Formally verifying Kyber

José Bacelar Almeida, Santiago Arranz Olmos, Manuel Barbosa, Gilles Barthe, François Dupressoir, Benjamin Grégoire, Vincent Laporte, Jean-Christophe Léchenet, Cameron Low, Tiago Oliveira, et al.

Formally verifying Kyber

Episode V: Machine-checked IND-CCA security and correctness of ML-KEM in EasyCrypt

José Bacelar Almeida[1,2], Santiago Arranz Olmos[3], Manuel Barbosa[2,3,4], Gilles Barthe[3,5], François Dupressoir[6], Benjamin Grégoire[7], Vincent Laporte[8], Jean-Christophe Léchenet[7], Cameron Low[6], Tiago Oliveira[9], Hugo Pacheco[2,4], Miguel Quaresma[3], Peter Schwabe[3,10], and Pierre-Yves Strub[12]

[1] Universidade do Minho, Portugal
[2] INESC TEC, Porto, Portugal
jba@di.uminho.pt, mbb@fc.up.pt, hpacheco@fc.up.pt
[3] Max Planck Institute for Security and Privacy, Bochum, Germany
santiago.arranz-olmos@mpi-sp.org, gilles.barthe@mpi-sp.org, miguel.quaresma@mpi-sp.org, peter@cryptojedi.org
[4] University of Porto (FCUP), Portugal
[5] IMDEA Software Institute, Spain
[6] University of Bristol, UK
f.dupressoir@bristol.ac.uk, cameron.low.2018@bristol.ac.uk
[7] Université Côte d'Azur, Inria, France
benjamin.gregoire@inria.fr, jean-christophe.lechenet@inria.fr
[8] Université de Lorraine, CNRS, Inria, LORIA, France, vincent.laporte@inria.fr
[9] SandboxAQ, USA, tiago.oliveira@sandboxquantum.com
[10] Radboud University, Nijmegen, The Netherlands
[11] PQShield, France, pierre-yves.strub@pqshield.com

Abstract. We present a formally verified proof of the correctness and IND-CCA security of ML-KEM, the Kyber-based Key Encapsulation Mechanism (KEM) undergoing standardization by NIST. The proof is machine-checked in EasyCrypt and it includes: 1) A formalization of the correctness (decryption failure probability) and IND-CPA security of the Kyber base public-key encryption scheme, following Bos et al. at Euro S&P 2018; 2) A formalization of the relevant variant of the Fujisaki-Okamoto transform in the Random Oracle Model (ROM), which follows closely (but not exactly) Hofheinz, Hövelmanns and Kiltz at TCC 2017; 3) A proof that the IND-CCA security of the ML-KEM specification and its correctness as a KEM follows from the previous results; 4) Two formally verified implementations of ML-KEM written in Jasmin that are provably constant-time, functionally equivalent to the ML-KEM specification and, for this reason, inherit the provable security guarantees established in the previous points. The top-level theorems give self-contained concrete bounds for the correctness and security of ML-KEM down to (a variant of) Module-LWE. We discuss how they are built modularly by leveraging various EasyCrypt features.

1 Introduction

The Module-Lattice-Based Key Encapsulation Mechanism (ML-KEM) is a post-quantum key-encapsulation mechanism that is currently being standardized by NIST as future standard FIPS-203 [54]. ML-KEM is based on Kyber [10,25] which was selected by NIST in 2022 after round 3 of their multi-year post-quantum cryptography standardization process [1]. Following NIST's announcement, major cryptographic libraries and widely used applications are already transitioning to post-quantum KEMs. Notably, Cloudflare, Google Chrome, and Signal are already using Kyber. It is expected that, once the FIPS-203 standard is ready, these deployments will migrate from Kyber to ML-KEM—for details of the differences between the two see below—and that the number of applications using ML-KEM will rapidly increase.

In this context, it is critically important to ensure that implementations of ML-KEM achieve the targeted security. At a high level, this means that we need to ensure the correctness of security reductions that link the IND-CCA security of Kyber to the module learning with errors (MLWE) problem; we need to ensure that implementations actually match the algorithm specification that has been proven secure; we need to give guarantees about implementation security; and finally we need to make sure that the underlying MLWE problem is indeed hard to solve for the concrete parameters used in ML-KEM.

In this paper we embrace the methods and tools of computer-aided cryptography [11] to give rigorous statements with regards to the first three aspects of this analysis. Concretely, we present a computer-verified proof of IND-CCA security of the ML-KEM draft standard specification down to a variant of

the MLWE assumption. We furthermore connect the machine-readable version of the ML-KEM draft specification used in this proof to two efficient implementations in the Jasmin programming language [5,7]. These implementations are proven functionally correct with respect to the specification and are proven to follow the "constant-time" programming paradigm. Overall, we obtain machine-verified proofs of ML-KEM reaching from the IND-CCA security notion down to the assembly level of highly optimized constant-time software.

The proofs are formalized and machine-checked in the EasyCrypt proof assistant [18]. EasyCrypt is tailored to code-based game-playing cryptographic proofs and supports many common proof techniques, including equivalence up to failure events, lazy sampling, hybrid arguments, and plug and pray. It has been used for proving security of a broad range of cryptographic constructions, and to connect security proofs with implementations (see related work below).

Security Proof. Our main theorem intuitively states the following—the full statement of Theorem 6 is given in Section 6.

Theorem 6 (Informal). *The ML-KEM specification is an IND-CCA secure KEM when the key-derivation hash is modeled as a random oracle, under a variant of the MLWE assumption where matrices are sampled pseudorandomly using SHAKE-128 and assuming that SHAKE-256 behaves as a pseudorandom function. The failure probability of ML-KEM can be bounded as described in [25].*

Our proof consolidates and strengthens prior work in three dimensions: first, our proof is fully machine-checked, and as such provides higher guarantees than pen-and-paper proofs. Second, our proof applies to a (machine-readable) specification of the very recently published ML-KEM draft standard. Third, our proof clarifies the underlying computational assumptions for ML-KEM and streamlines the overall structure of the security analysis. In particular, our work provides a machine-checked proof of the relevant variant of the Fujisaki-Okamoto transform, which we instantiate for the particular case of ML-KEM with concrete parameters. To this end, we lay down and verify all the details related to sampling noise using the non-trivial pseudorandom generation algorithms specified by ML-KEM, under the assumption that various algorithms in the SHA-3 family can be used as pseudorandom functions.

Technically, the security proof is decomposed in two steps. In the first step, we prove security of an abstract ML-KEM specification written in the EasyCrypt proof assistant. This applies to all ML-KEM variants. In the second step, we instantiate our abstract result with a fully concrete specification of the ML-KEM draft standard, with parameters fixed for ML-KEM-768. The final theorem provides a detailed bound for the IND-CCA security of the construction, showing that it is tightly related to a variant of the MLWE problem. This variant of MLWE is tailored to allow plugging in the security proof of the base PKE into the FO transform.[12] In the end our theorem reflects the practical security of ML-KEM-768 closely: MLWE is assumed to be hard when the underlying matrix is sampled exactly as prescribed in the draft standard. For a detailed discussion, see Sections 4 and 5. Extending these results to other ML-KEM variants is straightforward and will be done when verified implementations are available.

Functional Correctness. It is notoriously hard to develop practical implementations of cryptographic constructions. Subtle discrepancies between specifications, on which security proofs generally reason, and implementations, may have catastrophic security consequences. We provide two Jasmin implementations of ML-KEM-768, and prove that:

> *The reference and vectorized Jasmin ML-KEM implementations are constant-time and functionally equivalent to the ML-KEM specification.*

To obtain this result, we upgrade the two Jasmin implementations of Kyber from [8] to ML-KEM and adapt the proofs of functional correctness to establish the link to the EasyCrypt specification of ML-KEM. With this, we provide an end-to-end computer-verified proof that the highly optimized assembly code from [8] (with our adaptations) implements an IND-CCA secure KEM. In order to validate that this KEM is indeed most likely what NIST specified as the draft of ML-KEM, we confirmed that the implementations generate the extensive test vectors recently published by Schmieg from her independent implementation of ML-KEM (see Section 7). Finally, to ensure that we follow best practices on mitigating timing side-channel attacks, we used the Jasmin specialized type system to prove that our implementations follow the constant-time discipline.

[12] Proving IND-CPA security of the PKE down to standard MLWE is possible assuming that the matrix sampling procedure is a random oracle [44], but this then makes it hard (in a mechanized proof setting) to see the resulting PKE as a deterministic construction that can be plugged into the FO transform.

Relevance. The provable security of Kyber has been scrutinized extensively during the NIST competition. However, the algorithm has undergone several tweaks since the original round-1 submission [61] and the ML-KEM specification has introduced further modifications. Therefore, it is important to consolidate the security proofs during the standardization process. In this work we confirm to a high-level of assurance that the various modifications introduced in Kyber and, more recently in ML-KEM, allow for a proof of security that follows from well-known results published in the literature. We review the evolution of Kyber and the modifications introduced by ML-KEM in Section 3.2.

We also contend that connecting security proofs to implementations is necessary to ensure that security guarantees apply to deployed code. Gaps between specifications and implementations can lead to security issues, and arise in practice. For example, early implementations of Kyber computed the "rejection key" used for the implicit rejection in the FO transform, but also returned a non-zero return value in case of rejection, which makes implicit-rejection FO proofs inapplicable. Another example are discrepancies between the specification and implementations of Saber [27] pointed out in [34, Sec. 5.4]. These meant that security proofs in the QROM did not apply to the implemented algorithms.

It is equally important to prove formally that implementations do not leak through timing side channels. For example, it was recently reported by Bhargavan, Kiefer, and Tamvada[13] (and independently by Bernstein), and by Kannwischer and Ravi[14] that some compilers with certain optimization flags translate the reference implementations for Kyber and ML-KEM made available by the Kyber team to binaries containing variable-time DIV instructions operating on secret inputs. Our implementations have been verified to correctly implement the draft standard *and* to avoid secret branch conditions, secret memory addresses, and variable-time DIV and MOD instructions on secret inputs.

Limitations. We point out four important limitations of our work. First, our security proof is classical. An important direction for future work is to prove security of ML-KEM against quantum adversaries. This will require using an extension of EasyCrypt for reasoning about security in the Quantum Random Oracle Model [24]. Second, our guarantees are currently limited to the code generated by the Jasmin compiler. In the future, we plan to extend the security and functional correctness guarantees to other implementations using equivalence checking. Third, the complexity bounds of the reductions have been conducted manually. Fourth, our results for failure probability do not go all the way to formally verifying the numbers claimed in [62]. Our proof gives a precise specification of the noise distribution that needs to be computed to obtain such numbers, but refer to [25] for the analytical method to (heuristically) compute this bound. The details are given in Section 4.

Related work. There is extensive prior work on security proofs of lattice-based KEMs and related proof techniques (see, e.g., [63,60,45,67,22,30,42,50]), cryptanalysis of the underlying assumptions (see, e.g., [56,52,19,3,28,4,35,2,29]), and side-channel attacks (see, e.g., [58,36,37,57,55,39,38]). However, since the focus and novelty of this paper lie in the field of computer-aided cryptography (CAC), we summarize relevant related work in this field.

One unifying goal of CAC is to deliver machine-checked proofs of security for practical cryptographic implementations. Almeida *et al* [6] provide a general approach to decompose this goal into more classical tasks of functional correctness and side-channel protection of implementation and provable security of specification. This approach has been used successfully for proving indifferentiability from ROM of a Jasmin implementation of SHA3 [9]. Beringer *et al* [20] and Ye *et al* [66] use a similar approach for proving security of HMAC and HMAC-DRBG implementations. In addition, there is a large body of work that establishes either security or functional correctness [11,23]. Some of these works explicitly target lattice-based constructions. In particular, Hülsing, Meijers and Strub [44] use EasyCrypt to prove CPA security of Saber; Barbosa *et al* [14] use EasyCrypt to prove unforgeability of XMSS; and Barbosa *et al* [12] use EasyCrypt for proving unforgeability of the Dilithium signature scheme. Similarly, Almeida *et al* use EasyCrypt for proving functional correctness of reference and vectorized Jasmin implementations of Kyber [8], and Kreuzer [46] uses the Isabelle proof assistant for proving correctness and IND-CPA security of Kyber.

Most works in CAC establish security against classical adversaries. However, an increasing number of works considers security against quantum adversaries. AutoLWE [17] is an automated tool for proving security of lattice-based cryptographic constructions. It has been used to prove security of several case studies, including Dual Regev encryption [33] and the Micciancio-Peikert encryption scheme [51]. However, constructions such as ML-KEM are outside the scope of AutoLWE. EasyPQC [13] is an extension of EasyCrypt for mechanizing security proofs against quantum adversaries. EasyPQC supports some (but not

[13] See https://cryspen.com/post/ml-kem-implementation/.
[14] See https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/ldX0ThYJuBo.

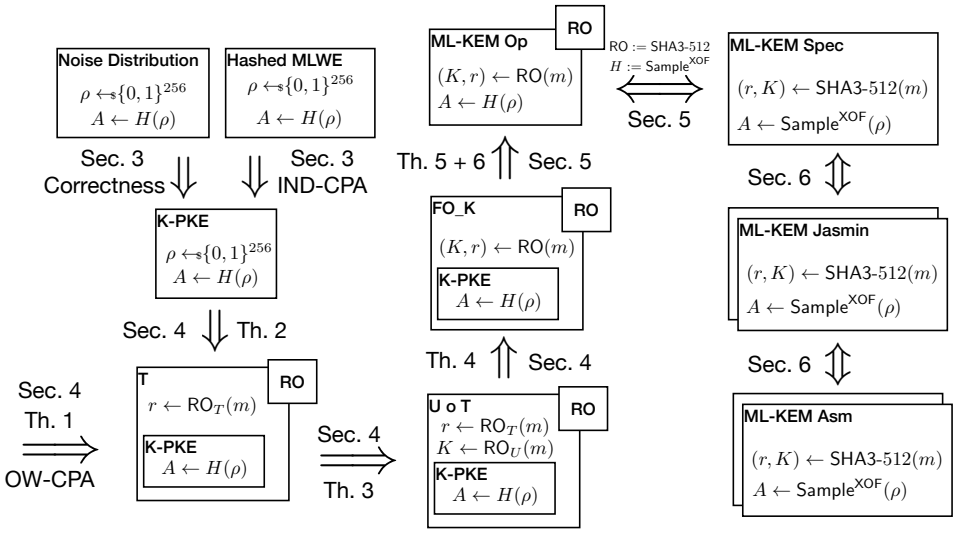

Fig. 1. Bird's eye view of development. Equivalence symbols represent functional correctness results. Implication symbols represent security and cryptographic correctness (failure probability) proofs.

all) common reasoning principles for QROM [24], and has been used for proving QROM security of the Full Domain Hash signature and of the Gentry-Peikert-Vaikuntanathan identity-based encryption [33]. Unfortunately, the current version of EasyPQC cannot be used for proving QROM security of Kyber. qrhl-tool is a tool for mechanizing security proofs of classical and quantum cryptographic constructions. The tool, which is implemented on top of the Isabelle proof assistant, is based on a relational Hoare logic for quantum programs [64]. The tool has been used for proving security of the same variant of the Fujisaki-Okamoto transformation we use in this paper [65] in the QROM, but it establishes incomparable bounds to our classical proof. Furthermore, these proofs are not connected to a security proof of a Kyber or ML-KEM specification, nor to an implementation. Finally, recent work, see e.g. [26], studies post-quantum applicability of automated tools for protocol verification. However, these tools cannot be used for verifying ML-KEM.

2 Bird's Eye View of Development

Figure 1 depicts the global structure of the formally verified development we present throughout the paper. We briefly describe it next.

On the top left, we show the results described in Section 4. There we prove that K-PKE, the public-key encryption scheme underlying Kyber and ML-KEM is IND-CPA secure and δ-correct. The IND-CPA proof shows that security holds under a variant of the MLWE assumption, which we call Hashed MLWE, where the matrix A is sampled using a deterministic procedure H, which is seeded with a uniform random seed ρ. The δ-correctness proof shows that the probability of a decryption failure can be upper-bounded by computing the probability of a noise distribution (independently of any adversarial action) producing a value that exceeds a predefined threshold.

On the right-hand side of the diagram, we show the functional correctness results described in Section 7: two Jasmin implementations of ML-KEM-768 are proved to be functionally equivalent to the ML-KEM specification in EasyCrypt. These implementations then give rise to functionally correct and constant-time assembly code via the Jasmin compiler. All of the artifacts on the right-hand side of the diagram are machine-checked representations of the ML-KEM draft standard at different levels of abstraction. In particular, they use SHA3-512 to compute the random coins r and the shared key K required by the FO_K transform. Furthermore, they use a complex sampling procedure $\mathsf{Sample}^{\mathsf{XOF}}$ based on the SHAKE-128 XOF to generate a random-looking matrix from a small seed ρ.

