perp\prime}$ transform). Let $\mathsf{PKE} = (\mathrm{KeyGen}, \mathrm{Enc}, \mathrm{Dec})$ be a public-key encryption scheme with message space $\mathcal{M}$ and ciphertext space $\mathcal{C}$, where the randomness space of Enc is $\mathcal{R}$. Let $\mathsf{len}_{\mathbf{s}}, \mathsf{len}_{\mathbf{k}}, \mathsf{len}_{\mathbf{pkh}}, \mathsf{len}_{\mathbf{ss}}$ be parameters. Let $G_1 \colon \{0,1\}^* \to \{0,1\}^{\mathsf{len}_{\mathbf{pkh}}}$, $G_2 \colon \{0,1\}^* \to \mathcal{R} \times \{0,1\}^{\mathsf{len}_{\mathbf{k}}}$, and $F \colon \{0,1\}^* \to \{0,1\}^{\mathsf{len}_{\mathbf{ss}}}$ be hash functions. Define $\mathsf{KEM}^{\not\perp\prime} = \mathsf{FO}^{\not\perp\prime}[\mathsf{PKE}, G_1, G_2, F]$ to be the key encapsulation mechanism as shown in Figure 3.

As observed by Guo, Johansson, and Nilsson [66], a timing side-channel enables key recovery if step 5 of $\mathsf{KEM}^{\not\perp\prime}$.Decaps is not performed in constant time.

$\underline{\mathsf{KEM}^{\not\perp'}.\text{KeyGen}():}$

$(pk, sk) \leftarrow\!\!\$\ \mathsf{PKE}.\text{KeyGen}()$
$\mathbf{s} \leftarrow\!\!\$\ \{0,1\}^{\mathsf{len}_{\mathbf{s}}}$
$\mathbf{pkh} \leftarrow G_1(pk)$
$sk' \leftarrow (sk, \mathbf{s}, pk, \mathbf{pkh})$
return (pk, sk')

$\underline{\mathsf{KEM}^{\not\perp'}.\text{Encaps}(pk):}$

$\mu \leftarrow\!\!\$\ \mathcal{M}$
$(\mathbf{r}, \mathbf{k}) \leftarrow G_2(G_1(pk) \| \mu)$
$c \leftarrow \mathsf{PKE}.\text{Enc}(\mu, pk; \mathbf{r})$
$\mathbf{ss} \leftarrow F(c \| \mathbf{k})$
return $(c, \mathbf{ss})$

$\underline{\mathsf{KEM}^{\not\perp'}.\text{Decaps}(c, (sk, \mathbf{s}, pk, \mathbf{pkh})):}$

$\mu' \leftarrow \mathsf{PKE}.\text{Dec}(c, sk)$
$(\mathbf{r}', \mathbf{k}') \leftarrow G_2(\mathbf{pkh} \| \mu')$
$\mathbf{ss}'_0 \leftarrow F(c \| \mathbf{k}')$
$\mathbf{ss}'_1 \leftarrow F(c \| \mathbf{s})$
(in constant time) $\mathbf{ss}' \leftarrow \mathbf{ss}'_0$ if $c = \mathsf{PKE}.\text{Enc}(\mu', pk; \mathbf{r}')$ else $\mathbf{ss}' \leftarrow \mathbf{ss}'_1$
return $\mathbf{ss}'$

Figure 3: Construction of an $\mathsf{IND\text{-}CCA}$-secure key encapsulation mechanism $\mathsf{KEM}^{\not\perp'} = \mathsf{FO}^{\not\perp'}[\mathsf{PKE}, G_1, G_2, F]$ from a public-key encryption scheme PKE and hash functions G_1, G_2, and F.

2.2.9 FrodoKEM: IND-CCA-secure key encapsulation mechanism

This section describes FrodoKEM, a key encapsulation mechanism that is derived from FrodoPKE by applying the $\mathsf{FO}^{\not\perp'}$ transform. FrodoKEM is parameterized by the following parameters:

- $q = 2^D$, a power-of-two integer modulus with exponent $D \le 16$;
- $n, \overline{m}, \overline{n}$, integer matrix dimensions with $n \equiv 0 \pmod 8$;
- $B \le D$, the number of bits encoded in each matrix entry;
- $\ell = B \cdot \overline{m} \cdot \overline{n}$, the length of bit strings to be encoded in an $\overline{m}$-by-$\overline{n}$ matrix;
- $\mathsf{len}_\mu = \ell$, the bit length of messages;
- $\mathcal{M} = \{0,1\}^{\mathsf{len}_\mu}$, the message space;
- $\mathsf{len}_{\mathsf{seed}_\mathbf{A}}$, the bit length of seeds used for pseudorandom matrix generation;
- $\mathsf{len}_{\mathsf{seed}_\mathbf{SE}}$, the bit length of seeds used for pseudorandom bit generation for error sampling;
- Frodo.Gen, pseudorandom matrix generation algorithm, either Algorithm 7 or Algorithm 8;
- T_χ, distribution table for sampling;
- $\mathsf{len}_\mathbf{s}$, the length of the bit vector $\mathbf{s}$ used for pseudorandom shared secret generation in the event of decapsulation failure in the $\mathsf{FO}^{\not\perp'}$ transform;
- $\mathsf{len}_\mathbf{z}$, the bit length of seeds used for pseudorandom generation of $\mathsf{seed}_\mathbf{A}$;
- $\mathsf{len}_\mathbf{k}$, the bit length of intermediate shared secret $\mathbf{k}$ in the $\mathsf{FO}^{\not\perp'}$ transform;
- $\mathsf{len}_\mathbf{pkh}$, the bit length of the hash $G_1(pk)$ of the public key in the $\mathsf{FO}^{\not\perp'}$ transform;
- $\mathsf{len}_\mathbf{ss}$, the bit length of shared secret $\mathbf{ss}$ in the $\mathsf{FO}^{\not\perp'}$ transform;

Algorithm 12 FrodoKEM.KeyGen.

Input: None.

Output: Key pair (pk, sk') with $pk \in \{0,1\}^{\mathsf{len}_{\mathsf{seed}_\mathbf{A}} + D\cdot n\cdot \overline{n}}$, $sk' \in \{0,1\}^{\mathsf{len}_\mathbf{s} + \mathsf{len}_{\mathsf{seed}_\mathbf{A}} + D\cdot n\cdot \overline{n}} \times \mathbb{Z}_q^{\overline{n}\times n} \times \{0,1\}^{\mathsf{len}_\mathbf{pkh}}$.

Choose uniformly random seeds $\mathbf{s}\|\mathsf{seed}_\mathbf{SE}\|\mathbf{z} \leftarrow_{\$} U(\{0,1\}^{\mathsf{len}_\mathbf{s}+\mathsf{len}_{\mathsf{seed}_\mathbf{SE}}+\mathsf{len}_\mathbf{z}})$
Generate pseudorandom seed $\mathsf{seed}_\mathbf{A} \leftarrow \mathrm{SHAKE}(\mathbf{z}, \mathsf{len}_{\mathsf{seed}_\mathbf{A}})$
Generate the matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times n}$ via $\mathbf{A} \leftarrow \mathsf{Frodo}.\mathrm{Gen}(\mathsf{seed}_\mathbf{A})$
Generate pseudorandom bit string $(\mathbf{r}^{(0)}, \mathbf{r}^{(1)}, \ldots, \mathbf{r}^{(2n\overline{n}-1)}) \leftarrow \mathrm{SHAKE}(\mathtt{0x5F}\|\mathsf{seed}_\mathbf{SE}, 2n\overline{n}\cdot \mathsf{len}_\chi)$
Sample error matrix $\mathbf{S}^\mathrm{T} \leftarrow \mathsf{Frodo}.\mathrm{SampleMatrix}((\mathbf{r}^{(0)}, \mathbf{r}^{(1)}, \ldots, \mathbf{r}^{(n\overline{n}-1)}), \overline{n}, n, T_\chi)$
Sample error matrix $\mathbf{E} \leftarrow \mathsf{Frodo}.\mathrm{SampleMatrix}((\mathbf{r}^{(n\overline{n})}, \mathbf{r}^{(n\overline{n}+1)}, \ldots, \mathbf{r}^{(2n\overline{n}-1)}), n, \overline{n}, T_\chi)$
Compute $\mathbf{B} \leftarrow \mathbf{AS} + \mathbf{E}$
Compute $\mathbf{b} \leftarrow \mathsf{Frodo}.\mathrm{Pack}(\mathbf{B})$
Compute $\mathbf{pkh} \leftarrow \mathrm{SHAKE}(\mathsf{seed}_\mathbf{A}\|\mathbf{b}, \mathsf{len}_\mathbf{pkh})$
return public key $pk \leftarrow \mathsf{seed}_\mathbf{A}\|\mathbf{b}$ and secret key $sk' \leftarrow (\mathbf{s}\|\mathsf{seed}_\mathbf{A}\|\mathbf{b}, \mathbf{S}^\mathrm{T}, \mathbf{pkh})$

Algorithm 13 FrodoKEM.Encaps.

Input: Public key $pk = \mathsf{seed}_\mathbf{A}\|\mathbf{b} \in \{0,1\}^{\mathsf{len}_{\mathsf{seed}_\mathbf{A}} + D\cdot n\cdot \overline{n}}$.

Output: Ciphertext $\mathbf{c}_1\|\mathbf{c}_2 \in \{0,1\}^{(\overline{m}\cdot n + \overline{m}\cdot\overline{n})D}$ and shared secret $\mathbf{ss} \in \{0,1\}^{\mathsf{len}_\mathbf{ss}}$.

Choose a uniformly random key $\mu \leftarrow_{\$} U(\{0,1\}^{\mathsf{len}_\mu})$
Compute $\mathbf{pkh} \leftarrow \mathrm{SHAKE}(pk, \mathsf{len}_\mathbf{pkh})$
Generate pseudorandom values $\mathsf{seed}_\mathbf{SE}\|\mathbf{k} \leftarrow \mathrm{SHAKE}(\mathbf{pkh}\|\mu, \mathsf{len}_{\mathsf{seed}_\mathbf{SE}} + \mathsf{len}_\mathbf{k})$
Generate pseudorandom bit string $(\mathbf{r}^{(0)}, \mathbf{r}^{(1)}, \ldots, \mathbf{r}^{(2\overline{m}n+\overline{m}\overline{n}-1)}) \leftarrow \mathrm{SHAKE}(\mathtt{0x96}\|\mathsf{seed}_\mathbf{SE}, (2\overline{m}n + \overline{m}\overline{n}) \cdot \mathsf{len}_\chi)$
Sample error matrix $\mathbf{S}' \leftarrow \mathsf{Frodo}.\mathrm{SampleMatrix}((\mathbf{r}^{(0)}, \mathbf{r}^{(1)}, \ldots, \mathbf{r}^{(\overline{m}n-1)}), \overline{m}, n, T_\chi)$
Sample error matrix $\mathbf{E}' \leftarrow \mathsf{Frodo}.\mathrm{SampleMatrix}((\mathbf{r}^{(\overline{m}n)}, \mathbf{r}^{(\overline{m}n+1)}, \ldots, \mathbf{r}^{(2\overline{m}n-1)}), \overline{m}, n, T_\chi)$
Generate $\mathbf{A} \leftarrow \mathsf{Frodo}.\mathrm{Gen}(\mathsf{seed}_\mathbf{A})$
Compute $\mathbf{B}' \leftarrow \mathbf{S}'\mathbf{A} + \mathbf{E}'$
Compute $\mathbf{c}_1 \leftarrow \mathsf{Frodo}.\mathrm{Pack}(\mathbf{B}')$
Sample error matrix $\mathbf{E}'' \leftarrow \mathsf{Frodo}.\mathrm{SampleMatrix}((\mathbf{r}^{(2\overline{m}n)}, \mathbf{r}^{(2\overline{m}n+1)}, \ldots, \mathbf{r}^{(2\overline{m}n+\overline{m}\overline{n}-1)}), \overline{m}, \overline{n}, T_\chi)$
Compute $\mathbf{B} \leftarrow \mathsf{Frodo}.\mathrm{Unpack}(\mathbf{b}, n, \overline{n})$
Compute $\mathbf{V} \leftarrow \mathbf{S}'\mathbf{B} + \mathbf{E}''$
Compute $\mathbf{C} \leftarrow \mathbf{V} + \mathsf{Frodo}.\mathrm{Encode}(\mu)$
Compute $\mathbf{c}_2 \leftarrow \mathsf{Frodo}.\mathrm{Pack}(\mathbf{C})$
Compute $\mathbf{ss} \leftarrow \mathrm{SHAKE}(\mathbf{c}_1\|\mathbf{c}_2\|\mathbf{k}, \mathsf{len}_\mathbf{ss})$
return ciphertext $\mathbf{c}_1\|\mathbf{c}_2$ and shared secret $\mathbf{ss}$

Algorithm 14 $\mathsf{FrodoKEM}$.Decaps.

Input: Ciphertext $\mathbf{c}_1\|\mathbf{c}_2 \in \{0,1\}^{(\overline{m}\cdot n+\overline{m}\cdot\overline{n})D}$, secret key $sk' = (\mathbf{s}\|\mathsf{seed}_\mathbf{A}\|\mathbf{b}, \mathbf{S}^\mathrm{T}, \mathbf{pkh}) \in \{0,1\}^{\mathsf{len}_\mathbf{s}+\mathsf{len}_{\mathsf{seed}_\mathbf{A}}+D\cdot n\cdot\overline{n}} \times \mathbb{Z}_q^{\overline{n}\times n} \times \{0,1\}^{\mathsf{len}_{\mathbf{pkh}}}$.

Output: Shared secret $\mathbf{ss} \in \{0,1\}^{\mathsf{len}_\mathbf{ss}}$.

1: $\mathbf{B}' \leftarrow \mathsf{Frodo}.\mathrm{Unpack}(\mathbf{c}_1, \overline{m}, n)$
2: $\mathbf{C} \leftarrow \mathsf{Frodo}.\mathrm{Unpack}(\mathbf{c}_2, \overline{m}, \overline{n})$
3: Compute $\mathbf{M} \leftarrow \mathbf{C} - \mathbf{B}'\mathbf{S}$
4: Compute $\mu' \leftarrow \mathsf{Frodo}.\mathrm{Decode}(\mathbf{M})$
5: Parse $pk \leftarrow \mathsf{seed}_\mathbf{A}\|\mathbf{b}$
6: Generate pseudorandom values $\mathsf{seed}_\mathbf{SE}'\|\mathbf{k}' \leftarrow \mathrm{SHAKE}(\mathbf{pkh}\|\mu', \mathsf{len}_{\mathsf{seed}_\mathbf{SE}} + \mathsf{len}_\mathbf{k})$
7: Generate pseudorandom bit string $(\mathbf{r}^{(0)}, \mathbf{r}^{(1)}, \ldots, \mathbf{r}^{(2\overline{m}n+\overline{mn}-1)}) \leftarrow \mathrm{SHAKE}(\mathtt{0x96}\|\mathsf{seed}_\mathbf{SE}', (2\overline{m}n + \overline{mn})\cdot \mathsf{len}_\chi)$
8: Sample error matrix $\mathbf{S}' \leftarrow \mathsf{Frodo}.\mathrm{SampleMatrix}((\mathbf{r}^{(0)}, \mathbf{r}^{(1)}, \ldots, \mathbf{r}^{(\overline{m}n-1)}), \overline{m}, n, T_\chi)$
9: Sample error matrix $\mathbf{E}' \leftarrow \mathsf{Frodo}.\mathrm{SampleMatrix}((\mathbf{r}^{(\overline{m}n)}, \mathbf{r}^{(\overline{m}n+1)}, \ldots, \mathbf{r}^{(2\overline{m}n-1)}), \overline{m}, n, T_\chi)$
10: Generate $\mathbf{A} \leftarrow \mathsf{Frodo}.\mathrm{Gen}(\mathsf{seed}_\mathbf{A})$
11: Compute $\mathbf{B}'' \leftarrow \mathbf{S}'\mathbf{A} + \mathbf{E}'$
12: Sample error matrix $\mathbf{E}'' \leftarrow \mathsf{Frodo}.\mathrm{SampleMatrix}((\mathbf{r}^{(2\overline{m}n)}, \mathbf{r}^{(2\overline{m}n+1)}, \ldots, \mathbf{r}^{(2\overline{m}n+\overline{mn}-1)}), \overline{m}, \overline{n}, T_\chi)$
13: Compute $\mathbf{B} \leftarrow \mathsf{Frodo}.\mathrm{Unpack}(\mathbf{b}, n, \overline{n})$
14: Compute $\mathbf{V} \leftarrow \mathbf{S}'\mathbf{B} + \mathbf{E}''$
15: Compute $\mathbf{C}' \leftarrow \mathbf{V} + \mathsf{Frodo}.\mathrm{Encode}(\mu')$
16: (in constant time) $\overline{\mathbf{k}} \leftarrow \mathbf{k}'$ if $(\mathbf{B}'\|\mathbf{C} = \mathbf{B}''\|\mathbf{C}')$ else $\overline{\mathbf{k}} \leftarrow \mathbf{s}$
17: Compute $\mathbf{ss} \leftarrow \mathrm{SHAKE}(\mathbf{c}_1\|\mathbf{c}_2\|\overline{\mathbf{k}}, \mathsf{len}_\mathbf{ss})$
18: **return** shared secret $\mathbf{ss}$

See Section 6.1 for a discussion on implementing this step 16 in constant time.

2.2.10 Correctness of IND-CCA KEM

The failure probability δ of FrodoKEM is the same as the failure probability of the underlying FrodoPKE as computed in Section 2.2.7.

2.2.11 Interconversion to IND-CCA PKE

FrodoKEM can be converted to an IND-CCA-secure public-key encryption scheme using standard conversion techniques as specified by NIST. In particular, shared secret **ss** can be used as the encryption key in an appropriate data encapsulation mechanism in the KEM/DEM (key encapsulation mechanism / data encapsulation mechanism) framework [46].

2.3 Cryptographic primitives

In FrodoKEM we use the following generic cryptographic primitives. We describe their security requirements and instantiations with NIST-approved cryptographic primitives. In what follows, we use SHAKE128/256 to denote the use of either SHAKE128 or SHAKE256; which one is used with which parameter set for FrodoKEM is indicated in Table 4.

- Frodo.Gen in FrodoKEM.KeyGen: The security requirement on Frodo.Gen is that it is a public function that generates pseudorandom matrices $\mathbf{A}$. Frodo.Gen is instantiated using either AES128 (as in Algorithm 7) or SHAKE128 (as in Algorithm 8).
- G_1, G_2, and F in transform $\mathsf{FO}^{\not\perp'}$: these are modeled as independent random oracles. We instantiate these using SHAKE128/256; see below for an explanation of domain separation to achieve independence.

