Proof-of-Possession for KEM Certificates using Verifiable Generation (full version)

Tim Güneysu  
Ruhr University Bochum, Horst Görtz  
Institute for IT-Security & DFKI GmbH, Cyber-Physical Systems  
Bochum, Germany  
tim.gueneysu@rub.de

Philip Hodges  
University of Waterloo  
Waterloo, Ontario, Canada  
pwhodges@uwaterloo.ca

Georg Land  
Ruhr University Bochum, Horst Görtz  
Institute for IT-Security & DFKI GmbH, Cyber-Physical Systems  
Bochum, Germany  
georg.land@rub.de

Mike Ounsworth  
Entrust  
Ottawa, Ontario, Canada  
mike.ounsworth@entrust.com

Douglas Stebila  
University of Waterloo  
Waterloo, Ontario, Canada  
dstebila@uwaterloo.ca

Greg Zaverucha  
Microsoft Research  
Redmond, Washington, USA  
gregz@microsoft.com

ABSTRACT
Certificate authorities in public key infrastructures typically require entities to prove possession of the secret key corresponding to the public key they want certified. While this is straightforward for digital signature schemes, the most efficient solution for public key encryption and key encapsulation mechanisms (KEMs) requires an interactive challenge-response protocol, requiring a departure from current issuance processes. In this work we investigate how to non-interactively prove possession of a KEM secret key, specifically for lattice-based KEMs, motivated by the recently proposed KEMTLS protocol which replaces signature-based authentication in TLS 1.3 with KEM-based authentication. Although there are various zero-knowledge (ZK) techniques that can be used to prove possession of a lattice key, they yield large proofs or are inefficient to generate. We propose a technique called verifiable generation, in which a proof of possession is generated at the same time as the key itself is generated. Our technique is inspired by the Picnic signature scheme and uses the multi-party-computation-in-the-head (MPCitH) paradigm; this similarity to a signature scheme allows us to bind attribute data to the proof of possession, as required by certificate issuance protocols. We show how to instantiate this approach for two lattice-based KEMs in Round 3 of the NIST post-quantum cryptography standardization project, Kyber and FrodoKEM, and achieve reasonable proof sizes and performance. Our proofs of possession are faster and an order of magnitude smaller than the previous best MPCitH technique for knowledge of a lattice key, and in size-optimized cases can be comparable to even state-of-the-art direct lattice-based ZK proofs for Kyber. Our approach relies on a new result showing the uniqueness of Kyber and FrodoKEM secret keys, even if the requirement that all secret key components are small is partially relaxed, which may be of independent interest for improving efficiency of zero-knowledge proofs for other lattice-based statements.

1 INTRODUCTION
Public key infrastructures (PKIs) operate by issuing certificates which bind an entity’s public key to identifying information for that entity [2, 26]. During the enrollment process, the issuing certificate authority, before agreeing to bind a public / private keypair to an entity, often requires the requester to demonstrate use of their private key as a proof of possession (PoP). The most common proof of possession mechanism is the PKCS#10 Certificate Signing Request (CSR) [60] which is used in many certificate enrollment protocols such as CMP [1], ACME [6], EST [62], and SCEP [42]. (See Appendix A for a discussion on the role of PoPs in PKIs.)

CSRs have become ubiquitous in part because they are fully non-interactive and therefore easily portable. They do not require the certificate authority (CA) to have real-time communication with the entity requesting the certificate, so a CSR can be transported and validated out of band. As an example, web PKI CAs tend to offer simple certificate enrollment workflows requiring a certificate requester to paste a CSR into the CA’s web page [6] first. This allows key generation and CSR creation to take place on a production server, after which the certificate request is initiated from the workstation outside the production network. Even fully automated certificate issuance protocols such as ACME [6] do not require the CSR to contain any protocol state information which would force it to be freshly-generated as part of the certificate issuance exchange.

One major drawback of CSRs is that they require a digital signature [60], and therefore can only be used for PoP of digital signature-type keys. Some enrollment protocols such as CMP [1] define PoP mechanisms for key agreement and encryption keys, however these are interactive mechanisms requiring either a half or full round trip of communication between the CA and the certificate requester.