The middle of the diagram shows machine-checked cryptographic proofs for ML-KEM, as described in Sections 5 and 6. At the top, MLKEM_Op represents the ML-KEM specification with two modifications. The first modification moves the proof to the Random Oracle Model by idealizing the SHA3-512 hash computation for the fixed input size that is used when computing the Fujisaki-Okamoto hash. The

second modification is merely syntactic to facilitate the machine-checked proof: we replace the sampling procedure $\mathsf{Sample}^{\mathsf{XOF}}$ with an operator we simply call H. This second modification does not introduce a gap between the ML-KEM specification and the security proof: we prove that the semantics of H is precisely the semantics of $\mathsf{Sample}^{\mathsf{XOF}}$. This allows, in particular, basing our entire proof on the variant of Hashed MLWE that results from fixing H in this way. We guarantee that, indeed, the idealization of SHA3-512 is the only difference between $\mathsf{MLKEM\_Op}$ and the ML-KEM specification by showing that the two are equivalent when the RO is simply taken to be SHA3-512.

As shown in the middle of the diagram, in Section 6 we prove that the IND-CCA security of $\mathsf{MLKEM\_Op}$ follows from the IND-CCA security of a fully instantiated FO_K transform (Theorems 5 and 6). A full instantiation means plugging in the results described above for K-PKE proved in Section 4 and making the algebraic setting concrete for a set of ML-KEM parameters. FO_K uses a single random oracle to obtain random coins r and the session key K, whereas the modularized Fujisaki-Okamoto transform $\mathsf{T} \circ \mathsf{U}^{\not\perp}_m$ uses two independent random oracles to do this. We also prove in Section 5, that the IND-CCA security of FO_K follows from the IND-CCA security of $\mathsf{T} \circ \mathsf{U}^{\not\perp}_m$ (Theorem 4). In turn, the security of $\mathsf{T} \circ \mathsf{U}^{\not\perp}_m$ relies on the proof of the T transform (Theorem 2). This theorem shows that one can obtain a OW-secure derandomized public-key encryption scheme starting from an IND-CPA secure one. Finally, as shown on the bottom left of the diagram, we also prove Theorem 1—not used in the proof of ML-KEM—showing that the T transform also works when applied to a randomized OW-secure public-key encryption scheme.

Artifact. This paper is accompanied by an artifact that permits independently checking the EasyCrypt proofs. The artifact includes a Readme with instructions and the file locations for the EasyCrypt lemmas matching theorems in this paper.

3 Preliminaries

We now briefly discuss the mechanized reasoning tools we use for our proofs and give an overview of ML-KEM and how it evolved from Kyber. The cryptographic definitions and notation we use are fairly standard and are given in Appendix A.

3.1 The EasyCrypt proof assistant

EasyCrypt[15] [16] is a proof assistant for formalizing proofs of cryptographic properties. Its primary feature is the Probabilistic Relational Hoare Logic (pRHL), which we use throughout to prove equivalences between games, sometimes conditioned by the non-occurrence of a failure event. We call such conditional equivalence steps *up-to-bad* reasoning in later section. pRHL is designed to support reasoning about equivalences of probabilistic programs while reasoning only locally (within oracles) and without reasoning about the distribution of specific variables—essentially keeping track only of the fact that variables in one program are distributed identically to variables in the other, but not keeping track of what that distribution may be. This logic has proved highly expressive for the bulk of cryptographic proof work. However, some steps require more global reasoning (about the entire execution) or keeping track of the distribution of individual variables. Logical rules to support such reasoning steps are implemented in EasyCrypt, but are often unwieldy to apply in concrete context. The EasyCrypt team has, over the years, developed a number of generic libraries that abstract those more complex reasoning rules into "game transformations" or equivalence results that can be instantiated as part of other proofs.

Our proof makes extensive use of three of those generic libraries, which we outline below and discuss more in depth in Appendix B:

plug-and-pray provides a formalized generic argument for bounding the security loss of reductions that guess an instance, session or query (among a bounded-sized set) will lead to a successful run;

hybrid provides a formalized and generic argument for bounding the distance between two games that differ only in one oracle, but where the transition must be done query-by-query for the purpose of the proof;

PROM provides a generic argument, initially intended to apply to programmable random oracles, that encapsulates the widely used argument that a "value is random and independent of the adversary's view".

[15] https://easycrypt.info

Algorithm 1 K-PKE.KeyGen(): key generation

Output: Secret key $sk \in \mathcal{R}_q^k$ and public key $pk \in \hat{\mathcal{R}}_q^{\ k} \times \{0,1\}^{256}$
1: $d \leftarrow\!\!\$\ \{0,1\}^{256}$
2: $(\rho, \sigma) \leftarrow \mathrm{G}(d)$
3: $\hat{\mathbf{A}} \leftarrow \mathsf{Parse}(\mathsf{XOF}(\rho))$
4: $\mathbf{s}, \mathbf{e} \leftarrow \mathsf{CBD}_{\eta_1}(\mathsf{PRF}_\sigma)$ ▷ $\mathbf{s}, \mathbf{e} \in \mathcal{R}_q^k$
5: $\hat{\mathbf{s}} \leftarrow \mathsf{NTT}(\mathbf{s})$
6: $\hat{\mathbf{e}} \leftarrow \mathsf{NTT}(\mathbf{e})$
7: $\hat{\mathbf{t}} \leftarrow \hat{\mathbf{A}} \circ \hat{\mathbf{s}} + \hat{\mathbf{e}}$
8: **return** $sk = \hat{\mathbf{s}}$ and $pk = (\hat{\mathbf{t}}, \rho)$

Algorithm 2 K-PKE.Enc(pk, m): encryption

Input: Public key $pk = (\hat{\mathbf{t}}, \rho) \in \mathcal{R}_q^k \times \{0,1\}^{256}$, message $m \in \{0,1\}^{256}$
Output: Ciphertext $c \in \mathcal{R}_{d_u}^k \times \mathcal{R}_{d_v}$
1: $r \leftarrow\!\!\$\ \{0,1\}^{256}$
2: $\hat{\mathbf{A}} \leftarrow \mathsf{Parse}(\mathsf{XOF}(\rho))$
3: $\mathbf{r} \leftarrow \mathsf{CBD}_{\eta_1}(\mathsf{PRF}_r)$ ▷ $\mathbf{r} \in \mathcal{R}_q^k$
4: $\mathbf{e}_1, e_2 \leftarrow \mathsf{CBD}_{\eta_2}(\mathsf{PRF}_r)$ ▷ $\mathbf{e}_1 \in \mathcal{R}_q^k, e_2 \in \mathcal{R}_q$
5: $\hat{\mathbf{r}} \leftarrow \mathsf{NTT}(\mathbf{r})$
6: $\mathbf{u} \leftarrow \mathsf{NTT}^{-1}(\hat{\mathbf{A}}^T \circ \hat{\mathbf{r}}) + \mathbf{e}_1$
7: $v \leftarrow \mathsf{NTT}^{-1}(\hat{\mathbf{t}}^T \circ \hat{\mathbf{r}}) + e_2 + \mathsf{ToPoly}(m)$
8: $\mathbf{c}_1 \leftarrow \mathsf{Compress}_q(\mathbf{u}, d_u)$
9: $c_2 \leftarrow \mathsf{Compress}_q(v, d_v)$
10: **return** $c = (\mathbf{c}_1, c_2)$

3.2 Background on ML-KEM

From Kyber to ML-KEM. The original NIST PQC round-1 submission of Kyber described the scheme as a two-stage construction consisting of a module-lattice instantiation of the IND-CPA-secure LPR encryption scheme [47,48] and a *"slightly tweaked"* Fujisaki-Okamoto (FO) transform [31,32] to build an IND-CCA-secure KEM. The reduction linking IND-CPA security of the encryption scheme to MLWE was not spelled out in the NIST submission document. For the proof of IND-CCA security of the full scheme the submission mostly referred to previous work on the security of the FO transform, most notably [40] and [59]. Less than two months after the submission documents were made public, D'Anvers pointed out[16] that there are major obstacles to proving IND-CPA security from MLWE in Kyber. The reason is that unlike the original LPR scheme, round-1 Kyber included a public-key-compression step, which invalidates an assumption in the LPR proof. As a consequence, the Kyber submission team decided to tweak the submission for round 2 of NIST PQC by removing the public-key compression (and adjusting some other parameters to obtain similar performance); some more tweaks to parameters were proposed for the round-3 version of Kyber.

Through the further course of the NIST PQC project, issues were identified also in the second part of Kyber's security argument, which establishes IND-CCA security through a "tweaked" FO transform. It turned out that existing QROM proofs of the FO transform, e.g., from [40], did not apply and that proving the particular variant of the FO transform is not straightforward [34, Sec. 5.4]. Although Kyber's variant of the FO transform was eventually proven secure [50,15], NIST decided, after discussion on the pqc-forum mailing list, to revert to a more "vanilla" version of the FO transform for ML-KEM.

[16] See https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/PX_Wd11BecI.

Algorithm 3 K-PKE.Dec(sk, c): decryption

Input: Secret key $sk = \hat{\mathbf{s}} \in \mathcal{R}_q^k$ and ciphertext $c = (\mathbf{c}_1, c_2) \in \mathcal{R}_{d_u}^k \times \mathcal{R}_{d_v}$
Output: Message $m \in \{0,1\}^{256}$
1: $\tilde{\mathbf{u}} \leftarrow \mathsf{Decompress}_q(\mathbf{c}_1, d_u)$
2: $\tilde{v} \leftarrow \mathsf{Decompress}_q(c_2, d_v)$
3: $m \leftarrow \mathsf{ToMsg}(\tilde{v} - \mathsf{NTT}^{-1}(\hat{\mathbf{s}}^T \circ \mathsf{NTT}(\tilde{\mathbf{u}})))$
4: **return** m

Algorithm 4 ML-KEM.KeyGen(): key generation

Output: Secret key $sk = (sk', pk, \mathrm{H}(pk), z) \in \mathcal{R}_q^k \times (\hat{\mathcal{R}_q}^k \times \{0,1\}^{256}) \times \{0,1\}^{256} \times \{0,1\}^{256}$ and public key $pk \in \hat{\mathcal{R}_q}^k \times \{0,1\}^{256}$

1: $(pk, sk') \leftarrow$ K-PKE.KeyGen()
2: $z \leftarrow \{0,1\}^{256}$
3: $sk \leftarrow (sk' \| pk \| \mathrm{H}(pk) \| z)$
4: **return** (sk, pk)

Algorithm 5 ML-KEM.Encap(pk): encapsulation

Input: Public key $pk \in \hat{\mathcal{R}_q}^k \times \{0,1\}^{256}$
Output: Ciphertext $c \in \mathcal{R}_{d_u}^k \times \mathcal{R}_{d_v}$ and shared key $K \in \{0,1\}^{256}$

1: $m \leftarrow \{0,1\}^{256}$
2: $(K, r) \leftarrow \mathrm{G}(m \| \mathrm{H}(pk))$ ▷ $(K, r) \in \{0,1\}^{256} \times \{0,1\}^{256}$
3: $c \leftarrow$ K-PKE.Enc(pk, m, r)
4: **return** (c, K)

Our mechanized proof validates the final outcome of these evolutions, which led to the draft specification of ML-KEM. In our models we do *not* include the public-key validation step at the beginning of encapsulation, which NIST introduced in the ML-KEM draft standard. The reason is that, as far as we know, this is the last major open discussion topic for the final standard and it seems rather unlikely that it will be included in the final version of ML-KEM.

Technical description of ML-KEM. We give high-level algorithmic descriptions of K-PKE, the IND-CPA-secure public-key encryption scheme underlying ML-KEM, in Algorithms 1 to 3 and of the full IND-CCA-secure KEM in Algorithms 4 to 6. For a more implementation-oriented description that operates on byte arrays, see [54, Algs. 12–14].

ML-KEM works in the ring $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$ with $q = 3329$ and $n = 256$. The core operations are on small-dimension vectors and matrices over $\mathcal{R}_q$; the dimension depends on the parameter k, which is different for different parameter sets of ML-KEM: ML-KEM (NIST security level 1) uses $k = 2$, ML-KEM (NIST security level 3) uses $k = 3$, and ML-KEM (NIST security level 5) uses $k = 4$. We denote elements in $\mathcal{R}_q$ with regular lower-case letters (e.g., v); vectors over $\mathcal{R}_q$ with bold-face lower-case letters (e.g., $\mathbf{u}$), and matrices over $\mathcal{R}_q$ with bold-face upper-case letters (e.g., $\mathbf{A}$).

In these descriptions, XOF is an extendable output function that in ML-KEM is instantiated with SHAKE-128 [53]. Parse interprets outputs of XOF as sequence of 12-bit unsigned integers and runs rejection sampling to obtain coefficients that look uniformly random modulo q. The function H is instantiated with SHA3-256, G is instantiated with SHA3-512 and J is instantiated with SHAKE256 with output length fixed to 32 bytes. CBD_η denotes sampling coefficients from a centered binomial distribution with parameter η;[17] extension from coefficients to (vectors of) polynomials is done by sampling each coefficient independently from CBD_η. For example, ML-KEM-768 uses $\eta_1 = \eta_2 = 2$. The sampling routine is parameterized by a pseudorandom function PRF_k with key k. NTT is the number-theoretic transform of a polynomial in $\mathcal{R}_q$. Both input and output of NTT can be written as a sequence of 256 coefficients in $\mathbb{Z}_q$ and typical implementations perform the transform inplace. However, output coefficients do not have any meaning as a polynomial in $\mathcal{R}_q$. We therefore denote the output domain as $\hat{\mathcal{R}_q}$; we apply the same

[17] This means we have $B(n, p)$ with $p = 1/2$, $n = 2\eta$ and expected value shifted to 0.

Algorithm 6 ML-KEM.Decap(c, sk): decapsulation

Input: Ciphertext $c \in \mathcal{R}_{d_u}^k \times \mathcal{R}_{d_v}$ and secret key $sk = (sk', pk, \mathrm{H}(pk), z) \in \mathcal{R}_q^k \times (\hat{\mathcal{R}_q}^k \times \{0,1\}^{256}) \times \{0,1\}^{256} \times \{0,1\}^{256}$
Output: Shared key $K \in \{0,1\}^{256}$

1: $m' \leftarrow$ K-PKE.Dec(sk', c)
2: $(K', r') \leftarrow \mathrm{G}(m' \| \mathrm{H}(pk))$
3: $c' \leftarrow$ K-PKE.Enc(pk, m', r')
4: $K^- \leftarrow \mathrm{J}(z \| c)$
5: **if** $c = c'$ **then** $K \leftarrow K'$ **else** $K \leftarrow K^-$ ▷ *Requires branch-free conditional*
6: **return** K

notation for elements in $\hat{\mathcal{R}}_q$, e.g., $\hat{u} = \mathsf{NTT}(u)$. Application of NTT to vectors and matrices over $\mathcal{R}_q$ is done element-wise. $\mathsf{Compress}_q$ compresses elements in $\mathcal{R}_q$ (or $\mathcal{R}_q^k$) by rounding coefficients to a smaller modulus d_v (or d_u). $\mathsf{Decompress}_q$ is an approximate inverse of $\mathsf{Compress}_q$. ToPoly maps 256-bit strings to elements in $\mathcal{R}_q$ by mapping a zero bit to a zero coefficient and mapping a one bit to a $\frac{q}{2}$ coefficient; ToMsg rounds coefficients to bits to recover a message from a noisy version of a polynomial generated by ToPoly.

4 K-PKE correctness and IND-CPA security

Our proof of ML-KEM has the same structure as proposed originally for Kyber in [25]. It starts from an IND-CPA public-key encryption scheme and then instantiates a variant of the Fujisaki-Okamoto transform. In this section we look at the underlying public-key encryption scheme, which we call K-PKE as in the ML-KEM draft [54]. We prove IND-CPA security down to a variant of the MLWE assumption; we prove correctness down to the probability of a statistical event that is defined over the randomness space of public-key generation and encryption but, crucially, not the adversary's.

Our presentation in this section uses EasyCrypt notation because the specifications and theorem statements are relatively non-verbose. Later in the paper, we switch to a more concise mathematical notation and point to the artifact for the EasyCrypt formulations. In our presentation we highlight how EasyCrypt allows us to construct the proof modularly. The specification of K-PKE over which we carry out the proof is parametric on all noise distributions, serialization and compression operations, and vector and matrix dimensions. This means it applies to all ML-KEM variants. However, to complete the proof we need a series of hypotheses on these abstract parameters. These are stated as axioms at this level, which means that EasyCrypt will force us to discharge them as proof obligations when we want to use these results for concrete instantiations. Indeed, as we will see in Section 6, all the axioms in this section are proved to hold when we apply these general results to the EasyCrypt specification of ML-KEM-768.

4.1 The Hashed Module LWE assumption

The computational hardness assumption underlying the security of K-PKE is formalized in our development by first considering an abstract ring R and matrices and vectors over elements in R. We obtain these types, called matrix and vector, and operations over them directly from the Matrix theory that exists in the EasyCrypt library, and then define distributions over them as follows.

```
op [lossless uniform full] duni_R : R distr.
op [lossless] dshort_R : R distr.
op duni = dvector duni_R.
op dshort = dvector dshort_R.
```

Here, the lossless, uniform and full attributes automatically generate axioms that represent assumptions on these distributions. A *full uniform* distribution assigns the same non-zero probability to all values in the support type. In a lossless distribution, the probability of the trivial event true is 1.[18] The dvector operator generalizes a distribution D over ring elements to distributions over (column) vectors by defining distributions that sample each vector element independently from D. Note that we have two distributions over ring elements and vectors thereof: the uniform distribution and an arbitrary distribution that we call *short*. In the variant of the MLWE assumption used in K-PKE this corresponds to sampling ring elements by choosing each polynomial coefficient independently from a centered binomial distribution with $p = 1/2$ and $n = 2$ or $n = 3$, depending on the security parameters.