Overall, FrodoKEM has the following uses of SHAKE:

1. Frodo.Gen using SHAKE128, line 3: SHAKE128($\mathbf{b}, \ldots$), input $16 + \mathsf{len}_{\mathsf{seed}_\mathbf{A}}$ bits
2. FrodoKEM.KeyGen, line 2: SHAKE($\mathbf{z}, \ldots$), input $\mathsf{len}_\mathbf{z}$ bits
3. FrodoKEM.KeyGen, line 4: SHAKE($\mathtt{0x5F}\|\mathsf{seed}_\mathbf{SE}, \ldots$), input $8 + \mathsf{len}_{\mathsf{seed}_\mathbf{SE}}$ bits

4. $\mathsf{FrodoKEM}$.KeyGen, line 9: SHAKE($\mathsf{seed}_{\mathbf{A}}\|\mathbf{b},\ldots$), input $\mathsf{len}_{\mathsf{seed}_{\mathbf{A}}} + D \cdot n \cdot \overline{n}$ bits
5. $\mathsf{FrodoKEM}$.Encaps, line 2: same as $\mathsf{FrodoKEM}$.KeyGen, line 9
6. $\mathsf{FrodoKEM}$.Encaps, line 3: SHAKE($\mathbf{pkh}\|\mu,\ldots$), input length $\mathsf{len}_{\mathbf{pkh}} + \mathsf{len}_{\mathsf{seed}_{\mathbf{SE}}}$ bits
7. $\mathsf{FrodoKEM}$.Encaps, line 4: SHAKE($\mathtt{0x96}\|\mathsf{seed}_{\mathbf{SE}},\ldots$), input length $8 + \mathsf{len}_{\mathsf{seed}_{\mathbf{SE}}}$ bits
8. $\mathsf{FrodoKEM}$.Encaps, line 15: SHAKE($\mathbf{c}_1\|\mathbf{c}_2\|\mathbf{k},\ldots$), input length $(\overline{m} \cdot n + \overline{m} \cdot \overline{n})D + \mathsf{len}_{\mathbf{k}}$ bits
9. $\mathsf{FrodoKEM}$.Decaps, line 6: same as $\mathsf{FrodoKEM}$.Encaps, line 3
10. $\mathsf{FrodoKEM}$.Decaps, line 7: same as $\mathsf{FrodoKEM}$.Encaps, line 4
11. $\mathsf{FrodoKEM}$.Decaps, line 17 and 19: same as $\mathsf{FrodoKEM}$.Encaps, line 15

Domain separation. Each distinct use of SHAKE in the list above should be cryptographically independent, which is achieved via one of two forms of domain separation.

SHAKE, and the underlying Keccak operation, employ padding to pad input strings to a multiple of the rate. The specific padding used is appending the string `10*1`. Thus, inputs of different lengths yield different padded strings.

For uses of SHAKE where the inputs are of different lengths (entries 1, 2, 4, 6, and 8 in the list above), we rely on Keccak's padding for domain separation.

For uses of SHAKE where the inputs are of the same length (entries 3 and 7 in the list above), we prepend distinct bytes as domain separators. These domain separators have bit patterns ($\mathtt{0x5F} = \mathtt{01011111}, \mathtt{0x96} = \mathtt{10010110}$) that were chosen to make it hard to use individual or consecutive bit flipping attacks to turn one into the other.

2.4 Parameters

This section outlines our methodology for choosing tunable parameters of the proposed algorithms.

2.4.1 High-level overview

Recall the main FrodoPKE parameters defined in Section 2.2:

- χ, a probability distribution on $\mathbb{Z}$;
- $q = 2^D$, a power-of-two integer modulus with exponent $D \leq 16$;
- $n, \overline{m}, \overline{n}$, integer matrix dimensions with $n \equiv 0 \pmod 8$;
- $B \leq D$, the number of bits encoded in each matrix entry;
- $\ell = B \cdot \overline{m} \cdot \overline{n}$ the length of bit strings to be encoded in an $\overline{m}$-by-$\overline{n}$ matrix.

The task of parameter selection is framed as a combinatorial optimization problem, where the objective function is the ciphertext's size, and the constraints are dictated by the target security level, probability of decryption failure, and computational efficiency. The optimization problem is solved by sweeping the parameter space, subject to simple pruning techniques. We perform this sweep of the parameter space using the Python scripts that accompany the submission, in the folder `Additional_Implementations/Parameter_Search_Scripts`.

2.4.2 Parameter constraints

Implementation considerations limit q to be at most 2^{16} and n to be a multiple of 16. Our cost function is the sum of the bit lengths of FrodoPKE's ciphertext and its public key, which is $D \cdot (n \cdot (\overline{m} + \overline{n}) + \overline{m}\,\overline{n}) + \mathsf{len}_{\mathsf{seed}_{\mathbf{A}}}$.

The standard deviation σ of the Gaussian error distribution is taken to exceed the "smoothing parameter" of the integers $\mathbb{Z}$, for a very small error parameter $\varepsilon > 0$. The specific values of σ are chosen following the methodology in Section 5.1.5, which demonstrates that our choices conform to a nontrivial reduction from the worst-case BDDwDGS problem to the corresponding average-case LWE decision problem.

The complexity of the error-sampling algorithm (Section 2.2.4) depends on the support of the distribution and the number of uniformly random bits per sample. We bound the number of bits per sample by 16. Since the distribution is symmetric, the sample's sign ($\mathbf{r}_0$ in Algorithm 5) can be chosen independently from its magnitude e, which leaves 15 bits for sampling from the non-negative part of the support. For each setting of the variance σ^2 we find a discrete distribution subject to the above constraints that minimizes its Rényi divergence (for several integral orders) from the target "ideal" distribution, which is the rounded Gaussian $\Psi_{\sigma\sqrt{2\pi}}$.

We estimate the concrete security of parameters for our scheme based on cryptanalytic attacks (Section 5.2), accounting for the loss due to substitution of a rounded Gaussian with its discrete approximation (Section 5.1.1). The probability of decryption failure is computed according to the procedure outlined in Section 2.2.6.

In case of ties, i.e., when different parameter sets result in identical ciphertext sizes (i.e., the same q and n), we chose the smaller σ for FrodoKEM-640 and FrodoKEM-1344 (minimizing the probability of decryption failure), and the larger σ for FrodoKEM-976 (prioritizing security).

2.4.3 Selected parameter sets

We present three parameter sets for FrodoKEM:

- Frodo-640 targets Level 1 in the NIST call for proposals (matching or exceeding the brute-force security of AES-128).
- Frodo-976, targets Level 3 (matching or exceeding the brute-force security of AES-192).
- Frodo-1344, targets Level 5 (matching or exceeding the brute-force security of AES-256).

The procedures outlined in this section can be adapted to support alternative cost functions and constraints. For instance, an objective function that takes into account computational costs or penalizes the public key size would lead to a different set of outcomes. For example, constraints can be also chosen to guarantee error-free decryption, or to select parameters that allow for a bounded number of homomorphic operations.

The three parameter sets are given in Table 1. The corresponding error distributions appear in Table 3. Table 2 summarizes security claims we can make about FrodoKEM and its components. The columns LWE security C, Q and P respectively denote security, in bits, for classical, quantum, and plausible attacks on $\overline{m}+\overline{n}$ instances of the normal-form (decisional) LWE problem with Gaussian error distribution (Section 2.1.3) as estimated by the methodology of Section 5.2. The column IND-CCA security C denotes IND-CCA security, in bits, for classical random oracle model attacks according to Theorem 5.1.

Table 1: **Parameters at a glance**

	n	q	σ	**support of χ**	B	$\bar{m}\times\bar{n}$	**c size (bytes)**
Frodo-640	640	2^{15}	2.8	$[-12\dots12]$	2	8×8	9,720
Frodo-976	976	2^{16}	2.3	$[-10\dots10]$	3	8×8	15,744
Frodo-1344	1344	2^{16}	1.4	$[-6\dots6]$	4	8×8	21,632

Table 2: **Security bounds**

	target level	**failure rate**	**LWE security**			**IND-CCA security**
			C	**Q**	**P**	**C**
Frodo-640	1	$2^{-138.7}$	145	132	104	141
Frodo-976	3	$2^{-199.6}$	210	191	150	206
Frodo-1344	5	$2^{-252.5}$	275	250	197	268

2.5 Summary of parameters

Table 4 summarizes all cryptographic parameters for Frodo-640, Frodo-976 and Frodo-1344. FrodoKEM-640-AES, FrodoKEM-976-AES and FrodoKEM-1344-AES use AES128 for generation of $\mathbf{A}$; FrodoKEM-640-SHAKE, FrodoKEM-976-SHAKE and FrodoKEM-1344-SHAKE use SHAKE for generation of $\mathbf{A}$.

Table 5 summarizes the sizes, in bytes, of the different inputs and outputs required by FrodoKEM. Note that we also include the size of the public key in the secret key sizes, in order to comply with NIST's API

Table 3: **Error distributions**

	σ	**Probability of (in multiples of 2^{-16})**													**Rényi**	
		0	±1	±2	±3	±4	±5	±6	±7	±8	±9	±10	±11	±12	**order**	**divergence**
$\chi_{\textsf{Frodo-640}}$	2.8	9288	8720	7216	5264	3384	1918	958	422	164	56	17	4	1	200	0.324×10^{-4}
$\chi_{\textsf{Frodo-976}}$	2.3	11278	10277	7774	4882	2545	1101	396	118	29	6	1			500	0.140×10^{-4}
$\chi_{\textsf{Frodo-1344}}$	1.4	18286	14320	6876	2023	364	40	2							1000	0.264×10^{-4}

Table 4: **Cryptographic parameters for Frodo-640, Frodo-976, and Frodo-1344**

	Frodo-640	Frodo-976	Frodo-1344
D	15	16	16
q	32768	65536	65536
n	640	976	1344
$\overline{m} = \overline{n}$	8	8	8
B	2	3	4
$\textsf{len}_{\textsf{seed}_{\mathbf{A}}}$	128	128	128
$\textsf{len}_{\mathbf{z}}$	128	128	128
$\textsf{len}_{\mu} = \ell$	128	192	256
$\textsf{len}_{\textsf{seed}_{\mathbf{SE}}}$	128	192	256
$\textsf{len}_{\mathbf{s}}$	128	192	256
$\textsf{len}_{\mathbf{k}}$	128	192	256
$\textsf{len}_{\mathbf{pkh}}$	128	192	256
$\textsf{len}_{\mathbf{ss}}$	128	192	256
$\textsf{len}_{\chi}$	16	16	16
χ	$\chi_{\textsf{Frodo-640}}$	$\chi_{\textsf{Frodo-976}}$	$\chi_{\textsf{Frodo-1344}}$
SHAKE	SHAKE128	SHAKE256	SHAKE256

guidelines. Specifically, since NIST's decapsulation API does not include an input for the public key, it needs to be included as part of the secret key.

Table 5: **Size (in bytes) of inputs and outputs of FrodoKEM.** Secret key size is the sum of the sizes of the actual secret value and of the public key (the NIST API does not include the public key as explicit input to decapsulation).

Scheme	**secret key** *sk*	**public key** *pk*	**ciphertext** *c*	**shared secret** **ss**
FrodoKEM-640	19,888 (10,272 + 9,616)	9,616	9,720	16
FrodoKEM-976	31,296 (15,664 + 15,632)	15,632	15,744	24
FrodoKEM-1344	43,088 (21,568 + 21,520)	21,520	21,632	32

2.6 Provenance of constants and tables

Constants used as domain separators in calls to SHAKE are described in Section 2.3.

The constants in Table 1 and Table 3 were generated by search scripts following the methodology described in Section 2.4.

3 Performance analysis

3.1 Associated implementations

The submission package includes:

- a reference implementation written exclusively in Python,
- a reference implementation written exclusively in portable C,
- an optimized implementation written exclusively in portable C that includes efficient algorithms to generate the matrix $\mathbf{A}$ and to compute the matrix operations $\mathbf{AS} + \mathbf{E}$ and $\mathbf{S}'\mathbf{A} + \mathbf{E}'$, and
- an additional, optimized implementation for x64 platforms that exploits Advanced Vector Extensions 2 (AVX2) intrinsic instructions.

The implementations in the submission package support all three security levels and both variants of matrix generation: FrodoKEM-640-AES, FrodoKEM-640-SHAKE, FrodoKEM-976-AES, FrodoKEM-976-SHAKE, FrodoKEM-1344-AES, and FrodoKEM-1344-SHAKE. The only difference between the reference and the optimized implementation is that the latter includes two efficient functions to generate the public matrix $\mathbf{A}$ and to compute the matrix operations $\mathbf{AS} + \mathbf{E}$ and $\mathbf{S}'\mathbf{A} + \mathbf{E}'$. Similarly, the only difference between the optimized and the additional implementation is that the latter uses AVX2 intrinsic instructions to speed up the implementation of the aforementioned functions. Hence, the different implementations share most of their codebase: this illustrates the simplicity of software based on FrodoKEM.

All our implementations avoid the use of secret address accesses and secret branches and, hence, are protected against timing and cache attacks.

3.2 Performance analysis on x64 Intel

In this section, we summarize the results of our performance evaluation using a machine equipped with a 3.4GHz Intel Core i7-6700 (Skylake) processor and running Ubuntu 16.04.3 LTS. As standard practice, TurboBoost was disabled during the tests. For compilation we used GNU GCC version 7.2.0 with the command `gcc -O3 -march=native`. The generation of the matrix $\mathbf{A}$ is the most expensive part of the computation. As described in Section 2.2.5, we support two ways of generating $\mathbf{A}$: one using AES128 and one using SHAKE128.

3.2.1 Performance using AES128

Table 6 details the performance of the optimized implementations and the additional x64 implementations when using AES128 for the generation of the matrix $\mathbf{A}$. The top two sets of results correspond to performance when using OpenSSL's AES implementation[4] and the bottom set presents the results when using a standalone AES implementation using Intel's Advanced Encryption Standard New Instructions (AES-NI).

As can be observed, the different implementation variants have similar performance, even when using hand-optimized AVX2 intrinsic instructions. This illustrates that FrodoKEM's algorithms, which are mainly based on matrix operations, facilitate automatic parallelization using vector instructions. Hence, the compiler is able to achieve close to "optimal" performance with little intervention from the programmer. The best results for FrodoKEM-640-AES, FrodoKEM-976-AES and FrodoKEM-1344-AES (i.e., 1.1 ms, 2.0 ms and 3.4 ms., respectively, obtained by adding the times for encapsulation and decapsulation) are achieved by the optimized implementation using C only. However, the difference in performance between the different implementations reported in Table 6 is, in all the cases, less than 1%.

We note that the performance of FrodoKEM using AES on Intel platforms greatly depends on AES-NI instructions. For example, when turning off the use of these instructions the computing cost of the optimized implementation of FrodoKEM-640-AES (resp., FrodoKEM-976-AES) is 26.5 ms (resp., 61.1 ms), which is roughly a 24-fold (resp., 31-fold) degradation in performance.

3.2.2 Performance using SHAKE128

Table 7 outlines the performance figures of the optimized implementations and the additional x64 implementations when using SHAKE128 for the generation of the matrix $\mathbf{A}$. The top set of results shows the

[4]Note that in order to enable AES-NI instructions in OpenSSL, we use the `EVP_aes_128_ecb` interface in OpenSSL.

Table 6: **Performance (in thousands of cycles) of FrodoKEM on a 3.4GHz Intel Core i7-6700 (Skylake) processor with matrix A generated using AES128.** Results are reported using OpenSSL's AES implementation and using a standalone AES implementation, all of which exploit AES-NI instructions. Cycle counts are rounded to the nearest 10^3 cycles.

Scheme	**KeyGen**	**Encaps**	**Decaps**	**Total (Encaps + Decaps)**
Optimized Implementation (AES from OpenSSL)				
FrodoKEM-640-AES	1,385	1,862	1,747	3,609
FrodoKEM-976-AES	2,836	3,560	3,398	6,958
FrodoKEM-1344-AES	4,747	5,963	5,740	11,703
Additional implementation using AVX2 intrinsic instructions (AES from OpenSSL)				
FrodoKEM-640-AES	1,384	1,861	1,751	3,612
FrodoKEM-976-AES	2,896	3,563	3,399	6,962
FrodoKEM-1344-AES	4,732	5,965	5,738	11,703
Additional implementation using AVX2 intrinsic instructions (standalone AES)				
FrodoKEM-640-AES	1,388	1,873	1,756	3,629
FrodoKEM-976-AES	2,821	3,598	3,449	7,047
FrodoKEM-1344-AES	4,784	6,061	5,798	11,859

performance of the optimized implementation written in C only, while the bottom set presents the results when using a 4-way implementation of SHAKE using AVX2 instructions ("SHAKE4x using AVX2"). Note that the use of such a vectorized implementation of SHAKE is necessary to boost the practical performance. In our use-case, it results in a two-fold speedup when compared to the version using a SHAKE implementation written in plain C.

Comparing Table 6 and Table 7, FrodoKEM using AES, when implemented with AES-NI instructions, is around 2.4–2.8× faster than the vectorized SHAKE implementation. Nevertheless, this comparative result could change drastically if hardware-accelerated instructions such as AES-NI are not available on the targeted platform, or if support for hardware-accelerated instructions for SHA-3 is added in the future.

3.2.3 Memory analysis

Table 8 shows the peak usage of stack memory per function. In addition, in the right-most column we show the size of the produced static libraries.

In order to determine the memory usage we ran `valgrind` (http://valgrind.org/) to obtain "memory use snapshots" during execution of the test program:

```
$ valgrind --tool=massif --stacks=yes --detailed-freq=1 ./frodo/test_KEM
```

This command produces a file of the form `massif.out.xxxxx`. We then ran `massif-cherrypick` (https://github.com/lnishan/massif-cherrypick), which is an extension that outputs memory usage per function:

```
$ ./massif-cherrypick massif.out.xxxxx kem_function
```

The results are summarized in Table 8. Note that in our implementations the use of SHAKE for generating **A** reduces peak memory usage in up to 22%. However, the vectorized AVX2 implementation of SHAKE increases the size of the produced static libraries significantly (implementations based on AES-NI instructions are indeed very compact).

Table 7: **Performance (in thousands of cycles) of FrodoKEM on a 3.4GHz Intel Core i7-6700 (Skylake) processor with matrix A generated using** SHAKE128**.** Results are reported for two test cases: (i) using a SHAKE implementation written in plain C and, (ii) using a 4-way implementation of SHAKE using AVX2 instructions. Cycle counts are rounded to the nearest 10^3 cycles.

Scheme	**KeyGen**	**Encaps**	**Decaps**	**Total (Encaps + Decaps)**
Optimized Implementation (plain C SHAKE)				
FrodoKEM-640-SHAKE	7,657	8,391	8,278	16,669
FrodoKEM-976-SHAKE	16,879	18,125	17,962	36,087
FrodoKEM-1344-SHAKE	30,300	32,604	32,395	64,999
Additional implementation using AVX2 intrinsics (SHAKE4x using AVX2)				
FrodoKEM-640-SHAKE	4,022	4,440	4,325	8,765
FrodoKEM-976-SHAKE	8,579	9,302	9,143	18,445
FrodoKEM-1344-SHAKE	15,191	16,357	16,148	32,505

Table 8: **Peak usage of stack memory (in bytes) and static library size (in bytes)** of the optimized and additional implementations of FrodoKEM on a 3.4GHz Intel Core i7-6700 (Skylake) processor. Compilation with GNU GCC version 7.2.0 using flags `-O3 -march=native`. Matrix **A** is generated with either SHAKE128 or AES128 (using OpenSSL's AES implementation or the standalone AES implementation).

