

FrodoKEM
Learning With Errors Key Encapsulation

Algorithm Specifications And Supporting Documentation

November 30, 2017

Contents

1	Introduction and design rationale	4
1.1	Pedigree	4
1.2	Design overview and rationale	5
1.2.1	Generic, algebraically unstructured lattices	5
1.2.2	Parameters from worst-case reductions and conservative cryptanalysis	6
1.2.3	Simplicity of design and implementation	7
1.3	Other features	8
2	Written specification	9
2.1	Background	9
2.1.1	Notation	9
2.1.2	Cryptographic definitions	9
2.1.3	Learning With Errors	11
2.1.4	Gaussians	12
2.1.5	Lattices	12
2.2	Algorithm description	13
2.2.1	Matrix encoding of bit strings	13
2.2.2	Packing matrices modulo q	13
2.2.3	Deterministic random bit generation	14
2.2.4	Sampling from the error distribution	14
2.2.5	Pseudorandom matrix generation	15
2.2.6	FrodoPKE: IND-CPA-secure public key encryption scheme	16
2.2.7	Correctness of IND-CPA PKE	17
2.2.8	Transform from IND-CPA PKE to IND-CCA KEM	18
2.2.9	FrodoKEM: IND-CCA-secure key encapsulation mechanism	19
2.2.10	Correctness of IND-CCA KEM	20
2.2.11	Interconversion to IND-CCA PKE	20
2.3	Cryptographic primitives	21
2.4	Parameters	21
2.4.1	High-level overview	21
2.4.2	Parameter constraints	21
2.4.3	Selected parameter sets	22
2.5	Summary of parameters	22
2.6	Provenance of constants and tables	23
3	Performance analysis	24
3.1	Associated implementations	24
3.2	Performance analysis on x64 Intel	24
3.2.1	Performance using AES128	24
3.2.2	Performance using cSHAKE128	24
3.2.3	Memory analysis	25
3.3	Performance analysis on ARM	26
4	Known Answer Test (KAT) values	28
5	Justification of security strength	29
5.1	Security reductions	29
5.1.1	IND-CCA Security of KEM	29
5.1.2	IND-CPA Security of PKE	30
5.1.3	Approximating the error distribution	30
5.1.4	Deterministic generation of \mathbf{A}	31
5.1.5	Reductions from worst-case lattice problems	32
5.2	Cryptanalytic attacks	34

5.2.1	Methodology: the core-SVP hardness	34
5.2.2	Primal attack	36
5.2.3	Dual attack	36
6	Advantages and limitations	37
6.1	Ease of implementation	37
6.2	Compatibility with existing deployments and hybrid schemes	37
6.3	Hardware implementations	38
6.4	Side-channel resistance	38

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 two 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).

Two variants of each of the above schemes are provided:

- FrodoKEM-640-AES and FrodoKEM-976-AES, which use AES128 to pseudorandomly generate a large public matrix (called \mathbf{A}).
- FrodoKEM-640-cSHAKE and FrodoKEM-976-cSHAKE, which use cSHAKE128 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 cSHAKE 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.

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 [82, 103, 92] 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 [57, 30, 81, 101, 84]. Notably, in work published in 2005, Regev [102] 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., [88, 13, 28, 48, 83, 94]),
- extensively analyzed the concrete security of LWE and closely related lattice problems (e.g., [85, 40, 76, 8, 39, 6, 7, 69, 65, 10, 11, 24, 5, 9], among countless others), and
- constructed LWE-based cryptosystems with improved efficiency or additional functionality (e.g., [96, 95, 55, 32, 29, 56, 22, 58]).

In particular, in work published in 2011, Lindner and Peikert [75] 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 [75] 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.

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

²As pointed out in [89], Regev’s encryption scheme implicitly contains an (unauthenticated) “approximate” key-exchange protocol analogous to the classic Diffie–Hellman protocol [46].

Frodo. FrodoPKE is closely related to an earlier work [24], 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 [47, 91, 25, 11]), whereas FrodoKEM uses simpler key transport via public-key encryption (as in [102, 75]).
- 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 [51] 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 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 [111] gave a modification of the Fujisaki–Okamoto transform that achieves IND-CCA security in the *quantum* random-oracle model. In 2017, Hofheinz, Hövelmanns, and Kiltz [61] 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. FrodoKEM is constructed from FrodoPKE using a slight variant of one of the constructions from [61], 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 [102] 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 [78, 94] and Module-LWE [27, 72], and problems associated with the classic NTRU cryptosystem [60], 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 [110], have been proven hard assuming the *worst-case* quantum hardness of certain problems on lattices corresponding to *ideals* or *modules* over the ring [78, 72, 94].

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 [40, 107, 62, 26, 70].³ However, some conventional NTRU parameterizations admit specialized attacks with significantly better asymptotic performance than on generic lattices with the same parameters [65, 66]. In addition, a series of recent works [31, 42, 43] 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 [74, 108]). 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), and the algorithms from [31, 42, 43] do not yet have any impact on Ring- or Module-LWE themselves.

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 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., [109, 86, 31, 42]. Indeed, Micciancio and Regev [85] 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 [102, 94] 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 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 [48, 83] 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 [102, 75], in the case of moderately small errors. Note that some limitation on the number of samples is necessary,

³Some unconventional parameterizations of Ring-LWE were specifically devised to be breakable by certain algebraic attacks [50, 37, 33, 38]. 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 [33, 93].

because LWE with errors bounded by (say) a constant is solvable in polynomial time, given a large enough polynomial number of samples [14, 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 $\sigma \geq 2.3$, and (automatically) limits the number of LWE samples available to the adversary. Although such parameters do not conform to the full quantum reductions from [102, 94] 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 [24] 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, 77, 102, 45]. 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” [84] 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 [85] (see also [36]), the known worst-case reduction does not yield any meaningful “end-to-end” security guarantee for our concrete parameters based on the conjecture 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, as stated in the above quote from [85], 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 [11] and Frodo [24] 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 [9], which concluded that its experimental results “confirm that lattice-reduction largely follows the behavior expected from the 2016 estimate [11].” 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 [\[15\]](#). See [Section 5.1.3](#) for full details.

Matrix-vector operations. Apart from error sampling and calls to symmetric primitives like AES or cSHAKE, the main operations in our schemes are simple matrix-vector products. Compared to systems like NewHope [\[11\]](#) or Kyber [\[23\]](#) 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 use the original method from Regev’s encryption scheme [\[102\]](#) of transmitting secret bits by simply adding their $(q/2)$ -multiples to pseudorandom values that the receiver can (approximately) subtract away. 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 [\[11\]](#), 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.

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 [\[29\]](#).

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 [\[18\]](#) and used for all keys (see [\[25\]](#) 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 [\[11, 24\]](#) 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 i th entry for $0 \leq i < n$ 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 i th row and j th column) for $0 \leq i < m$ and $0 \leq j < n$ is denoted by $\mathbf{A}_{i,j}$, and its i th row is denoted by $\mathbf{A}_i = (\mathbf{A}_{i,0}, \mathbf{A}_{i,1}, \dots, \mathbf{A}_{i,n-1})$.
- An m -bit string $\mathbf{k} \in \{0, 1\}^m$ is written as a vector over the set $\{0, 1\}$ and its i th bit for $0 \leq i < m$ 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 $\lceil a \rceil = \lfloor a + 1/2 \rfloor$.
- 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 key space \mathcal{K} :

- KeyGen() $\rightarrow (pk, sk)$: A probabilistic key generation algorithm that outputs a public key pk and a secret key sk .
- 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.
- 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

$$\Pr[\mathbf{ss}' \neq \mathbf{ss} : (pk, sk) \leftarrow \text{KEM.KeyGen}(); (c, \mathbf{ss}) \leftarrow \text{KEM.Encaps}(pk); \mathbf{ss}' \leftarrow \text{KEM.Decaps}(c, sk)] \leq \delta .$$

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 key space \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 $\text{Exp}_{\text{KEM}}^{\text{ind-cca}}(\mathcal{A})$ shown in [Figure 1](#). The advantage of \mathcal{A} in the experiment is

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

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 $\mathcal{O}_{\text{Decaps}}$ oracle.

Experiment $\text{Exp}_{\text{KEM}}^{\text{ind-cca}}(\mathcal{A})$:	Oracle $\mathcal{O}_{\text{Decaps}}(c)$:
1: $(pk, sk) \leftarrow_{\$} \text{KEM.KeyGen}()$	1: if $c = c^*$ then
2: $b \leftarrow_{\$} \{0, 1\}$	2: return \perp
3: $(c^*, \text{ss}_0) \leftarrow_{\$} \text{KEM.Encaps}(pk)$	3: else
4: $\text{ss}_1 \leftarrow_{\$} U(\mathcal{K})$	4: return $\text{KEM.Decaps}(c, sk)$
5: $b' \leftarrow_{\$} \mathcal{A}^{\mathcal{O}_{\text{Decaps}}(\cdot)}(pk, \text{ss}_b, c^*)$	
6: if $b' = b$ then	
7: return 1	
8: else	
9: return 0	

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 $(\text{KeyGen}, \text{Enc}, \text{Dec})$ along with a message space \mathcal{M} :

- $\text{KeyGen}() \rightarrow (pk, sk)$: A probabilistic key generation algorithm that outputs a public key pk and a secret key sk .
- $\text{Enc}(m, pk) \rightarrow c$: 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 $\text{Enc}(m, pk; r) \rightarrow c$, where the randomness $r \in \mathcal{R}$ is passed as an explicit input; \mathcal{R} is called the randomness space of the encryption algorithm.
- $\text{Dec}(c, sk) \rightarrow m'$ 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 [61]). A public key encryption scheme PKE with message space \mathcal{M} is δ -correct if

$$\mathbb{E} \left[\max_{m \in \mathcal{M}} \Pr [\text{PKE.Dec}(c, sk) \neq m : c \leftarrow_{\$} \text{PKE.Enc}(m, pk)] \right] \leq \delta , \quad (1)$$

where the expectation is taken over $(pk, sk) \leftarrow_{\$} \text{PKE.KeyGen}()$.