We define the relevant variant of the MLWE problem, which we call MLWE_H for *Hashed MLWE*, using the experiment in Figure 2 (left). There are several differences to the standard MLWE assumption [25]. First and foremost, matrix A is not sampled uniformly at random, but rather created using a deterministic sampling procedure H that expands A from a small seed.

```
type seed.
op H : seed → matrix.
op [lossless] dseed : seed distr.
```

Sampling A in this way is what K-PKE does in practice and, since our purpose is to connect our security proof to an exact specification of ML-KEM (and from there to an implementation) we cannot go around this aspect as in [46]. Note that, as in [44], we could still prove the security of K-PKE down to the

[18] EasyCrypt's distr type represents subdistributions.

```
module MLWE_H(Adv : HAdv_T) = {
  proc main(tr b : bool) : bool = {
    sd <-$ dseed; s <-$ dshort; e <-$ dshort;
    _A <- if tr then (H sd)^T else (H sd);
    u0 <- _As + e;
    u1 <-$ duni;
    t <-$ duni;
    e' <-$ dshort_R;
    v0 <- t^T s + e';
    v1 <-$ duni_R;
    b' <- Adv.guess(sd, t, if b
                           then (u1,v1)
                           else (u0,v0));
    return b';
  }

}.
```

```
module MLWE_PKE_HASH_PROC : Scheme = {
  proc kg() : pkey * skey = {
    sd <-$ dseed; s <-$ dshort; e <-$ dshort;
    t <- (H sd)s + e;
    return (pk_encode (sd,t),sk_encode s);
  }

  proc enc(pk : pkey, m : plaintext) : ciphertext = {
    (sd,t) <- pk_decode pk;
    r <-$ dshort; e1 <-$ dshort; e2 <-$ dshort_R;
    u <- (H sd)^T r + e1;
    v <- t^T r + e2 + (m_encode m);
    return c_encode (u,v);
  }

  proc dec(sk : skey, c : ciphertext) : plaintext option = {
    (u,v) <- c_decode c;
    return (Some (m_decode (v - (sk_decode sk^T u))));
  } }.
```

Fig. 2. Hashed MLWE assumption (left). K-PKE (right).

standard notion of MLWE by assuming that H is a random oracle. However, this would create another problem: when using K-PKE to instantiate the FO transform in the next section in a modular way, we must have a version of K-PKE that is *derandomizeable*. Modeling H as a random oracle would make it impossible to plug-in the K-PKE security and correctness results (where H is a random oracle) to the FO formalization (where H must be a deterministic function). We note that, in practice, the computational assumption underlying ML-KEM is closer to the one we formalize here: one assumes that MLWE holds when A is expanded via a deterministic rejection-sampling procedure based on SHAKE-256.

Another minor difference in our formalization of MLWE_H to the standard MLWE assumption is that, rather than having an MLWE assumption that allows a matrix with a configurable number of rows, our formalization of MLWE fixes A to be square (for some dimensions $k \times k$) and then consider an additional row t^T explicitly. This allows us to have a simpler formalization and reuse our MLWE_H experiment for both the case where we only need to assume that the output vector u is pseudorandom, and the case where we need to assume that both u and an additional ring element v are (jointly) pseudorandom. Finally, because we sample A as H(sd), we need to have two variants of the definition for the case where the challenge is computed over A and A^T.

4.2 The K-PKE construction.

K-PKE [25] is a variant of a construction introduced by Lyubashevsky, Peikert, and Regev [47,49] (LPR). We present our (abstract) specification in Figure 2 (right). This construction introduces several new operators: m_encode and m_decode that map plaintexts to ring elements and vice-versa; pk_encode and pk_decode that represent serialization procedures for the public key; sk_encode and sk_decode that represent serialization procedures for the secret key; and, finally, c_encode and c_decode that represent a combination of compression by rounding and serialization for K-PKE ciphertexts. For the security of this construction what these operators do is irrelevant: the IND-CPA security experiment only uses the encoding operators and, if they lose information, this can never hurt security. However, for correctness, we require additional properties that we will introduce later.

Security of K-PKE. We briefly discuss how we prove that K-PKE is IND-CPA secure under the standard EasyCrypt library definition.[19] This formalizes the notion given in Figure 16. The security proof uses the Hashed MLWE assumption twice to carry out two game hops. The first transformation modifies the key-generation procedure to obtain MLWE_PKE_HASH1 as follows:

```
proc kg() : pkey * skey = {
  sd <-$ dseed; s <-$ dshort; t <-$ duni; return (pk_encode (sd,t), sk_encode s); }
```

In this new game, the t component in the public key is now a uniform vector. We construct an attacker B1 against the Hashed MLWE game that bridges the game hop by running the K-PKE attacker A in a way that perfectly interpolates between the two games. In particular we show:

$$\Pr[\mathsf{CPA[MLWE\_PKE\_HASH\_PROC,A]}() \Rightarrow 1] = \Pr[\mathsf{MLWE\_H[B1(A)]}(\mathsf{false},\mathsf{false}) \Rightarrow 1]$$
$$\Pr[\mathsf{CPA[MLWE\_PKE\_HASH1,A]}() \Rightarrow 1] = \Pr[\mathsf{MLWE\_H[B1(A)]}(\mathsf{false},\mathsf{true}) \Rightarrow 1]$$

[19] Line 24 in https://github.com/EasyCrypt/easycrypt/blob/main/theories/crypto/PKE.ec

Here, notation M[·] means the parametrization of module/algorithm M with one or more modules/algorithms. In the proof, B1 attacks Hashed MLWE in the version where the matrix is not transposed (first argument to MLWE_H) and it just uses the first part of the Hashed MLWE challenge in its attack, which is a vector, and simply discards the additional ring element.

The next modification to the game constructs MLWE_PKE_HASH2 where the encryption procedure now simply uses uniformly distributed values to construct the ciphertext as follows:

```
proc enc(pk : pkey, m : plaintext) : ciphertext = {
  _A ← m_transpose (H (pk_decode pk)_1);
  u ←$ duni; v ←$ duni_R;
  return c_encode (u,v + m_encode m); }
```

Using a similar strategy as before, but reducing to the variant of MLWE_H that uses a transposed matrix, we obtain

$$\Pr[\mathsf{CPA[MLWE\_PKE\_HASH1,A]}(\,) \Rightarrow 1] = \Pr[\mathsf{MLWE\_H[B2(A)]}(\mathsf{true},\mathsf{false}) \Rightarrow 1].$$
$$\Pr[\mathsf{CPA[MLWE\_PKE\_HASH2,A]}(\,) \Rightarrow 1] = \Pr[\mathsf{MLWE\_H[B2(A)]}(\mathsf{true},\mathsf{true}) \Rightarrow 1].$$

Adversary B2 uses the full Hashed MLWE challenge to justify the pseudorandomness of the u and v ciphertext components. Finally, we can use the properties of abstract ring R to show that the ciphertext provided to the adversary is uniformly distributed and therefore independent of m, so that:

$$\Pr[\mathsf{CPA[MLWE\_PKE\_HASH2,A]}(\,) \Rightarrow 1] = 1/2\,.$$

The security result states that, for all A, we have:

$$\begin{aligned}\Pr[\mathsf{CPA[MLWE\_PKE\_HASH\_PROC,A]}(\,) \Rightarrow 1] - 1/2 = \\ \Pr[\mathsf{MLWE\_H[B1(A)]}(\mathsf{false},\mathsf{false}) \Rightarrow 1] - \Pr[\mathsf{MLWE\_H[B1(A)]}(\mathsf{false},\mathsf{true}) \Rightarrow 1] \\ + \Pr[\mathsf{MLWE\_H[B2(A)]}(\mathsf{true},\mathsf{false}) \Rightarrow 1] - \Pr[\mathsf{MLWE\_H[B2(A)]}(\mathsf{true},\mathsf{true}) \Rightarrow 1].\end{aligned}$$

Correctness of K-PKE. The correctness experiment we use is the standard definition when analyzing post-quantum secure KEMs using the Fusijaki-Okamoto transform and stated in [40,41].[20] This is given in Figure 14.In this experiment, a correctness adversary gets a freshly challenged *key pair*, including both public and secret key, and tries to find a message m for which the encryption scheme fails to decrypt correctly, i.e., encryption of m followed by decryption does not produce m. The advantage the adversary is given by δ, an upper bound on the failure probability, so that the probability of correct decryption is above $1 - \delta$.

The correctness proof for K-PKE outlined in [25] consists of two steps:

1. Showing that δ can be expressed as the probability that a (complex) noise distribution produces a value above a predefined threshold.
2. Computing the aforementioned probability using an analytic procedure implemented in Python and given in [25].[21]

As noted in [25] and [46], the connection between points (1) and (2) for Kyber and ML-KEM is, so far, heuristic. Indeed, the Python script given in (2) simplifies the analysis of the failure probability by assuming that the distribution of errors introduced by compression can be computed as if the compressed values come from a uniform distribution. This seems to follow from the Hashed MLWE assumption but, in fact and to the best of our knowledge, a formal proof that justifies the heuristic bound has not been given in the literature.[22] Our proof of correctness establishes point (1). We state and prove the following claim for all correctness adversaries A:

$$\begin{aligned}\Pr[\mathsf{CorrectnessAdv[MLWE\_PKE\_HASH\_PROC,\ A]}(\,) \Rightarrow 1] \leq \\ \Pr[\mathsf{CorrectnessBound}(\,) \Rightarrow 1].\end{aligned}$$

The CorrectnessBound experiment is defined as follows. This experiment has no adversary: it computes a noise value that follows a particular distribution and checks whether this noise value is above a certain bound—concretely, the check is whether any coefficient of the noise polynomial is above max_noise − cv_bound_max.

[20] As of this work, this is also included in the standard EasyCrypt library: Line 172 in https://github.com/EasyCrypt/easycrypt/blob/main/theories/crypto/PKE.ec

[21] The Python script was provided with the Kyber submission to the NIST Post Quantum competition: https://github.com/pq-crystals/security-estimates/blob/master/Kyber.py

[22] A reduction to MLWE is possible by expressing the failure probability as a disjunction of events that are simpler to analyze, but this results in significantly worse bounds than the one computed heuristically.

```
lemma noise_exp_val _A s e r e1 e2 m :
  noise_exp _A s e r e1 e2 m =
  let t = _As + e in
  let u = _A^T r + e1 in
  let v = t^T r + e2 + (m_encode m) in
  let cu = rnd_err_u u in
  let cv = rnd_err_v v in
   e^T r − s^T e1 − s^T cu + e2 + cv.
```

```
module CorrectnessAdvNoise(A : CORR_ADV) = {
  proc main() = {
    sd ←$ dseed; _A ← H sd;
    r ←$ dshort; s ←$ dshort; e ←$ dshort;
    e1 ←$ dshort; e2 ←$ dshort_R;
    m ← A.find(pk_encode (sd,_As + e),sk_encode s);
    n ← noise_exp _A s e r e1 e2 m;
    return (¬under_noise_bound n max_noise);
  }
}.
```

Fig. 3. Noise expression (left) and restated correctness experiment (right).

```
module CorrectnessBound = {
  proc main() = {
    sd ←$ dseed; _A ← H sd; r ←$ dshort; s ←$ dshort;
    e ←$ dshort; e1 ←$ dshort; e2 ←$ dshort_R;
    u  ← m_transpose _Ar + e1;
    cu ← rnd_err_u u;
    n ← e^T r − s^T e1 + e2 − s^T cu.
    return (¬under_noise_bound n (max_noise − cv_bound_max));
  } }.
```

So, in short, what our theorem states is that K-PKE is δ-correct according to the definition in Appendix A, where δ is the probability that the noise distribution, described in experiment CorrectnessBound, produces a value that is above a certain threshold. We explain how we derive this result next. For this, we need to introduce additional definitions and hypotheses. The first requirement is that the serialization operators for public keys and secret keys commute, i.e., that decoding inverts encoding.

```
axiom pk_encodeK : ∀  pk, pk_decode (pk_encode pk) = pk.
axiom sk_encodeK : ∀  sk, sk_decode (sk_encode sk) = sk.
```

Next we need to describe the errors introduced by compression and decompression of the ciphertext as additive noise. We do this by introducing two operators that have precisely this semantics. Note that the semantics of these operators is easy to define when c_encode, c_decode and the ring R are concrete.

```
op rnd_err_v : R → R.
op rnd_err_u : vector → vector.

axiom encode_noise u v :
  c_decode (c_encode (u,v)) = (u + rnd_err_u u, v + rnd_err_v v).
```

Finally we need to fix an upper bound for the noise, below which it is guaranteed that the decoded message matches the originally encrypted message.

```
op max_noise : int.
op under_noise_bound : R → int → bool.

axiom good_decode m n :
  under_noise_bound n max_noise ⇒  m_decode (m_encode m + n) = m.
```

With these operators we can define the full noise expression for K-PKE and restate the correctness experiment in terms of a noise threshold as in Figure 3.

In the next step of the correctness proof, we over-approximate the adversary's winning probability by splitting it as the sum of the probability of two (not necessarily disjoint) events. One event failprob2 corresponds to the possibility that the compression error associated with the v component in the ciphertext (cv in the noise expression) is above a certain bound cv_bound. The other event failprob1 is that all the other terms in the noise expression are above max_noise − cv_bound.

To remove the adversary's influence in the probability of failure, we take the value of cv_bound that guarantees failprob2 = 0. We call this cv_bound_max and note that, in practice, this is much smaller than max_noise (around 10%).

```
type randomness.
op [uniform full lossless]drand :
                    randomness distr.
op prg_kg : randomness →
                    seed * vector * vector.
op prg_enc : randomness →
                    vector * vector * R.

op kg(r : randomness) : pkey * skey =
  let (sd,s,e) = prg_kg r in
  let t =  (H sd)s + e in
     (pk_encode (sd,t),sk_encode s).

op enc(rr : randomness, pk : pkey, m : plaintext) : ciphertext =
  let (sd,t) = pk_decode pk in
  let (r,e1,e2) = prg_enc rr in
  let u = m_transpose (H sd)r + e1 in
  let v = t^T r) + e2 + (m_encode m) in
     c_encode (u,v).

op dec(sk : skey, c : ciphertext) : plaintext option =
  let (u,v) = c_decode c in
    Some (m_decode (v − (sk_decode sk)^T u)).
```

Fig. 4. De-randomized K-PKE.

```
op cv_bound_max : int.
axiom cv_bound_valid _A s e r e2 m :
  s ∈ dshort ⇒  e ∈ dshort ⇒  r ∈ dshort ⇒  e2 ∈ dshort_R ⇒
  let t = _As + e in
  let v = (t^T r) + e2 + (m_encode m) in
    under_noise_bound (rnd_err_v v) cv_bound_max.
```

At this point we can move to a correctness game where the probability of failure does not depend on the adversary and is simply failprob1. This game is CorrectnessBound that we use in our theorem statement. To summarize, we formally verify that K-PKE is δ-correct, according to the definition in Section A, for δ = failprob1.

We finally note that by "maxing-out" the adversary's contribution to the noise, we do not need to consider the distribution of cv at all. However, the distribution of cu to the noise expression computed in game CorrectnessBound still does not match the one in the heuristic computation proposed in [25], where cu results from compressing a uniform u. Justifying this simplification using (Hashed) MLWE is not immediate because r and e_1 occur elsewhere in the noise expression, so an efficient reduction that does not get r and e_1 cannot check for the failure event.

4.3 Derandomizing K-PKE

Looking at the K-PKE specification in Section 3, we see that the key generation and encryption algorithms take small random seeds as inputs and then use complex procedures to convert these seeds into the required noise values. Note also that these procedures internally use hash functions that we will later model as pseudorandom functions to expand the input seeds into longer sequences of pseudorandom bits. To bridge this gap to the abstract specification of K-PKE that we introduced in this section, and also to allow connecting K-PKE to the Fujisaki-Okamoto transform, we specify an alternative definition of K-PKE that explicitly uses abstract deterministic procedures to expand small seeds of type randomness into the required noise values. This is shown in Figure 4.

It is not hard to see that the correctness and security of derandomized K-PKE follow—and we prove this in EasyCrypt—from the results we proved earlier in this section, under the assumption that prg_kg and prg_enc produce distributions that are computationally close to the following ideal randomness distributions.

```
op prg_kg_ideal = dseed `*` dshort `*` dshort.
op prg_enc_ideal = dshort `*` dshort `*` dshort_R.
```

Where d_1 `*` d_2 represents the product of the distributions d_1 and d_2.

In the next section we discuss how we formalized the Fujisaki-Okamoto transform used in ML-KEM. When we instantiate it, we will use the derandomized K-PKE we just defined. Then, in Section 6, when we apply the result to a fully concrete specification of K-PKE, we prove that the procedures in ML-KEM that instantiate prg_kg and prg_enc are indeed producing distributions computationally close to the ideal ones, under the assumption that the underlying hash functions have standard pseudorandomness properties. This will allow us to have a bound for ML-KEM expressed as a function of the Hashed MLWE assumption, the δ-correctness of K-PKE as defined in the previous section, and the pseudorandomness of algorithms in the SHA3 family.