Scheme	**Peak stack memory usage**			**Static library size**
	KeyGen	**Encaps**	**Decaps**	
Optimized Implementation (AES from OpenSSL)				
FrodoKEM-640-AES	72,448	102,944	123,968	68,668
FrodoKEM-976-AES	111,424	158,944	189,080	66,236
FrodoKEM-1344-AES	152,688	216,552	259,784	64,732
Additional implementation using AVX2 intrinsics (standalone AES)				
FrodoKEM-640-AES	71,272	102,040	122,776	66,062
FrodoKEM-976-AES	110,200	157,752	189,240	63,630
FrodoKEM-1344-AES	151,496	216,696	259,258	62,158
Additional implementation using AVX2 intrinsics (SHAKE4x using AVX2)				
FrodoKEM-640-SHAKE	70,264	81,848	102,584	207,938
FrodoKEM-976-SHAKE	106,552	124,792	156,280	205,346
FrodoKEM-1344-SHAKE	144,856	169,752	211,528	203,850

3.3 Performance analysis on ARM

In this section, we summarize the results of our performance evaluation using a device powered by a 1.992GHz 64-bit ARM Cortex-A72 (ARMv8) processor and running Ubuntu 16.04.2 LTS. For compilation we used GNU GCC version 5.4.0 with the command `gcc -O3 -march=native`.

Table 9 details the performance of the optimized implementations when using AES128 and SHAKE128. Similar to the case of the x64 Intel platform, the overall performance of FrodoKEM is highly dependent on the performance of the primitive that is used for the generation of the matrix **A**. Hence, the best performance in this case is achieved when using OpenSSL's AES implementation, which exploits the efficient NEON engine. On the other hand, SHAKE performs significantly better when there is no support for specialized instructions

Table 9: **Performance (in thousands of cycles) of the optimized implementations of FrodoKEM on a 1.992GHz 64-bit ARM Cortex-A72 (ARMv8) processor.** Results are reported for three test cases: (i) using OpenSSL's AES implementation, (ii) using an AES implementation written in plain C, and (iii) using a SHAKE implementation written in plain C. Results have been scaled to cycles using the nominal processor frequency. Cycle counts are rounded to the nearest 10^3 cycles.

Scheme	**KeyGen**	**Encaps**	**Decaps**	**Total** (Encaps + Decaps)
Optimized Implementation (AES from OpenSSL)				
FrodoKEM-640-AES	3,470	4,057	3,969	8,026
FrodoKEM-976-AES	7,219	8,530	8,014	16,544
FrodoKEM-1344-AES	12,789	14,854	14,635	29,489
Optimized implementation (plain C AES)				
FrodoKEM-640-AES	44,354	44,766	44,765	89,531
FrodoKEM-976-AES	101,540	102,551	102,460	205,011
FrodoKEM-1344-AES	191,359	193,123	192,458	385,581
Optimized implementation (plain C SHAKE**)**				
FrodoKEM-640-SHAKE	11,278	12,411	12,311	24,722
FrodoKEM-976-SHAKE	24,844	27,033	26,936	53,969
FrodoKEM-1344-SHAKE	44,573	48,554	48,449	97,003

in the targeted platform: using a plain C version of SHAKE is more than 3 times faster than using a plain C version of AES.

4 Known Answer Test (KAT) values

The submission includes KAT values with tuples containing secret key (sk), public key (pk), ciphertext (c) and shared secret ($\mathbf{ss}$) values for the proposed KEM schemes. The KAT files can be found in the KAT folder of the submission:

Scheme	**KAT file**
FrodoKEM-640-AES	`\KAT\PQCkemKAT_19888.rsp`
FrodoKEM-976-AES	`\KAT\PQCkemKAT_31296.rsp`
FrodoKEM-1344-AES	`\KAT\PQCkemKAT_43088.rsp`
FrodoKEM-640-SHAKE	`\KAT\PQCkemKAT_19888_shake.rsp`
FrodoKEM-976-SHAKE	`\KAT\PQCkemKAT_31296_shake.rsp`
FrodoKEM-1344-SHAKE	`\KAT\PQCkemKAT_43088_shake.rsp`

In addition, we provide a test suite that can be used to verify the KAT values against any of the implementations. Instructions to compile and run the KAT test suite can be found in the README file (see Section 2, "Quick Instructions").

5 Justification of security strength

The security of FrodoKEM is supported both by security reductions and by analysis of the best known cryptanalytic attacks. A summary of the bit-security estimates based on these two methodologies is shown in Table 2.

5.1 Security reductions

A summary of the reductions supporting the security of FrodoKEM is as follows:

1. FrodoKEM, using the concrete error distributions χ_{Frodo} specified in Table 3, is an IND-CCA-secure KEM against classical attacks in the classical random oracle model, under the assumption that FrodoPKE using a rounded Gaussian error distribution is an IND-CPA-secure public-key encryption scheme against classical attacks. This is Theorem 5.1, and the reduction is tight. The argument combines results of [68] on the modular FO transform with results of [81] on Rényi divergence. We also note that the same conclusion follows from the assumption that FrodoPKE using the distributions χ_{Frodo} is IND-CPA secure, using the same proof but without any analysis of Rényi divergence.
2. FrodoKEM, using any error distribution, is an IND-CCA-secure KEM against quantum attackers in the quantum random oracle model, under the assumption that FrodoPKE using the same error distribution is an OW-CPA-secure public-key encryption scheme against quantum attackers. This is Theorem 5.8, and the reduction is non-tight. We view this theorem as giving support for the security of general constructions of LWE-based KEMs in the style of FrodoKEM against quantum adversaries, but it does not concretely support the bit-security of the six FrodoKEM instantiations in this document, which is why we omit the corresponding column from Table 2.
3. Changing the distribution of matrix $\mathbf{A}$ from a truly uniform distribution to one generated from a public random seed in a pseudorandom fashion does not affect the security of FrodoKEM or FrodoPKE, provided that the pseudorandom generator is modeled either as an ideal cipher (when using AES128) or a random oracle (when using SHAKE128). This is shown in Section 5.1.3.
4. FrodoPKE, using any error distribution and a uniformly random $\mathbf{A}$, is an IND-CPA-secure public-key encryption scheme under the assumption that the uniform-secret learning with errors decision problem is hard for the same parameters (except for a small additive loss in the number of samples), for either classical or quantum adversaries. This is a consequence of Theorem 5.9 and Theorem 5.10, and the result is tight.
5. The uniform-secret learning with errors decision problem, using a rounded Gaussian distribution with parameter σ from Table 1 and an appropriate bound on the number of samples, is hard under the assumption that the *worst-case* bounded-distance decoding with discrete Gaussian samples problem (BDDwDGS, Definition 5.11) is hard for related parameters. Theorem 5.12 gives a non-tight classical reduction against classical or quantum adversaries (in the standard model).

5.1.1 IND-CCA security in the classical random oracle model

The following theorem says that the transformation $\mathsf{FO}^{\not\perp\prime}$, which we use to construct FrodoKEM from FrodoPKE, generically yields an IND-CCA-secure KEM (in the classical random oracle model) from an IND-CPA-secure public-key encryption scheme, even if the KEM and PKE are parameterized by different distributions, provided that those distributions are sufficiently close in terms of Rényi divergence.

Theorem 5.1 **(IND-CPA PKE $\Longrightarrow$ IND-CCA KEM in classical ROM, with distribution switch).** *Let $\mathsf{PKE}_X = (\mathrm{KeyGen}, \mathrm{Enc}, \mathrm{Dec})$ be a δ-correct public-key encryption scheme with message space $\mathcal{M}$ that is parameterized by a distribution X, and let s be an upper bound on the total number of samples drawn from X by* KeyGen *and* Enc *combined. Let G_1, G_2 and F be independent random oracles, and let $\mathsf{KEM}_X^{\not\perp\prime} = \mathsf{FO}^{\not\perp\prime}[\mathsf{PKE}_X, G_1, G_2, F]$ be the KEM obtained by applying the $\mathsf{FO}^{\not\perp\prime}$ transform from Definition 2.19 to PKE_X. Let P, Q be any discrete distributions. There exists a classical algorithm (a reduction) $\mathcal{B}$ against the* IND-CPA *security of PKE_Q, which uses as a "black box" subroutine any $\mathcal{A}$ against the* IND-CCA *security* *of* $\mathsf{KEM}_P^{\not\perp\prime}$ *that makes at most* q_{RO} *oracle queries, for which*

$$\mathrm{Adv}^{\mathsf{ind\text{-}cca}}_{\mathsf{KEM}_P^{\not\perp\prime}}(\mathcal{A}) \leq \frac{q_{\mathsf{RO}}}{|\mathcal{M}|} + \left(\left(\frac{2 \cdot q_{\mathsf{RO}} + 1}{|\mathcal{M}|} + q_{\mathsf{RO}} \cdot \delta + 3 \cdot \mathrm{Adv}^{\mathsf{ind\text{-}cpa}}_{\mathsf{PKE}_Q}(\mathcal{B}) \right) \cdot \exp(s \cdot \mathrm{D}_\alpha(P \| Q)) \right)^{1-1/\alpha} \tag{3}$$

for any $\alpha > 1$*, where the Rényi divergence* D_α *is defined in Definition 5.4. The total running time of* $\mathcal{B}$ *is about that of* $\mathcal{A}$ *plus the time needed to simulate the random oracles.*

We point out that when $P = Q$ are the same distribution, we have $\exp(s \cdot \mathrm{D}_\alpha(P\|Q)) = 1$ for any $\alpha > 1$ and hence can take α to be arbitrarily large, making the exponent $1 - 1/\alpha$ approach 1 in the limit. This special case is a main theorem from [68], and it relates the IND-CCA security of FrodoKEM to the IND-CPA security of FrodoPKE when they use the same error distribution, e.g., χ_{Frodo}.

The proof of Theorem 5.1 combines components from two separate works: the modular analysis of the Fujisaki–Okamoto transform by HHK [68], and the work of Langlois, Stehlé, and Steinfeld relating the security of search problems when one distribution is substituted by another via analysis of the Rényi divergence [81]. More specifically, the proof of the theorem proceeds in the following steps:

1. We apply HHK's Theorem 3.2, which shows that their T transform converts an IND-CPA-secure public-key encryption scheme PKE_Q into an OW-PCA-secure public-key encryption scheme with deterministic encryption (in the random oracle model).
2. Next, we apply distribution substitution for the OW-PCA security experiment (which represents a search problem), to switch from distribution Q to P.
3. Finally, we apply HHK's Theorem 3.4, which shows that their $\mathsf{U}^{\not\perp}$ transform converts an OW-PCA-secure public-key encryption scheme into an IND-CCA-secure KEM (in the random oracle model).

HHK denote the composition of the T and $\mathsf{U}^{\not\perp}$ transforms as the $\mathsf{FO}^{\not\perp}$ transform. As described in Section 2.2.8, we use a slight variant of this transform called $\mathsf{FO}^{\not\perp\prime}$, which differs from $\mathsf{FO}^{\not\perp\prime}$ as follows:

- $\mathsf{FO}^{\not\perp\prime}$ uses a single hash function (with longer output) to compute r and K, whereas $\mathsf{FO}^{\not\perp}$ uses two separate functions, but these are equivalent when the hash functions are modeled as independent random oracles and have appropriate output lengths.
- The $\mathsf{FO}^{\not\perp\prime}$ computation of r and K also takes the hash $G_1(pk)$ of the public key pk as input, whereas $\mathsf{FO}^{\not\perp}$ does not; this change preserves the relevant theorems (with trivial changes to the proofs), and has the potential to provide stronger multi-target security.

Step 1: IND-CPA PKE to OW-PCA deterministic PKE_1. For completeness, we recall the definition of OW-PCA, following the presentation of Hofheinz et al. [68].

Definition 5.2 (OW-PCA for PKE [96]). Let PKE be a public-key encryption scheme with message space $\mathcal{M}$ and let $\mathcal{A}$ be an algorithm. The OW-PCA security experiment for $\mathcal{A}$ attacking PKE is $\mathrm{Exp}^{\mathsf{ow\text{-}pca}}_{\mathsf{PKE}}(\mathcal{A})$ from Figure 4. The advantage of $\mathcal{A}$ in the experiment is

$$\mathrm{Adv}^{\mathsf{ow\text{-}pca}}_{\mathsf{PKE}}(\mathcal{A}) := \Pr\left[\mathrm{Exp}^{\mathsf{ow\text{-}pca}}_{\mathsf{PKE}}(\mathcal{A}) \Rightarrow 1\right].$$

Experiment $\mathrm{Exp}^{\mathsf{ow\text{-}pca}}_{\mathsf{PKE}}(\mathcal{A})$:	Oracle $\mathcal{O}_{\mathrm{Pco}}(m, c)$:
1: $(pk, sk) \leftarrow \mathsf{PKE}.\mathrm{KeyGen}()$	1: **if** $\mathsf{PKE}.\mathrm{Dec}(sk, c) = m$ **then**
2: $m \leftarrow_{\$} \mathcal{M}$	2: **return** 1
3: $c^* \leftarrow \mathsf{PKE}.\mathrm{Enc}(m, pk)$	3: **else**
4: $m' \leftarrow \mathcal{A}^{\mathcal{O}_{\mathrm{Pco}}(\cdot,\cdot)}(pk, c^*)$	4: **return** 0
5: **return** $\mathcal{O}_{\mathrm{Pco}}(m', c^*)$	

Figure 4: Security experiment for OW-PCA.

The T transform of HHK converts a public-key encryption scheme PKE to a deterministic public-key encryption scheme PKE_1; see Figure 5. HHK's Theorem 3.2 tightly establishes the OW-PCVA-security of PKE_1 under, among others, the assumption that PKE is IND-CPA secure and γ-spread. (In the OW-PCVA

$\underline{\mathsf{PKE}_1.\mathrm{KeyGen}():}$

return $\mathsf{PKE}.\mathrm{KeyGen}()$

$\underline{\mathsf{PKE}_1.\mathrm{Enc}(\mu, pk):}$

$r \leftarrow G_2(\mu)$
$c \leftarrow \mathsf{PKE}.\mathrm{Enc}(\mu, pk; r)$
return c

$\underline{\mathsf{PKE}_1.\mathrm{Dec}(c, sk):}$

$\mu' \leftarrow \mathsf{PKE}.\mathrm{Dec}(c, sk)$
if $\mu' = \perp$ or $c \neq \mathsf{PKE}.\mathrm{Enc}(\mu', pk; G_2(\mu'))$ **then**
  return $\perp$
else
  return μ'

Figure 5: Construction of deterministic public-key encryption scheme $\mathsf{PKE}_1 = \mathsf{T}[\mathsf{PKE}, G_2]$ from a public-key encryption scheme PKE and hash function G_2.

security game, the attacker additionally has a ciphertext-validity oracle, which checks whether a queried ciphertext has a valid decryption.) However, they note that OW-PCA security follows (tightly) *without* the γ-spread assumption, because in the security bounds γ-spreadness is relevant only to ciphertext-validity queries. We state that adapted version here.

Lemma 5.3 ([68], Theorem 3.2, OW-PCA version). *Let* PKE *be a δ-correct public-key encryption scheme with message space $\mathcal{M}$. For any* OW-PCA *adversary $\mathcal{A}$ that issues at most q_G queries to the random oracle G_2 and q_P queries to the plaintext-checking oracle, there exists an* IND-CPA *adversary $\mathcal{B}$ such that*

$$\mathrm{Adv}^{\mathsf{ow\text{-}pca}}_{\mathsf{PKE}_1}(\mathcal{A}) \leq q_G \cdot \delta + \frac{2q_G + 1}{|\mathcal{M}|} + 3 \cdot \mathrm{Adv}^{\mathsf{ind\text{-}cpa}}_{\mathsf{PKE}}(\mathcal{B}) \ ,$$

and the running time of $\mathcal{B}$ is about that of $\mathcal{A}$ plus the time needed to simulate the random oracle.

It is straightforward to verify from the proof that $\mathcal{B}$ uses $\mathcal{A}$ solely as a "black box" subroutine.

Step 2: Approximating the error distribution. The rounded Gaussian distribution (Definition 2.11), which is important to the worst-case-to-average-case reduction, is difficult to sample on a finite computer (and impossible to sample in constant time). Following Langlois et al. [81], we replace this infinite-precision distribution with a finite approximation, and quantify the OW-PCA security loss using their Rényi divergence.

Definition 5.4 (Rényi divergence). The Rényi divergence of positive order $\alpha \neq 1$ of a discrete distribution P from a distribution Q is defined as

$$\mathrm{D}_\alpha(P\|Q) = \frac{1}{\alpha - 1} \ln\left(\sum_{x \in \operatorname{supp} P} P(x) \left(\frac{P(x)}{Q(x)}\right)^{\alpha-1}\right) \ .$$

Note that our definition differs from that of [81] in that we take the logarithm of the sum, and that Rényi divergence is not symmetric. The following result relates probabilities of a certain event occurring under two distributions as a function of their Rényi divergence.

Lemma 5.5 ([81, Lemma 4.1]). *Let S be an event defined in a probabilistic experiment G_Q in which s samples are drawn from distribution Q. Then the probability that S occurs in the same experiment but with Q replaced by P is bounded as follows:*

$$\Pr[G_P(S)] \leq (\Pr[G_Q(S)] \cdot \exp(s \cdot \mathrm{D}_\alpha(P\|Q)))^{1-1/\alpha} \ . \tag{4}$$

It immediately follows that reductions from any *search* problem, such as the one represented by the OW-PCA game, are preserved up to the relaxation in (4). For any given security relationship, and any concrete choice of the two distributions P and Q, the loss can be minimized by choosing an optimal value of the order α.

Corollary 5.6 (Distribution substitution for OW-PCA). *Let PKE_X be a public-key encryption scheme that is parameterized by a distribution X, and let s be an upper bound on the total number of samples drawn from X by $\mathsf{PKE}_X.\mathrm{Enc}$ and $\mathsf{PKE}_X.\mathrm{KeyGen}$ combined. Let $\mathcal{A}$ be an OW-PCA adversary against PKE_X, and let P and Q be discrete distributions. Then for any $\alpha > 1$,*

$$\mathrm{Adv}^{\mathsf{ow\text{-}pca}}_{\mathsf{PKE}_P}(\mathcal{A}) \leq \left(\mathrm{Adv}^{\mathsf{ow\text{-}pca}}_{\mathsf{PKE}_Q}(\mathcal{A}) \cdot \exp(s \cdot \mathrm{D}_\alpha(P\|Q))\right)^{1-1/\alpha} .$$

Proof. This follows immediately from Lemma 5.5, with S being the event that $\mathcal{A}$ "wins" the OW-PCA experiment from Figure 4, i.e., causes it to output 1. ☐

We use Corollary 5.6 to relate the OW-PCA security of $\mathsf{T}[\mathsf{FrodoPKE}_P, G_2]$ to the OW-PCA security of $\mathsf{T}[\mathsf{FrodoPKE}_\Psi, G_2]$ where $\mathsf{FrodoPKE}_\Psi$ is the same as FrodoPKE but with the error distribution $P = \chi_{\mathsf{Frodo}}$ replaced by a rounded Gaussian distribution $Q = \Psi$ (see Definition 2.11).