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

$$\Pr [\text{PKE.Dec}(c, sk) \neq m : (pk, sk) \leftarrow_{\$} \text{PKE.KeyGen}(); c \leftarrow_{\$} \text{PKE.Enc}(m, pk)] \leq \delta , \quad (2)$$

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 $\text{Exp}_{\text{PKE}}^{\text{ind-cpa}}(\mathcal{A})$ shown in Figure 2. The advantage of \mathcal{A} in the experiment is

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

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

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

- 1: $(pk, sk) \leftarrow_{\$} \text{PKE.KeyGen}()$
- 2: $(m_0, m_1, st) \leftarrow_{\$} \mathcal{A}(pk)$
- 3: $b \leftarrow_{\$} \{0, 1\}$
- 4: $c^* \leftarrow_{\$} \text{PKE.Enc}(m_b, pk)$
- 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., [20]) first defined by Regev [102]. 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 $\text{SLWE}_{n, m, q, \chi}$ (respectively, $\text{nf-SLWE}_{n, m, q, \chi}$), 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)

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

Similarly, define (for the normal-form case) $\text{Adv}_{n, m, q, \chi}^{\text{nf-slwe}}(\mathcal{A})$, 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 $\text{DLWE}_{n, m, q, \chi}$ (respectively, $\text{nf-DLWE}_{n, m, q, \chi}$), 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)

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

Similarly, define (for the normal-form case) $\text{Adv}_{n,m,q,\chi}^{\text{ndlwe}}(\mathcal{A})$, 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 $\rho_s: \mathbb{R} \rightarrow \mathbb{R}^+$, 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:

$$\Psi_s(x) = \int_{\{z: \lfloor z \rfloor = x\}} D_s(z) dz . \quad (3)$$

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 $\text{span}_{\mathbb{R}}(\mathcal{L}) = \mathbb{R}^n$. 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

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

The volume, or determinant, of \mathcal{L} is defined as $\text{vol}(\mathcal{L}) := |\det(\mathbf{B})|$. 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 i th 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 $_\gamma$)). 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 $_\gamma$)). Given a basis \mathbf{B} of an n -dimensional lattice $\mathcal{L} = \mathcal{L}(\mathbf{B})$, output a set $\{\mathbf{v}_1, \dots, \mathbf{v}_n\} \subset \mathcal{L}$ 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 $_{\varphi}$)). 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, \bar{m}, \bar{n} , integer matrix dimensions with $n \equiv 0 \pmod{8}$;
- $B \leq D$, the number of bits encoded in each matrix entry;
- $\ell = B \cdot \bar{m} \cdot \bar{n}$, the length of bit strings that are encoded as \bar{m} -by- \bar{n} matrices;
- $\text{len}_{\mathbf{A}}$, the bit length of seeds used for pseudorandom matrix generation;
- $\text{len}_{\mathbf{E}}$, 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 $\text{ec}(\cdot)$ encodes an integer $0 \leq k < 2^B$ as an element in \mathbb{Z}_q by multiplying it by $q/2^B = 2^{D-B}$:

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

Using this function, the function `Frodo.Encode` encodes bit strings of length $\ell = B \cdot \bar{m} \cdot \bar{n}$ as \bar{m} -by- \bar{n} -matrices with entries in \mathbb{Z}_q by applying $\text{ec}(\cdot)$ 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 $\text{ec}(k)$, 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 \bar{m} -by- \bar{n} matrix \mathbf{K} into a bit string of length $\ell = B \cdot \bar{m} \cdot \bar{n}$. It extracts B bits from each entry by applying the function $\text{dc}(\cdot)$:

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

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 $\text{dc}(\text{ec}(k)) = k$ for all $0 \leq k < 2^B$.

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 1 Frodo.Encode

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

```
1: for ( $i = 0$ ;  $i < \bar{m}$ ;  $i \leftarrow i + 1$ ) do
2:   for ( $j = 0$ ;  $j < \bar{n}$ ;  $j \leftarrow j + 1$ ) do
3:      $k \leftarrow \sum_{l=0}^{B-1} \mathbf{k}_{(i \cdot \bar{n} + j)B + l} \cdot 2^l$ 
4:      $\mathbf{K}_{i,j} \leftarrow \text{ec}(k) = k \cdot q / 2^B$ 
5: return  $\mathbf{K} = (\mathbf{K}_{i,j})_{0 \leq i < \bar{m}, 0 \leq j < \bar{n}}$ 
```

Algorithm 2 Frodo.Decode

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

```
1: for ( $i = 0$ ;  $i < \bar{m}$ ;  $i \leftarrow i + 1$ ) do
2:   for ( $j = 0$ ;  $j < \bar{n}$ ;  $j \leftarrow j + 1$ ) do
3:      $k \leftarrow \text{dc}(\mathbf{K}_{i,j}) = \lfloor \mathbf{K}_{i,j} \cdot 2^B / q \rfloor \bmod 2^B$ 
4:      $k = \sum_{l=0}^{B-1} k_l \cdot 2^l$  where  $k_l \in \{0, 1\}$ 
5:     for ( $l = 0$ ;  $l < B$ ;  $l \leftarrow l + 1$ ) do
6:        $\mathbf{k}_{(i \cdot \bar{n} + j)B + l} \leftarrow k_l$ 
7: return  $\mathbf{k}$ 
```

Algorithm 3 Frodo.Pack

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

```
1: for ( $i = 0$ ;  $i < n_1$ ;  $i \leftarrow i + 1$ ) do
2:   for ( $j = 0$ ;  $j < n_2$ ;  $j \leftarrow j + 1$ ) do
3:      $\mathbf{C}_{i,j} = \sum_{l=0}^{D-1} c_l \cdot 2^l$  where  $c_l \in \{0, 1\}$ 
4:     for ( $l = 0$ ;  $l < D$ ;  $l \leftarrow l + 1$ ) do
5:        $\mathbf{b}_{(i \cdot n_2 + j)D + l} \leftarrow c_{D-1-l}$ 
6: 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 $\mathbf{C} \in \mathbb{Z}_q^{n_1 \times n_2}$.

```
1: for ( $i = 0$ ;  $i < n_1$ ;  $i \leftarrow i + 1$ ) do
2:   for ( $j = 0$ ;  $j < n_2$ ;  $j \leftarrow j + 1$ ) do
3:      $\mathbf{C}_{i,j} \leftarrow \sum_{l=0}^{D-1} \mathbf{b}_{(i \cdot n_2 + j)D + l} \cdot 2^{D-1-l}$ 
4: 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 cSHAKE [64]. The function cSHAKE is taken as either cSHAKE128 or cSHAKE256 (indicated below for each parameter set of FrodoKEM), and takes as input a bit string X , a requested output bit length L , and a 16-bit customization value c as a domain separator. Interpreting c as a 16-bit integer, it is converted to an array of two bytes $[c_0, c_1]$, where $c = c_0 + 2^8 \cdot c_1$. The call to the function on these inputs is written $\text{cSHAKE}(X, L, c)$, and returns a string of L bits as output.

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 $S_\chi = \{-s, -s + 1, \dots, -1, 0, 1, \dots, s - 1, s\}$ 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

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

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:

$$T_\chi(0)/2^{\text{len}_\chi} = \chi(0)/2 \quad \text{and} \quad T_\chi(z)/2^{\text{len}_\chi} = \chi(0)/2 + \sum_{i=1}^z \chi(i) \text{ for } 1 \leq z \leq s .$$

Therefore, $T_\chi(s) = 2^{\text{len}_\chi - 1}$.

Sampling from χ via inversion sampling is done as shown in Algorithm 5. Given a string of len_χ uniformly random bits $\mathbf{r} \in \{0, 1\}^{\text{len}_\chi}$ and a distribution table T_χ , the algorithm Frodo.Sample returns a sample e from the distribution χ . 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 . Note also that the comparison in Step 4 needs to be implemented in a constant-time manner.

Algorithm 5 Frodo.Sample

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

```
1:  $t \leftarrow \sum_{i=1}^{\text{len}_\chi - 1} \mathbf{r}_i \cdot 2^{i-1}$ 
2:  $e \leftarrow 0$ 
3: for ( $z = 0; z < s; z \leftarrow z + 1$ ) do
4:   if  $t > T_\chi(z)$  then
5:      $e \leftarrow e + 1$ 
6:  $e \leftarrow (-1)^{\mathbf{r}_0} \cdot e$ 
7: return  $e$ 
```

An n_1 -by- n_2 matrix of $n_1 n_2$ samples from the error distribution is sampled by expanding a random seed to an $(n_1 n_2 \cdot \text{len}_\chi)$ -bit string, here written as a sequence of $n_1 n_2$ bit vectors $(\mathbf{r}^{(0)}, \mathbf{r}^{(1)}, \dots, \mathbf{r}^{(n_1 n_2 - 1)})$ of length len_χ , using the extendable output function cSHAKE, and then sampling $n_1 n_2$ error terms by calling 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: Seed $\text{seed}_\mathbf{E} \in \{0, 1\}^{\text{len}_\mathbf{E}}$, dimensions n_1, n_2 , the table T_χ , domain separator value c .
Output: A sample $\mathbf{E} \in \mathbb{Z}^{n_1 \times n_2}$.

```
1:  $(\mathbf{r}^{(0)}, \mathbf{r}^{(1)}, \dots, \mathbf{r}^{(n_1 n_2 - 1)}) \leftarrow \text{cSHAKE}(\text{seed}_\mathbf{E}, n_1 n_2 \cdot \text{len}_\chi, c)$  (here, each  $\mathbf{r}^{(i)}$  is a vector of  $\text{len}_\chi$  bits)
2: for ( $i = 0; i < n_1; i \leftarrow i + 1$ ) do
3:   for ( $j = 0; j < n_2; j \leftarrow j + 1$ ) do
4:      $\mathbf{E}_{i,j} \leftarrow \text{Frodo.Sample}(\mathbf{r}^{(i \cdot n_2 + j)}, T_\chi)$ 
5: return  $\mathbf{E}$ 
```

2.2.5 Pseudorandom matrix generation

The algorithm Frodo.Gen takes as input a seed $\text{seed}_\mathbf{A} \in \{0, 1\}^{\text{len}_\mathbf{A}}$ 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 cSHAKE128 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 $i = 0, 1, \dots, n - 1$ and column index $j = 0, 8, \dots, n - 8$, the algorithm generates a 128-bit block, which it uses to set the matrix entries $\mathbf{A}_{i,j}, \mathbf{A}_{i,j+1}, \dots, \mathbf{A}_{i,j+7}$ as follows. It applies AES128 with key $\text{seed}_\mathbf{A}$ 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. 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 $k = 0, 1, \dots, 7$. 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 $\text{seed}_\mathbf{A} \in \{0, 1\}^{\text{len}_\mathbf{A}}$.
Output: Matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times n}$.

```
1: for ( $i = 0; i < n; i \leftarrow i + 1$ ) do
2:   for ( $j = 0; j < n; j \leftarrow j + 8$ ) do
3:      $\mathbf{b} \leftarrow \langle i \rangle \| \langle j \rangle \| 0 \cdots 0 \in \{0, 1\}^{128}$  where  $\langle i \rangle, \langle j \rangle \in \{0, 1\}^{16}$ 
4:      $\langle c_{i,j} \rangle \| \langle c_{i,j+1} \rangle \cdots \| \langle c_{i,j+7} \rangle \leftarrow \text{AES128}_{\text{seed}_\mathbf{A}}(\mathbf{b})$  where each  $\langle c_{i,k} \rangle \in \{0, 1\}^{16}$ .
5:     for ( $k = 0; k < 8; k \leftarrow k + 1$ ) do
6:        $\mathbf{A}_{i,j+k} \leftarrow c_{i,j+k} \bmod q$ 
7: return  $\mathbf{A}$ 
```

Using cSHAKE128. Algorithm 8 generates a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times n}$ as follows. For each row index $i = 0, 1, \dots, n-1$, it calls cSHAKE128 with a main input of $\text{seed}_{\mathbf{A}}$ and customization value $2^8 + i$ to produce a $16n$ -bit output string. It splits this output into 16-bit integers $c_{i,j}$ for $j = 0, 1, \dots, n-1$, and sets $\mathbf{A}_{i,j} = c_{i,j} \bmod q$ for all j . Note that the offset of 2^8 in the customization value is used for domain separation between the generation of \mathbf{A} and other uses of cSHAKE128 in the key encapsulation functions below, which have customization values smaller than 2^8 .