This problem has been side-stepped in many applications, such as the web PKI, which almost exclusively use certificates containing digital signature public keys. However, this may not necessarily be the case in the future, especially in the context of the transition to post-quantum cryptography. In 2020, Schwabe, Stebila, and Wiggers [67] proposed the KEMTLS protocol as a replacement for the TLS 1.3 handshake [64] which uses key encapsulation mechanisms (KEMs) for authentication, rather than signatures. Since post-quantum digital signatures tend to have larger communication sizes than post-quantum KEMs, KEMTLS can reduce the bandwidth needed to establish a post-quantum channel. But KEMTLS requires certificates containing KEM keys, so one cannot avoid the question of how to prove possession of a KEM key, preferably in a non-interactive setting to maintain the flexibility and existing workflows that use certificate signing requests.
Lattice-based KEMs are among the leading candidates in the NIST post-quantum cryptography standardization project, due to a relatively good mix of speed and communication size. Three of the five Round 3 finalists are lattice-based KEMs (Kyber, NTRU, and Saber), as well as one Round 3 alternate candidate (FrodoKEM). In this paper we focus on Kyber [17, 66] (based on the module learning with errors (MLWE) problem [49]) and FrodoKEM [18, 58] (based on the learning with errors (LWE) problem [63]); at a high level, both have a similar design, which we briefly outline here. Roughly speaking, in FrodoKEM the public key is a pair of matrices \((A, B)\) where \(B = AS + E \text{ mod } q\) for secret matrices \(S\) and \(E\) consisting of small entries. At the 128-bit security level, \(q = 2^{15}\), "small" means in \([0, \pm 12]\), and \(S\) and \(E\) contain 10,240 entries together.

1.1 Existing Zero-Knowledge Techniques

The obvious approach for building a non-interactive proof of possession for a lattice-based KEM key would be to use zero-knowledge tools. There are two main techniques: directly building a zero-knowledge proof for the relevant mathematical operations, or using a generic technique such as SNARKs [14] or multi-party computation in-the-head (MPCitH) [44].

Direct lattice-based ZK constructions. There is a long line of research for ZK proofs of knowledge of a lattice secret key, starting with lattice-based identification schemes [53, 57]. Kawachi et al. [48] adapted techniques from Stern’s identification scheme [68] based on syndrome decoding to the lattice setting, which has influenced several subsequent constructions [50, 52]. Recent direct constructions focusing on proving knowledge of an LWE or short integer solution (SIS) sample include [12, 16, 36, 54–56], most of which focus on binary \(\{0, 1\}\) or ternary \(\{-1, 0, 1\}\) secrets. Some include ZK proofs of verifiable decryption for lattice-based KEMs [36, 54, 56]. The constructionyielding the smallest proofs is a recent paper by Lyubashevsky et al. [54]: its proof takes place in a larger space (e.g., a Kyber512 key with modulus \(q = 3329\) is embedded in a proof with modulus \(= 2^{36}\)) and depends on the hardness of both MSIS and MLWE in the larger space, so parameters for the proof must be chosen appropriately to both ensure the proof can be constructed and that the MSIS and MLWE problems are hard in the larger space. Generic ZK constructions. There are several constructions based on zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARKs), del Pina et al. [30] showed how to use Bulletproofs [20] to prove knowledge of lattice secret key with proofs as small as 1.25 kB, but at the cost of relying on a non-post-quantum security assumption (elliptic curve discrete logarithms) and a 10 second generation time. Boschini et al. [19] showed how to use the Aurora SNARK system [10] (for rank-1 constraint systems) for module LWE/SIS relations, and Beullens [12] gave estimates for using both Aurora and Ligero [3] (based on probabilistically checkable proofs) for LWE secrets; however Aurora’s runtimes are estimated to be dozens of seconds [19]. Baum and Nof [8] showed how to use MPCitH [44] with preprocessing [47] for ZK proof of knowledge of binary LWE secrets.