Step 3: OW-PCA deterministic PKE_1 to IND-CCA KEM. HHK define the $\mathsf{U}^{\not\perp}$ transform from a deterministic public-key encryption scheme PKE_1 to a key encapsulation mechanism $\mathsf{KEM}^{\not\perp}$; see Figure 6. HHK's Theorem 3.4 shows the IND-CCA security of $\mathsf{KEM}^{\not\perp} = \mathsf{U}^{\not\perp}[\mathsf{PKE}_1, F]$ assuming the OW-PCA security of the underlying PKE_1. Their result is stated below in Lemma 5.7.

$\underline{\mathsf{KEM}^{\not\perp}.\mathrm{KeyGen}():}$

1. $(pk, sk) \leftarrow_\$ \mathsf{PKE}_1.\mathrm{KeyGen}()$
2. $s \leftarrow_\$ \mathcal{M}$
3. $sk' \leftarrow (sk, s)$
4. **return** (pk, sk')

$\underline{\mathsf{KEM}^{\not\perp}.\mathrm{Encaps}(pk):}$

1. $\mu \leftarrow_\$ \mathcal{M}$
2. $c \leftarrow \mathsf{PKE}_1.\mathrm{Enc}(\mu, pk)$
3. $ss \leftarrow F(\mu, c)$
4. **return** (c, ss)

$\underline{\mathsf{KEM}^{\not\perp}.\mathrm{Decaps}(c, (sk, s)):}$

1. $\mu' \leftarrow \mathsf{PKE}.\mathrm{Dec}(c, sk)$
2. **if** $\mu' \neq \perp$ **then**
3.   **return** $ss' \leftarrow F(\mu', c)$
4. **else**
5.   **return** $ss' \leftarrow F(s, c)$

Figure 6: Construction of key encapsulation mechanism $\mathsf{KEM}^{\not\perp} = \mathsf{U}^{\not\perp}[\mathsf{PKE}_1, F]$ from a deterministic public-key encryption scheme PKE_1 and hash function F.

Lemma 5.7 ([68, Theorem 3.4]). *Model F as a random oracle. Then if PKE_1 is δ_1-correct, so is $\mathsf{KEM}^{\not\perp}$. For any IND-CCA adversary $\mathcal{A}$ against $\mathsf{KEM}^{\not\perp}$ issuing at most q_F queries to F, there exists an OW-PCA adversary $\mathcal{B}$ against PKE_1 that makes at most q_F queries to its plaintext-checking oracle, and for which*

$$\mathrm{Adv}^{\mathsf{ind\text{-}cca}}_{\mathsf{KEM}^{\not\perp}}(\mathcal{A}) \leq \frac{q_F}{|\mathcal{M}|} + \mathrm{Adv}^{\mathsf{ow\text{-}pca}}_{\mathsf{PKE}_1}(\mathcal{B}) ,$$

where the running time of $\mathcal{B}$ is about that of $\mathcal{A}$, plus the time to simulate the random oracle and decapsulation queries.

It is straightforward to verify from the proof that $\mathcal{B}$ uses $\mathcal{A}$ solely as a "black box" subroutine. Together, Lemma 5.3, Corollary 5.6, and Lemma 5.7 establish Theorem 5.1.

Applying Theorem 5.1. For an application of Theorem 5.1 to our schemes, consider the relation between the IND-CCA security of FrodoKEM-640 and the IND-CPA security of $\mathsf{FrodoPKE\text{-}640}_\Psi$, where the error distribution of the latter is taken to be the rounded Gaussian $\Psi_{2.8/\sqrt{2\pi}}$ as defined in Section 2.1.4.

To extract exact bounds on the IND-CCA security (in the classical ROM) of FrodoKEM-640, we make a number of assumptions about the underlying cost model. Specifically,

- We ignore the overhead of running the reduction of Theorem 5.1, including the cost of simulating random oracles.
- The cost to the adversary of *making* an oracle query is 2^{18} classical gates. This bound is based on the NIST Call for Proposals, Section 4.A.5, which estimates the cost of finding collisions in SHA-3 at all security levels. (Here we ignore the small performance differences between SHAKE128, SHAKE256, and SHA3-256.)
- We interpret "b bits of classical security" as a statement that the advantage in the corresponding game of a uniform t-gate classical adversary is bounded by $t/2^b$. For some tasks, such as collision finding, this upper bound can be quite loose for smaller values of t (and thus beneficial to the adversary).
- The $\mathsf{IND\text{-}CPA}$ security of $\mathsf{FrodoPKE\text{-}640}_\Psi$ is given by the smaller of the costs of the primal and dual attacks on the LWE problem (Table 10), discounted by the reduction factor of $\overline{n}+\overline{m} = 16$ (Theorem 5.9), yielding $2^{-145.6}$.

Under these assumptions, if an adversary $\mathcal{B}$ has uniform gate complexity t, then it has advantage $\mathrm{Adv}^{\mathsf{ind\text{-}cpa}}_{\mathsf{FrodoPKE\text{-}640}_\Psi}(\mathcal{B})$ bounded by $t \cdot 2^{-145.6}$.

The Rényi divergence of $\chi_{\mathsf{Frodo\text{-}640}}$ from the rounded Gaussian is $\mathrm{D}_\alpha(\chi_{\mathsf{Frodo\text{-}640}} \| \Psi_{2.8/\sqrt{2\pi}}) \leq 0.0000324$ for $\alpha = 200$ (Table 3). The number of samples drawn from the error distribution by $\mathsf{FrodoPKE}$.KeyGen is $2n\overline{n}$, and by $\mathsf{FrodoPKE}$.Enc is $2\overline{m}n + \overline{m}\overline{n}$, which for $n = 640$ and $\overline{m} = \overline{n} = 8$ totals $s = 2 \times (8+8) \times 640 + 64 = 20544$.

Substituting $q_{\mathsf{RO}} < t \cdot 2^{-18}$, $|\mathcal{M}| = 2^{128}$ and $\delta < 2^{-138.7}$ into (3), we can bound the advantage in $\mathrm{Exp}^{\mathsf{ind\text{-}cca}}_{\mathsf{FrodoKEM\text{-}640}}$ for an adversary $\mathcal{A}$ with gate count $t \geq 1$ as follows:

$$\begin{aligned}\mathrm{Adv}^{\mathsf{ind\text{-}cca}}_{\mathsf{FrodoKEM\text{-}640}}(\mathcal{A}) &< 2^{-146} \cdot t + \left((2.01 \cdot 2^{-146} + 3 \cdot 2^{-145.6}) \cdot t \cdot \exp(20544 \cdot 0.0000324)\right)^{0.995} \\ &< 2^{-141.6} \cdot t \ .\end{aligned}$$

Similarly computed bounds on the advantage of a classical $\mathsf{IND\text{-}CCA}$ adversary for other parameter settings appear in Table 2.

5.1.2 $\mathsf{IND\text{-}CCA}$ security in the quantum random oracle model

Jiang et al. [70] show that the $\mathsf{FO}^{\not\perp}$ transform yields an $\mathsf{IND\text{-}CCA}$-secure KEM from an $\mathsf{OW\text{-}CPA}$-secure public-key encryption scheme, in the *quantum* random oracle model. As noted above, we apply a slight variant $\mathsf{FO}^{\not\perp\prime}$ of the $\mathsf{FO}^{\not\perp}$ transform, which does not affect Jiang et al.'s results.

Theorem 5.8 ($\mathsf{OW\text{-}CPA}$ **PKE** $\Longrightarrow$ $\mathsf{IND\text{-}CCA}$ **KEM in quantum ROM; [70, Theorem 1]).** *Let* $\mathsf{PKE} = (\mathrm{KeyGen}, \mathrm{Enc}, \mathrm{Dec})$ *be a* δ*-correct public-key encryption scheme with message space* $\mathcal{M}$*. Let* G_2 *and* F *be independent random oracles. Let* $\mathsf{KEM}^{\not\perp} = \mathsf{FO}^{\not\perp}[\mathsf{PKE}, G_2, F]$ *be the KEM obtained by applying the* $\mathsf{FO}^{\not\perp} = \mathsf{U}^{\not\perp} \circ \mathsf{T}$ *transform to* PKE*. For any quantum algorithm* $\mathcal{A}$ *against the* $\mathsf{IND\text{-}CCA}$ *security of* $\mathsf{KEM}^{\not\perp}$ *that makes* q_F *quantum oracle queries to* F *and* q_G *quantum oracle queries to* G_2*, there exists a quantum algorithm* $\mathcal{B}$ *against the* $\mathsf{OW\text{-}CPA}$ *security of* PKE *such that*

$$\mathrm{Adv}^{\mathsf{ind\text{-}cca}}_{\mathsf{KEM}^{\not\perp}}(\mathcal{A}) \leq \frac{2q_F}{\sqrt{|\mathcal{M}|}} + 4q_G\sqrt{\delta} + 2(q_G + q_F)\sqrt{\mathrm{Adv}^{\mathsf{ow\text{-}cpa}}_{\mathsf{PKE}}(\mathcal{B})}.$$

Moreover, the running time of $\mathcal{B}$ *is about that of* $\mathcal{A}$*.*

Note that Theorem 5.8 is not tight due to the square-root on the size of the message space $\mathcal{M}$, the square-root on the correctness error δ, the multiplicative factors from the number of hash function queries, and the square-root on the $\mathrm{Adv}^{\mathsf{ow\text{-}cpa}}_{\mathsf{PKE}}(\mathcal{B})$ term. In our parameter selection, we ignore the tightness gap arising from Theorem 5.8.

In an eprint posted in February 2019, Jiang, Zhang, and Ma [71, Theorem 3] give a tighter proof of the QROM security of the $\mathsf{FO}^{\not\perp}$ transform, with a bound of

$$\mathrm{Adv}^{\mathsf{ind\text{-}cca}}_{\mathsf{KEM}^{\not\perp}}(\mathcal{A}) \leq \frac{2q_F}{\sqrt{|\mathcal{M}|}} + 4q_G\sqrt{\delta} + 2\sqrt{q_{\mathsf{RO}} \cdot \mathrm{Adv}^{\mathsf{ind\text{-}cpa}}_{\mathsf{PKE}}(\mathcal{B}) + \frac{2(q_{\mathsf{RO}}+1)^2}{|\mathcal{M}|}} \text{ where } q_{\mathsf{RO}} = q_F + q_G \ .$$

This theorem also applies to our $\mathsf{KEM}^{\not\perp\prime}$.

5.1.3 Deterministic generation of A

The matrix $\mathbf{A}$ in $\mathsf{FrodoKEM}$ and $\mathsf{FrodoPKE}$ is deterministically expanded from a short random seed in the function Frodo.Gen either using AES128 or SHAKE128. In order to relate $\mathsf{FrodoKEM}$ and $\mathsf{FrodoPKE}$'s security to the hardness of the learning with errors problem, we argue that we can replace a uniformly sampled $\mathbf{A} \in \mathbb{Z}_q^{n\times n}$ with matrices sampled according to Frodo.Gen. Although the matrix appears pseudorandom under standard security assumptions to an adversary without access to the seed, we argue security of this step against a stronger (and more realistic) adversary via the indifferentiability framework [88, 43].

Informally, a construction $\mathcal{C}$ with access to an ideal primitive $\mathcal{G}$ is said to be ε-indifferentiable from an ideal primitive $\mathcal{F}$ if there exists a simulator $\mathcal{S}$ such that for any polynomial time distinguisher $\mathcal{D}$ it holds that $\left|\Pr\left[\mathcal{D}^{\mathcal{C},\mathcal{G}} = 1\right] - \Pr\left[\mathcal{D}^{\mathcal{F},\mathcal{S}} = 1\right]\right| < \varepsilon$. An indifferentiability argument implies that any cryptosystem secure in the $\mathcal{F}$-model remains secure (in a tight sense) in the $\mathcal{G}$-model with $\mathcal{F}$ instantiated as $\mathcal{C}^{\mathcal{G}}$ [88]. In what follows, we consider the ideal primitive $\mathcal{F}$ to be an ideal "domain expansion" function expanding a small seed to a matrix $\mathbf{A}$. Critically, the security of the step depends on the properties of $\mathcal{G}$ rather than randomness of the seed. The construction $\mathcal{C}$ and primitive $\mathcal{G}$ depend on whether we use AES128 or SHAKE128, modeled below as an ideal cipher and an ideal extendable-output function (XOF) respectively.

Using AES128 to generate A. Algorithm 7 generates the entries of $\mathbf{A}$ as 16-bit values and then reduces each one modulo q. For simplicity, we assume that $\mathbf{A}$ consists of $N = 16n^2$ bits and we set $M = N/128$. This means that $\mathbf{A}$ consists of M 128-bit AES128 blocks. The pseudorandom bits in the ith block are generated by encrypting a fixed index idx_i with a uniformly random $\mathsf{seed}_{\mathbf{A}} \in \{0,1\}^{128}$ as the key. Throughout, we refer to the set $\mathrm{Idx} := \{\mathsf{idx}_1, \ldots, \mathsf{idx}_M\}$ as the set of indices used in the pseudorandom generation of $\mathbf{A}$.

The ideal domain expansion primitive $\mathcal{F}$ expands a uniformly random seed $\mathsf{seed}_{\mathbf{A}} \in \{0,1\}^{128}$ to a larger bit string $s_1\|s_2\|\cdots\|s_M \in \{0,1\}^{128M}$ subject to the condition that $s_i \neq s_j$ for any distinct pair of i, j. Observe that a uniformly sampled $\mathbf{A}$ satisfies this condition with probability at least $1 - M^2/2^{128}$. In our security reductions, the matrix $\mathbf{A}$ is constructed through $m = n$ calls to the LWE oracle (Definition 2.9). By increasing the number of calls to this oracle marginally, by setting $m = 1.01n > n\cdot(1 - M^2/2^{128})^{-1}$, we can construct an LWE matrix $\mathbf{A}$ sampled from the same distribution as the output of $\mathcal{F}$ with overwhelming probability without affecting its underlying security.

When Frodo.Gen uses AES128, we consider a construction $\mathcal{C}^{\mathcal{G}}$ in the Ideal Cipher model implementing $\mathcal{F}$ as $\mathrm{AES128}_{\mathsf{seed}_{\mathbf{A}}}(\mathsf{idx}_1)\|\cdots\|\mathrm{AES128}_{\mathsf{seed}_{\mathbf{A}}}(\mathsf{idx}_M)$. We show that $\mathcal{C}^{\mathcal{G}}$ is indifferentiable from $\mathcal{F}$ as follows. Consider the two worlds with which $\mathcal{D}$ interacts to make queries on the construction $\mathcal{C}$ and $\mathcal{G}$:

- **REAL.** In the real world, upon query $\mathcal{C}(k)$, $\mathcal{D}$ receives $\mathrm{AES128}_k(\mathsf{idx}_1)\|\cdots\|\mathrm{AES128}_k(\mathsf{idx}_M)$. Queries to $\mathcal{G}$ are answered naturally with $\mathrm{AES128}_{(\cdot)}(\cdot)$ or $\mathrm{AES128}^{-1}_{(\cdot)}(\cdot)$ as required.
- **IDEAL.** In the ideal world, upon query $\mathcal{C}(k)$, the simulator $\mathcal{S}$ simulates $\mathcal{F}$ as follows. $\mathcal{S}$ samples M uniformly random strings $s_1, \ldots, s_M$ subject to no collisions and outputs $\mathcal{F}(k) = s_1\|\cdots\|s_M$. It additionally stores a mapping M_k from $\{\mathsf{idx}_1, \ldots, \mathsf{idx}_M\}$ to $S = \{s_1, \ldots, s_M\}$. These will be used to answer $\mathcal{G}$ queries. Without loss of generality, we assume that whenever $\mathcal{G}$ is queried on a key k, $\mathcal{S}$ pretends that $\mathcal{C}(k)$ has been queried and sets up M_k.
  $\mathcal{D}$ can now effectively simulate an ideal cipher $\mathcal{G}$ as follows. For forward queries with an input in Idx or backward queries with an input in S_k, $\mathcal{S}$ uses the mapping M_k to answer the query in a manner consistent with $\mathcal{C}(\cdot)$ simulation. For all other queries, the simulator maintains an on-the-fly table to simulate an ideal cipher. It samples independent uniformly random responses for each input query (forward or backward) subject to the fact that the resulting table of input/output pairs (x, y) combined with (idx_i, s_i) pairs remains a permutation over $\{0,1\}^{128}$ for every key k.

It is easy to see that the simulator is efficient. Indifferentiability of the two worlds follows by construction as $\mathrm{AES128}(\cdot,\cdot)$ is modeled as an ideal cipher. Thus, in generating $\mathbf{A}$ starting with a seed $\mathsf{seed}_{\mathbf{A}}$ using AES128, we can effectively replace the ideal domain extension primitive $\mathcal{F}$ with our construction in the ideal cipher model.

Using SHAKE128 **to generate A.** An argument in using SHAKE128 to expand $\mathsf{seed}_{\mathbf{A}}$ to the matrix $\mathbf{A}$ is significantly simpler. In the random oracle model, SHAKE128 is an ideal XOF [55]. In fact, for every distinct prefix str, we can model $\mathrm{SHAKE128}(str\|\cdot, \ell)$ as an independent hash function mapping $\{0,1\}^{128}$ to $\{0,1\}^{\ell}$.

The domain expansion step is constructed by computing SHAKE128($\langle i\rangle \| \mathsf{seed}_{\mathbf{A}}, 16n$) for $1 \leq i \leq n$ where $\langle i\rangle \in \{0,1\}^{16}$; each step fills up the ith row of the matrix $\mathbf{A}$. As each row is independently constructed via an ideal hash function, this construction maps a uniformly random seed $\mathsf{seed}_{\mathbf{A}}$ to a much larger uniformly random matrix $\mathbf{A}$ thereby implementing the ideal functionality $\mathcal{F}$ perfectly.

Reusing A. Finally, we point out that generating $\mathbf{A}$ from $\mathsf{seed}_{\mathbf{A}}$ can be a significant computational burden, but this cost can be amortized by relaxing the requirement that a fresh $\mathsf{seed}_{\mathbf{A}}$ be used for every instance of key encapsulation, e.g., by caching and reusing $\mathbf{A}$ for a small period of time. In this case, we observe that the cost of generating $\mathbf{A}$ represents roughly 40% of the cost of encapsulation and decapsulation on the targeted x64 Intel machine used in Section 3. A straightforward argument shows that the amortization above is compatible with all the security reductions in this section. But importantly, it now allows for an all-for-the-price-of-one attack against those key encapsulations that share the same $\mathbf{A}$. This can be mitigated by making sure that we cache and reuse $\mathbf{A}$ only for a small number of uses, but we need to do this in a very careful manner.