Algorithm 8 Frodo.Gen using cSHAKE128

Input: Seed $\text{seed}_{\mathbf{A}} \in \{0, 1\}^{\text{len}_{\mathbf{A}}}$.

Output: Pseudorandom matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times n}$.

```

1: for ( $i = 0; i < n; i \leftarrow i + 1$ ) do
2:    $\langle c_{i,0} \rangle \| \langle c_{i,1} \rangle \| \dots \| \langle c_{i,n-1} \rangle \leftarrow \text{cSHAKE128}(\text{seed}_{\mathbf{A}}, 16n, 2^8 + i)$  where each  $\langle c_{i,j} \rangle \in \{0, 1\}^{16}$ .
3:   for ( $j = 0; j < n; j \leftarrow j + 1$ ) do
4:      $\mathbf{A}_{i,j} \leftarrow c_{i,j} \bmod q$ 
5: return  $\mathbf{A}$ 

```

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 [75], with the following adaptations and specializations:

- The matrix \mathbf{A} is generated from a seed using the function Gen specified in Section 2.2.5.
- Several (\bar{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 $D \leq 16$;
- n, \bar{m}, \bar{n} , integer matrix dimensions with $n \equiv 0 \pmod{8}$;
- $B \leq D$, the number of bits encoded in each matrix entry;
- $\ell = B \cdot \bar{m} \cdot \bar{n}$, the length of bit strings that are encoded as \bar{m} -by- \bar{n} matrices;
- $\text{len}_{\mu} = \ell$, the bit length of messages;
- $\mathcal{M} = \{0, 1\}^{\text{len}_{\mu}}$, the message space;
- $\text{len}_{\mathbf{A}}$, the bit length of seeds used for pseudorandom matrix generation;
- $\text{len}_{\mathbf{E}}$, the bit length of seeds used for pseudorandom bit generation for error sampling;
- Gen, the matrix-generation algorithm, either Algorithm 7 or Algorithm 8;
- T_{χ} , the distribution table for sampling.

In the notation of [75], their n_1 and n_2 both equal n here, and their dimension ℓ is \bar{n} here.

Algorithm 9 FrodoPKE.KeyGen.

Input: None.

Output: Key pair $(pk, sk) \in (\{0, 1\}^{\text{len}_{\mathbf{A}}} \times \mathbb{Z}_q^{n \times \bar{n}}) \times \mathbb{Z}_q^{n \times \bar{n}}$.

```

1: Choose a uniformly random seed  $\text{seed}_{\mathbf{A}} \leftarrow_{\$} U(\{0, 1\}^{\text{len}_{\mathbf{A}}})$ 
2: Generate the matrix  $\mathbf{A} \in \mathbb{Z}_q^{n \times n}$  via  $\mathbf{A} \leftarrow \text{Frodo.Gen}(\text{seed}_{\mathbf{A}})$ 
3: Choose a uniformly random seed  $\text{seed}_{\mathbf{E}} \leftarrow_{\$} U(\{0, 1\}^{\text{len}_{\mathbf{E}}})$ 
4: Sample error matrix  $\mathbf{S} \leftarrow \text{Frodo.SampleMatrix}(\text{seed}_{\mathbf{E}}, n, \bar{n}, T_{\chi}, 1)$ 
5: Sample error matrix  $\mathbf{E} \leftarrow \text{Frodo.SampleMatrix}(\text{seed}_{\mathbf{E}}, n, \bar{n}, T_{\chi}, 2)$ 
6: Compute  $\mathbf{B} = \mathbf{A}\mathbf{S} + \mathbf{E}$ 
7: return public key  $pk \leftarrow (\text{seed}_{\mathbf{A}}, \mathbf{B})$  and secret key  $sk \leftarrow \mathbf{S}$ 

```

Algorithm 10 FrodoPKE.Enc.

Input: Message $\mu \in \mathcal{M}$ and public key $pk = (\text{seed}_{\mathbf{A}}, \mathbf{B}) \in \{0, 1\}^{\text{len}_{\mathbf{A}}} \times \mathbb{Z}_q^{n \times \bar{n}}$.

Output: Ciphertext $c = (\mathbf{C}_1, \mathbf{C}_2) \in \mathbb{Z}_q^{\bar{m} \times n} \times \mathbb{Z}_q^{\bar{m} \times \bar{n}}$.

- 1: Generate $\mathbf{A} \leftarrow \text{Frodo.Gen}(\text{seed}_{\mathbf{A}})$
 - 2: Choose a uniformly random seed $\text{seed}_{\mathbf{E}} \leftarrow_s U(\{0, 1\}^{\text{len}_{\mathbf{E}}})$
 - 3: Sample error matrix $\mathbf{S}' \leftarrow \text{Frodo.SampleMatrix}(\text{seed}_{\mathbf{E}}, \bar{m}, n, T_{\chi}, 4)$
 - 4: Sample error matrix $\mathbf{E}' \leftarrow \text{Frodo.SampleMatrix}(\text{seed}_{\mathbf{E}}, \bar{m}, n, T_{\chi}, 5)$
 - 5: Sample error matrix $\mathbf{E}'' \leftarrow \text{Frodo.SampleMatrix}(\text{seed}_{\mathbf{E}}, \bar{m}, \bar{n}, T_{\chi}, 6)$
 - 6: Compute $\mathbf{B}' = \mathbf{S}'\mathbf{A} + \mathbf{E}'$ and $\mathbf{V} = \mathbf{S}'\mathbf{B} + \mathbf{E}''$
 - 7: **return** ciphertext $c \leftarrow (\mathbf{C}_1, \mathbf{C}_2) = (\mathbf{B}', \mathbf{V} + \text{Frodo.Encode}(\mu))$
-

Algorithm 11 FrodoPKE.Dec.

Input: Ciphertext $c = (\mathbf{C}_1, \mathbf{C}_2) \in \mathbb{Z}_q^{\bar{m} \times n} \times \mathbb{Z}_q^{\bar{m} \times \bar{n}}$ and secret key $sk = \mathbf{S} \in \mathbb{Z}_q^{n \times \bar{n}}$.

Output: Decrypted message $\mu' \in \mathcal{M}$.

- 1: Compute $\mathbf{M} = \mathbf{C}_2 - \mathbf{C}_1\mathbf{S}$
 - 2: **return** message $\mu' \leftarrow \text{Frodo.Decode}(\mathbf{M})$
-

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 $\text{dc}(\text{ec}(k) + e) = k$ 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 $\text{dc}(\text{ec}(k) + e) = \lfloor k + e2^B/q \rfloor \bmod 2^B$. \square

Correctness of decryption: The decryption algorithm FrodoPKE.Dec computes

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

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 [51] constructs an IND-CCA2-secure public key encryption scheme from a one-way-secure public key encryption scheme in the classical random oracle model (with an assumption on the distribution of ciphertexts for each plaintext being sufficiently close to uniform). Targhi and Unruh [111] 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 not the case for lattice-based schemes. Hofheinz, Hövelmanns, and Kiltz [61] gave a variety of constructions in a modular fashion. We apply their $\text{QFO}_m^{\mathcal{K}}$ (“Quantum FO with implicit rejection and K being a hash of m ”) transform which constructs an IND-CCA-secure key encapsulation mechanism from an IND-CPA public key encryption scheme and three hash functions; following [23], we make the following modifications (see Figure 3 for notation), denoting the resulting transform $\text{QFO}_m^{\mathcal{K}'}$:

- A single hash function (with longer output) is used to compute \mathbf{r} , \mathbf{k} , and \mathbf{d} .
- The computation of \mathbf{r} , \mathbf{k} , and \mathbf{d} 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 (QFO $_m^{\mathcal{K}'}$ transform). Let $\text{PKE} = (\text{KeyGen}, \text{Enc}, \text{Dec})$ 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 $\text{len}_{\mathbf{s}}$, $\text{len}_{\mathbf{k}}$, $\text{len}_{\mathbf{d}}$, $\text{len}_{\mathbf{ss}}$ be parameters. Let $G : \{0, 1\}^* \rightarrow \mathcal{R} \times \{0, 1\}^{\text{len}_{\mathbf{k}}} \times \{0, 1\}^{\text{len}_{\mathbf{d}}}$ and $F : \{0, 1\}^* \rightarrow \{0, 1\}^{\text{len}_{\mathbf{ss}}}$ be hash functions. Define $\text{QKEM}_m^{\mathcal{K}'} = \text{QFO}_m^{\mathcal{K}'}[\text{PKE}, G, F]$ be the key encapsulation mechanism with $\text{QKEM}_m^{\mathcal{K}'}. \text{KeyGen}$, $\text{QKEM}_m^{\mathcal{K}'}. \text{Encaps}$ and $\text{QKEM}_m^{\mathcal{K}'}. \text{Decaps}$ as shown in Figure 3.