In terms of proof sizes (excluding non-post-quantum constructions [30]), among existing constructions, direct ZK constructions generally yield the smallest proof sizes, followed by SNARK systems, followed by MPCitH. Generally, the most relevant input features affecting performance are the modulus \(q\), the total number \(\sigma\) of entries among the secrets \(S\) and \(E\) (whether they be matrix entries in a plain LWE instance, or coefficients in a ring-LWE instance, or both in a module-LWE instance), and the bounds \(\beta\) on the absolute value of the size of the secret entries. However, conducting an apples-to-apples comparison among the literature is challenging: it is rare that two papers in this field give example parameterizations that are directly comparable, they rarely provide direct non-asymptotic formulas or scripts for calculating parameter sizes, and very few report runtime results of implementations. Additionally, many constructions are principally designed for binary or ternary secrets; for some there are techniques that can extend to larger secrets bounds \(\beta\) by transforming the problem instance, but determining parameters for this extension is non-trivial. Table 1 presents results from the literature of various schemes in various parameter regimes.

1.2 Our Approach and Contributions

Our overall contribution is non-interactive protocols for proof of possession of lattice-based KEM keys (with instantiations for Kyber and FrodoKEM) using a technique called verifiable generation, in which the PoP is generated at the same time as the key itself.

Definitions for proof of possession. In Section 3, we begin by giving the syntax for a key generation and proof of possession (KGPOP) scheme, and consider a special class of combined KGPOP schemes in which the proof of possession is generated at the same time as the key. We define two security properties for such schemes: unforgeability and zero-knowledge of proofs of possession. Roughly speaking, unforgeability says that it is hard to generate a valid proof of possession for well-formed public key without the secret key (defined similarly to signature unforgeability), and zero knowledge says that proofs leak no information about the secret key (defined via simulatability in the random oracle model). However, since the key pair exists not just for the purposes of proof of possession, but has some intended application purpose (for example, authentication in KEMTLS), we have to consider composability of the PoP with the intended application purpose. In the context of an IND-CCA-secure KEM, in principle it is possible that (i) the PoP for a KEM key could undermine the confidentiality of the KEM shared secret, or conversely (ii) that use of the KEM secret key in decapsulation could undermine the unforgeability of proofs of possession. We handle this in both directions as follows. For (i), zero-knowledge naturally models the privacy notion we need. For (ii), we give the adversary access to an auxiliary secret key usage algorithm that represents use of the secret key in its intended application; proof of unforgeability must be shown even in the presence of this oracle. We give a quick example showing that certificate signing requests (CSRs) [60] are PoPs within our framework assuming the unforgeability of the signature scheme and appropriate domain separation between how CSRs are formatted and how application messages are formatted.

Non-interactive verifiable generation for lattice-based KEMs. Our main contribution, in Section 4, is a combined key generation and proof of possession scheme for lattice-based KEMs, based on the MPCitH paradigm, inspired in part by the Picnic signature scheme [22, 69]. The design of our scheme is as a 5-round interactive protocol to which the Fiat–Shamir transform [38] is applied to make a
to prove that \( N \) are the inputs to a simple key from which we will construct the public key. These small values remaining unrevealed values will be used as the entries of the secret that only a fraction of secret key components are small. Then the FrodoKEM secret keys: we can show that it is sufficient to prove this step uses a new result showing the uniqueness of high number of the remaining values are small. Our analysis of the secret key, and commit to all of them, then reveal some fraction follows. First, we generate more small values than are needed for a relatively simple cut-and-choose approach. The basic idea is as by combining generation of the key and PoP, since we can use that the values in \( (S, E) \) are small – and this is where we benefit from combining generation of the key and PoP, since we can use a relatively simple cut-and-choose approach. The basic idea is as follows. First, we generate more small values than are needed for the secret key, and commit to all of them, then reveal some fraction of them to demonstrate that, with high probability, a sufficiently high number of the remaining values are small. Our analysis of this step uses a new result showing the uniqueness of Kyber and FrodoKEM secret keys: we can show that it is sufficient to prove that only a fraction of secret key components are small. Then the remaining unrevealed values will be used as the entries of the secret key from which we will construct the public key. These small values are the inputs to a simple \( N \)-party MPC protocol, which is repeated \( r \) times to prove that \( B \) is is computed correctly.