Generating A from joint randomness. It is also possible to generate $\mathbf{A}$ from joint randomness or using protocol random nonces. For example, when integrating FrodoKEM into the TLS protocol, $\mathbf{A}$ could be generated from a seed consisting of the random nonces `client_random` and `server_random` sent by the client and server in their `ClientHello` and `ServerHello` messages in the TLS handshake protocol. This functionality does not match the standard description of a KEM and the API provided by NIST, but is possible in general. A design with both parties contributing entropy to the seed might better protect against all-for-the-price-of-one attacks by being more robust to faulty random number generation at one of the parties.

5.1.4 IND-CPA security

In this section we show that FrodoPKE, using any error distribution χ and uniformly random $\mathbf{A}$, is an IND-CPA-secure public-key encryption scheme based on the hardness of the learning with errors decision problem with the same error distribution. We first tightly relate the IND-CPA security of FrodoPKE to the normal-form DLWE problem, where the secret coordinates have the same distribution as the errors.

Theorem 5.9 (normal-form DLWE $\implies$ IND-CPA security of FrodoPKE). *Let $n, q, \overline{m}, \overline{n}$ be positive integers, and χ be a probability distribution on $\mathbb{Z}$. There exist classical algorithms $\mathcal{B}_1, \mathcal{B}_2$ that use as a "black box" subroutine any (quantum or classical) algorithm $\mathcal{A}$ against the IND-CPA security of FrodoPKE (with a uniformly random $\mathbf{A}$), for which*

$$\mathrm{Adv}^{\mathsf{ind\text{-}cpa}}_{\mathsf{FrodoKEM}}(\mathcal{A}) \leq \overline{n} \cdot \mathrm{Adv}^{\mathsf{nf\text{-}dlwe}}_{n,n,q,\chi}(\mathcal{B}_1) + \overline{m} \cdot \mathrm{Adv}^{\mathsf{nf\text{-}dlwe}}_{n,n+\overline{n},q,\chi}(\mathcal{B}_2) \enspace .$$

The running times of $\mathcal{B}_1$ and $\mathcal{B}_2$ are approximately that of $\mathcal{A}$.

The proof of Theorem 5.9 is the same as that of [83, Theorem 3.2] or [26, Theorem 5.1].

The following theorem relates the LWE decision problem in its normal form to one where the secret is *uniformly random* over $\mathbb{Z}_q$. We need this only for connecting the latter variant, which arises in the reduction from worst-case lattice problems described in the next subsection, to the normal form as used in FrodoPKE. (In particular, our cryptanalysis and concrete security bounds are for the normal form.) The theorem is specialized to power-of-two modulus q (our case of interest), and the stated bounds in the advantage and number of LWE samples are more precise than those given in the original work. These bounds follow from the fact that, by a straightforward calculation, a uniformly random n-by-$(n+k)$ matrix over $\mathbb{Z}_q$ has an invertible n-by-n submatrix except with probability at most 2^{-k}.

Theorem 5.10 (uniform-secret DLWE $\implies$ normal-form DLWE; [15], Lemma 2). *Let n, m, k, q be positive integers with $q \geq 2$ a power of two, and let χ be a probability distribution on $\mathbb{Z}$. There exists a classical algorithm $\mathcal{B}$ that uses as a "black box" subroutine any (quantum or classical) algorithm $\mathcal{A}$ against the normal-form LWE decision problem, for which*

$$\mathrm{Adv}^{\mathsf{nf\text{-}dlwe}}_{n,m,q,\chi}(\mathcal{A}) \leq \mathrm{Adv}^{\mathsf{dlwe}}_{n,m+n+k,q,\chi}(\mathcal{B}) + 2^{-k} \enspace .$$

The running time of $\mathcal{B}$ is approximately that of $\mathcal{A}$.

5.1.5 Reductions from worst-case lattice problems

When choosing parameters for LWE, one needs to choose an error distribution, and in particular its "width." Certain choices (e.g., sufficiently wide Gaussians) are supported by *reductions* from worst-case lattice problems to LWE; see, e.g., [113, 97, 30, 103]. At a high level, such a reduction transforms any algorithm that solves LWE *on the average*—i.e., for random instances sampled according to the prescribed distribution—into an algorithm of related efficiency that solves *any instance* of certain lattice problems (not just random instances).

The original work of [113] and a follow-up work [103] gave quantum polynomial-time reductions, from the worst-case GapSVP_γ (Definition 2.15), SIVP_γ (Definition 2.16), and DGS_φ (Definition 2.17) problems on n-dimensional lattices, to n-dimensional LWE (for an unbounded polynomial $m = \mathrm{poly}(n)$ number of samples) with Gaussian error of standard deviation $\sigma \geq c\sqrt{n}$. The constant factor c was originally stated as $c = \sqrt{2/\pi}$, but can easily be improved to any $c > 1/(2\pi)$ via a tighter analysis of essentially the same proof.[5] However, for efficiency reasons our choices of σ (see Table 3) are somewhat smaller than required by these reductions.

Instead, following [113, Section 1.1], below we obtain an alternative *classical* (i.e., non-quantum) reduction from a variant of the worst-case bounded-distance decoding (BDD) problem to our LWE parameterizations. In contrast to the quantum reductions described above, which requires Gaussian error of standard deviation $\sigma \geq c\sqrt{n}$, the alternative reduction supports a smaller error width—as small as the "smoothing parameter" [92] of the lattice of integers $\mathbb{Z}$. For the BDD variant we consider, which we call "BDD with Discrete Gaussian Samples" ($\mathsf{BDDwDGS}$), the input additionally includes discrete Gaussian samples over the dual lattice, but having a larger width than known algorithms are able to exploit [85, 48]. Details follow.

Bounded-distance decoding with discrete Gaussian samples. We first define a variant of the bounded-distance decoding problem, which is implicit in prior works that consider "BDD with preprocessing," [2, 85, 48] and recall the relevant aspects of known algorithms for the problem.

Definition 5.11 (Bounded-distance decoding with discrete Gaussian samples). For a lattice $\mathcal{L} \subset \mathbb{R}^n$ and positive reals $d < \lambda_1(\mathcal{L})/2$ and $r > 0$, an instance of the *bounded-distance decoding with discrete Gaussian samples* problem $\mathsf{BDDwDGS}_{\mathcal{L},d,r}$ is a point $\mathbf{t} \in \mathbb{R}^n$ such that $\mathrm{dist}(\mathbf{t}, \mathcal{L}) \leq d$, and access to an oracle that samples from $D_{\mathcal{L}^*,s}$ for any (adaptively) queried $s \geq r$. The goal is to output the (unique) lattice point $\mathbf{v} \in \mathcal{L}$ closest to $\mathbf{t}$.

Remark. For a given distance bound d, known $\mathsf{BDDwDGS}$ algorithms use discrete Gaussian samples that all have the same width parameter s. However, the reduction to LWE will use the ability to vary s. Alternatively, we mention that when $r \geq \eta_\varepsilon(\mathcal{L}^*)$ for some very small $\varepsilon > 0$ (which will always be the case in our setting), we can replace the variable-width DGS oracle from Definition 5.11 with a fixed-width one that samples from $D_{\mathbf{w}+\mathcal{L}^*,r}$ for any queried coset $\mathbf{w} + \mathcal{L}^*$, always for the same width r. This is because we can use the latter oracle to implement the former one (up to statistical distance 8ε), by sampling $\mathbf{e}$ from the continuous Gaussian of parameter $\sqrt{s^2 - r^2}$ and then adding a sample from $D_{\mathcal{L}^*-\mathbf{e},r}$. See [99, Theorem 3.1] for further details.

The state-of-the-art algorithms for solving $\mathsf{BDDwDGS}$ [2, 85, 48] employ a certain $\mathcal{L}$-periodic function $f_{\mathcal{L},1/r} \colon \mathbb{R}^n \to [0, 1]$, defined as

$$f_{\mathcal{L},1/r}(\mathbf{x}) := \frac{\rho_{1/r}(\mathbf{x} + \mathcal{L})}{\rho_{1/r}(\mathcal{L})} = \mathop{\mathbb{E}}_{\mathbf{w}\sim D_{\mathcal{L}^*,r}}[\cos(2\pi\langle \mathbf{w}, \mathbf{x}\rangle)] \ , \tag{5}$$

where the equality on the right follows from the Fourier series of $f_{\mathcal{L},1/r}$ (see [2]). To solve $\mathsf{BDDwDGS}$ for a target point $\mathbf{t}$, the algorithms use several discrete Gaussian samples $\mathbf{w}_i \sim D_{\mathcal{L}^*,r}$ to estimate the value of $f_{\mathcal{L},1/r}$ at $\mathbf{t}$ and nearby points via Equation (5), to "hill climb" from $\mathbf{t}$ to the nearest lattice point. For the relevant points $\mathbf{t}$ we have the (very sharp) approximation

$$f_{\mathcal{L},1/r}(\mathbf{t}) \approx \exp(-\pi r^2 \cdot \mathrm{dist}(\mathbf{t}, \mathcal{L})^2) \ ,$$

[5] The approximation factor γ for GapSVP and SIVP is $\tilde{O}(qn/\sigma) = (qn/\sigma)\log^{O(1)} n$, and the parameter φ for DGS is $\Theta(q\sqrt{n}/\sigma)$ times the "smoothing parameter" of the lattice.

so by the Chernoff-Hoeffding bound, approximating $f_{\mathcal{L},1/r}(\mathbf{t})$ to within (say) a factor of two uses at least

$$\frac{1}{f_{\mathcal{L},1/r}(\mathbf{t})^2} \approx \exp(2\pi r^2 \cdot \mathrm{dist}(\mathbf{t},\mathcal{L})^2)$$

samples.[6] Note that without enough samples, the "signal" of $f_{\mathcal{L},1/r}(\mathbf{t})$ is overwhelmed by measurement "noise," which prevents the hill-climbing from making progress toward the answer.

In summary, when limited to N discrete Gaussian samples, the known approaches to solving $\mathsf{BDDwDGS}$ are limited to distance

$$\mathrm{dist}(\mathbf{t},\mathcal{L}) \leq r^{-1}\sqrt{\ln(N)/(2\pi)}\ . \tag{6}$$

Having such samples does not appear to provide any speedup in decoding at distances that are larger than this bound by some constant factor greater than one. In particular, if $d \cdot r \geq \omega(\sqrt{\log n})$ (which is the smoothing parameter of the integer lattice $\mathbb{Z}$ for negligible error ε), then having $N = \mathrm{poly}(n)$ samples does not seem to provide any help in solving $\mathsf{BDDwDGS}_{\mathcal{L},d,r}$ (versus having no samples at all).

Reduction from $\mathsf{BDDwDGS}$ to LWE. We now recall the following result from [103], which generalizes a key theorem from [113] to give a reduction from $\mathsf{BDDwDGS}$ to the LWE decision problem.

Theorem 5.12 ($\mathsf{BDDwDGS}$ hard $\implies$ decision-LWE hard [103, Lemma 5.4]). *Let $\varepsilon = \varepsilon(n)$ be a negligible function and let $m = poly(n)$ and $C = C(n) > 1$ be arbitrary. There is a probabilistic polynomial-time (classical) algorithm that, given access to an oracle that solves $\mathsf{DLWE}_{n,m,q,\alpha}$ with non-negligible advantage and input a number $\alpha \in (0,1)$, an integer $q \geq 2$, a lattice $\mathcal{L} \subset \mathbb{R}^n$, and a parameter $r \geq Cq \cdot \eta_\varepsilon(\mathcal{L}^*)$, solves $\mathsf{BDDwDGS}_{\mathcal{L},d,r}$ using $N = m \cdot poly(n)$ samples, where $d = \sqrt{1 - 1/C^2} \cdot \alpha q/r$.*

Remark. The above statement generalizes the fixed choice of $C = \sqrt{2}$ in the original statement (inherited from [113, Section 3.2.1]), using [113, Corollary 3.10]. In particular, for any constant $\delta > 0$ there is a constant $C > 1$ such that $d = (1-\delta) \cdot \alpha q/r$.

In particular, by Equation (6), if the Gaussian parameter αq of the LWE error sufficiently exceeds $\sqrt{\ln(N)/(2\pi)}$ (e.g., by a constant factor greater than one), then the $\mathsf{BDDwDGS}_{\mathcal{L},d,r}$ problem is plausibly hard (in the worst case), hence so is the corresponding LWE problem from Theorem 5.12 (on the average). An interesting direction is to obtain a more precise bound on, and improve, the "sample overhead" of the reduction, i.e., the $\mathrm{poly}(n)$ factor connecting the number of LWE samples m and the number of DGS samples N.

Concrete parameters. Concretely, for the extremely large bound $N = 2^{256}$ on the number of discrete Gaussian samples, the threshold for Gaussian parameters αq that conform to Theorem 5.12 is $\sqrt{\ln(N)/(2\pi)} \approx 5.314$, which corresponds to a standard deviation threshold of $\sqrt{\ln(N)}/(2\pi) \approx 2.120$. Our $\mathsf{FrodoPKE}$ parameters for security Levels 1 and 3, which use standard deviation $\sigma \geq 2.3$ (see Table 3), exceed this threshold by a comfortable margin. (Indeed, $\sigma = 2.3$ corresponds to $N \approx 2^{300}$.) For efficiency reasons, our parameters for security Level 5 use a somewhat smaller standard deviation of $\sigma = 1.4$; this corresponds to the very large bound $N \approx 2^{111}$. While this N is smaller than the running time for the Level 5 brute-force security level, we stress that these two quantities are not comparable; N is merely a bound on the *number of samples* provided in a $\mathsf{BDDwDGS}$ input, and it controls the decoding distance for known efficient algorithms.

5.2 Cryptanalytic attacks

In this section, we explain our methodology to estimate the security level of our proposed parameters. The methodology is similar to the one proposed in [13], with slight modifications taking into account the fact that some quasi-linear accelerations [118, 28] over sieving algorithms [18, 77] are not available without the ring structure.

[6]In fact, the algorithms need approximation factors much better than two, so the required number of samples is even larger by a sizable constant factor. However, the above crude bound will be sufficient for our purposes.

We also remark that this methodology is significantly more conservative than what is usually used in the literature [12], at least since recently. Indeed, we must acknowledge that lattice cryptanalysis is far less mature than that for factoring and computing discrete logarithms, for which the best-known attacks can more safely be considered best-possible attacks.

5.2.1 Methodology: the core-SVP hardness

In this section, let m_{samp} denote the number of LWE samples available to the attacker. Due to the small number of samples (i.e., $m_{\mathsf{samp}} \approx n$ in our schemes) we are not concerned with either BKW types of attacks [73] or linearization attacks [16]. This essentially leaves us with two BKZ [42] attacks, usually referred to as primal and dual attacks that we will briefly recall below.

Formally, BKZ with block-size b requires up to polynomially many calls to an SVP oracle in dimension b, but some heuristics allow to decrease the number of calls to be essentially linear [41]. To account for further improvement, we shall count only the cost of one such call to the SVP oracle: the core-SVP hardness. Such precaution is motivated by the fact that there are ways to amortize the cost of SVP calls inside BKZ, especially when sieving is to be used as the SVP oracle. Such a strategy was suggested in a talk, but has so far not been experimentally tested, as more implementation effort is required to integrate sieving within BKZ.

Even evaluating the concrete cost of one SVP oracle call in dimension b is difficult, because the numerically optimized pruned enumeration strategy does not yield a closed formula [59, 42]. Yet, asymptotically, enumeration is super-exponential (even with pruning), while sieving algorithms are exponential $2^{cb+o(b)}$ with a well understood constant c in the exponent. A sound and simple strategy is therefore to give a lower bound for the cost of an attack by 2^{cb} vector operations (i.e., about $b2^{cb}$ CPU cycles[7]), and to make sure that the block-size b is in a range where enumeration costs more than 2^{cb}. From the estimates of [42], it is argued in [13] that this is the case both classically and quantumly whenever $b \geq 200$.

The best known constant in the exponent for classical algorithms is $c_{\mathrm{C}} = \log_2 \sqrt{3/2} \approx 0.292$, as provided by the sieve algorithm of [18]. For quantum algorithms it is $c_{\mathrm{Q}} = \log_2 \sqrt{13/9} \approx 0.265$ [77, Sec. 14.2.10]. Because all variants of the sieve algorithm require building a list of $\sqrt{4/3}^b$ many vectors, the constant $c_{\mathrm{P}} = \log_2 \sqrt{4/3} \approx .2075$ can plausibly serve as a "worst-possible" lower bound for sieving algorithms.

Conservatism: lower bounds vs. experiments. These estimates are very conservative compared to the state of the art implementation of [87], which has practical complexity of about $2^{0.405b+11}$ cycles in the range $b = 60 \dots 80$. The classical lower bound of $2^{0.292b}$ corresponds to a margin factor of 2^{20} at blocksize $b = 80$, and this margin should continue increasing with the blocksize (abusing the linear fit suggests a margin of 2^{45} at blocksize $b = 300$).

Conservatism: future improvements. Of course, one could assume further improvements on known techniques. At least asymptotically, it may be reasonable to assume that $2^{0.292b+o(b)}$ is optimal for SVP considering that the underlying technique of [18] has been shown to reach lower bounds for the generic nearest-neighbor search problem [14]. As for concrete improvements, we note that this algorithm has already been subject to some fine-tuning in [87], so we may conclude that there is not much more to be gained without introducing new ideas. We therefore consider our margin sufficient to absorb such future improvements.

Conservatism: cost models. The NIST call for proposals suggested a particular cost model, inspired by the estimates of a Grover search attack on AES, essentially accounting for the quantum gate count. In comparison, the literature on sieving algorithms mostly focuses on analysis in the RAM model and quantumly accessible RAM models, and considers the amount of memory they use. Their cost in the area-time model should be higher by polynomial, if not exponential, factors.

Firstly, our model accounts for arithmetic operations rather than gates (used to compute inner products and evaluate norms of vectors). The conversion to gate count may not be trivial as it is unclear how many bits of precision are required.

[7] Because of the additional ring-structure, [13] chooses to ignore this factor b to the advantage of the adversary, assuming the techniques of [118, 28] can be adapted to more advanced sieve algorithms [13]. But for plain LWE, we can safely include this factor.

Secondly, even in the classical setting, the cost of sieving in large dimensions may not be accurately captured by the count of elementary operations in the RAM model, as those algorithms use an exponential amount of memory. Admittedly, the most basic sieve algorithm (with theoretical complexity $2^{0.415b+o(b)}$) has sequential memory access, and can therefore be efficiently implemented by a large circuit without memory access delays. But more advanced ones [18] have much less predictable memory access patterns, and memory complexities as large as time complexities ($2^{0.292b+o(b)}$). It is unclear if they can be adapted to reach a complexity $2^{0.292b+o(b)}$ in the area-time model; one might expect extra polynomial factors to appear. (Following an idea of [19], Becker et al. [18] also claim a version that only requires $2^{0.2015b+o(b)}$ memory, but we suspect this would come at some hidden cost on the running time.)

Moreover, the quantum versions of all sieving algorithms work in the quantumly accessible RAM model [79]. Again, the conversion to an efficient quantum circuit will induce extra costs—at least polynomial ones.