$\text{QKEM}_m^{\mathcal{K}'}. \text{KeyGen}():$ 1: $(pk, sk) \leftarrow_{\mathcal{S}} \text{PKE}. \text{KeyGen}()$ 2: $\mathbf{s} \leftarrow_{\mathcal{S}} \{0, 1\}^{\text{len}_{\mathbf{s}}}$ 3: $sk' \leftarrow (sk, \mathbf{s})$ 4: return (pk, sk')	$\text{QKEM}_m^{\mathcal{K}'}. \text{Decaps}((c, \mathbf{d}), (sk, \mathbf{s}, pk)):$ 1: $\mu' \leftarrow \text{PKE}. \text{Dec}(c, sk)$ 2: $(\mathbf{r}', \mathbf{k}', \mathbf{d}') \leftarrow G(pk \parallel \mu')$ 3: if $c = \text{PKE}. \text{Enc}(\mu', pk; \mathbf{r}')$ and $\mathbf{d} = \mathbf{d}'$ then 4: return $\mathbf{ss}' \leftarrow F(c \parallel \mathbf{k}' \parallel \mathbf{d})$ 5: else 6: return $\mathbf{ss}' \leftarrow F(c \parallel \mathbf{s} \parallel \mathbf{d})$
$\text{QKEM}_m^{\mathcal{K}'}. \text{Encaps}(pk):$ 1: $\mu \leftarrow_{\mathcal{S}} \mathcal{M}$ 2: $(\mathbf{r}, \mathbf{k}, \mathbf{d}) \leftarrow G(pk \parallel \mu)$ 3: $c \leftarrow \text{PKE}. \text{Enc}(\mu, pk; \mathbf{r})$ 4: $\mathbf{ss} \leftarrow F(c \parallel \mathbf{k} \parallel \mathbf{d})$ 5: return $((c, \mathbf{d}), \mathbf{ss})$	

Figure 3: Construction of an IND-CCA-secure key encapsulation mechanism $\text{QKEM}_m^{\mathcal{K}'} = \text{QFO}_m^{\mathcal{K}'}[\text{PKE}, G, F]$ from a public key encryption scheme PKE and hash functions G and F .

Remark. In October and November 2017, Jiang et al. posted an eprint giving a proof for two FO-like transforms that yield an IND-CCA-secure KEM in the quantum random oracle model and do not require the extra hash value d introduced by Targhi and Unruh [63]. As of this writing, it is too early to know if the results in the Jiang et al. eprint are correct. If they are, then we can remove the d value in the $\text{QKEM}_m^{\mathcal{K}'}$ transform of Figure 3, and from the resulting FrodoKEM in Algorithm 13 and Algorithm 14. The transform becomes sufficiently similar to Jiang et al.’s FO-I transform to have its security implied by their result; the differences are: the computation of r and K also take the public key pk as input, and the value random coins μ are hashed before they are input to the hash function that computes the shared secret.

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 $\text{QFO}_m^{\chi'}$ transform. FrodoKEM is parameterized by the following parameters:

- $q = 2^D$, a power-of-two integer modulus with exponent $D \leq 16$;
- n, \bar{m}, \bar{n} , integer matrix dimensions with $n \equiv 0 \pmod{8}$;
- $B \leq D$, the number of bits encoded in each matrix entry;
- $\ell = B \cdot \bar{m} \cdot \bar{n}$, the length of bit strings to be encoded in an \bar{m} -by- \bar{n} matrix;
- $\text{len}_\mu = \ell$, the bit length of messages;
- $\mathcal{M} = \{0, 1\}^{\text{len}_\mu}$, the message space;
- $\text{len}_\mathbf{A}$, the bit length of seeds used for pseudorandom matrix generation;
- $\text{len}_\mathbf{E}$, the bit length of seeds used for pseudorandom bit generation for error sampling;
- Gen , pseudorandom matrix generation algorithm, either [Algorithm 7](#) or [Algorithm 8](#);
- T_χ , distribution table for sampling;
- $\text{len}_\mathbf{s}$, the length of the bit vector \mathbf{s} used for pseudorandom shared secret generation in the event of decapsulation failure in the $\text{QFO}_m^{\chi'}$ transform;
- $\text{len}_\mathbf{z}$, the bit length of seeds used for pseudorandom generation of $\text{seed}_\mathbf{A}$;
- $\text{len}_\mathbf{k}$, the bit length of intermediate shared secret \mathbf{k} in the $\text{QFO}_m^{\chi'}$ transform;
- $\text{len}_\mathbf{d}$, the bit length of check value \mathbf{d} in the $\text{QFO}_m^{\chi'}$ transform;
- $\text{len}_{\mathbf{ss}}$, the bit length of shared secret \mathbf{ss} in the $\text{QFO}_m^{\chi'}$ transform;
- $H : \{0, 1\}^* \rightarrow \{0, 1\}^{\text{len}_\mathbf{A}}$, a hash function, either $\text{cSHAKE128}(\cdot, \text{len}_\mathbf{A}, 0)$ or $\text{cSHAKE256}(\cdot, \text{len}_\mathbf{A}, 0)$;
- $G : \{0, 1\}^* \rightarrow \{0, 1\}^{\text{len}_\mathbf{E}} \times \{0, 1\}^{\text{len}_\mathbf{k}} \times \{0, 1\}^{\text{len}_\mathbf{d}}$, a hash function, either $\text{cSHAKE128}(\cdot, \text{len}_\mathbf{E} + \text{len}_\mathbf{k} + \text{len}_\mathbf{d}, 3)$ or $\text{cSHAKE256}(\cdot, \text{len}_\mathbf{E} + \text{len}_\mathbf{k} + \text{len}_\mathbf{d}, 3)$;
- $F : \{0, 1\}^* \rightarrow \{0, 1\}^{\text{len}_{\mathbf{ss}}}$, a hash function, either $\text{cSHAKE128}(\cdot, \text{len}_{\mathbf{ss}}, 7)$ or $\text{cSHAKE256}(\cdot, \text{len}_{\mathbf{ss}}, 7)$.

Algorithm 12 FrodoKEM.KeyGen.

Input: None.

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

- 1: Choose uniformly random seeds $\mathbf{s} \parallel \text{seed}_\mathbf{E} \parallel \mathbf{z} \leftarrow_s U(\{0, 1\}^{\text{len}_\mathbf{s} + \text{len}_\mathbf{E} + \text{len}_\mathbf{z}})$
 - 2: Generate pseudorandom seed $\text{seed}_\mathbf{A} \leftarrow H(\mathbf{z})$
 - 3: Generate the matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times n}$ via $\mathbf{A} \leftarrow \text{Frodo.Gen}(\text{seed}_\mathbf{A})$
 - 4: Sample error matrix $\mathbf{S} \leftarrow \text{Frodo.SampleMatrix}(\text{seed}_\mathbf{E}, n, \bar{n}, T_\chi, 1)$
 - 5: Sample error matrix $\mathbf{E} \leftarrow \text{Frodo.SampleMatrix}(\text{seed}_\mathbf{E}, n, \bar{n}, T_\chi, 2)$
 - 6: Compute $\mathbf{B} \leftarrow \mathbf{A}\mathbf{S} + \mathbf{E}$
 - 7: Compute $\mathbf{b} \leftarrow \text{Frodo.Pack}(\mathbf{B})$
 - 8: **return** public key $pk \leftarrow \text{seed}_\mathbf{A} \parallel \mathbf{b}$ and secret key $sk' \leftarrow (\mathbf{s} \parallel \text{seed}_\mathbf{A} \parallel \mathbf{b}, \mathbf{S})$
-

Algorithm 13 FrodoKEM.Encaps.

Input: Public key $pk = \text{seed}_A \| \mathbf{b} \in \{0, 1\}^{\text{len}_A + D \cdot n \cdot \bar{n}}$.

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

- 1: Choose a uniformly random key $\mu \leftarrow_s U(\{0, 1\}^{\text{len}_\mu})$
 - 2: Generate pseudorandom values $\text{seed}_E \| \mathbf{k} \| \mathbf{d} \leftarrow G(pk \| \mu)$
 - 3: Sample error matrix $\mathbf{S}' \leftarrow \text{Frodo.SampleMatrix}(\text{seed}_E, \bar{m}, n, T_\chi, 4)$
 - 4: Sample error matrix $\mathbf{E}' \leftarrow \text{Frodo.SampleMatrix}(\text{seed}_E, \bar{m}, n, T_\chi, 5)$
 - 5: Generate $\mathbf{A} \leftarrow \text{Frodo.Gen}(\text{seed}_A)$
 - 6: Compute $\mathbf{B}' \leftarrow \mathbf{S}' \mathbf{A} + \mathbf{E}'$
 - 7: Compute $\mathbf{c}_1 \leftarrow \text{Frodo.Pack}(\mathbf{B}')$
 - 8: Sample error matrix $\mathbf{E}'' \leftarrow \text{Frodo.SampleMatrix}(\text{seed}_E, \bar{m}, \bar{n}, T_\chi, 6)$
 - 9: Compute $\mathbf{B} \leftarrow \text{Frodo.Unpack}(\mathbf{b}, n, \bar{n})$
 - 10: Compute $\mathbf{V} \leftarrow \mathbf{S}' \mathbf{B} + \mathbf{E}''$
 - 11: Compute $\mathbf{C} \leftarrow \mathbf{V} + \text{Frodo.Encode}(\mu)$
 - 12: Compute $\mathbf{c}_2 \leftarrow \text{Frodo.Pack}(\mathbf{C})$
 - 13: Compute $\mathbf{ss} \leftarrow F(\mathbf{c}_1 \| \mathbf{c}_2 \| \mathbf{k} \| \mathbf{d})$
 - 14: **return** ciphertext $\mathbf{c}_1 \| \mathbf{c}_2 \| \mathbf{d}$ and shared secret \mathbf{ss}
-

Algorithm 14 FrodoKEM.Decaps.

Input: Ciphertext $\mathbf{c}_1 \| \mathbf{c}_2 \| \mathbf{d} \in \{0, 1\}^{(\bar{m} \cdot n + \bar{m} \cdot \bar{n})D + \text{len}_d}$, secret key $sk' = (\mathbf{s} \| \text{seed}_A \| \mathbf{b}, \mathbf{S}) \in \{0, 1\}^{\text{len}_s + \text{len}_A + D \cdot n \cdot \bar{n}} \times \mathbb{Z}_q^{n \times \bar{n}}$.

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

- 1: $\mathbf{B}' \leftarrow \text{Frodo.Unpack}(\mathbf{c}_1)$
 - 2: $\mathbf{C} \leftarrow \text{Frodo.Unpack}(\mathbf{c}_2)$
 - 3: Compute $\mathbf{M} \leftarrow \mathbf{C} - \mathbf{B}' \mathbf{S}$
 - 4: Compute $\mu' \leftarrow \text{Frodo.Decode}(\mathbf{M})$
 - 5: Parse $pk \leftarrow \text{seed}_A \| \mathbf{b}$
 - 6: Generate pseudorandom values $\text{seed}'_E \| \mathbf{k}' \| \mathbf{d}' \leftarrow G(pk \| \mu')$
 - 7: Sample error matrix $\mathbf{S}' \leftarrow \text{Frodo.SampleMatrix}(\text{seed}'_E, \bar{m}, n, T_\chi, 4)$
 - 8: Sample error matrix $\mathbf{E}' \leftarrow \text{Frodo.SampleMatrix}(\text{seed}'_E, \bar{m}, n, T_\chi, 5)$
 - 9: Generate $\mathbf{A} \leftarrow \text{Frodo.Gen}(\text{seed}_A)$
 - 10: Compute $\mathbf{B}'' \leftarrow \mathbf{S}' \mathbf{A} + \mathbf{E}'$
 - 11: Sample error matrix $\mathbf{E}'' \leftarrow \text{Frodo.SampleMatrix}(\text{seed}'_E, \bar{m}, \bar{n}, T_\chi, 6)$
 - 12: Compute $\mathbf{B} \leftarrow \text{Frodo.Unpack}(\mathbf{b}, n, \bar{n})$
 - 13: Compute $\mathbf{V} \leftarrow \mathbf{S}' \mathbf{B} + \mathbf{E}''$
 - 14: Compute $\mathbf{C}' \leftarrow \mathbf{V} + \text{Frodo.Encode}(\mu')$
 - 15: **if** $\mathbf{B}' \| \mathbf{C} = \mathbf{B}'' \| \mathbf{C}'$ and $\mathbf{d} = \mathbf{d}'$ **then**
 - 16: **return** shared secret $\mathbf{ss} \leftarrow F(\mathbf{c}_1 \| \mathbf{c}_2 \| \mathbf{k}' \| \mathbf{d})$
 - 17: **else**
 - 18: **return** shared secret $\mathbf{ss} \leftarrow F(\mathbf{c}_1 \| \mathbf{c}_2 \| \mathbf{s} \| \mathbf{d})$
-

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 \mathbf{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 [44].