Strictly speaking our proof does not guarantee that a lattice secret key is perfectly well-formed; it remains possible for a dishonest party to generate a key with some non-small entries. But this approach suffices for the purposes of proof of possession: if Alice has generated a key honestly, then there is no way for Mallory to generate a valid proof of possession for Alice’s public key without effectively recovering Alice’s secret key.

Outline of the security analysis. Using techniques developed for MPCih-based signatures, we prove that the scheme is both zero-knowledge (Lemma 7) and straight-line extractable (Lemma 7, also known in the literature as online extractable) in the ROM. Our extractor uses the random oracle queries from a successful prover to recover a secret key that has mostly small entries and satisfies the equation \( B = AS + E \). Together with Lemma 4 this immediately implies unforgeability, since in Lemma 4 we show that the extracted secret key is unique, even though the PoP may allow a bounded number of non-small entries. Using the explicit bound given by Lemma 4, we choose concrete parameters so that the probability of non-unique keys is negligible, and consequently the secret key extracted must with high probability be the original one, yielding unforgeability. (Lemma 4 allows relaxation without sacrificing uniqueness, and may help efficiency of zero-knowledge proofs for other lattice-based statements; for example the performance of [8]).
may be improved by checking that only a certain fraction of values are small.)

For our KEMTLS scenario we must handle unforgeability in the presence of the auxiliary secret key usage oracle corresponding to KEM decapsulation. We observe that IND-CCA-secure KEMs constructed via the Fujisaki–Okamoto transform [39, 43] (the class that includes FrodoKEM and Kyber) have a property we call decapsulation simulatability, meaning that in the ROM decapsulation can be simulated without the secret key; details appear in Appendix B. This allows us to simulate the auxiliary secret key usage algorithm in the ROM during the proof unforgeability experiment. Conversely, since our proof is zero-knowledge, security of the KEM is maintained when it is added to the public key (Appendix C).

Overall, security only relies on the random oracle model plus a hiding commitment scheme and a pseudorandom generator, assumptions that are already required by FrodoKEM and Kyber and which follow from a random oracle. Compared with direct ZK constructions based on lattice assumptions, such as the state-of-the-art [54], one major benefit of our approach is that we do not rely on any significant security assumptions beyond the original scheme (e.g., [54] depends on both MLWE and MSIS). Moreover, direct lattice-based assumptions such as [54] have to select security parameters for the lattice problem on which the proof is based, which in general will be different from (and larger than) the security parameters for the lattice-based key the proof is about. Given the subtleties and expertise required to pick lattice parameters, it is an advantage of our scheme that the proof system’s parameters are based on MPCitH, which is quite a bit simpler to pick parameters for.

Generality of our construction. Our exposition in Section 4 is phrased in terms of FrodoKEM, but most of it applies directly to Kyber as well, and we give a version of the uniqueness lemma (Lemma 4) for Kyber in Section 5. Our approach is flexible and we expect variants of our design would apply to other LWE, Ring-LWE or MLWE-based KEMs such as [61].

Implementation. In Section 6 we report on a software implementation of our scheme including runtime and proof size, for both Kyber and FrodoKEM at the 128-, 192-, and 256-bit security levels. We are able to tune the trade-off between proof size and runtime by varying the number of simulated MPC parties N and the number of MPC repetitions τ; the range of trade-offs can be seen in Figure 8.1 Overall, our MPCitH-based approach for verifiable generation yields proofs that are at least an order of magnitude smaller than the previous best MPCitH approach for proof of knowledge of lattice secrets by Baum and Nof [8]. Compared to direct lattice constructions and post-quantum zkSNARKs, our MPCitH approach remains generally competitive, and our most sized-optimized parameterizations can in some cases nearly match the state-of-the-art direct lattice constructions. We summarize some comparative results below; see Table 1 for detailed comparisons across several parameter regimes. As noted above, direct comparisons with proof of knowledge schemes in the literature can be challenging due to difference in parameterizations and the non-triviality of extrapolating existing results to other parameter regimes, especially for non-ternary secrets. Nonetheless we are able to offer some direct comparisons.