5.2.2 Primal attack

The primal attack consists of constructing a unique-SVP instance from the LWE problem and solving it using BKZ. We examine how large the block dimension b is required to be for BKZ to find the unique solution. Given the matrix LWE instance $(\mathbf{A}, \mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e})$ one builds the lattice $\Lambda = \{\mathbf{x} \in \mathbb{Z}^{m+n+1} : (\mathbf{A}|\mathbf{I}_m| - \mathbf{b})\mathbf{x} = 0 \bmod q\}$ of dimension $d = m + n + 1$, volume q^m, and with a unique-SVP solution $\mathbf{v} = (\mathbf{s}, \mathbf{e}, 1)$ of norm $\lambda \approx \sigma\sqrt{n+m}$. The number of used samples m may be chosen between 0 and m_{samp}, and we numerically optimize this choice.

Using the typical models of BKZ (geometric series assumption, Gaussian heuristic [41, 12]) one concludes that the primal attack is successful if and only if

$$\sigma\sqrt{b} \leq \delta^{2b-d-1} \cdot q^{m/d} \quad \text{where } \delta = ((\pi b)^{1/b} \cdot b/(2\pi e))^{1/(2b-2)} \ . \tag{7}$$

We note that this condition, introduced in [13], is substantially different from the one suggested in [58] and used in many previous security analyses, such as [12]. The recent study [11] showed that this new condition predicts significantly smaller security levels than the older, and is corroborated by extensive experiments.

5.2.3 Dual attack

The dual attack searches for a short nonzero vector in the lattice $\hat{\Lambda} = \{(\mathbf{v}, \mathbf{w}) \in \mathbb{Z}^{n+m} : \mathbf{v}^t = \mathbf{w}^t\mathbf{A} \pmod q\}$ of dimension $d = n + m$ and volume q^n, which is generated by the rows of the basis matrix

$$\mathbf{B} = \begin{pmatrix} q\mathbf{I}_n & \\ \bar{\mathbf{A}} & \mathbf{I}_m \end{pmatrix} \in \mathbb{Z}^{(n+m)\times(n+m)},$$

where each entry of $\bar{\mathbf{A}}$ is an arbitrary mod-q integer representative of the corresponding entry of $\mathbf{A}$. As above, the BKZ algorithm with block size b will output such a vector of length $\ell \approx \delta^{d-1}q^{n/d}$. The dual attack then uses this vector as a distinguisher for LWE, as described next.

For convenience we actually analyze the attack against a *continuous* form of LWE, with Gaussian $\mathbf{s}, \mathbf{e}$ over the reals $\mathbb{R}$. An instance of this problem has the form $(\mathbf{A}, \mathbf{b})$, where $\mathbf{b} \in (\mathbb{R}/q\mathbb{Z})^{n+m}$ either is uniformly random and independent of $\mathbf{A}$, or has the form

$$\mathbf{b} = \begin{pmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{pmatrix} = \mathbf{B} \cdot \begin{pmatrix} \mathbf{s} \\ \mathbf{e} \end{pmatrix} = \begin{pmatrix} q \cdot \mathbf{s} \\ \bar{\mathbf{A}}\mathbf{s} + \mathbf{e} \end{pmatrix} \bmod q\mathbb{Z}^{n+m},$$

where the entries of $(\mathbf{s}, \mathbf{e}) \in \mathbb{R}^{n+m}$ are independent continuous Gaussians of standard deviation σ. Because there is a trivial reduction from this LWE decision problem to the discrete one of interest that has *rounded Gaussian* $\mathbf{s}, \mathbf{e}$ (over $\mathbb{Z}$), the latter problem is no easier than the former.[8]

Having found some $(\mathbf{v}, \mathbf{w}) \in \hat{\Lambda}$ of length ℓ, the attacker computes

$$z = (\mathbf{v}^t, \mathbf{w}^t) \cdot \mathbf{B}^{-1} \cdot \mathbf{b} = q^{-1}(\mathbf{v}^t - \mathbf{w}^t\bar{\mathbf{A}}) \cdot \mathbf{b}_1 + \mathbf{w}^t \cdot \mathbf{b}_2 \bmod q,$$

[8] The reduction just replaces $\mathbf{b}$ with $\lfloor \mathbf{b}_2 - \bar{\mathbf{A}}(\mathbf{b}_1/q \bmod \mathbb{Z}^n) \rceil \in \mathbb{Z}_q^m$, where the mod operation returns the unique representative (i.e., fractional part) in $[-1/2, 1/2)^n$, and $\lfloor \cdot \rceil$ rounds to the nearest integer by subtracting the fractional part. This reduction maps the (continuous) uniform distribution to the (discrete) uniform distribution, and in the LWE case, subtracts $\bar{\mathbf{A}}$ times the fractional part of $\mathbf{s}$ and rounds away the fractional part of $\mathbf{e}$, yielding $\bar{\mathbf{A}}\lfloor \mathbf{s} \rceil + \lfloor \mathbf{e} \rceil \bmod q\mathbb{Z}^m$.

Table 10: **Primal and dual attacks on a single instance of an LWE problem.** Attack costs are given as the base-2 logarithm.

Scheme	Attack Mode	Classical	Quantum	Plausible
Frodo-640	Primal	150.8	137.6	109.6
	Dual	149.6	136.5	108.7
Frodo-976	Primal	216.0	196.7	156.0
	Dual	214.5	195.4	154.9
Frodo-1344	Primal	281.6	256.3	202.6
	Dual	279.8	254.7	201.4

and attempts to distinguish it from uniformly random over $\mathbb{R}/q\mathbb{Z}$. Its advantage in doing so can be bounded as follows. First, suppose that $\mathbf{b} = \mathbf{B} \cdot \left(\begin{smallmatrix}\mathbf{s}\\ \mathbf{e}\end{smallmatrix}\right)$ is from a continuous LWE instance as defined above. Then

$$z = (\mathbf{v}^t, \mathbf{w}^t) \cdot \begin{pmatrix}\mathbf{s}\\ \mathbf{e}\end{pmatrix} \bmod q$$

is distributed as a Gaussian of standard deviation $\ell\sigma$, modulo q. On the other hand, if $\mathbf{b}$ is uniformly random, then z is uniformly random in $\mathbb{R}/q\mathbb{Z}$. By the results of [92], these two distributions have statistical distance at most $\varepsilon = 2\delta$ for $\delta = \exp(-2\pi^2\tau^2) < 1/8$ where $\tau = \ell\sigma/q$, so this ε bounds the distinguishing advantage.

Because the value μ encrypted by the underlying FrodoPKE (using LWE) actually serves as a seed to pseudorandomly generate the FrodoKEM KEM key **ss** (see Algorithm 13), a small advantage ε—say, below $1/2$—in distinguishing μ from a uniform random string does not significantly decrease the brute-force search space. We therefore require the attacker to amplify its distinguishing advantage by obtaining about $1/\varepsilon^2$ short lattice vectors, which we can model (most favorably to the attacker) as being Gaussian distributed and independent. Because the lattice-sieve algorithms provide about $2^{.2075b}$ vectors, the dual attack must be repeated at least $R = \max(1, 1/(2^{.2075b}\varepsilon^2))$ times. (This view is also favorable to the attacker, as the other vectors output by the sieving algorithm are a bit larger than the shortest one.)

Primal and dual attacks for our suggested parameters are given in Table 10. The costs are listed for a single instance of the LWE problem. (Our ultimate security claims, such as those listed in Table 2, result from a series of reductions and thus are weaker.)

5.2.4 Beyond Core-SVP Hardness

At the time the core-SVP hardness measure was introduced by [13], the best implementations of sieving [18, 87] had performance significantly worse than the $2^{.292b}$ CPU cycles proposed as a conservative estimate by this methodology. This was due to substantial polynomial, or even sub-exponential, overheads hidden in the $2^{.292b+o(b)}$ complexity given in the analysis of [18]. Before [13], security estimates of lattice schemes were typically based on the cost of solving SVP via enumeration as given in [42, 12], leading to much more aggressive parameters. Beyond affecting the cost of SVP-calls, this methodology also introduced a different prediction of when BKZ solves LWE which was later confirmed by [11] and refined in [47].

While doubts were expressed to whether enumeration [42] with its super-exponential, yet practically smaller, costs would ever be outperformed by sieving for relevant cryptographic dimensions, significant progress on sieving algorithms [53, 7] has brought the cross-over point down to dimension about $b = 80$. In fact, the current SVP records are now held by algorithms that employ sieving.[9] These developments mandate a revision and refinement of security estimates for the FrodoKEM parameters, especially regarding classical attacks. In particular, while experiments indicated that, before those improvements were made, the costs hidden in the $o(b)$ were positive both in practice and asymptotically, the dimensions-for-free technique of [53] offers a sub-exponential speed-up, making it a priori unclear whether the total $o(b)$ term is positive or negative, both asymptotically and concretely.

[9] https://www.latticechallenge.org/svp-challenge/

We follow the same methodology as the one detailed in the Round-3 specification document of the Kyber key encapsulation mechanism [25] to count the number of gates required to solve the LWE problem. This analysis refines the core-SVP methodology in the following ways:

- It uses the probabilistic simulation of [47] rather than the GSA-intersect model of [13, 11] to determine the BKZ blocksize b for a successful attack. The required blocksize is somewhat larger (by an added term between 10 and 25 for our parameters), because of a "tail" phenomenon [123].
- It accounts for the "few dimensions for free" introduced in [53], which permits to solve SVP in dimension b while running sieving in a somewhat smaller dimension $b' = b - o(b)$.
- It relies on the concrete estimation for the gate cost of sieving from [10].
- It accounts for the number of calls to the SVP oracle.
- It dismisses the dual attack as realistically more expensive than the primal one, noting that the analysis of [13] (also used here) assumes that the sieve provides $2^{.208b}$ many vectors as short as the shortest one. But first, most of those vectors are larger by a factor of $\sqrt{4/3}$, and second, the trick of exploiting all of those vectors is not compatible with the "dimension-for-free" trick of [53].

The scripts for these refined estimates are provided in a git branch of the leaky-LWE-estimator of [47],[10] and lead to the estimates given in Table 11. We refer to the Kyber Round-3 specification document for the details of this analysis. We point out that it is paired with a detailed discussion of the "known unknowns", providing a plausible confidence interval for these estimates. For the classical hardness of the LWE problem at Levels 1 and 2, it is estimated in the Kyber Round-3 document that the true cost is no more than 16 bits away from this estimate, in either direction.

We also note that a similarly refined count of quantum gates seems to be essentially irrelevant: the work of [10] concluded that obtaining a quantum speed-up for sieving is rather tenuous, while the quantum security target for each level is significantly lower than the classical target.

Table 11: Refined estimates for the LWE hardness, where n is the optimal lattice dimension for the attack, b the BKZ blocksize and b' the sieving dimension accounting for "dimensions for free"

	n	b	b'	$\log_2$(gates)	$\log_2$(memory in bits)
Frodo-640	1297	496	453	175.1	110.4
Frodo-976	1969	724	668	240.0	155.8
Frodo-1344	2634	957	888	305.4	202.1

The refined analysis above covers recent improvements for solving the SVP and models the current state of the art. The resulting gate counts show that the FrodoKEM parameter sets comfortably match their respective target security levels with a large margin. While these numbers might encourage parameter modifications to more closely match security targets and improve performance and bandwidth, we prefer to leave parameter sets unchanged. This aligns with FrodoKEM's conservative design approach and hedges against improvements for cryptanalytic algorithms solving general lattice problems.

5.2.5 Decryption failures

The concrete FrodoPKE parameters induce a tiny probability of incorrect decryption (see Table 1), for honestly generated keys and ciphertexts. This is because a ciphertext may decrypt to a different message than the encrypted one, if the combination of the short error matrices in the key and the ciphertext is too large (see Section 2.2.7). This aspect of the scheme carries over to the transformed, CCA-targeting FrodoKEM, where incorrect decryption in the underlying PKE typically causes a decryption failure.

It has long been well understood that the ability to induce incorrect decryption or decryption failure in LWE-based schemes can leak information about the secret key, up to and including full key recovery (with sufficiently many failures). In brief, this is because such failures indicate some correlation between the secret key and the encryption randomness.

[10] https://github.com/lducas/leaky-LWE-Estimator/tree/NIST-round3

In the context of chosen-ciphertext attacks on FrodoKEM and similarly transformed schemes, the attacker can attempt to create ciphertexts whose underlying error matrices—which are derived pseudorandomly using an attacker-chosen seed—are atypically large. Such "weak" ciphertexts have an increased probability of inducing decryption failures when submitted to a decryption oracle. The process of searching for such ciphertexts, which can be done offline (without using a decryption oracle), is known as "failure boosting."

Recently, D'Anvers et al. [49] performed a detailed study of the complexity of failure-boosting attacks (in both the classical and quantum setting) against a variety of NIST candidates, including FrodoKEM. In summary, they found that our original Level 3 parameterization Frodo-976 suffered *no loss* in our claimed security (either classical or quantum) under such attacks. This is essentially because the cost of finding weak ciphertexts exceeds the benefit obtained from the corresponding increase in decryption failure probability.

Subsequently, we ran the scripts from [49] on the parameters for Frodo-640 (updated), Frodo-976, and Frodo-1344, and confirmed that applying the failure-boosting attack does not violate security Levels 1, 3, and 5, respectively. (Note that for Frodo-1344, failure boosting did not provide any improvement over the intrinsic failure probability of $2^{-252.5}$. We consider this to be consistent with the Level 5 requirement of 256 bits of brute-force security, because the overhead in using decryption failures to win the CCA security game exceeds 3.5 bits.)

6 Advantages and limitations

6.1 Ease of implementation

One of the features of FrodoKEM is that it is easy to implement and naturally facilitates writing implementations that are compact and run in constant-time. This latter feature aids to avoid common cryptographic implementation mistakes which can lead to key-extraction based on, for instance, timing differences when executing the code.

Nonetheless, care must be taken to avoid timing leaks. In 2020, Guo, Johansson, and Nilsson [66] demonstrated a key-recovery attack on the reference implementation in the Round 2 submission of FrodoKEM by exploiting branching in the computation of $\mathbf{ss}'_0$ and $\mathbf{ss}'_1$ in FrodoKEM.Decaps. This attack can be avoided by ensuring the implementation reads both $\mathbf{k}'$ and $\mathbf{s}$, compares $\mathbf{B}'\|\mathbf{C}$ and $\mathbf{B}''\|\mathbf{C}'$ in a constant-time way that avoids early termination, and sets $\overline{\mathbf{k}}$ using data-independent evaluation. The Round 3 reference implementation provides an example of one such implementation.

The additional x64 implementation of the full KEM scheme accompanying this submission consists of slightly more than 250 lines of plain C code.[11] This same code is used for all three security levels to implement FrodoKEM-640, FrodoKEM-976, and FrodoKEM-1344, with parameters changed by a small number of macros at compile-time.

Computing on matrices —the basic operation in FrodoKEM— allows for easy scaling to different dimensions n. In addition, FrodoKEM uses a modulus q that is always equal or less than 2^{16}. These two combined aspects allow for the full reuse of the matrix functions for the different security levels by instantiating them with the right parameters at build time. Since the modulus q used is always a power of two, implementing arithmetic modulo q is simple, efficient and ease to do in constant-time in modern computer architectures: for instance, computing modulo 2^{16} comes for free when using 16-bit data-types. Moreover, the dimension values were chosen to be divisible by 16 in order to facilitate vectorization optimizations and to simplify the use of AES128 for the generation of the matrix $\mathbf{A}$.

Also the error sampling is designed to be simple and facilitates code reuse: for any security level, FrodoKEM requires 16 bits per sample, and the tables T_χ corresponding to the discrete cumulative density functions always consist of values that are less than 2^{15}. Hence, a simple function applying inversion sampling (see Algorithm 5) can be instantiated using different precomputed tables T_χ. Moreover, due to the small sizes of these pre-computed tables constant-time table lookups, needed to protect against attacks based on timing differences, can be implemented almost for free in terms of effort and performance impact.

6.2 Compatibility with existing deployments and hybrid schemes

FrodoKEM-640, FrodoKEM-976, and FrodoKEM-1344 do have larger public key / encapsulation sizes than traditional RSA and elliptic curve cryptosystems, and some other post-quantum candidates such as ring-LWE-based schemes. Nonetheless, their communication sizes are sufficiently small that they are still compatible with many existing deployments. In our original research paper on FrodoCCS [26], we integrated FrodoCCS as well as several other key encapsulation mechanisms into OpenSSL v1.0.1f and added ciphersuites, both hybrid and non-hybrid, to the TLS 1.2 implementation in OpenSSL. We compiled the Apache httpd v2.4.20 web server against our modified OpenSSL, and tested compatibility and performance of the web server. We encountered no problems with existing clients despite using larger ephemeral public keys / encapsulations, and did not need to make any modifications to data structures (e.g., existing 16-bit length fields were large enough to hold our values).

We measured throughput (connections per second) for a variety of page sizes, and latency (connection establishment time) for a server with or without heavy load, of both hybrid and non-hybrid ciphersuites. Detailed results including the exact methodology can be found in [26]. To highlight a few results: the connection time of an ECDHE (nistp256) ciphersuite with an RSA certificate on an unloaded server was 16.1 milliseconds (over a network with ping time 0.62 ms); it was 20.7 ms for FrodoCCS, and 24.5 ms for hybrid FrodoCCS+ECDHE[12]. The number of connections (with 1 KiB HTTP payload) supported per second with

[11] This count does not include header files and the additional symmetric primitives.

[12] Note that the results in [26] use a different parameter set than in this proposal which had slightly larger communication (22.1 KiB in [26] versus 18.9 KiB for FrodoKEM-640 in this proposal; the IND-CCA-secure FrodoKEM-640 in this proposal has an additional runtime cost in decapsulation due to the application of the FO transform compared to the IND-CPA-secure scheme in

an ECDHE ciphersuite with an RSA certificate was 810, compared to 700 for FrodoCCS and 551 for hybrid FrodoCCS+ECDHE. These results indicate that, despite their larger communication sizes, FrodoKEM remains practical for Internet applications.

In our experience with testing the performance of the original Frodo construction in an end-to-end testbed OpenSSL deployment, we observed a few trends that let us extrapolate these results to our current proposal. First, we note that even with the significantly larger bandwidth of the original FrodoCCS proposal, as compared to the original NewHope proposal, we observed a slowdown of less than $1.6\times$ when comparing connection times for 1 KiB webpages. This slowdown factor only decreases with increasing sizes of webpages and considering our smaller bandwidth (18.9 KiB for FrodoKEM-640 versus 22.1 KiB for the original FrodoCCS construction) we expect to be competitive for typical connection sizes.

Moreover, we can state with some measure of confidence that the additional costs when applying the FO transform will have a very small impact on the connection throughput as well as on the connection times. We state this with two supporting arguments. First, with a microbenchmark a whole order of magnitude faster than the original FrodoCCS construction, the original NewHope construction only improves connection times and throughputs by 30–50% and we expect various other bottlenecks in the entire Web serving ecosystem to have a larger impact. To compare, our FO-transformed implementations run in time a small constant factor larger than the microbenchmarks of FrodoCCS. Second—and as stated previously to support the practical application of the original FrodoCCS construction [26]—deployments in the near-term will necessarily involve both a post-quantum and a traditional EC-based construction which would result in any drastic improvements in post-quantum microbenchmarks having a small or even negligible impact in practical deployments. The costs of these small impacts are well worth the long-term post-quantum security afforded by a conservative scheme based only on generic lattices.