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 cSHAKE128/256 to denote the use of either cSHAKE128 or cSHAKE256; which one is used with which parameter set for FrodoKEM is indicated in Table 3. Every use of cSHAKE in different contexts listed below is performed with a different domain separator.

- *Gen*: The security requirement on *Gen* is that it is a public random function that generates pseudorandom matrices \mathbf{A} . *Gen* is instantiated using either AES128 (as in Algorithm 7) or cSHAKE128 (as in Algorithm 8).
- *H*: The security requirement on *H* is that it is a pseudorandom generator. *H* is instantiated using cSHAKE128/256 with the input being a uniformly random seed \mathbf{z} .
- *G*: The security requirement on *G* is that it is a pseudorandom function. *G* is instantiated using cSHAKE128/256 with the input being the public key pk concatenated with a random key μ .
- *F*: The security requirement on *F* is that it is a pseudorandom function. *F* is instantiated using cSHAKE128/256 with the input being the ciphertext components c_1 and c_2 , concatenated (during encapsulation and successful decapsulation) with the intermediate shared secret \mathbf{k} or (during failed decapsulation) the seed \mathbf{s} .
- cSHAKE: We also use cSHAKE directly in the SampleMatrix algorithm as a pseudorandom function. cSHAKE128/256 is used with the input being a seed $\text{seed}_{\mathbf{E}}$ and distinct context strings for the different matrices generated from the same seed.

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, \bar{m}, \bar{n} , integer matrix dimensions with $n \equiv 0 \pmod{8}$;
- $B \leq D$, the number of bits encoded in each matrix entry;
- $\ell = B \cdot \bar{m} \cdot \bar{n}$ the length of bit strings to be encoded in an \bar{m} -by- \bar{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 8. Our cost function is the bit length of the FrodoPKE ciphertext, which is $D \cdot \bar{m} \cdot (n + \bar{n})$.

The standard deviation σ of the Gaussian error distribution is bounded from below by the constant 2.12. This bound is chosen following the argument in Section 5.1.5, which demonstrates that $\sigma > 2.12$ conforms to a reduction from the worst-case BDDwDGS problem to the 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 5.1.3). 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 (minimizing the probability of decryption failure), and the larger σ for FrodoKEM-976 (prioritizing security).

2.4.3 Selected parameter sets

We present two parameter sets for FrodoKEM. The first, denoted Frodo-640, targets Level 1 in the NIST call for proposals (matching or exceeding the brute-force security of AES-128). The second, denoted Frodo-976, targets Level 3 (matching or exceeding the brute-force security of AES-192). At present, we are not proposing parameters for Level 5, the highest security target (matching or exceeding the brute-force security of AES-256).

In addition to targeting Level 5 and other security objectives, 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 two parameter sets are given in Table 1. The corresponding error distributions are given in Table 2. Security columns C and Q respectively denote security, in bits, for classical and quantum attacks as estimated by the methodology of Section 5.2.

Table 1: Parameter sets

	n	q	σ	support of χ	B	$\bar{m} \times \bar{n}$	failure prob.	c size (bytes)	Security C	Q
Frodo-640	640	2^{15}	2.75	$[-11 \dots 11]$	2	8×8	$2^{-148.8}$	9,736	143	103
Frodo-976	976	2^{16}	2.3	$[-10 \dots 10]$	3	8×8	$2^{-199.6}$	15,768	209	150

Table 2: Error distributions

	σ	Probability of (in multiples of 2^{-15})												Rényi	
		0	± 1	± 2	± 3	± 4	± 5	± 6	± 7	± 8	± 9	± 10	± 11	order	divergence
$\chi_{\text{Frodo-640}}$	2.75	9456	8857	7280	5249	3321	1844	898	384	144	47	13	3	500.0	0.72×10^{-4}
$\chi_{\text{Frodo-976}}$	2.3	11278	10277	7774	4882	2545	1101	396	118	29	6	1		500.0	0.14×10^{-4}

2.5 Summary of parameters

Table 3 summarizes all cryptographic parameters for Frodo-640 and Frodo-976. FrodoKEM-640-AES and FrodoKEM-976-AES use AES128 as Gen for generation of \mathbf{A} ; FrodoKEM-640-cSHAKE and FrodoKEM-976-cSHAKE use cSHAKE as Gen for generation of \mathbf{A} .

Table 4 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 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 3: Cryptographic parameters for Frodo-640 and Frodo-976

	Frodo-640	Frodo-976
D	15	16
q	32768	65536
n	640	976
$\bar{m} = \bar{n}$	8	8
B	2	3
$\text{len}_{\mathbf{A}}$	128	128
$\text{len}_{\mu} = \ell$	128	192
$\text{len}_{\mathbf{E}}$	128	192
$\text{len}_{\mathbf{z}}$	128	192
$\text{len}_{\mathbf{s}}$	128	192
$\text{len}_{\mathbf{k}}$	128	192
$\text{len}_{\mathbf{d}}$	128	192
len_{ss}	128	192
len_{χ}	16	16
χ	$\chi_{\text{Frodo-640}}$	$\chi_{\text{Frodo-976}}$
H	cSHAKE128(\cdot , 128, 0)	cSHAKE256(\cdot , 128, 0)
G	cSHAKE128(\cdot , 384, 3)	cSHAKE256(\cdot , 576, 3)
F	cSHAKE128(\cdot , 128, 7)	cSHAKE256(\cdot , 192, 7)

Table 4: 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	public key	ciphertext	shared secret
	sk	pk	c	ss
FrodoKEM-640	19,872 (10,256 + 9,616)	9,616	9,736	16
FrodoKEM-976	31,272 (15,640 + 15,632)	15,632	15,768	24

2.6 Provenance of constants and tables

Constants used as domain separators in calls to cSHAKE are integers selected incrementally starting at 0.

The constants in Table 2 and Table 1 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 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 four schemes: FrodoKEM-640-AES, FrodoKEM-640-cSHAKE, FrodoKEM-976-AES, and FrodoKEM-976-cSHAKE. 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 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 cSHAKE128.

3.2.1 Performance using AES128

Table 5 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 implementation⁴ 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 and FrodoKEM-976-AES (i.e., 1.1 ms and 2.1 ms, respectively, obtained by adding the times for encapsulation and decapsulation) are achieved by the optimized implementation using C only and by the additional implementation using AVX2 intrinsic instructions, respectively. However, the difference in performance between the different implementations reported in Table 5 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 26.4 ms (resp. 61.2 ms), which is roughly a 24-fold (resp. 29-fold) degradation in performance.

3.2.2 Performance using cSHAKE128

Table 6 outlines the performance figures of the optimized implementations and the additional x64 implementations when using cSHAKE128 for the generation of the matrix \mathbf{A} . The top set of results shows the performance of the optimized implementation written in C only, while the bottom set presents the results

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

Table 5: **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	1,287	1,810	1,811	3,621
FrodoKEM-976-AES	2,715	3,572	3,588	7,160
Additional implementation using AVX2 intrinsic instructions (AES from OpenSSL)				
FrodoKEM-640-AES	1,293	1,828	1,829	3,657
FrodoKEM-976-AES	2,663	3,565	3,580	7,145
Additional implementation using AVX2 intrinsic instructions (standalone AES)				
FrodoKEM-640-AES	1,288	1,834	1,837	3,671
FrodoKEM-976-AES	2,677	3,577	3,580	7,157

Table 6: **Performance (in thousands of cycles) of FrodoKEM on a 3.4GHz Intel Core i7-6700 (Skylake) processor with matrix A generated using cSHAKE128.** Results are reported for two test cases: (i) using a cSHAKE implementation written in plain C and, (ii) using a 4-way implementation of cSHAKE using AVX2 instructions. Cycle counts are rounded to the nearest 10^3 cycles.

Scheme	KeyGen	Encaps	Decaps	Total (Encaps + Decaps)
Optimized Implementation (plain C cSHAKE)				
FrodoKEM-640-cSHAKE	8,297	9,082	9,077	18,159
FrodoKEM-976-cSHAKE	17,798	19,285	19,299	38,584
Additional implementation using AVX2 intrinsics (cSHAKE4x using AVX2)				
FrodoKEM-640-cSHAKE	4,212	4,671	4,672	9,343
FrodoKEM-976-cSHAKE	8,822	9,749	9,720	19,469

when using a 4-way implementation of cSHAKE using AVX2 instructions (“cSHAKE4x using AVX2”). Note that the use of such vectorized implementation of cSHAKE is necessary to boost the practical performance. In our use-case, the usage of such a vectorized implementation results in a two-fold speedup when compared to the version using a cSHAKE implementation written in plain C.

Comparing Table 5 and Table 6, FrodoKEM using AES, when implemented with AES-NI instructions, is around 2.6–2.7× faster than the vectorized cSHAKE 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 7 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 7. Note that in our implementations the use of cSHAKE for generating **A** reduces peak memory usage in up to 26%. However, the vectorized AVX2 implementation of cSHAKE increases the size of the produced static libraries significantly (implementations based on AES-NI instructions are indeed very compact).

Table 7: **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 cSHAKE128 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	72,192	103,072	123,968	81,836
FrodoKEM-976-AES	111,424	159,136	189,176	79,700
Additional implementation using AVX2 intrinsics (standalone AES)				
FrodoKEM-640-AES	72,192	103,088	124,016	81,184
FrodoKEM-976-AES	111,424	157,832	189,176	79,048
Additional implementation using AVX2 intrinsics (cSHAKE4x using AVX2)				
FrodoKEM-640-cSHAKE	70,184	81,832	101,016	257,654
FrodoKEM-976-cSHAKE	105,560	124,856	156,248	255,398

3.3 Performance analysis on ARM

In this section, we summarize 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 8 details the performance of the optimized implementations when using AES128 and cSHAKE128. 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, cSHAKE performs significantly better when there is no support for specialized instructions in the targeted platform: using a plain C version of cSHAKE is more than 3 times faster than using a plain C version of AES.