$\underline{\mathsf{Enc}_1(pk, m)}$:
1: $c \leftarrow \mathsf{Enc}(pk, m; \mathsf{G}(m))$
2: **return** c

$\underline{\mathsf{Dec}_1((pk, sk), c)}$:
1: $m' \leftarrow \mathsf{Dec}(sk, c)$
2: **if** $m' = \perp$ **or** $\mathsf{Enc}(pk, m'; \mathsf{G}(m')) \neq c$
3: | **return** $\perp$
4: **else return** m'

Fig. 5. The T transform: algorithms for $\mathsf{T}[\mathsf{PKE}, \mathsf{G}]$ for $\mathsf{PKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$. Gen_1 runs Gen and sets the secret key to be (pk, sk).

5 Formalizing the FO Transform used in ML-KEM

Hofheinz, Hövelmanns and Kiltz [40,41] (HHK) give a modular treatment of the Fujisaki-Okamoto (FO) transform [31,32] that splits it into two constructions presented and analyzed independently: the T and the U transforms.

The T transform (Fig. 5) constructs an actively OW-secure *deterministic* PKE scheme (active in the sense that the adversary is given access to both a ciphertext validity oracle and to a plaintext checking oracle—see Definition 1), from a passively secure *probabilistic* PKE scheme (one can start from $\mathsf{IND\text{-}CPA}$ or $\mathsf{OW\text{-}CPA}$). HHK [41, Theorems 3.1 and 3.2] bound the insecurity of this construction.

The U transform (Fig. 6) constructs an $\mathsf{IND\text{-}CCA}$-secure KEM from a OW-secure PKE scheme (for example, such as that constructed by T). It operates by encrypting (using the PKE scheme), a random message m which is treated as a pre-key from which a shared key k is derived through a random oracle. HHK describe four variants of U: $\mathsf{U}^{\perp}$, $\mathsf{U}^{\not\perp}$, $\mathsf{U}^{\perp}_{m}$ and $\mathsf{U}^{\not\perp}_{m}$. The superscript denotes whether rejection when the underlying PKE fails to recover a pre-key is explicit ($\perp$), or implicit ($\not\perp$) (that is, it returns a random-looking shared key instead of an error). The subscript indicates whether the key derivation step includes only the pre-key m, or also the ciphertext (no subscript). HHK [41, Theorems 3.3-3.6] demonstrate the $\mathsf{IND\text{-}CCA}$ security of the KEM resulting from the different versions of the U transform, with varying one-wayness assumptions on the underlying PKE based on the U variant used. For example, it is convenient to assume one-wayness in the presence of a ciphertext validity oracle when the KEM does explicit rejection.

Each of the transforms uses a random oracle. Following HHK, *in this section*, we call these random oracles G (used by T) and H (used by U). These are distinct from the oracles H and G used in other sections, and in particular in the description of ML-KEM in Section 3.

Overview of our results on FO. ML-KEM can be seen as an instance of the composed transform $\mathsf{U}^{\not\perp}_{m} \circ \mathsf{T}$. However, we do not follow the purely modular proof structure introduced by HHK [40,41]. As pointed out in various works [21,43], the abstraction boundary introduced between the T and U transforms in [40,41] makes the proof of the U transform unnecessarily complex. For example, the U transform requires a property called *rigidity*[23] from the underlying PKE, which is guaranteed by the T transform by construction.

In our formalization we consider the $\mathsf{U}^{\not\perp}_{m}$ transform specifically in the case where it is composed with the T transform. This eases some proof steps, in particular when query counting must across the abstraction boundary.

5.1 The T transform

Let $\mathsf{PKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a derandomized (as in Section 4) public-key encryption scheme with message space $\mathcal{M}$ and randomness space $\mathcal{R}$, and $\mathsf{G} : \mathcal{M} \to \mathcal{R}$ be a random oracle. The T transform defines a new public-key encryption scheme $\mathsf{PKE}_1 = \mathsf{T}[\mathsf{PKE}, \mathsf{G}] = (\mathsf{Gen}_1, \mathsf{Enc}_1, \mathsf{Dec}_1)$, where algorithms Gen_1, Enc_1 and Dec_1 are shown in Figure 5.

Our formal verification of the ML-KEM $\mathsf{IND\text{-}CCA}$ security proof includes a machine-checked proof of the following security and correctness theorem for the T transform.

Theorem 1. *Assume* PKE *to be* δ*-correct and* γ*-spread. Then* $\mathsf{PKE}_1 = \mathsf{T}[\mathsf{PKE}, \mathsf{G}]$ *is* $(q_G + 1) \cdot \delta$*-correct, where* q_G *is the number of random oracle queries placed by the correctness attacker. Furthermore, for any* $\mathsf{OW\text{-}PCVA}$ *adversary* $\mathcal{B}$ *that issues at most* q_{G} *queries to the random oracle* G*,* q_P *queries to a plaintext*

[23] The PKE must guarantee consistent re-encryption unless decryption outputs $\perp$.

checking oracle PCO*, and* q_V *queries to a validity checking oracle* CVO*, there exists a* OW-CPA *adversary* $\mathcal{A}$ *whose running time is about that of* $\mathcal{B}$*, and such that*

$$\mathsf{Adv}^{\mathsf{OW-PCVA}}_{\mathsf{PKE}_1}(\mathcal{B}) \leq (q_\mathsf{G} + q_P + 2) \cdot \delta + q_V \cdot 2^{-\gamma} + (q_\mathsf{G} + q_P + 1) \cdot \mathsf{Adv}^{\mathsf{OW-CPA}}_{\mathsf{PKE}}(\mathcal{A})$$

The theorem statement differs from that given in HHK [40,41, Theorem 3.1] only in that there is an extra additive δ term in the correctness bound, and an extra $2 \cdot \delta$ additive term in the security bound. In both cases, the high-level proof is identical to those given by HHK, except for the fact that our δ-correctness reductions make one additional call to the random oracle before terminating, to ensure that a specific message is in the random oracle table. This simplifies some intermediate steps in the proof. We sketch the proof next, highlighting the steps that were more challenging to formalize in EasyCrypt.

Proof (Formalization Sketch). The correctness proof, which takes a paragraph to write [40,41], is surprisingly involved in EasyCrypt. Its intuition is easy to understand: if the adversary finds a correctness error for the T transform, then there must be some message in the random oracle table for which the underlying PKE has failed. Writing this in game-based form requires: 1) wrapping the correctness adversary to ensure the problematic message is in the random oracle table; 2) guessing which of the adversary's queries to the random oracle will be problematic; 3) proving that, if a correct guess is made, the probability of error in T is the same as in the base PKE. Step 3) is particularly laborious in EasyCrypt: it entails moving a randomness sampling that occurs due to an adversarial call to the random oracle, in order to place it at the beginning of the game. This aligns the problematic sampling with the single randomness sampling that occurs in the correctness game for PKE. An instance of the PROM library(App. B) forms the core of the argument, but remains difficult to instantiate since we cannot predict the inputs the adversary will query the random oracle on. Step (2) is performed using the generic *plug-and-pray* argument(App. B).

The security proof generally follows along the steps outlined by HHK [40,41]. The first game hop (Game 1) changes the ciphertext validity oracle to reject all ciphertexts that cannot be reconstructed using a prior query to G. This hop is summarized in the handwritten proof as following from a union bound over all queries to the validity oracle. In EasyCrypt this must be expressed as a rather intricate hybrid argument that handles one query at a time. This is formalized using the existing generic *hybrid* argument.

The next step (Game 2) removes the use of the secret key from both the ciphertext validity and the plaintext checking oracles. Both checks are replaced by re-encryption; an inconsistency can occur only if the correctness of the T transform itself fails. This hop is formalized as a simple *up-to-bad* argument, followed by (tightly) bounding the probability of the failure event—a failure in the correctness of T—using the existing result on T correctness. This reduction places at most $(q_\mathsf{G} + q_P + 1)$ queries to its random oracle, which justifies the δ-correctness term in the security theorem statement.

The final game (Game 3) is reached by declaring the adversary a loser if she queries the OW challenge message to G—an event we call QUERY. Since the secret key is not used in any of the oracles at this point, the probability of success in the final game follows from a simple reduction from the one-wayness of the underlying PKE. Formally bounding the probability of the QUERY event is harder, since it once again involves guessing which random-oracle query made by the adversary will actually break the one-wayness of the underlying PKE. This is resolved using a similar argument as that deployed in the correctness proof.

Combined, the two reductions from one-wayness yield the $(q_\mathsf{G} + q_P + 1) \cdot \mathsf{Adv}^{\mathsf{OW-CPA}}_{\mathsf{PKE}}(\mathcal{A})$ term in the theorem, where $(q_\mathsf{G} + q_P)$ is the guessing loss. □

Theorem 1 does not directly apply to the ML-KEM construction, which starts from an IND-CPA —rather than OW-CPA —encryption scheme. For this reason, we extend the proof of Theorem 1 to obtain the analogous of [40,41, Theorem 3.2], as follows.

Theorem 2. *Assume* PKE *to be* δ*-correct and* γ*-spread. Then, for any* OW-PCVA *adversary* $\mathcal{B}$ *that issues at most* q_G *queries to the random oracle* G*,* q_P *queries to a plaintext checking oracle* PCO*, and* q_V *queries to a validity checking oracle* CVO*, there exists an* IND-CPA *adversary* $\mathcal{A}$ *whose running time is about that of* $\mathcal{B}$ *and such that*

$$\mathsf{Adv}^{\mathsf{OW-PCVA}}_{\mathsf{PKE}_1}(\mathcal{B}) \leq (q_\mathsf{G} + q_P + 3) \cdot \delta + q_V \cdot 2^{-\gamma} + \frac{2(q_\mathsf{G} + q_P + 1)}{|\mathcal{M}|} + 4 \cdot \mathsf{Adv}^{\mathsf{IND-CPA}}_{\mathsf{PKE}}(\mathcal{A})$$

Proof (Sketch). There are two differences to the proof of Theorem 1, where we replace the reductions from OW-CPA with reductions from IND-CPA. However, as discussed by HHK [40,41], here we can

$\mathsf{Gen}_2()$:	$\mathsf{Enc}_2(pk)$:	$\mathsf{Dec}_2((sk, s), c)$:
1: $s \leftarrow\!\!\$\ \mathcal{K}$	1: $m \leftarrow\!\!\$\ \mathcal{M}$	1: $m' \leftarrow \mathsf{Dec}_1^{\mathsf{G}}(sk, c)$
2: $(pk, sk) \leftarrow\!\!\$\ \mathsf{Gen}_1()$	2: $c \leftarrow \mathsf{Enc}_1^{\mathsf{G}}(pk, m)$	2: **if** $m' \neq \bot$
3: **return** $(pk, (sk, s))$	3: $k \leftarrow \mathsf{H}(m)$	3: $\quad$ **return** $\mathsf{H}(m')$
	4: **return** (k, c)	4: **else return** $\mathsf{J}(s, c)$

Fig. 6. The $\mathsf{U}_m^{\not\perp}$ transform.

```
proc kg() : pkey * skey = {
  (pk,sk) <$- kg;
  k <$- dK;
  return (pk, ((pk,sk),k));
}
```

```
proc enc(pk : pkey)
    : ciphertext * key = {
  m <$- dplaintext;
  c <-TT(HT).enc(pk,m);
  k <- HU.get(m);
  return (c,k); }
```

```
proc dec(sk : skey, c : ciphertext)
                : key option = {
  m' <- TT(HT).dec(sk[1],c);
  if (m' = None) { k <- J sk[2] c; }
  else { k <- HU.get(oget m'); }
  return (Some k); }
```

Fig. 7. EasyCrypt specification of $\mathsf{KEM}_m^{\not\perp} = \mathsf{U}_m^{\not\perp}[\mathsf{PKE}_1, \mathsf{H}]$.

take advantage of the fact that IND-CPA is a stronger property than OW-CPA, in which the attacker controls almost fully the encrypted message, to obtain a tighter reduction. In particular, IND-CPA security implies a multi-guess version of one-wayness where the OW-CPA adversary outputs a list of guesses. This (Lemma 1) is proved generically in EasyCrypt. It enables us to more tightly bound the probability of QUERY, constructing a OW adversary that outputs the full contents of the random oracle table rather than trying to guess which of its entries is the correct one. Our formalization is structured so that the core arguments in the proofs of Theorem 1 and 2 are proven once and used twice, rather than repeated.

Remark. We use Lemma 1 twice to derive Theorem 2. The most intuitive use is in tightly bounding QUERY, as we described above. However, we also use it to convert the OW adversary constructed for Game 3 in the proof of Theorem 1 into an IND-CPA adversary in a black-box way by making it return a list of size 1. This second use justifies the minor difference to the bound given by HHK [40,41].

5.2 The $\mathsf{U}_m^{\not\perp}$ transform.

Let $\mathsf{PKE}_1 = (\mathsf{Gen}, \mathsf{Enc}_1, \mathsf{Dec}_1)$ be a deterministic PKE scheme that relies on a random oracle G. The $\mathsf{U}_m^{\not\perp}$ transform uses a new random oracle $\mathsf{H} : \mathcal{M} \to \{0,1\}^n$ and a pseudorandom function $\mathsf{J} : \mathcal{K} \times \mathcal{M} \to \{0,1\}^n$, where $\mathcal{K}$ is the key space, to define a new key encapsulation mechanism $\mathsf{KEM}_m^{\not\perp} = \mathsf{U}_m^{\not\perp}[\mathsf{PKE}_1, \mathsf{H}] = (\mathsf{Gen}_2, \mathsf{Enc}_2, \mathsf{Dec}_2)$ whose algorithms are shown in Figure 6. We note an important difference to the version of U given by HHK [40,41]: we use a PRF primitive rather than a random oracle to generate (random-looking) shared keys when performing implicit rejection in decapsulation. This simplifies the proof and implies that our bounds include PRF distinguishing advantage terms rather than concrete probability terms to account for the possibility that the adversary exploits a weakness in J in its attack.

Also note that, if one does not fix a concrete instance for PKE_1, then one must treat the interaction with G generically in the proof. For example, in [40,41, Theorem 3.6], the bound is given as a function of the maximum number of queries placed by Enc_1 and Dec_1 to G. This reasoning is possible in EasyCrypt, but it would significantly complicate the proof. For this reason, we consider explicitly the case where $\mathsf{PKE}_1 = \mathsf{T}[\mathsf{PKE}, \mathsf{G}]$. The EasyCrypt specification of this composed transform is given in Figure 7. We prove the following result in EasyCrypt, which is the analogue of HHK's Theorem 3.6 [40,41] in this more concrete setting.

Theorem 3. *If* $\mathsf{PKE}_1 = \mathsf{T}[\mathsf{PKE}, \mathsf{G}]$ *is* $\delta_1(\cdot)$*-correct, where* δ_1 *is parameterized by the number of queries of a correctness adversary to* G*, then* $\mathsf{KEM}_m^{\not\perp}$ *is* $\delta_1(0)$ *correct. Furthermore, for any* IND-CCA *adversary* $\mathcal{B}$ *against* $\mathsf{KEM}_m^{\not\perp}$*, issuing at most* q_D *queries to the decapsulation oracle, and at most* q_{G}*, and* q_{H} *queries to its random oracles* G *and* H *respectively, there exist a* OW-CPA *adversary* $\mathcal{A}$ *against* PKE_1 *and a* PRF *adversary* $\mathcal{D}$ *against* J *whose running times are about that of* $\mathcal{B}$ *and such that*

$$\mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{KEM}_m^{\not\perp}}(\mathcal{B}) \leq \mathsf{Adv}^{\mathsf{OW\text{-}CPA}}_{\mathsf{PKE}_1}(\mathcal{A}) + \mathsf{Adv}^{\mathsf{PRF}}_{\mathsf{J}}(\mathcal{D}) + 2 \cdot \delta_1(q_{\mathsf{G}} + q_H + 2)$$

Adversary $\mathcal{A}$ *makes at most* $q_{\mathsf{H}} + q_{\mathsf{G}} + 1$ *queries to* G.

Since the proof of Theorem 3.6 is not given in detail by HHK [40,41], we provide the full details. The proof is close, at a high level, to that of Theorem 3.5 [40,41], with simplifications that come from our specific setting.

Game 1	**Game 2**
$\mathsf{Decaps}(c)$:	$\mathsf{Decaps}(c)$:
1: If $c = c^*$ **return** $\perp$	1: If $c = c^*$ **return** $\perp$
2: $m' \leftarrow \mathsf{Dec}(sk, c)$	2: If $c \in \mathcal{L}_D$ **return** $L_D[c]$
3: **if** $m' = \perp$ **or** $\mathsf{Enc}(pk, m'; \mathsf{G}(m')) \neq c$	3: $k \leftarrow\!\!\$\ \{0,1\}^n$
4: $\quad$ **return** $\mathsf{J}(c)$	4: $\mathcal{L}_D[c] \leftarrow k$
5: **else return** $\mathsf{H}(m')$	5: **return** k
$\mathsf{H}(m)$:	$\mathsf{H}(m)$:
1: **if** $m \in T_{\mathsf{H}}$ **return** $T_{\mathsf{H}}[m]$	1: **if** $m \in T_{\mathsf{H}}$ **return** $T_{\mathsf{H}}[m]$
2: $k \leftarrow\!\!\$\ \{0,1\}^n$	2: $c \leftarrow \mathsf{Enc}(pk, m; \mathsf{G}(m))$
3: $T_{\mathsf{H}}[m] \leftarrow k$	3: **if** $\mathsf{Dec}(sk, c) = \perp$ **or** $\mathsf{Dec}(sk, c) \neq m$
4: **return** k	4: $\quad \mathsf{bad}_2 \leftarrow 1$
	5: **if** $m = m^*$ **and** $\mathsf{Dec}(sk, c^*) = m^*$
	6: $\quad \mathsf{bad}_3 \leftarrow 1$
	7: **if** $c \in \mathcal{L}_D$ **then** $k \leftarrow \mathcal{L}_D[c]$
	8: **else** $k \leftarrow\!\!\$\ \{0,1\}^n$; $\mathcal{L}_D[c] \leftarrow k$
	9: $T_{\mathsf{H}}[m] \leftarrow k$
	10: **return** k

Fig. 8. Game 1 to Game 2 hop. In blue: bad event for Game 3.