6.3 Hardware implementations

Hardware implementations of lattice-based cryptographic schemes have mainly considered the ring learning with errors based schemes (see, e.g., [65, 107, 108, 117]) since these schemes allow to compute polynomial multiplication with the number-theoretic transform, e.g. the discrete Fourier transform over a finite field. Computing the fast Fourier transform (FFT) is a well-known primitive for hardware implementations.

Schemes based on the original learning with errors problem work with matrices instead. Fortunately, the FPGA design and implementation of, for instance, matrix multiplication architectures is a well-studied area and very efficient (in terms of either area, energy or performance) implementations are known (cf. [110] and the related literature mentioned therein). Hence, the proposed schemes FrodoKEM-640, FrodoKEM-976, and FrodoKEM-1344 are a natural fit for hardware implementations.

6.4 Side-channel resistance

Side-channel attacks are a family of attacks which use meta-information such as power consumption (e.g, in a differential power analysis (DPA) attack [76]) or electromagnetic usage (e.g., in a differential electromagnetic analysis (DEMA) attack [60]) in a statistical analysis by correlating this information obtained when executing a cryptographic primitive to a key-dependent guess. Besides such passive side-channel attacks (cf. [75]) there are also active attacks which might *inject faults* [23, 21] and use the potentially corrupted output to obtain information about the secret key used.

This is a well-studied and active research area used to protect software and hardware implementations where such attacks are realistic. In the setting of implementations based on the ring LWE problems not much work has been done yet. For ring LWE masking techniques [36] have been studied to protect implementations such as in [95, 115, 116].

In a more recent work [109] it is shown how to perform a single trace attack on ring LWE encryption using side-channel template matching [37]. Hence, it can also be applied to attack masked implementations. This single trace behaviour makes it immediately applicable to key-exchange algorithms.

No side-channel attacks nor countermeasures are currently known for LWE key encapsulation mechanisms but the generic attacks methods as well as the countermeasures which apply to ring LWE also do apply to LWE. However, since our LWE-based schemes do not use FFT-based multiplication techniques (the point

[26]; and used somewhat different symmetric primitives. Nonetheless the results provide some indication of suitability.

of attack used in [109]), the attack surface against FrodoKEM is significantly reduced. This might result in cheap and easy-to-apply countermeasures against a large set of the known side-channel attacks applied in practice.

References

[1] D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J. A. Halderman, N. Heninger, D. Springall, E. Thomé, L. Valenta, B. VanderSloot, E. Wustrow, S. Zanella-Béguelin, and P. Zimmermann. Imperfect forward secrecy: How Diffie-Hellman fails in practice. In I. Ray, N. Li, and C. Kruegel, editors, *ACM CCS 2015: 22nd Conference on Computer and Communications Security*, pages 5–17. ACM Press, Oct. 2015.

[2] D. Aharonov and O. Regev. Lattice problems in NP $\cap$ coNP. *Journal of the ACM*, 52(5):749–765, 2005. Preliminary version in FOCS 2004.

[3] M. Ajtai. Generating hard instances of lattice problems (extended abstract). In *28th Annual ACM Symposium on Theory of Computing*, pages 99–108. ACM Press, May 1996.

[4] M. Ajtai and C. Dwork. A public-key cryptosystem with worst-case/average-case equivalence. In *29th Annual ACM Symposium on Theory of Computing*, pages 284–293. ACM Press, May 1997.

[5] M. R. Albrecht. On dual lattice attacks against small-secret LWE and parameter choices in HElib and SEAL. In J.-S. Coron and J. B. Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017, Part II*, volume 10211 of *Lecture Notes in Computer Science*, pages 103–129. Springer, Heidelberg, Apr. / May 2017.

[6] M. R. Albrecht, C. Cid, J.-C. Faugère, and L. Perret. Algebraic algorithms for LWE. Cryptology ePrint Archive, Report 2014/1018, 2014. http://eprint.iacr.org/2014/1018.

[7] M. R. Albrecht, L. Ducas, G. Herold, E. Kirshanova, E. W. Postlethwaite, and M. Stevens. The general sieve kernel and new records in lattice reduction. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 717–746. Springer, 2019.

[8] M. R. Albrecht, J.-C. Faugère, R. Fitzpatrick, and L. Perret. Lazy modulus switching for the BKW algorithm on LWE. In H. Krawczyk, editor, *PKC 2014: 17th International Conference on Theory and Practice of Public Key Cryptography*, volume 8383 of *Lecture Notes in Computer Science*, pages 429–445. Springer, Heidelberg, Mar. 2014.

[9] M. R. Albrecht, R. Fitzpatrick, and F. Göpfert. On the efficacy of solving LWE by reduction to unique-SVP. In H.-S. Lee and D.-G. Han, editors, *ICISC 13: 16th International Conference on Information Security and Cryptology*, volume 8565 of *Lecture Notes in Computer Science*, pages 293–310. Springer, Heidelberg, Nov. 2014.

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

[11] M. R. Albrecht, F. Göpfert, F. Virdia, and T. Wunderer. Revisiting the expected cost of solving uSVP and applications to LWE. In T. Takagi and T. Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017, Part I*, volume 10624 of *Lecture Notes in Computer Science*, pages 297–322. Springer, Heidelberg, Dec. 2017.

[12] M. R. Albrecht, R. Player, and S. Scott. On the concrete hardness of Learning with Errors. *Journal of Mathematical Cryptology*, 9(3):169–203, Nov 2015.

[13] E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe. Post-quantum key exchange - A new hope. In T. Holz and S. Savage, editors, *USENIX Security 2016: 25th USENIX Security Symposium*, pages 327–343. USENIX Association, Aug. 2016.

[14] A. Andoni, T. Laarhoven, I. P. Razenshteyn, and E. Waingarten. Optimal hashing-based time-space trade-offs for approximate near neighbors. In P. N. Klein, editor, *28th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 47–66. ACM-SIAM, Jan. 2017.

[15] B. Applebaum, D. Cash, C. Peikert, and A. Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In S. Halevi, editor, *Advances in Cryptology – CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 595–618. Springer, Heidelberg, Aug. 2009.

[16] S. Arora and R. Ge. New algorithms for learning in presence of errors. In L. Aceto, M. Henzinger, and J. Sgall, editors, *ICALP 2011: 38th International Colloquium on Automata, Languages and Programming, Part I*, volume 6755 of *Lecture Notes in Computer Science*, pages 403–415. Springer, Heidelberg, July 2011.

[17] S. Bai, A. Langlois, T. Lepoint, D. Stehlé, and R. Steinfeld. Improved security proofs in lattice-based cryptography: Using the Rényi divergence rather than the statistical distance. In T. Iwata and J. H. Cheon, editors, *Advances in Cryptology – ASIACRYPT 2015, Part I*, volume 9452 of *Lecture Notes in Computer Science*, pages 3–24. Springer, Heidelberg, Nov. / Dec. 2015.

[18] A. Becker, L. Ducas, N. Gama, and T. Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In R. Krauthgamer, editor, *27th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 10–24. ACM-SIAM, Jan. 2016.

[19] A. Becker, N. Gama, and A. Joux. Speeding-up lattice sieving without increasing the memory, using sub-quadratic nearest neighbor search. Cryptology ePrint Archive, Report 2015/522, 2015. http://eprint.iacr.org/2015/522.

[20] D. J. Bernstein, T. Chou, C. Chuengsatiansup, A. Hülsing, E. Lambooij, T. Lange, R. Niederhagen, and C. van Vredendaal. How to manipulate curve standards: A white paper for the black hat. In L. Chen and S. Matsuo, editors, *Security Standardisation Research (SSR) 2015*, volume 9497 of *Lecture Notes in Computer Science*, pages 109–139. Springer, 2015.

[21] E. Biham and A. Shamir. Differential fault analysis of secret key cryptosystems. In B. S. Kaliski Jr., editor, *Advances in Cryptology – CRYPTO'97*, volume 1294 of *Lecture Notes in Computer Science*, pages 513–525. Springer, Heidelberg, Aug. 1997.

[22] A. Blum, M. L. Furst, M. J. Kearns, and R. J. Lipton. Cryptographic primitives based on hard learning problems. In D. R. Stinson, editor, *Advances in Cryptology – CRYPTO'93*, volume 773 of *Lecture Notes in Computer Science*, pages 278–291. Springer, Heidelberg, Aug. 1994.

[23] D. Boneh, R. A. DeMillo, and R. J. Lipton. On the importance of checking cryptographic protocols for faults (extended abstract). In W. Fumy, editor, *Advances in Cryptology – EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51. Springer, Heidelberg, May 1997.

[24] D. Boneh, C. Gentry, S. Gorbunov, S. Halevi, V. Nikolaenko, G. Segev, V. Vaikuntanathan, and D. Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In P. Q. Nguyen and E. Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 533–556. Springer, Heidelberg, May 2014.

[25] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé. CRYSTALS — Kyber: A CCA-secure module-lattice-based KEM. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 353–367, Apr. 2018.

[26] J. W. Bos, C. Costello, L. Ducas, I. Mironov, M. Naehrig, V. Nikolaenko, A. Raghunathan, and D. Stebila. Frodo: Take off the ring! Practical, quantum-secure key exchange from LWE. In E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, editors, *ACM CCS 2016: 23rd Conference on Computer and Communications Security*, pages 1006–1018. ACM Press, Oct. 2016.

[27] J. W. Bos, C. Costello, M. Naehrig, and D. Stebila. Post-quantum key exchange for the TLS protocol from the ring learning with errors problem. In *2015 IEEE Symposium on Security and Privacy*, pages 553–570. IEEE Computer Society Press, May 2015.

[28] J. W. Bos, M. Naehrig, and J. van de Pol. *Int. J. of Applied Cryptography*, 3(4):313–329, Jan. 2017.

[29] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computer Theory*, 6(3):13, 2014. Preliminary version in ITCS 2012.

[30] Z. Brakerski, A. Langlois, C. Peikert, O. Regev, and D. Stehlé. Classical hardness of learning with errors. In D. Boneh, T. Roughgarden, and J. Feigenbaum, editors, *45th Annual ACM Symposium on Theory of Computing*, pages 575–584. ACM Press, June 2013.

[31] Z. Brakerski and V. Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In R. Ostrovsky, editor, *52nd Annual Symposium on Foundations of Computer Science*, pages 97–106. IEEE Computer Society Press, Oct. 2011.

[32] J. Cai and A. Nerurkar. An improved worst-case to average-case connection for lattice problems. In *38th Annual Symposium on Foundations of Computer Science*, pages 468–477. IEEE Computer Society Press, Oct. 1997.

[33] P. Campbell, M. Groves, and D. Shepherd. Soliloquy: a cautionary tale. ETSI 2nd Quantum-Safe Crypto Workshop, 2014. http://docbox.etsi.org/Workshop/2014/201410_CRYPTO/S07_Systems_and_Attacks/S07_Groves_Annex.pdf.

[34] D. Cash, D. Hofheinz, E. Kiltz, and C. Peikert. Bonsai trees, or how to delegate a lattice basis. *Journal of Cryptology*, 25(4):601–639, Oct. 2012.

[35] W. Castryck, I. Iliashenko, and F. Vercauteren. Provably weak instances of ring-LWE revisited. In M. Fischlin and J.-S. Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part I*, volume 9665 of *Lecture Notes in Computer Science*, pages 147–167. Springer, Heidelberg, May 2016.

[36] S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi. Towards sound approaches to counteract power-analysis attacks. In M. J. Wiener, editor, *Advances in Cryptology – CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, Heidelberg, Aug. 1999.

[37] S. Chari, J. R. Rao, and P. Rohatgi. Template attacks. In B. S. Kaliski Jr., Çetin Kaya. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 13–28. Springer, Heidelberg, Aug. 2003.

[38] S. Chatterjee, N. Koblitz, A. Menezes, and P. Sarkar. Another look at tightness II: Practical issues in cryptography. In *Paradigms in Cryptology – Mycrypt 2016*, volume 10311 of *Lecture Notes in Computer Science*, pages 21–25. Springer, Heidelberg, 2017.

[39] H. Chen, K. Lauter, and K. E. Stange. Attacks on the search RLWE problem with small errors. *SIAM J. on Applied Algebra and Geometry*, 1(1):665–682, 2017.

[40] H. Chen, K. E. Lauter, and K. E. Stange. Security considerations for galois non-dual RLWE families. In R. Avanzi and H. M. Heys, editors, *SAC 2016: 23rd Annual International Workshop on Selected Areas in Cryptography*, volume 10532 of *Lecture Notes in Computer Science*, pages 443–462. Springer, Heidelberg, Aug. 2016.

[41] Y. Chen. *Lattice reduction and concrete security of fully homomorphic encryption.* PhD thesis, Université Paris Diderot, 2013.

[42] Y. Chen and P. Q. Nguyen. BKZ 2.0: Better lattice security estimates. In D. H. Lee and X. Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 1–20. Springer, Heidelberg, Dec. 2011.

[43] J.-S. Coron, Y. Dodis, C. Malinaud, and P. Puniya. Merkle-Damgård revisited: How to construct a hash function. In V. Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 430–448. Springer, Heidelberg, Aug. 2005.

[44] R. Cramer, L. Ducas, C. Peikert, and O. Regev. Recovering short generators of principal ideals in cyclotomic rings. In M. Fischlin and J.-S. Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 559–585. Springer, Heidelberg, May 2016.

[45] R. Cramer, L. Ducas, and B. Wesolowski. Short stickelberger class relations and application to ideal-SVP. In J.-S. Coron and J. B. Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017, Part I*, volume 10210 of *Lecture Notes in Computer Science*, pages 324–348. Springer, Heidelberg, Apr. / May 2017.

[46] R. Cramer and V. Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal on Computing*, 33(1):167–226, 2003.

[47] D. Dachman-Soled, L. Ducas, H. Gong, and M. Rossi. LWE with side information: Attacks and concrete security estimation. 2020. https://eprint.iacr.org/2020/292.pdf.

[48] D. Dadush, O. Regev, and N. Stephens-Davidowitz. On the closest vector problem with a distance guarantee. In *IEEE Conference on Computational Complexity*, pages 98–109, 2014.

[49] J.-P. D'Anvers, Q. Guo, T. Johansson, A. Nilsson, F. Vercauteren, and I. Verbauwhede. Decryption failure attacks on IND-CCA secure lattice-based schemes. In D. Lin and K. Sako, editors, *PKC 2019: 22nd International Conference on Theory and Practice of Public Key Cryptography, Part II*, volume 11443 of *Lecture Notes in Computer Science*, pages 565–598. Springer, Heidelberg, Apr. 2019.

[50] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[51] J. Ding, X. Xie, and X. Lin. A simple provably secure key exchange scheme based on the learning with errors problem. Cryptology ePrint Archive, Report 2012/688, 2012. http://eprint.iacr.org/2012/688.

[52] N. Döttling and J. Müller-Quade. Lossy codes and a new variant of the learning-with-errors problem. In T. Johansson and P. Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 18–34. Springer, Heidelberg, May 2013.

[53] L. Ducas. Shortest vector from lattice sieving: a few dimensions for free. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 125–145. Springer, 2018.

[54] L. Ducas, M. Plançon, and B. Wesolowski. On the shortness of vectors to be found by the Ideal-SVP quantum algorithm. Cryptology ePrint Archive, Report 2019/234, 2019. https://eprint.iacr.org/2019/234.

[55] M. J. Dworkin. SHA-3 standard: Permutation-based hash and extendable-output functions. Federal Information Processing Standards (FIPS) 202, National Institute of Standards and Technology, Aug. 2015.

[56] Y. Elias, K. E. Lauter, E. Ozman, and K. E. Stange. Provably weak instances of ring-LWE. In R. Gennaro and M. J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 63–92. Springer, Heidelberg, Aug. 2015.

[57] E. Fujisaki and T. Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In M. J. Wiener, editor, *Advances in Cryptology – CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 537–554. Springer, Heidelberg, Aug. 1999.

[58] N. Gama and P. Q. Nguyen. Predicting lattice reduction. In N. P. Smart, editor, *Advances in Cryptology – EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 31–51. Springer, Heidelberg, Apr. 2008.

[59] N. Gama, P. Q. Nguyen, and O. Regev. Lattice enumeration using extreme pruning. In H. Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 257–278. Springer, Heidelberg, May / June 2010.

[60] K. Gandolfi, C. Mourtel, and F. Olivier. Electromagnetic analysis: Concrete results. In Çetin Kaya. Koç, D. Naccache, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*, pages 251–261. Springer, Heidelberg, May 2001.

[61] C. Gentry, C. Peikert, and V. Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In R. E. Ladner and C. Dwork, editors, *40th Annual ACM Symposium on Theory of Computing*, pages 197–206. ACM Press, May 2008.

[62] C. Gentry, A. Sahai, and B. Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In R. Canetti and J. A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 75–92. Springer, Heidelberg, Aug. 2013.

[63] O. Goldreich, S. Goldwasser, and S. Halevi. Collision-free hashing from lattice problems. Cryptology ePrint Archive, Report 1996/009, 1996. http://eprint.iacr.org/1996/009.

[64] S. Gorbunov, V. Vaikuntanathan, and H. Wee. Predicate encryption for circuits from LWE. In R. Gennaro and M. J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 503–523. Springer, Heidelberg, Aug. 2015.

[65] N. Göttert, T. Feller, M. Schneider, J. Buchmann, and S. A. Huss. On the design of hardware building blocks for modern lattice-based encryption schemes. In E. Prouff and P. Schaumont, editors, *Cryptographic Hardware and Embedded Systems – CHES 2012*, volume 7428 of *Lecture Notes in Computer Science*, pages 512–529. Springer, Heidelberg, Sept. 2012.

[66] Q. Guo, T. Johansson, and A. Nilsson. A key-recovery timing attack on post-quantum primitives using the Fujisaki-Okamoto transformation and its application on FrodoKEM. In D. Micciancio and T. Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part II*, volume 12171 of *Lecture Notes in Computer Science*, pages 359–386. Springer, Heidelberg, Aug. 2020.

[67] J. Hoffstein, J. Pipher, and J. H. Silverman. NTRU: A ring-based public key cryptosystem. In J. P. Buhler, editor, *Algorithmic Number Theory, Third International Symposium*, pages 267–288, 1998.

[68] D. Hofheinz, K. Hövelmanns, and E. Kiltz. A modular analysis of the Fujisaki-Okamoto transformation. In Y. Kalai and L. Reyzin, editors, *TCC 2017: 15th Theory of Cryptography Conference, Part I*, volume 10677 of *Lecture Notes in Computer Science*, pages 341–371. Springer, Heidelberg, Nov. 2017.

[69] T. Ishiguro, S. Kiyomoto, Y. Miyake, and T. Takagi. Parallel gauss sieve algorithm: Solving the SVP challenge over a 128-dimensional ideal lattice. In H. Krawczyk, editor, *PKC 2014: 17th International Conference on Theory and Practice of Public Key Cryptography*, volume 8383 of *Lecture Notes in Computer Science*, pages 411–428. Springer, Heidelberg, Mar. 2014.