Table 8: **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 cSHAKE 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	3,343	4,082	4,082	8,164
FrodoKEM-976-AES	7,056	8,428	8,382	16,810
Optimized implementation (plain C AES)				
FrodoKEM-640-AES	44,007	45,013	44,967	89,980
FrodoKEM-976-AES	100,996	102,074	102,323	204,397
Optimized implementation (plain C cSHAKE)				
FrodoKEM-640-AES	12,081	13,458	13,446	26,904
FrodoKEM-976-AES	26,125	28,735	28,506	57,241

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 (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_19872.rsp
FrodoKEM-976-AES	\KAT\PQCKemKAT_31272.rsp
FrodoKEM-640-cSHAKE	\KAT\PQCKemKAT_19872_cshake.rsp
FrodoKEM-976-cSHAKE	\KAT\PQCKemKAT_31272_cshake.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.

5.1 Security reductions

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

1. FrodoKEM is an IND-CCA-secure KEM under the assumption that FrodoPKE is an IND-CPA-secure public-key encryption scheme, and where G and F are modeled as random oracles. [Theorem 5.1](#) gives a tight, classical reduction against classical adversaries in the classical random oracle model. [Theorem 5.2](#) gives a non-tight, classical reduction against quantum adversaries in the quantum random oracle model.
2. FrodoPKE is an IND-CPA secure public key encryption scheme under the assumption that the corresponding normal-form learning with errors decision problem is hard. [Theorem 5.3](#) gives a tight, classical reduction against classical or quantum adversaries in the standard model.
3. [Section 5.1.3](#) provides justification and bounds exact security loss for substituting exact rounded Gaussian distributions with distributions from [Table 2](#).
4. [Section 5.1.4](#) provides an overview of the security reduction when replacing \mathbf{A} sampled from a truly uniform distribution with one generated in a pseudorandom fashion from a seed. The reduction models AES128 as an ideal cipher when considering Frodo.Gen with AES128 ([Algorithm 7](#)) and cSHAKE128 as a random oracle when considering Frodo.Gen with cSHAKE128 ([Algorithm 8](#)) and preserve the security up to a small multiplicative loss in the number of samples of the underlying LWE problem.
5. The normal-form learning with errors decision problem is hard 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. [Theorem 5.4](#) gives a tight, classical reduction against classical or quantum adversaries (in the standard model).
6. The (average-case) uniform-secret learning with errors decision problem, with the particular values of 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 problem with discrete Gaussian samples problem (BDDwDGS, [Definition 5.7](#)) is hard for related parameters. [Theorem 5.8](#) gives a non-tight classical reduction against classical or quantum adversaries (in the standard model).

5.1.1 IND-CCA Security of KEM

Theorem 5.1 (IND-CPA PKE \implies IND-CCA KEM in classical ROM). *Let PKE be a public key encryption scheme with algorithms (KeyGen, Enc, Dec), message space \mathcal{M} , and which is δ -correct. Let G and F be independent random oracles. Let $\text{QKEM}_m^{\mathcal{X}'} = \text{QFO}_m^{\mathcal{X}'}[\text{PKE}, G, F]$ be the KEM obtained by applying the $\text{QFO}_m^{\mathcal{X}'}$ transform as in [Definition 2.19](#). For any classical algorithm \mathcal{A} against the IND-CCA security of $\text{QKEM}_m^{\mathcal{X}'}$ that makes q_G and q_F queries to its G and F oracles, there exists a classical algorithm \mathcal{B} against the IND-CPA security of PKE such that*

$$\text{Adv}_{\text{QKEM}_m^{\mathcal{X}'}}^{\text{ind-cca}}(\mathcal{A}) \leq \frac{4 \cdot q_{\text{RO}} + 1}{|\mathcal{M}|} + q_{\text{RO}} \cdot \delta + 3 \cdot \text{Adv}_{\text{PKE}}^{\text{ind-cpa}}(\mathcal{B})$$

where $q_{\text{RO}} = q_G + q_F$. Moreover, the running time of \mathcal{B} is about that of \mathcal{A} .

[Theorem 5.1](#) follows from [Theorems 3.2 and 3.4](#) of Hofheinz, Hövelmanns, and Kiltz (HHK) [[61](#)], with the following modifications. In the application of HHK's [Theorem 3.2](#), we take $q_V = 0$. Note that [Theorems 3.2 and 3.4](#) of HHK are about the $\text{FO}^{\mathcal{X}}$ transform, which differs from the $\text{QFO}_m^{\mathcal{X}'}$ in the following ways.

1. $\text{QFO}_m^{\mathcal{X}'}$ uses a single hash function (with longer output) to compute r and K whereas $\text{FO}^{\mathcal{X}}$ uses two; but this is equivalent in the random oracle model with appropriate output lengths.
2. $\text{QFO}_m^{\mathcal{X}'}$'s computation of r and K also takes the public key pk as input whereas $\text{FO}^{\mathcal{X}}$ does not; this does not negatively affect any of the theorems, and has the potential to provide multi-target security.
3. $\text{QFO}_m^{\mathcal{X}'}$ includes the d value in the ciphertext, whereas $\text{FO}^{\mathcal{X}}$ does not; since d is computed by applying a random oracle G to the secret $\mu \in \mathcal{M}$, taking advantage of d requires querying G on μ , which occurs with the additional $q_{\text{RO}}/|\mathcal{M}|$ probability term added in the theorem.

Theorem 5.2 (IND-CPA PKE \implies IND-CCA KEM in quantum ROM). *Let PKE be a public key encryption scheme with algorithms (KeyGen, Enc, Dec), message space \mathcal{M} , and which is δ -correct. Let G and F be independent random oracles. Let $\text{QKEM}_m^{\mathcal{X}'} = \text{QFO}_m^{\mathcal{X}'}[\text{PKE}, G, F]$ be the KEM obtained by applying the $\text{QFO}_m^{\mathcal{X}'}$ transform as in Definition 2.19. For any quantum algorithm \mathcal{A} against the IND-CCA security of $\text{QKEM}_m^{\mathcal{X}'}$ that makes q_G and q_F queries to its quantum G and F oracles, there exists a quantum algorithm \mathcal{B} against the IND-CPA security of PKE such that*

$$\text{Adv}_{\text{QKEM}_m^{\mathcal{X}'}}^{\text{ind-cca}}(\mathcal{A}) \leq 9 \cdot q_{\text{RO}} \cdot \sqrt{q_{\text{RO}}^2 \cdot \delta + q_{\text{RO}} \cdot \sqrt{\text{Adv}_{\text{PKE}}^{\text{ind-cpa}}(\mathcal{B}) + \frac{1}{|\mathcal{M}|}}}$$

where $q_{\text{RO}} = q_G + q_F$. Moreover, the running time of \mathcal{B} is about that of \mathcal{A} .

Theorem 5.2 follows from Lemma 2.3 and Theorems 4.4 and 4.6 of Hofheinz, Hövelmanns, and Kiltz [61], with the following modifications. Note that Theorems 4.4 and 4.6 of HHK are about the $\text{QFO}_m^{\mathcal{X}}$ transform, which differs from the $\text{QFO}_m^{\mathcal{X}'}$ in the following ways. 1) $\text{QFO}_m^{\mathcal{X}'}$ uses a single hash function (with longer output) to compute r , K , and d whereas $\text{FO}^{\mathcal{X}}$ uses two; but this is equivalent in the random oracle model with appropriate output lengths. 2) $\text{QFO}_m^{\mathcal{X}'}$'s computation of r , K , and d also takes the public key pk as input whereas $\text{FO}^{\mathcal{X}}$ does not; this does not negatively affect any of the theorems, and has the potential to provide multi-target security. 3) $\text{QFO}_m^{\mathcal{X}'}$'s computation of the shared secret k also takes the encapsulation c as input; this does not negatively affect any of the theorems, and provides robustness against ciphertext modification.

Note that Theorem 5.2 is far from being tight due to the fourth-root between the $\text{Adv}_{\text{PKE}}^{\text{ind-cpa}}(\mathcal{B})$ term and the $\text{Adv}_{\text{QKEM}_m^{\mathcal{X}'}}^{\text{ind-cca}}(\mathcal{A})$ term. As of writing there is no known attack that takes advantage of the tightness gap, and so it seems to be an artifact of the proof technique. As noted in Section 2.2.8, a recent eprint of Jiang et al. [63] presents an alternative transform from IND-CPA PKE to IND-CCA KEM in the quantum random oracle model, and their theorems give only a square-root gap between the advantages; our IND-CPA PKE satisfies the conditions needed for their Theorem 1 to yield IND-CCA security of the resulting KEM. In our parameter selection, we ignore the tightness gap arising from Theorem 5.2.

5.1.2 IND-CPA Security of PKE

Theorem 5.3 (Normal Form DLWE \implies IND-CPA security of FrodoPKE). *Let n, q, \bar{m}, \bar{n} be positive integers, and χ be a probability distribution on \mathbb{Z} . For any quantum algorithm \mathcal{A} against the IND-CPA security of FrodoPKE (with a uniformly random \mathbf{A}), there exist quantum algorithms \mathcal{B}_1 and \mathcal{B}_2 against the normal-form LWE decision problem such that*

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

Moreover, the running times of \mathcal{B}_1 and \mathcal{B}_2 are approximately that of \mathcal{A} .

The proof of Theorem 5.3 is the same as that of [75, Theorem 3.2] or [24, Theorem 5.1].

Theorem 5.4 (uniform-secret DLWE \implies normal-form DLWE). *Let n, q, m be integers, and χ be a probability distribution on \mathbb{Z} . For any quantum algorithm \mathcal{A} against the normal-form LWE decision problem, there exists a quantum algorithm \mathcal{B} against the uniform-secret LWE decision problem such that*

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

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

The proof of Theorem 5.4 is the same as that of [13, Lemma 2].

5.1.3 Approximating the error distribution

The discrete Gaussian distribution (Definition 2.14), whose properties are key to the worst-to-average-case reduction, is difficult to sample from on a finite computer (and impossible to do so in constant time). Following Langlois et al. [73], we replace an infinite-precision distribution with its discrete approximation and quantify the loss of security by computing the Rényi divergence between the two distributions.

Definition 5.5 (Rényi divergence). *Rényi divergence of order α between two discrete distributions P and Q is defined as*

$$D_\alpha(P\|Q) = \frac{1}{\alpha - 1} \ln \sum_{x \in \text{supp } P} P(x) \left(\frac{P(x)}{Q(x)} \right)^{\alpha-1}.$$

(Note that our definition differs from Langlois et al. in that we take the logarithm of the sum.)