Proof. The correctness proof is straightforward. The KEM correctness experiment (see Section A)has no adversary, which means that we can construct an adversary against the correctness of PKE_1 by simply sampling a random message and outputting it. The resulting PKE_1 correctness game is identical to the KEM correctness game. Correctness follows from the fact that this PKE_1 adversary makes no calls to G.

For the security proof, we construct a sequence of games. The first hop (Game 1) simply replaces J with a random function, that is outside of the adversary's view, with the same name $\mathsf{J} : \mathcal{M} \to \{0,1\}^n$. The proof of this hop is a direct reduction ($\mathcal{D}$ in Theorem 3) from the PRF property of J.

The jump to Game 2 modifies the decapsulation oracle to rely only on a table $\mathcal{L}_D$. If the input ciphertext c is not in that table, a fresh shared key k is sampled and returned to the adversary; $\mathcal{L}_D$ is updated to ensure consistent simulation of later queries. We also modify random oracle H to guarantee consistency with $\mathcal{L}_D$: whenever a fresh message m is queried to H, we check whether its encryption under PKE_1 is in $\mathcal{L}_D$. If so, then we program H consistently with the corresponding entry in $\mathcal{L}_D$. Otherwise, we lazily sampling a fresh key k, and update *both* $\mathcal{L}_D$ and H. We also set a bad_2 flag in oracle H whenever a correctness error occurs in PKE that would lead to an inconsistency in $\mathcal{L}_D$, and declare the adversary a loser if bad_2 ever occurs. Figure 8 shows pseudocode for Games 1 and 2.

Interestingly, we can show that the probability of bad_2 can be bounded by the correctness of PKE_1, which allows us to avoid performing once again an intricate reduction that involves dealing with random oracle G. Our reduction places at most $q_{\mathsf{H}} + q_{\mathsf{G}} + 2$ queries to its G oracle: 2 queries from the KEM challenge, q_{H} queries to simulate the H oracle as in Game 2, and q_{G} queries to forward $\mathcal{A}$'s G queries. This hop accounts for one of the δ_1 terms in Theorem 3.

It is less straightforward to prove that the two games are, in fact, identical until bad_2 occurs. It implies proving that there are only two possible cases for entries in $\mathcal{L}_D$ created during a call to the decapsulation oracle:

- Those that produce keys that would correspond to implicit rejections;
- Those that would lead to the decapsulation oracle calling H to return a valid shared key, in which case the patching done inside H leads to an adversarial view that is identical to the previous game.

We prove this step by working over a version of the games where random oracle G is eagerly sampled as a random function that is fixed throughout both games—this is done by instantiating the PROM library (Appendix B). Eagerly sampling random oracle outputs is important in the proof, as it allows us to consider PKE_1 as a fully deterministic scheme (rather than a scheme that merely appears deterministic), which is one of the requirements of HHK's Theorem 3.6 [40,41]. It is also very convenient from a formal verification point of view, as it permits writing an intuitive, yet non-trivial, invariant that keeps track of the case analysis above. (See Figure 9.) Note that this invariant only needs to hold until bad_2 occurs. We distinguish the variables in Games 1 and 2 by denoting them with $\{1\}$ and $\{2\}$, respectively, and capture with $\mathsf{goodc}(c, sk, \mathsf{G})$ the fact that a ciphertext is consistent under re-encryption (as defined by sk and G). Here is a breakdown of the invariant point by point:

$$
\begin{aligned}
&\forall c, && c \in l_D\{2\} \wedge \neg \mathsf{goodc}(c, sk, \mathsf{G}) \Rightarrow && c \in \mathsf{J}\{1\} && (1)\\
\wedge\ &\forall c, && c \in \mathsf{J}\{1\} \Rightarrow && \mathcal{L}_D\{2\}[c] = \mathsf{J}\{1\}[c] && (2)\\
\wedge\ &\forall c, && c \in \mathcal{L}_D\{2\} \wedge \mathsf{goodc}(c, sk, \mathsf{G}) \Rightarrow && \mathsf{Dec}_1(sk, c) \in \mathsf{H}\{1\} && (3)\\
\wedge\ &\forall m, && m \in \mathsf{H}\{2\} \Rightarrow && \mathsf{Enc}(m, pk, \mathsf{G}(m)) \in \mathcal{L}_D\{2\} && (4)\\
&&& && \wedge\ \mathcal{L}_D\{2\}[\mathsf{Enc}(m, pk, \mathsf{G}(m))] = \mathsf{H}\{2\}(m) && (5)\\
&&& && \wedge\ \mathsf{H}\{1\}(m) = \mathsf{H}\{2\}(m) && (6)\\
\wedge\ &\forall m, && m \in \mathsf{H}\{1\} \wedge m \notin \mathsf{H}\{2\} \Rightarrow && \mathcal{L}_D\{2\}[\mathsf{Enc}(m, pk, \mathsf{G}(m))] = \mathsf{H}\{1\}(m) && (7)
\end{aligned}
$$

Fig. 9. Invariant for Game 1 to Game 2 hop in the proof of Theorem 3.

$\mathsf{Gen}_K()$:	$\mathsf{Enc}_K(pk)$:	$\mathsf{Dec}_K((sk, s), c)$:
1: $s \leftarrow\!\$\ \mathcal{K}$	1: $m \leftarrow\!\$\ \mathcal{M}$	1: $m' \leftarrow \mathsf{Dec}(sk, c)$
2: $(pk, sk) \leftarrow\!\$\ \mathsf{Gen}()$	2: $(r, k) \leftarrow \mathsf{GH}(m)$	2: $(r, k) \leftarrow \mathsf{GH}(m')$
3: **return** $(pk, (sk, s))$	3: $c \leftarrow \mathsf{Enc}(pk, m; r)$	3: $k' \leftarrow \mathsf{J}(s, c)$
	4: **return** (k, c)	4: **if** $m' = \perp$ **or** $\mathsf{Enc}(pk, m'; r) \neq c$
		5: $\quad$ **return** k'
		6: **else return** k

Fig. 10. The FO transform used in ML-KEM, denoted FO_K.

- Equations 1 and 2: There is a consistency between invalid ciphertexts in $\mathcal{L}_D$ in Game 2 and the values stored in J in Game 1.
- Equations 3,6 and 7: There is a consistency between valid ciphertexts in $\mathcal{L}_D$ in Game 2 and the values stored in H in Game 1, which may include ciphertexts not yet synchronized with H_2 in Game 2.
- Equations 4,5,6: H_2 in Game 2 guarantees consistency with H_1 and Decaps in Game 1 via $\mathcal{L}_D$.

Game 3 is modified to create the challenge encapsulation without calling H, sampling the shared key uniformly at random. In this game the adversary has no advantage in distinguishing the challenge encapsulation key from a random one. This claim is relatively straightforward to verify in EasyCrypt using probabilistic Hoare logic. It remains to bound the distance between Games 2 and 3.

Again, this is done by an up-to-bad argument: the two games are identical unless H is queried on the message m^* encrypted under PKE_1 in the challenge encapsulation. This is shown as bad_3 in Figure 8, and we note that we actually capture a more fine-grained event that is closer to the winning condition of the $\mathsf{OW\text{-}CPA}$ game for PKE_1, as given in Definition 1. Proving that the two games are identical until bad_3 occurs is straightforward using the logics of EasyCrypt, and relying on the fact that the adversary can only win at this point if no correctness error has occurred due to the action of bad_2.

It remains to bound the probability of bad_3 occurring, using a reduction from the OW security of PKE_1. The reduction, of course, cannot directly check whether bad_3 has occurred using the secret key, and instead simply checks whether a solution to the OW challenge is found by checking that $\mathsf{Enc}(pk, m; \mathsf{G}(m)) = c^*$. There is therefore a small gap between the probability of the reduction winning the one-wayness game and the occurrence of bad_3: this occurs if the challenge ciphertext decrypts to something other than m^*. We carry out a final up-to-bad analysis of our reduction, and show that it does break the one-wayness of the underlying scheme, except if there is a correctness error in PKE_1. This accounts for the remaining δ_1 terms in Theorem 3. □

5.3 The FO transform in ML-KEM

The FO transform used in ML-KEM, FO_K, is shown in Figure 10. It uses a *single* random oracle GH, but it is identical to the $\mathsf{KEM}_m^{\not\perp}$ transform considered above if one sees GH as the product of two independent random oracles $\mathsf{G} \times \mathsf{H}$. Using exactly this intuition, we prove the following theorem in EasyCrypt.

Theorem 4. *Assume* PKE *to be* δ*-correct. Then* FO_K *is* δ*-correct. Furthermore, for any* $\mathsf{IND\text{-}CCA}$ *adversary* $\mathcal{B}$ *against* FO_K*, issuing at most* q_D *queries to the decapsulation oracle, and at most* q_{GH} *queries to its random oracle* GH*, there exist an* $\mathsf{IND\text{-}CPA}$ *adversary* $\mathcal{A}$ *against* PKE *and a* PRF *adversary* $\mathcal{D}$ *against* J *s.t.*

$$\mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{FO}_K}(\mathcal{B}) \leq \mathsf{Adv}^{\mathsf{PRF}}_{\mathsf{J}}(\mathcal{D}) + Q \cdot \delta + \frac{2(2q_{\mathsf{GH}} + 2)}{|\mathcal{M}|} + 4 \cdot \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{PKE}}(\mathcal{A})$$

for $Q = 3 \cdot (2q_{\mathsf{GH}} + 3) + 1$ *and the running times of* $\mathcal{A}$ *and* $\mathcal{D}$ *are about that of* $\mathcal{B}$.

Proof (Sketch). Both the correctness and the security statements follow from the correctness and security of $\mathsf{KEM}_m^{\not\perp}$ by direct reductions. These reductions perfectly simulate calls to GH by calling both G and H. A simple invariant in EasyCrypt then permits showing that all values that $\mathcal{B}$ has seen in GH exactly match those queried by the reduction to G and H. Furthermore, this invariant is preserved on every fresh call by arguing that a sample at the output (r, k) of GH of type $\mathcal{R} \times \mathcal{K}$ is identically distributed to the value that results from grouping (r, k) coming from independent (guaranteed to be fresh) calls to G and H. Using this invariant, it is straightforward that the reductions perfectly simulate FO_K while attacking $\mathsf{KEM}_m^{\not\perp}$ with the same advantage as $\mathcal{B}$. Then, the theorem follows from instantiating Theorem 3 with the bound in Theorem 2. □

6 Main result: Security of ML-KEM

The main theorem in our machine-checked proof of ML-KEM applies to an EasyCrypt specification of the ML-KEM draft standard [54], obtained with minor and necessary adaptations from the Kyber specification used in [8]. This choice allows us, in Section 7, to combine the algorithmic security result in this section with the implementation equivalence proofs [8] (again with minor adaptations) and connect the security theorems we prove here to concrete assembly implementations of ML-KEM.

6.1 Instantiating FO_K with K-PKE

We first instantiate the results on FO_K obtained in Section 5 with those obtained in Section 4. This instantiation is straightforward in EasyCrypt, which easily supports purely modular results. Note, however that, at this point, correctness is still conditional on the assumptions about the encoding and decoding operators introduced in Section 4.

Theorem 5. *Let* δ *be the maximum probability that experiment* $\mathsf{CorrectnessBound}$ *returns* 1*, and* ϵ_{kg} *and* ϵ_{enc} *denote the pseudorandomness advantage against the derandomizing of key generation and encryption algorithms in* $\mathsf{K\text{-}PKE}$*, as defined in Section 4. Then,* $\mathsf{FO}_K[\mathsf{K\text{-}PKE}, \mathsf{J}]$ *is* $(\delta + \epsilon_{\mathsf{kg}} + \epsilon_{\mathsf{enc}})$*-correct. Furthermore, let* $\epsilon_{\mathsf{MLWE\_H}}$ *denote the maximum advantage of the reductions against Hashed MLWE constructed Section 4.*[24]

Then for any $\mathsf{IND\text{-}CCA}$ *adversary* $\mathcal{B}$ *against* $\mathsf{FO}_K[\mathsf{K\text{-}PKE}, \mathsf{J}]$*, issuing at most* q_D *queries to the decapsulation oracle, and at most* q_{GH} *queries to its random oracle* GH*, there exists a* PRF *adversary* $\mathcal{D}$ *against* J *such that*

$$\mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{FO}_K[\mathsf{K\text{-}PKE},\mathsf{J}]}(\mathcal{B}) \leq \mathsf{Adv}^{\mathsf{PRF}}_{\mathsf{J}}(\mathcal{D}) + Q \cdot (\delta + \epsilon_{\mathsf{kg}} + \epsilon_{\mathsf{enc}}) + \frac{2(2q_{\mathsf{GH}} + 2)}{|\mathcal{M}|} + 8 \cdot \epsilon_{\mathsf{MLWE\_H}}$$

for $Q = 3 \cdot (2q_{\mathsf{GH}} + 3) + 5$ *and the running times of the reductions to Hashed MLWE and* $\mathcal{D}$ *are about that of* $\mathcal{B}$.

6.2 Connection to the ML-KEM specification

The ML-KEM-768 specification can be found in the artifact that accompanies this paper under `crypto-specs/mlkem`. It is, for the most part, identical to the specification of Kyber described in detail in [8]. The only difference resides in the file that implements the top-level KEM construction, where the hashing pattern of the Fusijaki-Okamoto transform was modified to match the change introduced in the ML-KEM draft standard and briefly discussed in Section 3.2. Let us denote the EasyCrypt module that fully specifies ML-KEM by MLKEM and note that the key generation and encryption algorithms are given in derandomized form, i.e., taking all randomness as explicit additional input. We obtain a security theorem that applies to this specification in four steps:

1. Instantiate Theorem 5 by making concrete all remaining abstractions, and discharging assumptions. These proofs reuse some lemmas on the NTT and other operations from [8].

[24] In this section we simplify the theorem statements by using quantities such as $\epsilon_{\mathsf{MLWE\_H}}$ to denote the maximum advantage among a small number of fully defined reductions to the same assumption. They do not represent the computation of a maximum advantage over all adversaries in a class.

```
op pk_encode(pk : W8.t Array32.t * polyvec) : pkey =
                              (encode12_vec (toipolyvec (nttv pk[2])), pk[1]).
op pk_decode(pk : pkey) = (pk[2], invnttv (ofipolyvec (sem_decode12_vec (pk[1])))).
op sk_encode(sk : polyvec) : skey = encode12_vec_aux (toipolyvec (nttv sk)).
op sk_decode(sk : skey) = invnttv (ofipolyvec (sem_decode12_vec sk)).
op m_encode(m : plaintext) : poly = decompress_poly 1 (sem_decode1 m).
op m_decode(p : poly) : plaintext = encode1 (compress_poly 1 p).
op c_encode(c : polyvec * poly) : ciphertext =
   (encode10_vec (compress_polyvec 10 c[1]), encode4 (compress_poly 4 c[2])).
op c_decode(c : ciphertext) =
   (decompress_polyvec 10 (sem_decode10_vec c[1]),
                                          decompress_poly 4 (sem_decode4 c[2])).
op rnd_err_v = compress_poly_err 4.
op rnd_err_u = mapv (compress_poly_err 10).
op max_noise = q %/ 4 − 1.
op under_noise_bound (p : poly) (b : int) = all (fun cc ⇒ | as_sint cc| ≤ b) p.
op cv_bound_max : int = 104.
```

Fig. 11. Snippet of the instantiation of Theorem 5.

2. Define an alternative version of MLKEM, which we call MLKEM_Op below, where we introduce two modifications: 1) the hash function used in the FO transform is turned into an oracle, aligning with the GH oracle in FO_K; and 2) the rejection sampling procedures for the matrix A used in key generation and encapsulation are replaced by a deterministic function H.[25]
3. Prove that, when H is implemented as in ML-KEM, and when GH is instantiated with SHA3-512, MLKEM_Op is the same as MLKEM. This step clarifies how our results apply to MLKEM: GH is modeled as a random oracle, and the matrix sampling procedure fixes the concrete parameters of the Hashed MLWE assumption.
4. Prove that MLKEM_Op is a secure KEM when the random coins of key generation and key encapsulation are sampled uniformly at random. This is done by showing that the advantage of any attacker in breaking MLKEM_Op in the ROM can be bounded using (the concrete version of) Theorem 5.

We now discuss each of these steps next.