[70] H. Jiang, Z. Zhang, L. Chen, H. Wang, and Z. Ma. IND-CCA-secure key encapsulation mechanism in the quantum random oracle model, revisited. In H. Shacham and A. Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part III*, volume 10993 of *Lecture Notes in Computer Science*, pages 96–125. Springer, Heidelberg, Aug. 2018.

[71] H. Jiang, Z. Zhang, and Z. Ma. Tighter security proofs for generic key encapsulation mechanism in the quantum random oracle model. Cryptology ePrint Archive, Report 2019/134, 2019. https://eprint.iacr.org/2019/134.

[72] A. Kawachi, K. Tanaka, and K. Xagawa. Multi-bit cryptosystems based on lattice problems. In T. Okamoto and X. Wang, editors, *PKC 2007: 10th International Conference on Theory and Practice of Public Key Cryptography*, volume 4450 of *Lecture Notes in Computer Science*, pages 315–329. Springer, Heidelberg, Apr. 2007.

[73] P. Kirchner and P.-A. Fouque. An improved BKW algorithm for LWE with applications to cryptography and lattices. In R. Gennaro and M. J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 43–62. Springer, Heidelberg, Aug. 2015.

[74] P. Kirchner and P.-A. Fouque. Revisiting lattice attacks on overstretched NTRU parameters. In J.-S. Coron and J. B. Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017, Part I*, volume 10210 of *Lecture Notes in Computer Science*, pages 3–26. Springer, Heidelberg, Apr. / May 2017.

[75] P. C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In N. Koblitz, editor, *Advances in Cryptology – CRYPTO'96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, Heidelberg, Aug. 1996.

[76] P. C. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In M. J. Wiener, editor, *Advances in Cryptology – CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, Heidelberg, Aug. 1999.

[77] T. Laarhoven. *Search problems in cryptography.* PhD thesis, Eindhoven University of Technology, 2015.

[78] T. Laarhoven. Sieving for shortest vectors in lattices using angular locality-sensitive hashing. In R. Gennaro and M. J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 3–22. Springer, Heidelberg, Aug. 2015.

[79] T. Laarhoven, M. Mosca, and J. van de Pol. Finding shortest lattice vectors faster using quantum search. *Designs, Codes and Cryptography*, 77(2-3):375–400, 2015.

[80] A. Langlois and D. Stehlé. Worst-case to average-case reductions for module lattices. *Designs, Codes and Cryptography*, 75(3):565–599, 2015.

[81] A. Langlois, D. Stehlé, and R. Steinfeld. GGHLite: More efficient multilinear maps from ideal lattices. In P. Q. Nguyen and E. Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 239–256. Springer, Heidelberg, May 2014.

[82] A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, December 1982.

[83] R. Lindner and C. Peikert. Better key sizes (and attacks) for LWE-based encryption. In A. Kiayias, editor, *Topics in Cryptology – CT-RSA 2011*, volume 6558 of *Lecture Notes in Computer Science*, pages 319–339. Springer, Heidelberg, Feb. 2011.

[84] M. Liu and P. Q. Nguyen. Solving BDD by enumeration: An update. In E. Dawson, editor, *Topics in Cryptology – CT-RSA 2013*, volume 7779 of *Lecture Notes in Computer Science*, pages 293–309. Springer, Heidelberg, Feb. / Mar. 2013.

[85] Y.-K. Liu, V. Lyubashevsky, and D. Micciancio. On bounded distance decoding for general lattices. In *APPROX-RANDOM*, volume 4110 of *Lecture Notes in Computer Science*, pages 450–461. Springer, Heidelberg, 2006.

[86] V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. *Journal of the ACM*, 60(6):43:1–43:35, November 2013. Preliminary version in Eurocrypt 2010.

[87] A. Mariano, T. Laarhoven, and C. Bischof. A parallel variant of LDSieve for the SVP on lattices. In *25th Euromicro Int. Conf. on Parallel, Distributed and Network-based Processing (PDP)*, pages 23–30. IEEE, 2017.

[88] U. M. Maurer, R. Renner, and C. Holenstein. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In M. Naor, editor, *TCC 2004: 1st Theory of Cryptography Conference*, volume 2951 of *Lecture Notes in Computer Science*, pages 21–39. Springer, Heidelberg, Feb. 2004.

[89] D. Micciancio. Improved cryptographic hash functions with worst-case/average-case connection. In *34th Annual ACM Symposium on Theory of Computing*, pages 609–618. ACM Press, May 2002.

[90] D. Micciancio. Cryptographic functions from worst-case complexity assumptions. Information Security and Cryptography, pages 427–452. Springer, Heidelberg, 2010.

[91] D. Micciancio and C. Peikert. Hardness of SIS and LWE with small parameters. In R. Canetti and J. A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 21–39. Springer, Heidelberg, Aug. 2013.

[92] D. Micciancio and O. Regev. Worst-case to average-case reductions based on Gaussian measures. *SIAM J. Comput.*, 37(1):267–302, 2007. Preliminary version in FOCS 2004.

[93] D. Micciancio and O. Regev. Lattice-based cryptography. In *Post Quantum Cryptography*, pages 147–191. Springer, February 2009.

[94] P. Q. Nguyen and O. Regev. Learning a parallelepiped: Cryptanalysis of GGH and NTRU signatures. *Journal of Cryptology*, 22(2):139–160, Apr. 2009.

[95] T. Oder, T. Schneider, T. Pöppelmann, and T. Güneysu. Practical CCA2-secure masked Ring-LWE implementations. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(1):142–174, 2018. https://tches.iacr.org/index.php/TCHES/article/view/836.

[96] T. Okamoto and D. Pointcheval. REACT: Rapid Enhanced-security Asymmetric Cryptosystem Transform. In D. Naccache, editor, *Topics in Cryptology – CT-RSA 2001*, volume 2020 of *Lecture Notes in Computer Science*, pages 159–175. Springer, Heidelberg, Apr. 2001.

[97] C. Peikert. Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In M. Mitzenmacher, editor, *41st Annual ACM Symposium on Theory of Computing*, pages 333–342. ACM Press, May / June 2009.

[98] C. Peikert. Some recent progress in lattice-based cryptography. In O. Reingold, editor, *TCC 2009: 6th Theory of Cryptography Conference*, volume 5444 of *Lecture Notes in Computer Science*, page 72. Springer, Heidelberg, Mar. 2009. Invited talk. Slides available at http://web.eecs.umich.edu/~cpeikert/pubs/slides-tcc09.pdf.

[99] C. Peikert. An efficient and parallel Gaussian sampler for lattices. In T. Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 80–97. Springer, Heidelberg, Aug. 2010.

[100] C. Peikert. Lattice cryptography for the internet. In M. Mosca, editor, *Post-Quantum Cryptography - 6th International Workshop, PQCrypto 2014*, pages 197–219. Springer, Heidelberg, Oct. 2014.

[101] C. Peikert. A decade of lattice cryptography. *Foundations and Trends in Theoretical Computer Science*, 10(4):283–424, 2016.

[102] C. Peikert. How (not) to instantiate ring-LWE. In V. Zikas and R. De Prisco, editors, *SCN 16: 10th International Conference on Security in Communication Networks*, volume 9841 of *Lecture Notes in Computer Science*, pages 411–430. Springer, Heidelberg, Aug. / Sept. 2016.

[103] C. Peikert, O. Regev, and N. Stephens-Davidowitz. Pseudorandomness of ring-LWE for any ring and modulus. In H. Hatami, P. McKenzie, and V. King, editors, *49th Annual ACM Symposium on Theory of Computing*, pages 461–473. ACM Press, June 2017.

[104] C. Peikert, V. Vaikuntanathan, and B. Waters. A framework for efficient and composable oblivious transfer. In D. Wagner, editor, *Advances in Cryptology – CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 554–571. Springer, Heidelberg, Aug. 2008.

[105] C. Peikert and B. Waters. Lossy trapdoor functions and their applications. In R. E. Ladner and C. Dwork, editors, *40th Annual ACM Symposium on Theory of Computing*, pages 187–196. ACM Press, May 2008.

[106] L. T. Phong. Official comment: Frodo, Apr. 2018. NIST pqc-forum, https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-1/official-comments/Frodo-official-comment.pdf.

[107] T. Pöppelmann and T. Güneysu. Towards efficient arithmetic for lattice-based cryptography on reconfigurable hardware. In A. Hevia and G. Neven, editors, *Progress in Cryptology - LATINCRYPT 2012: 2nd International Conference on Cryptology and Information Security in Latin America*, volume 7533 of *Lecture Notes in Computer Science*, pages 139–158. Springer, Heidelberg, Oct. 2012.

[108] T. Pöppelmann and T. Güneysu. Towards practical lattice-based public-key encryption on reconfigurable hardware. In T. Lange, K. Lauter, and P. Lisonek, editors, *SAC 2013: 20th Annual International Workshop on Selected Areas in Cryptography*, volume 8282 of *Lecture Notes in Computer Science*, pages 68–85. Springer, Heidelberg, Aug. 2014.

[109] R. Primas, P. Pessl, and S. Mangard. Single-trace side-channel attacks on masked lattice-based encryption. In W. Fischer and N. Homma, editors, *Cryptographic Hardware and Embedded Systems – CHES 2017*, volume 10529 of *Lecture Notes in Computer Science*, pages 513–533. Springer, Heidelberg, Sept. 2017.

[110] S. M. Qasim, A. A. Telba, and A. Y. AlMazroo. FPGA design and implementation of matrix multiplier architectures for image and signal processing applications. *International Journal of Computer Science and Network Security*, 10(2):168–176, 2010.

[111] P. Ravi, D. B. Roy, S. Bhasin, A. Chattopadhyay, and D. Mukhopadhyay. Number "not used" once — practical fault attack on pqm4 implementations of NIST candidates. Cryptology ePrint Archive, Report 2018/211, 2018. https://eprint.iacr.org/2018/211.

[112] O. Regev. New lattice-based cryptographic constructions. *J. ACM*, 51(6):899–942, 2004. Preliminary version in STOC 2003.

[113] O. Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM*, 56(6):34, 2009. Preliminary version in STOC 2005.

[114] O. Regev. The learning with errors problem (invited survey). In *IEEE Conference on Computational Complexity*, pages 191–204, 2010.

[115] O. Reparaz, R. de Clercq, S. Sinha Roy, F. Vercauteren, and I. Verbauwhede. Additively homomorphic ring-LWE masking. In T. Takagi, editor, *PQCrypto 2016*, volume 9606 of *Lecture Notes in Computer Science*, pages 233–244. Springer, Heidelberg, 2016.

[116] O. Reparaz, S. Sinha Roy, R. de Clercq, F. Vercauteren, and I. Verbauwhede. Masking ring-LWE. *Journal of Cryptographic Engineering*, 6(2):139–153, 2016.

[117] S. S. Roy, F. Vercauteren, N. Mentens, D. D. Chen, and I. Verbauwhede. Compact ring-LWE cryptoprocessor. In L. Batina and M. Robshaw, editors, *Cryptographic Hardware and Embedded Systems – CHES 2014*, volume 8731 of *Lecture Notes in Computer Science*, pages 371–391. Springer, Heidelberg, Sept. 2014.

[118] M. Schneider. Sieving for shortest vectors in ideal lattices. In A. Youssef, A. Nitaj, and A. E. Hassanien, editors, *AFRICACRYPT 13: 6th International Conference on Cryptology in Africa*, volume 7918 of *Lecture Notes in Computer Science*, pages 375–391. Springer, Heidelberg, June 2013.

[119] C.-P. Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theoretical Computer Science*, 53:201–224, 1987.

[120] A. Shamir. A polynomial time algorithm for breaking the basic Merkle-Hellman cryptosystem. In D. Chaum, R. L. Rivest, and A. T. Sherman, editors, *Advances in Cryptology – CRYPTO'82*, pages 279–288. Plenum Press, New York, USA, 1982.

[121] D. Stehlé and R. Steinfeld. Making NTRU as secure as worst-case problems over ideal lattices. In K. G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 27–47. Springer, Heidelberg, May 2011.

[122] E. E. Targhi and D. Unruh. Post-quantum security of the Fujisaki-Okamoto and OAEP transforms. In M. Hirt and A. D. Smith, editors, *TCC 2016-B: 14th Theory of Cryptography Conference, Part II*, volume 9986 of *Lecture Notes in Computer Science*, pages 192–216. Springer, Heidelberg, Oct. / Nov. 2016.

[123] Y. Yu and L. Ducas. Second order statistical behavior of LLL and BKZ. In *International Conference on Selected Areas in Cryptography*, pages 3–22. Springer, 2017.

A Revision history

November 30, 2017

- Initial public release and submission to NIST Post-Quantum Cryptography standardization process round 1.

March 30, 2019

- Second public release and submission to NIST Post-Quantum Cryptography standardization process round 2.
- A third parameter set was added, Frodo-1344, which targets NIST Level 5 (matching or exceeding the brute-force security of AES-256). This resulted in the addition of the instantiations FrodoKEM-1344-AES and FrodoKEM-1344-SHAKE.
- The Gaussian parameter σ for Frodo-640 was changed from 2.75 to 2.8, and the distribution $\chi_{\mathsf{Frodo\text{-}640}}$ was updated as a result. This change was due to a miscalculation in Section 5.1.1 in the number of samples included in the Rényi divergence calculation as pointed out by Phong [106].
- The generation of pseudorandom bits for error matrices $\mathbf{S}, \mathbf{E}$ in FrodoPKE.KeyGen and error matrices $\mathbf{S}', \mathbf{E}', \mathbf{E}''$ in FrodoPKE.Enc has been changed. In the previous version of the specification the same seed was used in multiple pseudorandom expansions with distinct domain separators. In this version of the specification, the seed is used in a single pseudorandom expansion that outputs the total required output, which is then split among the different matrices. This reduces the number of calls to SHAKE and avoids the instruction-skipping fault attacks described in [111]. As part of implementing this change, we reorganized how pseudorandom bits are passed to the various calls to Frodo.SampleMatrix.
- We replaced all calls to cSHAKE with calls to SHAKE. We previously used customization strings with cSHAKE to achieve domain separation; however, since cSHAKE pads the customization string to the rate, this resulted in at least one additional call to the Keccak permutation for each use of cSHAKE, even when the input was smaller than the rate. We now use SHAKE throughout; this reduces the number of calls to the Keccak permutation. Our strategy for maintaining domain separation is described in Section 2.3.
- The transformation from the IND-CPA-secure FrodoPKE to the IND-CCA-secure FrodoKEM has been simplified. In the previous version of the specification, we used (a minor variant of) the $\mathsf{QFO}^{\not\perp}$ transformation of [68]; to achieve a security proof in the quantum random oracle model, it uses the technique of [122] which includes an extra hash value $\mathbf{d}$ in the ciphertext. In the current version of the specification, we use (a minor variant of) the $\mathsf{FO}^{\not\perp}$ transformation, which does not include the extra hash value $\mathbf{d}$ in the specification; a QROM proof of this version was given by [70] and is cited in Section 5.1.1.
- Known answer tests were updated as a result of the changes listed above.
- Performance measurements were updated as a result of the changes listed above.
- Section 5.2.5 was added to provide further details on the probability and impact of decryption failures.
- A few minor typos have been fixed, and some symbols have been renamed for greater clarity. Thanks to Martin Ekerå for identifying some mistakes.

July 2, 2019

- The March 30, 2019, revision introduced a mistranscription of the result of HHK [68] on the IND-CCA security of the FO transform in the classical random oracle model, incorrectly stating that it tightly relied on OW-CPA security, instead of IND-CPA security, of the underlying public-key encryption scheme. This error was pointed out by D. Bernstein on the NIST PQC mailing list. Bernstein also pointed out uncertainty in how the chain of reductions interacts with the argument involving Rényi divergence for the concrete distributions used by FrodoKEM. Section 5.1 has been revised and reorganized to address these issues. In particular, the following revisions appear in Sections 5.1.1–5.1.4:
  - A new Theorem 5.1 shows the IND-CCA security (in the classical random oracle model) of FrodoKEM, with its actual error distribution, under the assumption that FrodoPKE, with a rounded Gaussian error distribution, is IND-CPA-secure.

- The chain of reductions (specifically, Steps 2 and 3) also yields the same conclusion as above under the assumption that the “T-transformed” FrodoPKE, with either a rounded Gaussian or the actual FrodoKEM error distribution, is OW-PCA secure.
- Calculations for the bit security of FrodoKEM parameterizations based on Theorem 5.1 appear in a new Table 2.
- Results of [68] and [70] have been restated in a form closer to their original versions.

- In Section 5.2.3, the argumentation in the analysis of the dual attack was made more precise (though the ultimate conclusion remains the same).
- Ciphertext sizes were incorrectly stated in Table 1 and have been fixed (they are decreased by the size of the derived key **ss**). The March 30 version of Table 1 included estimates of OW-CPA security for FrodoPKE at different levels; since the current argument in support of IND-CCA security of FrodoKEM does not reference OW-CPA of FrodoPKE, these estimates have been removed.
- Table 2 was added to consolidate security estimates from all of the methodologies in the document.
- Tables 1–5 (and the Python scripts, under `3-media/Additional_Implementations`, generating these tables) were updated to reflect updated security estimates and ciphertext sizes.
- The text around Algorithms 7 and 8 was updated to clarify that 16-bit integers are represented in little-endian byte order.
- A typo in Table 9 that incorrectly labeled some rows was fixed.
- Typos (extra parentheses) were fixed in descriptions of Frodo, FrodoPKE, and FrodoKEM.
- An obsolete reference to hash function H was removed in Section 2.3.
- Table 10 was updated with revised estimates based on the number of samples available for each instance.

March 25, 2020

- Definition of table T_χ in Section 2.2.4 is fixed.
- The secret key matrix $\mathbf{S}$ is sampled in transposed form. This updates the specification to reflect how the C reference implementation works.
- Typos were fixed in Algorithms 10, 13, and 14.
- A Python reference implementation was made available.

September 30, 2020

- Submission to NIST Post-Quantum Cryptography standardization process round 3.
- Guo, Johansson, and Nilsson [66] identified a key-recovery timing attack on the reference implementation of FrodoKEM due to branching in FrodoKEM.Decaps. The pseudocode of Algorithm 14 (FrodoKEM.Decaps) was updated to make it clearer that these operations needed to be completed in constant-time. Our reference and optimized implementations have been updated accordingly. A brief discussion of this issue was added to Section 6.1 and performance figures were updated in Section 3.
- Parameter search scripts are now compatible with Python3.
- The parameter search procedure (Section 2.4.2) was reconciled with the scripts. Thanks to Sabrina Sewer for pointing out the discrepancy.
- A new Section 5.2.4 has been added, detailing a refinement of the security analysis with respect to concrete cryptanalytic attacks taking into account the recent state-of-the-art.
- The security estimates columns in Tables 2 and 10 were renamed and an extra column was added to use the “classical/quantum/plausible” terminology appearing elsewhere in the literature.