The following theorem relates probabilities of observing a certain event under two distributions as a function of their Rényi divergence.

Theorem 5.6 ([73, Lemma 4.1]). *If there is an event S defined in a game G_Q where n samples are drawn from distribution Q , the probability of S in the same game where Q is replaced with P is bounded as follows:*

$$\Pr[G_P(S)] \leq (\Pr[G_Q(S)] \cdot \exp(n \cdot D_\alpha(P\|Q)))^{1-1/\alpha}. \quad (4)$$

Reduction to any *search* problem, such as the ones that appear in the proof of [Theorem 5.3](#) (specifically, winning in the OW-PCVA game as defined in [\[61\]](#)) are preserved subject to the relaxation (4). For each exact security argument, and any concrete choice of the two distributions P and Q , the bound can be minimized by choosing an optimal value of the Rényi order α .

For a concrete example of application of [Theorem 5.6](#) consider the distribution $\chi_{\text{Frodo-640}}$ specified according to [Table 2](#). The distribution approximates the rounded Gaussian $\Psi_{2.75/\sqrt{2\pi}}$ as defined in [Section 2.1.4](#). During a single run of FrodoKEM the parties sample from the distribution $(8+8) \times 640 + 64 = 10,304$ times. Assume that the adversary attacking the underlying search problem (of recovering the shared secret key *before* applying the random oracle) has probability of winning 2^{-145} when the parties sample from $\Psi_{2.75/\sqrt{2\pi}}$. According to [Table 2](#) the Rényi divergence between the two distributions is $D_{500}(\chi_{\text{Frodo-640}}\|\Psi_{2.75/\sqrt{2\pi}}) = .000074$. Substituting the rounded Gaussian distribution with $\chi_{\text{Frodo-640}}$ and applying [Theorem 5.6](#) will lead to the following bound on classical security of the resulting protocol (cf. [Table 1](#)): $(2^{-145} \cdot \exp(10304 \cdot .000074))^{.998} \approx 2^{-143.6}$.

5.1.4 Deterministic generation of \mathbf{A}

The matrix \mathbf{A} in FrodoKEM is deterministically expanded from a short random seed in the function `Frodo.Gen` either using AES128 or cSHAKE128. In order to relate FrodoKEM’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 indistinguishability framework [\[80, 41\]](#).

Informally, a construction \mathcal{C} with access to an ideal primitive \mathcal{G} is said to be ε -indistinguishable 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 $|\Pr[\mathcal{D}^{\mathcal{C}, \mathcal{G}} = 1] - \Pr[\mathcal{D}^{\mathcal{F}, \mathcal{S}} = 1]| < \varepsilon$. An indistinguishability 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 $\mathcal{C}^{\mathcal{G}}$ [\[80\]](#). 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 cSHAKE128, modeled below as an ideal cipher and an ideal extendable-output function (XOF) respectively.

Using AES128 to generate \mathbf{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 i th block are generated by encrypting a fixed index idx_i with a uniformly random seed $\text{seed}_{\mathbf{A}} \in \{0, 1\}^{128}$ as the key. Throughout, we refer to the set $\text{Idx} := \{\text{idx}_1, \dots, \text{idx}_M\}$ 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 $\text{seed}_{\mathbf{A}} \in \{0, 1\}^{128}$ to a larger bit string $s_1 \| s_2 \| \dots \| s_M \in \{0, 1\}^{128M}$ 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 $\text{AES128}_{\text{seed}_{\mathbf{A}}}(\text{id}_{x_1}) \parallel \cdots \parallel \text{AES128}_{\text{seed}_{\mathbf{A}}}(\text{id}_{x_M})$. We show that $\mathcal{C}^{\mathcal{G}}$ is indistinguishable 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 $\text{AES128}_k(\text{id}_{x_1}) \parallel \cdots \parallel \text{AES128}_k(\text{id}_{x_M})$. Queries to \mathcal{G} are answered naturally with $\text{AES128}_{(\cdot)}(\cdot)$ or $\text{AES128}_{(\cdot)}^{-1}(\cdot)$ 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, \dots, s_M subject to no collisions and outputs $\mathcal{F}(k) = s_1 \parallel \cdots \parallel s_M$. It additionally stores a mapping M_k from $\{\text{id}_{x_1}, \dots, \text{id}_{x_M}\}$ to $S = \{s_1, \dots, s_M\}$. 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 (id_{x_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. Indistinguishability of the two worlds follows by construction as $\text{AES128}_{(\cdot)}$ is modeled as an ideal cipher. Thus, in generating \mathbf{A} starting with a seed $\text{seed}_{\mathbf{A}}$ using AES128, we can effectively replace the ideal domain extension primitive \mathcal{F} with our construction in the ideal cipher model.

Using cSHAKE128 to generate \mathbf{A} . An argument in using cSHAKE128 to expand $\text{seed}_{\mathbf{A}}$ to the matrix \mathbf{A} is significantly simpler. In the random oracle model, cSHAKE128 is an ideal XOF [49]. In fact, for every customization string str , we can model $\text{cSHAKE128}(\cdot, \ell, str)$ as an independent hash function mapping $\{0, 1\}^{128}$ to $\{0, 1\}^{\ell}$. The domain expansion step is constructed by computing $\text{cSHAKE128}(\text{seed}_{\mathbf{A}}, 16n, 2^8 + i)$ for $1 \leq i \leq n$ where each step fills up the i th row of the matrix \mathbf{A} . As each row is independently constructed via an ideal hash function, this construction maps a uniformly random seed $\text{seed}_{\mathbf{A}}$ to a much larger uniformly random matrix \mathbf{A} thereby implementing the ideal functionality \mathcal{F} perfectly.

Reusing \mathbf{A} . Finally, we point out that generating \mathbf{A} from $\text{seed}_{\mathbf{A}}$ can be a significant computational burden, but this cost can be amortized by relaxing the requirement that a fresh $\text{seed}_{\mathbf{A}}$ 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 \mathbf{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.

5.1.5 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., [102, 88, 28, 94]. 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 [102] and a follow-up work [94] 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 $m = \text{poly}(n)$ 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 2) are somewhat smaller than required by these reductions.

Instead, following [102, 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 $\sigma = \Theta(\sqrt{n})$, the alternative reduction supports a smaller error width—as small as the “smoothing parameter” [84] of the lattice of integers \mathbb{Z} . For the BDD variant we consider, which we call “BDD with Discrete Gaussian Samples” (BDDwDGS), the input additionally includes discrete Gaussian samples over the dual lattice, but having a larger width than known algorithms are able to exploit [77, 45]. 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, 77, 45] and recall the relevant aspects of known algorithms for the problem.

Definition 5.7 (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 $\text{BDDwDGS}_{\mathcal{L},d,r}$ is a point $\mathbf{t} \in \mathbb{R}^n$ such that $\text{dist}(\mathbf{t}, \mathcal{L}) \leq d$, 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 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 5.7 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 [90, Theorem 3.1] for further details.

The state-of-the-art algorithms for solving BDDwDGS [2, 77, 45] deal with a certain \mathcal{L} -periodic function $f_{\mathcal{L},1/r} : \mathbb{R}^n \rightarrow [0, 1]$, defined as

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

where the equality on the right follows from the Fourier series of $f_{\mathcal{L},1/r}$ (see [2]). To solve 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

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

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

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

⁵The 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.

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 BDDwDGS are limited to distance

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

and having such samples does not seem to provide any speedup in decoding at somewhat distances that are larger 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 $N = \text{poly}(n)$ samples does not seem to provide any help in solving BDDwDGS $_{\mathcal{L},d,r}$ (versus having no samples at all).

Reduction from BDDwDGS to LWE. We now recall the following result from [94], which generalizes a key theorem from [102] to give a reduction from BDDwDGS to the LWE decision problem.

Theorem 5.8 (BDDwDGS hard \implies decision-LWE hard [94, Lemma 5.4]). *Let $\varepsilon = \varepsilon(n)$ be a negligible function and let $m = \text{poly}(n)$ and $C = C(n) > 1$ be arbitrary. There is a probabilistic polynomial-time (classical) algorithm that, given access to an oracle that solves 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 BDDwDGS $_{\mathcal{L},d,r}$ using $N = m \cdot \text{poly}(n)$ 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 [102, Section 3.2.1]), using [102, 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 BDDwDGS $_{\mathcal{L},d,r}$ problem is plausibly hard in the worst case, hence so is the corresponding LWE problem from Theorem 5.8.

Concretely, if we use an extremely large bound $N \leq 2^{256}$ on the number of discrete Gaussian samples, then the threshold for Gaussian parameters αq that conform to Theorem 5.8 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 FrodoPKE parameters, which use standard deviation $\sigma \geq 2.3$ (see Table 2), are above this threshold by a comfortable margin. (Note that a standard deviation threshold of 2.3 corresponds to $N \approx 2^{300}$.)

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 [11], with slight modifications taking into account the fact that some quasi-linear accelerations [107, 26] over sieving algorithms [16, 69] are not available without the ring structure.

We also remark that this methodology is significantly more conservative than what is usually used in the literature [10], 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., $m_{\text{samp}} \approx n$ in our schemes) we are not concerned with either BKW types of attacks [65] or linearization attacks [14]. This essentially leaves us with two BKZ [40] 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 [39]. To account for further improvement, we shall count only the cost of one such call to the SVP oracle: the core-SVP hardness. Such

⁶In 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.

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 [53, 40]. 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 cycles⁷), and to make sure that the block-size b is in a range where enumeration costs more than 2^{cb} . From the estimates of [40], it is argued in [11] that this is the case both classically and quantumly whenever $b \geq 200$.

The best known constant in the exponent is for classical algorithms is $c_C = \log_2 \sqrt{3/2} \approx 0.292$, as provided by the sieve algorithm of [16]. For quantum algorithms it is $c_Q = \log_2 \sqrt{13/9} \approx 0.265$ [69, Sec. 14.2.10]. Because all variants of the sieve algorithm require building a list of $\sqrt{4/3}^b$ many vectors, the constant $c_P = \log_2 \sqrt{4/3} \approx .2075$ can plausibly serve as a “worst-possible” lower bound for sieving algorithm.

Conservatism: lower bounds vs. experiments. These estimates are very conservative compared to the state of the art implementation of [79], 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 [16] has been shown to reach lower bounds for the generic nearest-neighbor search problem [12]. As for concrete improvements, we note that this algorithm has already been subject to some fine-tuning in [79], 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.

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 [16] 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 [17], Becker et al. [16] also claims 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 [71]. Again, the conversion to an efficient quantum circuit will induce extra costs—at least polynomial ones.