Refining Theorem 5. We refine Theorem 5 via theory cloning and parameter instantiation: some of the types and operators left abstract in Section 4 are given concrete definitions, and all axioms about them are discharged. The abstract parameters we instantiate here include the polynomial ring, matrix and vector dimensions, the types of keys, ciphertexts, shared keys, etc., the randomness distributions and the randomness expansion operators, the compression and serialization procedures, the noise thresholds required for bounding the failure probability, and the type of the RO.

Figure 11 shows (some) of the concrete instantiations. For example, the pk_encode and sk_encode operations make it explicit that public keys and secret keys are encoded in the NTT domain, so proving the correctness of deserialization implies reasoning about the cancellation of the NTT. Important lemmas discharged at this point include the proof that rnd_err_v and rnd_err_u correctly describe compression errors as additive noise, the proof that decode_msg correctly recovers an encoded message as long as all the coefficients are affected by a noise value under $q/4$, and the proof that the maximum rounding error occurring in a coefficient of v is cv_bound_max = 104.

MLKEM_Op and Relation to MLKEM. Figure 12 shows MLKEM_Op, an operator-based ML-KEM specification that we use to connect the security theorem we obtain above to the ML-KEM specification. We prove that MLKEM_Op is equivalent to MLKEM when H is defined as the semantics of the matrix sampling procedure specified by MLKEM, and when the random oracle passed to MLKEM_Op is instantiated with SHA3-512.

The proof is mostly boiler plate; it implies providing functional descriptions of serialization routines and randomness expansion procedures and then proving that these operators are correctly expressing the

[25] Note that we see SHA3-512 as a random oracle only for the fixed input size used in the FO transform, which includes a message and the hash of the public-key. When SHA3-512 is used in K-PKE with a different input size—to smooth the key generation randomness—we model it as a stateless, fixed-length, pseudorandom generator.

```
module InnerPKE_Op = {
  proc kg_derand(coins: W8.t Array32.t)
          : pkey * skey = {
    (rho,s,e) ← prg_kg coins;
    a ← nttm (H rho); s ← nttv s; e ← nttv e;
    t ← ntt_mmul a s + e;
    tv ← encode12_vec_aux (toipolyvec t);
    sv ← encode12_vec_aux (toipolyvec s);
    return ((tv,rho),sv); }

  proc enc_derand(pk : pkey, m : plaintext,
                  r : W8.t Array32.t) : ciphertext = {
    (rv,e1,e2) ← prg_enc r; (tv,rho) ← pk;
    thati ← decode12_vec_aux tv;
    that ← ofipolyvec thati;
    aT ← nttm (trmx (H rho));
    rhat ← nttv rv;
    u ← invnttv (ntt_mmul aT rhat) + e1;
    mp ← decode1 m;
    v ← invntt (ntt_dotp that rhat) +
              e2 + decompress_poly 1 mp;
    c1 ← encode10_vec_aux (compress_polyvec 10 u);
    c2 ← encode4 (compress_poly 4 v);
    return (c1,c2); }

  proc dec(sk : skey, cph : ciphertext)
                    : plaintext option = {
    (c1,c2) ← cph;
    ui ← decode10_vec_aux c1;
    u ← decompress_polyvec 10 ui;
    vi ← decode4 c2;
    v ← decompress_poly 4 vi;
    si ← decode12_vec_aux sk;
    s ← ofipolyvec si;
    mp ← v + ((−) (invntt (ntt_dotp s (nttv u))));
    m ← encode1 (compress_poly 1 mp);
    return Some m; }
}.
```

```
module (MLKEM_Op : Scheme) (O : POracle) = {
  proc kg_derand(coins : W8.t Array32.t * W8.t Array32.t)
           : publickey * secretkey = {
    kgs ← coins[1]; z ← coins[2];
    (pk, sk) ← InnerPKE_Op.kg_derand(kgs);
    hpk ← H_pk pk;
    return (pk, (sk, pk, hpk, z));
  }

  proc kg() : publickey * secretkey = {
    coins ←$ srand; k ←$ srand;
    (pk, sk) ← kg_derand((coins,k));
    return (pk,sk); }

  proc enc_derand(pk : publickey, coins : W8.t Array32.t)
                       : ciphertext * sharedsecret = {
    m ← coins; hpk ← H_pk pk;
    (_K, r) ← O.get(m,hpk);
    c ← InnerPKE_Op.enc_derand(pk, m, r);
    return (c, _K); }

  proc enc(pk : publickey)
                : ciphertext * sharedsecret = {
    coins ←$ srand; (c, _K) ← enc_derand(pk,coins);
    return (c,_K); }

  proc dec(sk : secretkey, cph : ciphertext)
                : sharedsecret option = {
    (skp, pk, hpk, z) ← sk;
    m ← InnerPKE_Op.dec(skp, cph);
    (_K, r) ← O.get(oget m,hpk);
    _K' ← J z cph;
    c ← InnerPKE_Op.enc_derand(pk, oget m, r);
    if (c ≠ cph) _K ← _K';
    return (Some _K);
  }
}.
```

Fig. 12. Operator-based version of the ML-KEM specification.

semantics of the corresponding MLKEM procedures. This proof applies to the derandomized entry points kg_derand, enc_derand, and dec. However, when we analyze correctness and security of MLKEM_Op, we use the entry points kg and enc for key generation and encapsulation that simply sample the necessary coins uniformly at random before calling the derandomized entry points.

Security Theorem for MLKEM_Op. Having defined MLKEM_Op, we can prove the following theorem.

Theorem 6 (Main Theorem). *Define for notation conciseness the following quantities:*

- *let* δ *be the maximum probability that experiment* CorrectnessBound *returns* 1
- *let* ϵ_{prf} *denote the pseudorandomness advantage against the PRF used in the noise sampling procedure of ML-KEM, which is SHAKE-256 with a fixed output size of* 128 *bytes,*
- *let* ϵ_{J} *denote the pseudorandomness advantage against the PRF used in the implicit rejection step of ML-KEM, which is SHAKE-256 with a fixed output size of* 32 *bytes,*
- *let* ϵ_{hs} *denote the pseudorandomness advantage against the smoothing function* G_coins *used in ML-KEM key generation, and*
- *let* $\epsilon_{\mathsf{MLWE\_H}}$ *denote the advantage against Hashed MLWE, when* H *is defined as the matrix rejection sampling procedure defined by ML-KEM.*

Then, MLKEM_Op *is* $(\delta + \epsilon_{\mathsf{hs}} + 2\epsilon_{\mathsf{prf}})$*-correct. Furthermore, for any* IND-CCA *adversary* $\mathcal{B}$ *against* MLKEM_Op*, issuing at most* q_D *queries to the decapsulation oracle, and at most* q_{GH} *queries to its random oracle* GH*, we have that*

$$\mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{MLKEM\_Op}}(\mathcal{B}) \leq \epsilon_{\mathsf{J}} + Q \cdot (\delta + \epsilon_{\mathsf{hs}} + 2\epsilon_{\mathsf{prf}}) + \frac{2(2q_{\mathsf{GH}} + 2)}{|\mathcal{M}|} + 8 \cdot \epsilon_{\mathsf{MLWE\_H}}$$

for $Q = 3 \cdot (2q_{\mathsf{GH}} + 3) + 5$, $|\mathcal{M}| = 2^{256}$ *and the running times of all reductions are about that of* $\mathcal{B}$*.*

Proof (Sketch). The proof of this theorem in EasyCrypt has two parts. The first part is technically a simple equivalence. We show that any attacker $\mathcal{B}$ against MLKEM_Op is actually executing in an environment which is identical to that of an attacker against (the instantiated version of) Theorem 5.

```
proc prg_kg(coins:W8.t Array32.t)
      : W8.t Array32.t * polyvec * polyvec = {
  (rho, noiseseed) ← G_coins coins;
  PRF.k ← noiseseed;
  _N ← 0;  i ← 0;
  while (i < kvec) {
    c ← CBD2_PRF(PRF).sample(_N);
    noise1 ← noise1[i←c];
    _N ← _N + 1; i ← i + 1; }
  i ← 0;
  while (i < kvec) {
    c ← CBD2_PRF(PRF).sample(_N);
    noise2 ← noise2[i←c];
    _N ← _N + 1; i ← i + 1;   }
  return (rho,noise1,noise2);
}
```

```
proc prg_enc(noiseseed:W8.t Array32.t)
           : polyvec * polyvec * poly = {
  PRF.k ← noiseseed;
  _N ← 0; i ← 0;
  while (i < kvec) {
    c ← CBD2_PRF(PRF).sample(_N);
    noise1 ← noise1[i←c];
    _N ← _N + 1;  i ← i + 1; }
  i ← 0;
  while (i < kvec) {
    c ← CBD2_PRF(PRF).sample(_N);
   noise2 ← noise2[i←c];
    _N ← _N + 1; i ← i + 1; }
  e2 ← CBD2_PRF(PRF).sample(_N);
  return (noise1,noise2, e2);
}
```

Fig. 13. Randomness expansion procedures for ML-KEM.

Again, proving this step is mostly boiler-plate, showing that the operations on both experiments exactly match semantically, even though they may be written in a slightly different way.

The second part is a bit more challenging and much more interesting. Note that Theorem 5 talks about the pseudorandomness properties of operators prg_kg and prg_enc that expand the coins passed to key generation and encapsulation to approximate the distributions specified by the underlying Hashed MLWE assumption. However, the theorem we are proving states the final advantage expression as a function of the pseudorandomness security bounds offered by SHAKE-256. The prg_kg and prg_enc operators are defined as follows:

```
op prg_kg(coins : W8.t Array32.t) : W8.t Array32.t * polyvec * polyvec =
      ((G_coins coins)[1],
      offunv (fun i ⇒ cbd2sample (PRF (G_coins coins)[2] (W8.of_int i))),
      offunv (fun i ⇒ cbd2sample (PRF (G_coins coins)[2] (W8.of_int (i + 3))))).
op prg_enc(coins : W8.t Array32.t) : polyvec * polyvec * poly =
      (offunv (fun i ⇒ cbd2sample (PRF coins (W8.of_int i))),
      offunv (fun i ⇒ cbd2sample (PRF coins (W8.of_int (i + 3)))),
      cbd2sample (PRF coins (W8.of_int 6))).
```

They are the functional counterparts of the procedures in Figure 13, which we use in an intermediate step of the proof that MLKEM_Op and MLKEM are equivalent (i.e., in the first part of this proof). We omit from Figure 13 the details of procedure CBD2_PRF(PRF).sample that specifies the procedure CBD_η procedure introduced in Section 3 for $\eta = 2$. However, it was proved in [8] that this procedure generates the correct distribution of binomial noise when the output of PRF is replaced with uniform random bytes on each call. This allows us to show that the sampling procedures in Figure 13, when fed with uniform byte strings, generate distributions that are computationally close to those specified by operators prg_kg_ideal and prg_enc_ideal in Section 4. The proof for prg_enc is a direct reduction to the pseudorandomness of PRF, whereas the proof for prg_og requires us to first replace the output of G_coins with random byte strings, which gives us a uniform rho, and then mimic the prg_enc proof to justify the noise distribution. To the best of our knowledge, these low-level details of the operation of ML-KEM, although intuitive, have never been laid down in a concrete security bound. In our theorem, these are accounted for as the $(\epsilon_{\mathsf{hs}} + 2\epsilon_{\mathsf{prf}})$ terms, which make it explicit that a randomness expansion advantage term must be accounted for every time K-PKE correctness and security are used in the FO proof. □

7 Connecting to ML-KEM implementations

Provably secure specifications are good; provably secure implementations are better. Following [8] and the methodology first outlined in [6], we 1) adapt the Kyber implementation from [8] to implement ML-KEM; 2) prove that the resulting implementation is functionally correct with respect to the ML-KEM specification from Section 6; and 3) prove that the implementation is "cryptographic constant-time". This last task is carried out using Jasmin's type system, which ensures that well-typed code does not branch on secrets, does not access memory at secret locations, and uses basic operations which are not constant time—like division and modulo—on public arguments only. We note that our security proof allows us to clarify the assumptions that justify constant-time security of the implementations.[26] We can therefore

[26] For example, our proof of constant-time explicitly requires us to declassify the public seed ρ that gives rise to the matrix A. This σ is generated as $(\rho, \sigma) \leftarrow \mathrm{G}(d)$ in key generation. Here, σ is used to generate the secret

state that the implementations are *as secure* as the ML-KEM specification itself: an adversary interacting with the implementation can be converted into one that attacks the specification with the same success probability, as long as the attacker is not exploiting implementation leakage beyond what we consider in the constant-time paradigm.

Finally, we test that the test vectors produced by these implementations match those produced by the reference implementation of the draft standard available from the `pq-crystals/kyber` code package[27] and most of the more extensive test vectors that were made available by Schmieg[28]. The only mismatch between the test vectors by our implementations and the ones by Schmieg's implementation are relating to public-key validation, which Schmieg implements, but we deliberately do not (see Section 3.2). Also, the C function-call ABI used by the Jasmin implementations passes keys trough pointers; it is the caller's responsibility to ensure that those pointers point to memory regions of appropriate length and the callee has no way of checking this. We thus skip the tests with keys and ciphertexts of wrong length from Schmieg's set of test vectors.

The Jasmin compiler is certified to preserve semantics through compilation. Hence, through our proof of equivalence between EasyCrypt specification and Jasmin implementations, we conclude that our specification of ML-KEM is correct with respect to the test vectors used by library developers. Dually, we conclude that the assembly implementations produced by the Jasmin compiler correctly implement the EasyCrypt specification of the ML-KEM standard, and thus satisfy IND-CCA security down to cryptographic assumptions.

Acknowledgments We gratefully acknowledge discussions and support from the Formosa Crypto community, and Andreas Hülsing in particular.

This research was supported by Deutsche Forschungsgemeinschaft (DFG, German research Foundation) as part of the Excellence Strategy of the German Federal and State Governments – EXC 2092 CASA - 390781972; by the European Commission through the ERC Starting Grant 805031 (EPOQUE); by the German Federal Ministry of Education and Research (BMBF) in the course of the 6GEM research hub under grant number 16KISK038; by the *Agence Nationale de la Recherche* (French National Research Agency) as part of the France 2030 programme – ANR-22-PECY-0006; by Amazon Web Services, as an Amazon Research Award supporting the Formosa Crypto consortium; by an EPSRC Doctoral Training Partnership (EP/T517872/1); and by the InnovateUK ATI programme (10065634). This work was supported by European Structural and Investment Funds in the FEDER component, and through the Operational Competitiveness and Internationalization Programme (COMPETE 2020) (Project No. 047264; Funding Reference: POCI-01-0247-FEDER-047264).

References

1. Alagic, G., Apon, D., Cooper, D., Dang, Q., Dang, T., Kelsey, J., Lichtinger, J., Miller, C., Moody, D., Peralta, R., Perlner, R., Robinson, A., Smith-Tone, D., Liu, Y.K.: Status report on the third round of the NIST post-quantum cryptography standardization process. NISTIR 8413 (2022), https://csrc.nist.gov/publications/detail/nistir/8413/final 1
2. Albrecht, M., Ducas, L.: Lattice attacks on NTRU and LWE: A history of refinements. Cryptology ePrint Archive, Report 2021/799 (2021), https://eprint.iacr.org/2021/799 3
3. Albrecht, M.R.: On dual lattice attacks against small-secret LWE and parameter choices in HElib and SEAL. In: Coron, J.S., Nielsen, J.B. (eds.) EUROCRYPT 2017, Part II. LNCS, vol. 10211, pp. 103–129. Springer, Heidelberg (Apr / May 2017). https://doi.org/10.1007/978-3-319-56614-6_4 3
4. Albrecht, M.R., Ducas, L., Herold, G., Kirshanova, E., Postlethwaite, E.W., Stevens, M.: The general sieve kernel and new records in lattice reduction. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part II. LNCS, vol. 11477, pp. 717–746. Springer, Heidelberg (May 2019). https://doi.org/10.1007/978-3-030-17656-3_25 3
5. Almeida, J.B., Barbosa, M., Barthe, G., Blot, A., Grégoire, B., Laporte, V., Oliveira, T., Pacheco, H., Schmidt, B., Strub, P.Y.: Jasmin: High-assurance and high-speed cryptography. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017. pp. 1807–1823. ACM Press (Oct / Nov 2017). https://doi.org/10.1145/3133956.3134078 2

key, so clearly d cannot be public. In fact, the only way to justify this declassify statement is by looking at the security proof and observing that G is modeled as a PRG, and so ρ can be considered to be independent of σ and d under that assumption.

[27] https://github.com/pq-crystals/kyber/blob/standard/

[28] See https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/aCAX-2QrUFw/m/hy5gwcESAAAJ.

6. Almeida, J.B., Barbosa, M., Barthe, G., Dupressoir, F.: Verifiable side-channel security of cryptographic implementations: Constant-time MEE-CBC. In: Peyrin, T. (ed.) FSE 2016. LNCS, vol. 9783, pp. 163–184. Springer, Heidelberg (Mar 2016). https://doi.org/10.1007/978-3-662-52993-5_9 3, 21
7. Almeida, J.B., Barbosa, M., Barthe, G., Grégoire, B., Koutsos, A., Laporte, V., Oliveira, T., Strub, P.Y.: The last mile: High-assurance and high-speed cryptographic implementations. In: 2020 IEEE Symposium on Security and Privacy. pp. 965–982. IEEE Computer Society Press (May 2020). https://doi.org/10.1109/SP40000.2020.00028 2
8. Almeida, J.B., Barbosa, M., Barthe, G., Grégoire, B., Laporte, V., Léchenet, J.C., Oliveira, T., Pacheco, H., Quaresma, M., Schwabe, P., Séré, A., Strub, P.Y.: Formally verifying kyber episode IV: Implementation correctness. IACR TCHES **2023**(3), 164–193 (2023). https://doi.org/10.46586/tches.v2023.i3.164-193 2, 3, 18, 21
9. Almeida, J.B., Baritel-Ruet, C., Barbosa, M., Barthe, G., Dupressoir, F., Grégoire, B., Laporte, V., Oliveira, T., Stoughton, A., Strub, P.Y.: Machine-checked proofs for cryptographic standards: Indifferentiability of sponge and secure high-assurance implementations of SHA-3. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019. pp. 1607–1622. ACM Press (Nov 2019). https://doi.org/10.1145/3319535.3363211 3, 28
10. Avanzi, R., Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS-Kyber: Algorithm specifications and supporting documentation (version 3.02). Round-3 submission to the NIST PQC standardization project (2021), https://pq-crystals.org/kyber/data/kyber-specification-round3-20210804.pdf 1
11. Barbosa, M., Barthe, G., Bhargavan, K., Blanchet, B., Cremers, C., Liao, K., Parno, B.: SoK: Computer-aided cryptography. In: 2021 IEEE Symposium on Security and Privacy. pp. 777–795. IEEE Computer Society Press (May 2021). https://doi.org/10.1109/SP40001.2021.00008 1, 3
12. Barbosa, M., Barthe, G., Doczkal, C., Don, J., Fehr, S., Grégoire, B., Huang, Y.H., Hülsing, A., Lee, Y., Wu, X.: Fixing and mechanizing the security proof of Fiat-Shamir with aborts and Dilithium. In: Handschuh, H., Lysyanskaya, A. (eds.) CRYPTO 2023, Part V. LNCS, vol. 14085, pp. 358–389. Springer, Heidelberg (Aug 2023). https://doi.org/10.1007/978-3-031-38554-4_12 3
13. Barbosa, M., Barthe, G., Fan, X., Grégoire, B., Hung, S.H., Katz, J., Strub, P.Y., Wu, X., Zhou, L.: EasyPQC: Verifying post-quantum cryptography. In: Vigna, G., Shi, E. (eds.) ACM CCS 2021. pp. 2564–2586. ACM Press (Nov 2021). https://doi.org/10.1145/3460120.3484567 3
14. Barbosa, M., Dupressoir, F., Grégoire, B., Hülsing, A., Meijers, M., Strub, P.Y.: Machine-checked security for rmXMSS as in RFC 8391 and SPHINCS$^+$. In: Handschuh, H., Lysyanskaya, A. (eds.) CRYPTO 2023, Part V. LNCS, vol. 14085, pp. 421–454. Springer, Heidelberg (Aug 2023). https://doi.org/10.1007/978-3-031-38554-4_14 3
15. Barbosa, M., Hülsing, A.: The security of Kyber's FO-transform. Cryptology ePrint Archive, Report 2023/755 (2023), https://eprint.iacr.org/2023/755 6
16. Barthe, G., Dupressoir, F., Grégoire, B., Kunz, C., Schmidt, B., Strub, P.Y.: EasyCrypt: A Tutorial, pp. 146–166. Springer (2014), 5
17. Barthe, G., Fan, X., Gancher, J., Grégoire, B., Jacomme, C., Shi, E.: Symbolic proofs for lattice-based cryptography. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) ACM CCS 2018. pp. 538–555. ACM Press (Oct 2018). https://doi.org/10.1145/3243734.3243825 3
18. Barthe, G., Grégoire, B., Heraud, S., Zanella Béguelin, S.: Computer-aided security proofs for the working cryptographer. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 71–90. Springer, Heidelberg (Aug 2011). https://doi.org/10.1007/978-3-642-22792-9_5 2
19. Becker, A., Ducas, L., Gama, N., Laarhoven, T.: New directions in nearest neighbor searching with applications to lattice sieving. In: SODA '16: Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms. pp. 10–24. Society for Industrial and Applied Mathematics (2016) 3
20. Beringer, L., Petcher, A., Ye, K.Q., Appel, A.W.: Verified correctness and security of OpenSSL HMAC. In: Jung, J., Holz, T. (eds.) USENIX Security 2015. pp. 207–221. USENIX Association (Aug 2015) 3
21. Bernstein, D.J., Persichetti, E.: Towards KEM unification. Cryptology ePrint Archive, Report 2018/526 (2018), https://eprint.iacr.org/2018/526 13
22. Bindel, N., Hamburg, M., Hövelmanns, K., Hülsing, A., Persichetti, E.: Tighter proofs of CCA security in the quantum random oracle model. In: Hofheinz, D., Rosen, A. (eds.) TCC 2019, Part II. LNCS, vol. 11892, pp. 61–90. Springer, Heidelberg (Dec 2019). https://doi.org/10.1007/978-3-030-36033-7_3 3
23. Blanchette, J., Mahboubi, A. (eds.): Handbook of Proof Assistants. Springer-Verlag (2025), to appear 3
24. Boneh, D., Dagdelen, Ö., Fischlin, M., Lehmann, A., Schaffner, C., Zhandry, M.: Random oracles in a quantum world. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 41–69. Springer, Heidelberg (Dec 2011). https://doi.org/10.1007/978-3-642-25385-0_3 3, 4
25. Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Stehlé, D.: CRYSTALS – Kyber: a CCA-secure module-lattice-based KEM. In: 2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018. pp. 353–367. IEEE (2018), https://eprint.iacr.org/2017/634 1, 2, 3, 8, 9, 10, 12
26. Cremers, C., Fontaine, C., Jacomme, C.: A logic and an interactive prover for the computational post-quantum security of protocols. In: 2022 IEEE Symposium on Security and Privacy. pp. 125–141. IEEE Computer Society Press (May 2022). https://doi.org/10.1109/SP46214.2022.9833800 4

27. D'Anvers, J.P., Karmakar, A., Roy, S.S., Vercauteren, F., Mera, J.M.B., Beirendonck, M.V., Basso, A.: SABER. Tech. rep., National Institute of Standards and Technology (2020), available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions 3
28. Ducas, L.: Shortest vector from lattice sieving: A few dimensions for free. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part I. LNCS, vol. 10820, pp. 125–145. Springer, Heidelberg (Apr / May 2018). https://doi.org/10.1007/978-3-319-78381-9_5 3
29. Ducas, L., Pulles, L.N.: Does the dual-sieve attack on learning with errors even work? In: Handschuh, H., Lysyanskaya, A. (eds.) CRYPTO 2023, Part III. LNCS, vol. 14083, pp. 37–69. Springer, Heidelberg (Aug 2023). https://doi.org/10.1007/978-3-031-38548-3_2 3
30. Duman, J., Hövelmanns, K., Kiltz, E., Lyubashevsky, V., Seiler, G.: Faster lattice-based KEMs via a generic fujisaki-okamoto transform using prefix hashing. In: Vigna, G., Shi, E. (eds.) ACM CCS 2021. pp. 2722–2737. ACM Press (Nov 2021). https://doi.org/10.1145/3460120.3484819 3
31. Fujisaki, E., Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. In: Wiener, M.J. (ed.) CRYPTO'99. LNCS, vol. 1666, pp. 537–554. Springer, Heidelberg (Aug 1999). https://doi.org/10.1007/3-540-48405-1_34 6, 13
32. Fujisaki, E., Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. Journal of Cryptology **26**(1), 80–101 (Jan 2013). https://doi.org/10.1007/s00145-011-9114-1 6, 13
33. Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: Ladner, R.E., Dwork, C. (eds.) 40th ACM STOC. pp. 197–206. ACM Press (May 2008). https://doi.org/10.1145/1374376.1374407 3, 4
34. Grubbs, P., Maram, V., Paterson, K.G.: Anonymous, robust post-quantum public key encryption. In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022, Part III. LNCS, vol. 13277, pp. 402–432. Springer, Heidelberg (May / Jun 2022). https://doi.org/10.1007/978-3-031-07082-2_15 3, 6
35. Guo, Q., Johansson, T.: Faster dual lattice attacks for solving LWE with applications to CRYSTALS. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021, Part IV. LNCS, vol. 13093, pp. 33–62. Springer, Heidelberg (Dec 2021). https://doi.org/10.1007/978-3-030-92068-5_2 3
36. Guo, Q., Johansson, T., Nilsson, A.: A key-recovery timing attack on post-quantum primitives using the Fujisaki-Okamoto transformation and its application on FrodoKEM. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part II. LNCS, vol. 12171, pp. 359–386. Springer, Heidelberg (Aug 2020). https://doi.org/10.1007/978-3-030-56880-1_13 3
37. Hamburg, M., Hermelink, J., Primas, R., Samardjiska, S., Schamberger, T., Streit, S., Strieder, E., van Vredendaal, C.: Chosen ciphertext k-trace attacks on masked CCA2 secure kyber. IACR TCHES **2021**(4), 88–113 (2021). https://doi.org/10.46586/tches.v2021.i4.88-113, https://tches.iacr.org/index.php/TCHES/article/view/9061 3
38. Hermelink, J., Mårtensson, E., Samardjiska, S., Pessl, P., Rodosek, G.D.: Belief propagation meets lattice reduction: Security estimates for error-tolerant key recovery from decryption errors. IACR TCHES **2023**(4), 287–317 (2023). https://doi.org/10.46586/tches.v2023.i4.287-317 3
39. Hermelink, J., Streit, S., Strieder, E., Thieme, K.: Adapting belief propagation to counter shuffling of NTTs. IACR TCHES **2023**(1), 60–88 (2023). https://doi.org/10.46586/tches.v2023.i1.60-88 3
40. Hofheinz, D., Hövelmanns, K., Kiltz, E.: A modular analysis of the Fujisaki-Okamoto transformation. In: Kalai, Y., Reyzin, L. (eds.) TCC 2017, Part I. LNCS, vol. 10677, pp. 341–371. Springer, Heidelberg (Nov 2017). https://doi.org/10.1007/978-3-319-70500-2_12 6, 10, 13, 14, 15, 16, 26
41. Hofheinz, D., Hövelmanns, K., Kiltz, E.: A modular analysis of the Fujisaki-Okamoto transformation. Cryptology ePrint Archive, Report 2017/604 (2017), https://eprint.iacr.org/2017/604 10, 13, 14, 15, 16, 26
42. Hövelmanns, K., Hülsing, A., Majenz, C.: Failing gracefully: Decryption failures and the fujisaki-okamoto transform. In: Agrawal, S., Lin, D. (eds.) ASIACRYPT 2022, Part IV. LNCS, vol. 13794, pp. 414–443. Springer, Heidelberg (Dec 2022). https://doi.org/10.1007/978-3-031-22972-5_15 3
43. Hövelmanns, K., Kiltz, E., Schäge, S., Unruh, D.: Generic authenticated key exchange in the quantum random oracle model. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds.) PKC 2020, Part II. LNCS, vol. 12111, pp. 389–422. Springer, Heidelberg (May 2020). https://doi.org/10.1007/978-3-030-45388-6_14 13
44. Hülsing, A., Meijers, M., Strub, P.Y.: Formal verification of Saber's public-key encryption scheme in EasyCrypt. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022, Part I. LNCS, vol. 13507, pp. 622–653. Springer, Heidelberg (Aug 2022). https://doi.org/10.1007/978-3-031-15802-5_22 2, 3, 8
45. Jiang, H., Zhang, Z., Chen, L., Wang, H., Ma, Z.: IND-CCA-secure key encapsulation mechanism in the quantum random oracle model, revisited. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part III. LNCS, vol. 10993, pp. 96–125. Springer, Heidelberg (Aug 2018). https://doi.org/10.1007/978-3-319-96878-0_4 3
46. Kreuzer, K.: Verification of correctness and security properties for CRYSTALS-KYBER. Cryptology ePrint Archive, Report 2023/087 (2023), https://eprint.iacr.org/2023/087 3, 8, 10
47. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 1–23. Springer, Heidelberg (May / Jun 2010). https://doi.org/10.1007/978-3-642-13190-5_1 6, 9

48. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. Slides of the talk given by Chris Peikert at Eurocrypt 2010 (2010), https://iacr.org/conferences/eurocrypt2010/talks/slides-ideal-lwe.pdf 6
49. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. Cryptology ePrint Archive, Report 2012/230 (2012), https://eprint.iacr.org/2012/230 9
50. Maram, V., Xagawa, K.: Post-quantum anonymity of Kyber. In: Boldyreva, A., Kolesnikov, V. (eds.) PKC 2023, Part I. LNCS, vol. 13940, pp. 3–35. Springer, Heidelberg (May 2023). https://doi.org/10.1007/978-3-031-31368-4_1 3, 6
51. Micciancio, D., Peikert, C.: Trapdoors for lattices: Simpler, tighter, faster, smaller. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 700–718. Springer, Heidelberg (Apr 2012). https://doi.org/10.1007/978-3-642-29011-4_41 3
52. Micciancio, D., Voulgaris, P.: Faster exponential time algorithms for the shortest vector problem. In: SODA '10: Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete algorithms. pp. 1468–1480. Society for Industrial and Applied Mathematics (2010) 3
53. National Institute of Standards and Technology: FIPS PUB 202 – SHA-3 standard: Permutation-based hash and extendable-output functions (2015), http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf 7
54. National Institute of Standards and Technology: FIPS PUB 203 (Initial Public Draft) – module-lattice-based key-encapsulation mechanism standard (2023), https://csrc.nist.gov/pubs/fips/203/ipd 1, 7, 8, 18
55. Ngo, K., Dubrova, E., Guo, Q., Johansson, T.: A side-channel attack on a masked IND-CCA secure saber KEM implementation. IACR TCHES **2021**(4), 676–707 (2021). https://doi.org/10.46586/tches.v2021.i4.676-707, https://tches.iacr.org/index.php/TCHES/article/view/9079 3
56. Nguyen, P.Q., Vidick, T.: Sieve algorithms for the shortest vector problemare practical. J. Math. Crypt. pp. 181–207 (2008), https://doi.org/10.1515/JMC.2008.009 3
57. Pessl, P., Prokop, L.: Fault attacks on CCA-secure lattice KEMs. IACR TCHES **2021**(2), 37–60 (2021). https://doi.org/10.46586/tches.v2021.i2.37-60, https://tches.iacr.org/index.php/TCHES/article/view/8787 3
58. Primas, R., Pessl, P., Mangard, S.: Single-trace side-channel attacks on masked lattice-based encryption. In: Fischer, W., Homma, N. (eds.) CHES 2017. LNCS, vol. 10529, pp. 513–533. Springer, Heidelberg (Sep 2017). https://doi.org/10.1007/978-3-319-66787-4_25 3
59. Saito, T., Xagawa, K., Yamakawa, T.: Tightly-secure key-encapsulation mechanism in the quantum random oracle model. Cryptology ePrint Archive, Report 2017/1005 (2017), https://eprint.iacr.org/2017/1005 6
60. Saito, T., Xagawa, K., Yamakawa, T.: Tightly-secure key-encapsulation mechanism in the quantum random oracle model. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part III. LNCS, vol. 10822, pp. 520–551. Springer, Heidelberg (Apr / May 2018). https://doi.org/10.1007/978-3-319-78372-7_17 3
61. Schwabe, P., Avanzi, R., Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Seiler, G., Stehlé, D.: CRYSTALS-KYBER. Tech. rep., National Institute of Standards and Technology (2017), available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-1-submissions 3
62. Schwabe, P., Avanzi, R., Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Seiler, G., Stehlé, D., Ding, J.: CRYSTALS-KYBER. Tech. rep., National Institute of Standards and Technology (2022), available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022 3
63. Targhi, E.E., Unruh, D.: Post-quantum security of the Fujisaki-Okamoto and OAEP transforms. In: Hirt, M., Smith, A.D. (eds.) TCC 2016-B, Part II. LNCS, vol. 9986, pp. 192–216. Springer, Heidelberg (Oct / Nov 2016). https://doi.org/10.1007/978-3-662-53644-5_8 3
64. Unruh, D.: Quantum relational hoare logic. Proc. ACM Program. Lang. **3**(POPL), 33:1–33:31 (2019). https://doi.org/10.1145/3290346, https://doi.org/10.1145/3290346 4
65. Unruh, D.: Post-quantum verification of Fujisaki-Okamoto. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020, Part I. LNCS, vol. 12491, pp. 321–352. Springer, Heidelberg (Dec 2020). https://doi.org/10.1007/978-3-030-64837-4_11 4
66. Ye, K.Q., Green, M., Sanguansin, N., Beringer, L., Petcher, A., Appel, A.W.: Verified correctness and security of mbedTLS HMAC-DRBG. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017. pp. 2007–2020. ACM Press (Oct / Nov 2017). https://doi.org/10.1145/3133956.3133974 3
67. Zhandry, M.: How to record quantum queries, and applications to quantum indifferentiability. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part II. LNCS, vol. 11693, pp. 239–268. Springer, Heidelberg (Aug 2019). https://doi.org/10.1007/978-3-030-26951-7_9 3

A Cryptographic Definitions

For a set S, $|S|$ denotes the cardinality of S. For a finite set S, we denote the sampling of a uniform random element x by $x \leftarrow\!\!\$\, S$. We denote deterministic computation of an algorithm A on input x by $y \leftarrow A(x)$. We denote algorithms with access to an oracle O by A^O. Unless stated otherwise, we assume all our algorithms to be probabilistic and denote the computation by $y \leftarrow\!\!\$\, A(x)$. When deterministically

Game COR:	Game COR-RO:
1: $(pk, sk) \hookleftarrow_\$ \mathsf{Gen}()$	1: $(pk, sk) \hookleftarrow_\$ \mathsf{Gen}()$
2: $m \hookleftarrow_\$ \mathcal{A}(pk, sk)$	2: $m \hookleftarrow_\$ \mathcal{A}^{\mathsf{G}}(pk, sk)$
3: $c \hookleftarrow_\$ \mathsf{Enc}(pk, m)$	3: $c \hookleftarrow_\$ \mathsf{Enc}^{\mathsf{G}}(pk, m)$
4: **return** $(m \neq \mathsf{Dec}(sk, c))$	4: **return** $(m \neq \mathsf{Dec}^{\mathsf{G}}(sk, c))$

Fig. 14. Correctness of a PKE in the standard model and ROM.

computing a randomized algorithms by fixing its random coins r, we write: $y \leftarrow A(x; r)$. For the most part of this section we follow [40,41].