⁷Because of the additional ring-structure, [11] chooses to ignore this factor b to the advantage of the adversary, assuming the techniques of [107, 26] can be adapted to more advanced sieve algorithms [11]. But for plain LWE, we can safely include this factor.

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 $\Lambda = \{\mathbf{x} \in \mathbb{Z}^{m+n+1} : (\mathbf{A} | -\mathbf{I}_m | -\mathbf{b})\mathbf{x} = 0 \pmod{q}\}$ 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} in our case and we numerically optimize this choice.

Using the typical models of BKZ (geometric series assumption, Gaussian heuristic [39, 10]) one concludes that the primal attack is successful if and only if

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

We note that this condition, introduced in [11], is substantially different from the one suggested in [52] and is used in many previous security analyses, such as [10]. The recent study [9] 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 vector in the dual lattice $\mathbf{w} \in \hat{\Lambda} = \{(\mathbf{x}, \mathbf{y}) \in \mathbb{Z}^m \times \mathbb{Z}^n : \mathbf{A}^t \mathbf{x} = \mathbf{y} \pmod{q}\}$ and intends to use it as a distinguisher for LWE. The algorithm BKZ with block size b will output such a vector of length $\ell = \delta^{d-1} q^{n/d}$.

Having found $(\mathbf{x}, \mathbf{y}) \in \hat{\Lambda}$ of length ℓ , an attacker computes $z = \mathbf{v}^t \cdot \mathbf{b} = \mathbf{v}^t \mathbf{A}\mathbf{s} + \mathbf{v}^t \mathbf{e} = \mathbf{w}^t \mathbf{s} + \mathbf{v}^t \mathbf{e} \pmod{q}$ which is distributed as a Gaussian of standard deviation $\ell\sigma$ if (\mathbf{A}, \mathbf{b}) is indeed an LWE sample (otherwise it is uniform mod q). Those two distributions have maximal variation distance bounded by $\varepsilon = \exp(-\pi\tau^2)$ where $\tau = \ell\sigma/q$: given such a vector of length ℓ the attacker may distinguish LWE samples from random with advantage at most ε .

Note that small advantages ε are not meaningful to attack a key exchange protocol: as the agreed key is to be used as a symmetric cipher key, any advantage below $1/2$ does not significantly decrease the search space to bruteforce the symmetric cipher.⁸

We therefore require the attacker to amplify his success probability by building about $1/\varepsilon^2$ many such short vectors. Because the sieve algorithms provide 2^{2075b} vectors, the attack must be repeated at least R times where

$$R = \max(1, 1/(2^{2075b} \varepsilon^2)).$$

This is again quite pessimistic, as the other vectors outputted by the sieving algorithm are a bit larger than the shortest one.

⁸To make this more formal, one may simply hash the agreed key with a random oracle before using it for any other purposes.

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. For example, the additional x64 implementation of the full KEM scheme accompanying this submission consists of slightly more than 250 lines of plain C code.⁹ This same code is used for the two security levels FrodoKEM-640 and FrodoKEM-976, 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 does 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, FrodoKEM’s communication size is sufficiently small that it is still compatible with many existing deployments. In our original research paper on FrodoCCS [24], 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 [24]. 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+ECDHE¹⁰. The number of connections (with 1 KiB HTTP payload) supported per second with 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 its 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

⁹This count does not include header files and the additional symmetric primitives.

¹⁰Note that the results in [24] use a different parameter set than in this proposal which had slightly larger communication (22.1 KiB in [24] 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 [24]; and used somewhat different symmetric primitives. Nonetheless the results provide some indication of suitability.

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 [24]—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., [59, 97, 98, 106]) 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. [100] and the related literature mentioned therein). Hence, the proposed schemes FrodoKEM-640 and FrodoKEM-976 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 [68]) or electromagnetic usage (e.g., in a differential electromagnetic analysis (DEMA) attack [54]) 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. [67]) there are also active attacks which might *inject faults* [21, 19] 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 [34] have been studied to protect implementations such as in [87, 104, 105].

In a more recent work [99] it is shown how to perform a single trace attack on ring LWE encryption using side-channel template matching [35]. 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 FrodoKEM-640 and FrodoKEM-976 do not use FFT-based multiplication techniques (the point of attack used in [99]), 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. Z. Béguelin, and P. Zimmermann. Imperfect forward secrecy: How Diffie-Hellman fails in practice. In I. Ray, N. Li, and C. Kruegel, editors, *ACM CCS 15: 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. 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, 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, 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.
- [8] 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.
- [9] M. R. Albrecht, F. Göpfert, F. Virdia, and T. Wunderer. Revisiting the expected cost of solving uSVP and applications to LWE. Cryptology ePrint Archive, Report 2017/815, 2017. <http://eprint.iacr.org/2017/815>.
- [10] 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.
- [11] E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe. Post-quantum key exchange - a new hope. In *25th USENIX Security Symposium*, pages 327–343, 2016.
- [12] 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.
- [13] 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.
- [14] 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.

- [15] 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.
- [16] 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.
- [17] 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>.
- [18] D. J. Bernstein, T. Chou, C. Chuengsatiansup, A. Hülsing, E. Lambooi, 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.
- [19] 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.
- [20] 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.
- [21] 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.
- [22] 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.
- [23] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, and D. Stehlé. CRYSTALS – kyber: a CCA-secure module-lattice-based KEM. Cryptology ePrint Archive, Report 2017/634, 2017. <http://eprint.iacr.org/2017/634>.
- [24] 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 16: 23rd Conference on Computer and Communications Security*, pages 1006–1018. ACM Press, Oct. 2016.
- [25] 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.
- [26] J. W. Bos, M. Naehrig, and J. van de Pol. Sieving for shortest vectors in ideal lattices: a practical perspective. *International Journal of Applied Cryptography*, 2016. to appear, <http://eprint.iacr.org/2014/880>.
- [27] 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.
- [28] 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.

- [29] 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.
- [30] 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.
- [31] 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_CRYPTOS07_Systems_and_Attacks/S07_Groves_Annex.pdf.
- [32] 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.
- [33] 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.
- [34] 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.
- [35] 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.
- [36] S. Chatterjee, N. Kobitz, 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.
- [37] H. Chen, K. Lauter, and K. E. Stange. Attacks on search RLWE. Cryptology ePrint Archive, Report 2015/971, 2015. <http://eprint.iacr.org/2015/971>.
- [38] H. Chen, K. Lauter, and K. E. Stange. Vulnerable galois RLWE families and improved attacks. Cryptology ePrint Archive, Report 2016/193, 2016. <http://eprint.iacr.org/2016/193>.
- [39] Y. Chen. *Lattice reduction and concrete security of fully homomorphic encryption*. PhD thesis, l’Université Paris Diderot, 2013. <http://www.di.ens.fr/~ychen/research/these.pdf>.
- [40] 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.
- [41] 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.
- [42] 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.
- [43] R. Cramer, L. Ducas, and B. Wesolowski. Short stickelberger class relations and application to ideal-SVP. In J. 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, May 2017.
- [44] 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.

- [45] 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.
- [46] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.
- [47] 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>.
- [48] 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.
- [49] 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.
- [50] 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.
- [51] 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.
- [52] 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.
- [53] 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 2010.
- [54] 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.
- [55] 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.
- [56] 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.
- [57] 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>.
- [58] 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.
- [59] 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.

- [60] J. Hoffstein, J. Pipher, and J. H. Silverman. NTRU: A ring-based public key cryptosystem. In *ANTS*, pages 267–288, 1998.
- [61] 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.
- [62] 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.
- [63] H. Jiang, Z. Zhang, L. Chen, H. Wang, and Z. Ma. Post-quantum IND-CCA-secure KEM without additional hash. Cryptology ePrint Archive, Report 2017/1096, 2017. <https://eprint.iacr.org/2017/1096>.
- [64] J. Kelsey, S. Chang, and R. Perlner. SHA-3 derived functions: cSHAKE, KMAC, TupleHash and ParallelHash, 2016. NIST Special Publication 800-185. <http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-185.pdf>.
- [65] 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.
- [66] P. Kirchner and P.-A. Fouque. Revisiting lattice attacks on overstretched NTRU parameters. In J. 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, May 2017.
- [67] P. C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In N. Kobitz, editor, *Advances in Cryptology – CRYPTO’96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, Heidelberg, Aug. 1996.
- [68] 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.
- [69] T. Laarhoven. *Search problems in cryptography*. PhD thesis, Eindhoven University of Technology, 2015.
- [70] 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.
- [71] 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.
- [72] A. Langlois and D. Stehlé. Worst-case to average-case reductions for module lattices. *Designs, Codes and Cryptography*, 75(3):565–599, 2015.
- [73] 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.
- [74] A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, December 1982.
- [75] 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.

- [76] 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.
- [77] 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.
- [78] 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.
- [79] 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.
- [80] 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.
- [81] 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.
- [82] D. Micciancio. Cryptographic functions from worst-case complexity assumptions. *Information Security and Cryptography*, pages 427–452. Springer, Heidelberg, 2010.
- [83] 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.
- [84] 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.
- [85] D. Micciancio and O. Regev. Lattice-based cryptography. In *Post Quantum Cryptography*, pages 147–191. Springer, February 2009.
- [86] P. Q. Nguyen and O. Regev. Learning a parallelepiped: Cryptanalysis of GGH and NTRU signatures. *Journal of Cryptology*, 22(2):139–160, Apr. 2009.
- [87] T. Oder, T. Schneider, T. Pöppelmann, and T. Güneysu. Practical CCA2-secure and masked ring-LWE implementation. *Cryptology ePrint Archive*, Report 2016/1109, 2016. <http://eprint.iacr.org/2016/1109>.
- [88] 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.
- [89] 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>.
- [90] 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.
- [91] C. Peikert. Lattice cryptography for the Internet. In M. Mosca, editor, *PQCrypto 2014*, volume 8772 of *Lecture Notes in Computer Science*, pages 197–219. Springer, Heidelberg, 2014.

- [92] C. Peikert. A decade of lattice cryptography. *Foundations and Trends in Theoretical Computer Science*, 10(4):283–424, 2016.
- [93] 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.
- [94] 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.
- [95] 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.
- [96] 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.
- [97] 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.
- [98] 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.
- [99] 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.
- [100] 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.
- [101] O. Regev. New lattice-based cryptographic constructions. *J. ACM*, 51(6):899–942, 2004. Preliminary version in STOC 2003.
- [102] 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.
- [103] O. Regev. The learning with errors problem (invited survey). In *IEEE Conference on Computational Complexity*, pages 191–204, 2010.
- [104] 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.
- [105] 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.
- [106] 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.

- [107] 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.
- [108] C.-P. Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theoretical Computer Science*, 53:201–224, 1987.
- [109] 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.
- [110] 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.
- [111] 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.