Pseudorandomness. Let $\mathsf{f} : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ be a function. We define the PRF advantage of an adversary $\mathcal{A}$ as

$$\mathsf{Adv}^{\mathsf{PRF}}_{\mathsf{f}}(\mathcal{A}) = \left| \Pr[\mathcal{A}^{\mathsf{PRF}(K,\cdot)} \Rightarrow 1 : K \hookleftarrow_\$ \mathcal{K}] - \Pr[\mathcal{A}^{\mathsf{F}(\cdot)} \Rightarrow 1 : \mathsf{F} \hookleftarrow_\$ (\mathcal{X} \to \mathcal{Y})] \right| .$$

When this advantage is small for all efficient adversaries, we say f is a pseudorandom function. Let $\mathsf{g} : \mathcal{K} \to \mathcal{Y}$ be a function. We define the PRG advantage of an adversary $\mathcal{A}$ as

$$\mathsf{Adv}^{\mathsf{PRG}}_{\mathsf{g}}(\mathcal{A}) = |\Pr[\mathcal{A}(y) \Rightarrow 1 : K \hookleftarrow_\$ \mathcal{K}; y \leftarrow \mathsf{g}(K)] - \Pr[\mathcal{A}(y) \Rightarrow 1 : y \hookleftarrow_\$ \mathcal{Y}]| .$$

When this advantage is small for all efficient adversaries, we say g is a pseudorandom generator.

Public-Key Encryption. A public-key encryption scheme consists of three algorithms $\mathsf{PKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ and a finite message space $\mathcal{M}$. The key generation algorithm Gen outputs a key pair (pk, sk). The encryption algorithm Enc, on input pk and a message $m \in \mathcal{M}$, outputs a ciphertext $c \hookleftarrow_\$ \mathsf{Enc}(pk, m)$. If necessary to derandomize encryption, we writing $c \leftarrow \mathsf{Enc}(pk, m; r)$, where $r \hookleftarrow_\$ \mathcal{R}$ and $\mathcal{R}$ is the randomness space. The decryption algorithm Dec, on input sk and a ciphertext c, outputs either a message $m \leftarrow \mathsf{Dec}(sk, c) \in \mathcal{M}$ or a special symbol $\perp \notin \mathcal{M}$ to indicate that c is not a valid ciphertext.

Correctness. Correctness of a PKE is defined as in Figure 14. We give both a standard model definition (left) and a definition in the random oracle model (ROM, right). Note that the adversary gets the secret key as an input. In the standard model, δ-correctness means that for all (possibly unbounded) adversaries $\mathcal{A}$, $\Pr[\mathsf{COR}^{\mathcal{A}}_{\mathsf{PKE}} \Rightarrow 1] \leq \delta$. In the ROM, we extend the definition so that δ is parameterized by an upper bound on the number of queries placed by $\mathcal{A}$ to its random oracle.

Ciphertext Spreadness. We say that PKE is γ-spread if, for every key pair $(pk, sk) \hookleftarrow_\$ \mathsf{Gen}()$ and every ciphertext c, $\Pr_{r \hookleftarrow_\$ \mathcal{R}}[c = \mathsf{Enc}(pk, m; r)] \leq 2^{-\gamma}$.

Security. We now define several security notions for public-key encryption.

Definition 1 (OW-Atk). *Let* $\mathsf{PKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *be a public-key encryption scheme with message space* $\mathcal{M}$. *For* $\mathsf{Atk} \in \{\mathsf{CPA}, \mathsf{PCVA}\}$, *we give the* OW-Atk *security games in Figure 15. Note that, if* Atk=CPA, *we have* $\mathcal{O} = \emptyset$; *and if* Atk=PCVA, *we have* $\mathcal{O} = \{\textsc{Pco}, \textsc{Cvo}\}$. *We define the* OW-Atk *advantage function of an adversary* $\mathcal{A}$ *against* PKE *as* $\mathsf{Adv}^{\mathsf{OW\text{-}Atk}}_{\mathsf{PKE}}(\mathcal{A}) = \Pr[\mathsf{OW\text{-}Atk}^{\mathcal{A}}_{\mathsf{PKE}} \Rightarrow 1]$.

Note that the adversary wins the one-way game if it guesses the result of decrypting the challenge ciphertext, rather than the standard definition of finding the message that is originally encrypted. The two games have statistical distance δ, if PKE is δ-correct. We also define the more standard notion, allowing the adversary to output a list of guesses.

Definition 2. OWL-CPA Let $\mathsf{PKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a public-key encryption scheme with message space $\mathcal{M}$. We give the OWL-CPA security game in Figure 15. We define the OWL-CPA advantage function of an adversary $\mathcal{A}$ against PKE as $\mathsf{Adv}^{\mathsf{OWL\text{-}CPA}}_{\mathsf{PKE}}(\mathcal{A}) = \Pr[\mathsf{OW\text{-}CPA}^{\mathcal{A}}_{\mathsf{PKE}} \Rightarrow 1]$.

We conclude this section with the standard definition of IND-CPA security for a PKE.

Definition 3. IND-CPA Let $\mathsf{PKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a public-key encryption scheme with message space $\mathcal{M}$. We define the IND-CPA game as in Figure 16, and the IND-CPA advantage function of an adversary $\mathcal{A} = (\mathcal{A}_1, A_2)$ against PKE as $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{PKE}}(\mathcal{A}) = |\Pr[\mathsf{IND\text{-}CPA}^{\mathcal{A}}_{\mathsf{PKE}} \Rightarrow 1] - 1/2|$.

In the proof of the FO transform (see Section 5) we use this general lemma that is verified in EasyCrypt.

Game OW-Atk:	Oracle $\textsc{Pco}(m \in \mathcal{M}, c)$:	Game OWL-CPA:
1: $(pk, sk) \leftarrow_\$ \mathsf{Gen}()$	1: **return** $(m = \mathsf{Dec}(sk, c))$	1: $(pk, sk) \leftarrow_\$ \mathsf{Gen}()$
2: $m^* \leftarrow_\$ \mathcal{M}$		2: $m^* \leftarrow_\$ \mathcal{M}$
3: $c^* \leftarrow_\$ \mathsf{Enc}(pk, m^*)$	Oracle $\textsc{Cvo}(c \neq c^*)$:	3: $c^* \leftarrow_\$ \mathsf{Enc}(pk, m^*)$
4: $m' \leftarrow \mathcal{A}^{\mathcal{O}}(pk, c^*)$	1: $m \leftarrow \mathsf{Dec}(sk, c)$	4: $L \leftarrow \mathcal{A}(pk, c^*)$
5: **return** $\textsc{Pco}(m', c^*)$	2: **return** $m \neq \perp$	5: **return** $m^* \in L$

Fig. 15. One-wayness of a PKE.

Game IND-CPA:	Game IND-CCA:	Oracle $\mathsf{Decaps}(c)$:
1: $(pk, sk) \leftarrow_\$ \mathsf{Gen}()$	1: $(pk, sk) \leftarrow_\$ \mathsf{Gen}()$	1: If $c = c^*$ **return** $\perp$
2: $(m_0, m_1, st) \leftarrow_\$ \mathcal{A}_1(pk)$	2: $(k_0, c^*) \leftarrow_\$ \mathsf{Enc}(pk)$	2: $k \leftarrow \mathsf{Dec}(sk, c)$
3: $b \leftarrow_\$ \{0,1\}$	3: $k_1 \leftarrow_\$ \mathcal{K}$	3: **return** k
4: $c^* \leftarrow_\$ \mathsf{Enc}(pk, m_b)$	4: $b \leftarrow_\$ \{0,1\}$	
5: $b' \leftarrow_\$ \mathcal{A}_2(st, c^*)$	5: $b' \leftarrow_\$ \mathcal{A}^{\mathsf{Decaps}}(pk, c^*, k_b)$	
6: **return** $b' = b$	6: **return** $b' = b$	

Fig. 16. IND-CPA security of a PKE and IND-CCA security for a KEM.

Lemma 1. *For all* OWL-CPA *adversary* $\mathcal{B}$ *returning a list of size at most* Max*, there exists an* IND-CPA *adversary* $\mathcal{A}$ *such that*

$$\mathsf{Adv}^{\mathsf{OWL-CPA}}_{\mathsf{PKE}}(\mathcal{B}) \leq 2 \cdot \left(\frac{\textsc{Max}}{|\mathcal{M}|} + \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{PKE}}(\mathcal{A}) \right)$$

Proof (Sketch of formalization). We define an IND-CPA adversary $\mathcal{A}$ that uses the OWL-CPA adversary $\mathcal{B}$ in the obvious way: i. choose challenge IND-CPA messages (m_0, m_1) uniformly at random and run $\mathcal{B}$ on the challenge ciphertext; ii. if the list output by $\mathcal{B}$ contains m_b, but not $m_{\neg b}$, return b; iii. otherwise return a random bit b'.

We re-express the IND-CPA advantage using the standard form that conditions on the hidden bit, and get that:

$$2 \cdot \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{PKE}}(\mathcal{A}) = \left| \Pr[\mathsf{CPA}^{\mathsf{PKE}}_{\mathcal{A}}(1) \Rightarrow 1] - \Pr[\mathsf{CPA}^{\mathsf{PKE}}_{\mathcal{A}}(0) \Rightarrow 1] \right|$$

We then define a bad event in both experiments on the right-hand side of the above equation, which corresponds to $\mathcal{A}$ returning random bit b'. Clearly, this event occurs with the same probability in both experiments, and the output of both experiments are identically distributed when this event does occur. This means that

$$2 \cdot \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{PKE}}(\mathcal{A}) = |\Pr[\mathsf{CPA}^{\mathsf{PKE}}_{\mathcal{A}}(1) \Rightarrow 1 \wedge \neg\mathsf{bad}] - \Pr[\mathsf{CPA}^{\mathsf{PKE}}_{\mathcal{A}}(0) \Rightarrow 1 \wedge \neg\mathsf{bad}]|$$

The next step in the proof shows that

$$\Pr[\mathsf{CPA}^{\mathsf{PKE}}_{\mathcal{A}}(0) \Rightarrow 1 \wedge \neg\mathsf{bad}] \leq \textsc{Max}/|\mathcal{M}|\,.$$

which follows from the fact that m_1 is information theoretically hidden from $\mathcal{B}$. A similar argument allows us to show that

$$\left| \mathsf{Adv}^{\mathsf{OWL\text{-}CPA}}_{\mathsf{PKE}}(\mathcal{B}) - \Pr[\mathsf{CPA}^{\mathsf{PKE}}_{\mathcal{A}}(1) \Rightarrow 1 \wedge \neg\mathsf{bad}] \right| \leq \textsc{Max}/|\mathcal{M}|\,.$$

The lemma follows from the above inequalities. The arguments above are also good examples of game-hopping proofs that the EasyCrypt logic was designed to handle. □

Key Encapsulation Mechanisms. A key encapsulation mechanism $\mathsf{KEM} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ consists of three algorithms. The key generation algorithm Gen outputs a key pair (pk, sk), where pk also defines a finite key space $\mathcal{K}$. The encapsulation algorithm Enc, on input pk, outputs a tuple (k, c) where c is said to be an encapsulation of the key k which is contained in key space $\mathcal{K}$. The deterministic decapsulation algorithm Dec, on input sk and an encapsulation c, outputs either a key $k \leftarrow \mathsf{Dec}(sk, c) \in \mathcal{K}$ or a special symbol $\perp \notin \mathcal{K}$ to indicate that c is not a valid encapsulation. We call KEM δ-correct if $\Pr[\mathsf{Dec}(sk, c) \neq k \,|\, (pk, sk) \leftarrow_\$ Gen; (k, c) \leftarrow_\$ \mathsf{Enc}(pk)] \leq \delta$. Note that the above definition does not include a correctness adversary—and for this reason also makes sense in the random oracle model—because KEM ciphertexts do not depend on maliciously chosen messages.

Security. We now define indistinguishability under chosen ciphertext attacks (IND-CCA) for a KEM.

Definition 4. IND-CCA. We define the IND-CCA game also in Figure 16 and the IND-CCA advantage function of an adversary $\mathcal{A}$ against KEM as

$$\mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{KEM}}(\mathcal{A}) = \left|\Pr[\mathsf{IND\text{-}CCA}^{\mathcal{A}}_{\mathsf{KEM}} \Rightarrow 1] - 1/2\right| .$$

B Generic EasyCrypt Arguments

We now describe more in depth the generic arguments we make use of in our formal proofs, and some of the issues faced in applying them to the specific ML-KEM proof.

B.1 Plug-and-Pray

The first generic result we use heavily in this proof is a *plug-and-pray* argument. In this argument, a parametric reduction R_i—for example, a reduction that simulates the ith query to an oracle in a specific way—is replaced, with some security loss, with a reduction R that 1) makes a uniformly random guess as to the value of the parameter i (within a bounded-size set $\mathcal{S}$ of possible values); 2) runs the parametric reduction R_i; and 3) returns its result.

In such a setting, and for any event E, we will have, for every possible value j of the index

$$\Pr[\mathsf{R} : E] = |\mathcal{S}| \cdot \Pr[\mathsf{R} : E \wedge i = j]$$

The plug-and-pray library formalizes the argument generically, allowing arbitrary instantiations or the reduction, guess and event, and further allowing the value of the index to be derived from the state of the experiment, instead of being externally quantified.

B.2 Hybrids

The second result we heavily rely on in our ML-KEM proof is a generic hybrid argument, which captures generically union bound-style results in indistinguishability steps.

Consider two oracles $\mathcal{O}_l$ and $\mathcal{O}_r$ of the same type, and a distinguisher D trying to distinguish them in at most q queries. Our generic hybrid proves (entirely generically) that, for some distinguisher D_1 that makes at most 1 query to its oracle, we have

$$|\Pr[\mathsf{D}(\mathcal{O}_l)] - \Pr[\mathsf{D}(\mathcal{O}_r)]| \leq q \cdot |\Pr[\mathsf{D}_1(\mathcal{O}_l)] - \Pr[\mathsf{D}(\mathcal{O}_r)]|$$

The hybrid library formalizes the argument generically, allowing arbitrary instantiations of the oracle and distinguisher types, and also supporting the result where the oracle being replaced shares state with oracles that do not change from one side to the other.

B.3 PROM

The final generic result we use formally captures the cryptographic argument that "a value is random and independent of the adversary's view." This was initially produced as part of the formalization of indifferentiability from a random oracle for SHA3 [9], and is therefore framed as an indistinguishability result between two different implementations of a programmable random oracle interface. This result shows that no context (or unbounded adversary) can distinguish between the *eager* and *lazy* versions of the oracles presented in Figure 17, where the latter omits line 1 of the sample algorithm. These oracles allow for globally initializing or resetting (init), or locally observing associated values (get), programming the random oracle (set), or eagerly (resp. lazily) sampling values associated with given entries (sample). In essence, the library establishes generically that sampling unused values can be deferred until they (or some behavior that depends on them) first become visible to the adversary or context.

The result itself is quite general, and hides away some of the complex program logics involved in proving it, allowing less expert users to more easily formalize complex proofs. However, its application in our setting, where we need to identify a specific query which will be sampled eagerly, is somewhat complex: we cannot simply sample the entire random function eagerly over an infinite domain, and neither can we predict the values of the queries the adversary will make. Most of the applications we make of this result in the ML-KEM proof first involve an indirection mapping query values to their index (essentially, and up to repeat queries, the first query would have index 1, the second query would have index 2, and so on), and eagerly sampling a map from integers (between 1 and the maximum number of queries). This essentially allows to partially sample the outputs of the function and later assign them to particular inputs as the adversary makes queries.

Oracle $\mathsf{init}()$:
1: $m \leftarrow \emptyset$
2: **return** ()

Oracle $\mathsf{get}(x)$:
1: **If**$(x \notin m)$ $m[x] \hookleftarrow_{\$} R$
2: **return** $m[x]$

Oracle $\mathsf{set}(x, r)$:
1: $m[x] \leftarrow r$
2: **return** ()

Oracle $\mathsf{sample}(x)$:
1: **If**$(x \notin m)$ $m[x] \hookleftarrow_{\$} R$
2: **return** ()

Fig. 17. PROM oracles, in the lazy version the dashed box is removed