Comparison to MPCitH. For the parameter set given in [8] targeted towards somewhat homomorphic encryption applications (q ≈ 261, σ = 4096, binary secrets), our system yields proof sizes for the same parameters at sizes as small as 272 kB and of 342 kB for a reasonable (N, τ) trade-off, which is 7–15× smaller than Baum and Nof’s solution with a 4.0 MB proof and 2.4 second generation time. To apply Baum and Nof’s protocol to FrodoKEM parameters, one has change from handling binary secrets to small (absolute value ≤ β = 12). Applying the edalits construction [35] for integer comparisons in MPC, we extrapolate that checking Frodo640’s secret values have absolute value at most β = 12 would lead to a proof of at least 8.42 MB (at 90 bits of security) with estimated multi-second runtimes. Our verifiable generation construction yields Frodo640 proofs of possession at sizes as small as 203 kB; as shown in Figure 8, other points in the size–speed tradeoff space include 402 kB in 0.63 seconds and 799 kB in 0.07 seconds, which is 11–41× smaller than the Baum and Nof proof.

Comparison to direct lattice constructions. The most extensive results available for comparison across multiple papers are for a parameter suite with modulus q ≈ 232, σ = 2048 samples, and ternary secrets. zkSNARKs achieve proof sizes of 72 kB (Aurora [10]) or 157 kB (Liger [3]). Recent ZK proofs based on MLWE and MSIS achieve proof sizes of 47 kB [36], 33 k [56], and 14 k [54]. Our MPCitH-based verifiable generation construction yields PoPs for this parameter regime at sizes as small as 79 kB and at 156 kB for a reasonable (N, τ) trade-off, which is comparable with zkSNARKs but not as good as MLWE- and MSIS-based ZK proofs.

For the Kyber512 parameters (q = 3329, σ = 1024 samples, secrets ≤ β = 2) our scheme compares more favorably with direct lattice constructions since it scales better with increased secret bound β. Only two papers in the literature report proof sizes for Kyber512 parameters [54, 56], and even then they only report sizes for proofs of verifiable decryption, rather than proof of knowledge of the secret key. Verifiable decryption proofs are a bit larger than proof of secret key knowledge; although size differences are not given in [54, 56], in [36] verifiable decryption proofs were about 1.3× bigger. [56]’s proof size for Kyber512 verifiable decryption is 43.6 kB, and for [54] it is 19.0 kB. Our MPCitH-based verifiable generation construction yields Kyber512 proofs of possession as small as 17.8 kB (runtime 3.77 seconds); as shown in Figure 8, other points in the size–speed tradeoff space include 25.7 kB in 109 milliseconds, 33.4 kB in 28 milliseconds, and 52.9 kB in 6 milliseconds.

2 PRELIMINARIES

In this section we review notation and background notions needed in the rest of the paper.

All adversaries are assumed to be probabilistic polynomial time algorithms (in our security parameter λ), that are stateful, i.e. if A appears twice in an security experiment, state may be shared between instances.
FrodoKEM.KeyGen()  
1: seed_A ← {0, 1}^{ln{seed_A}}  
2: A ← Frodo.Gen(seed_A) ∈ Z_q^{n \times n}  
3: seed_E ← {0, 1}^{ln{seed_E}}  
4: (r(0), r(1), ..., r((2nπ)^{-1})) ← SHAKE(8 | 5F | seed_E, 2nπ | len_x)  
5: S^T ← Frodo.SampleMatrix(r(0), r(1), ..., r((nπ)^{-1})), π, n, χ)  
6: E ← Frodo.SampleMatrix((r(nπ), r((nπ)^{-1})), n, π, χ)  
7: B ← AS + E  
8: return (pk = (seed_A, B), sk = S^T)  
Frodo.SampleMatrix(r(0), ..., r(n_1, n_2-1), where r(i) ∈ {0, 1}^{len_x} 
1: for i = 0, ..., n_1 - 1 do  
2: for j = 0, ..., n_2 - 1 do  
3: E_{ij} ← Frodo.Sample(r(i,n_2), χ)  
4: return E  

Figure 1: FrodoKEM key generation algorithm.

A key encapsulation mechanism (KEM) is a triple of algorithms KEM = (KeyGen, Encaps, Decaps) used to establish randomly chosen keys. The probabilistic algorithm KeyGen(1^n) outputs a key pair (pk, sk). The probabilistic algorithm Encaps(pk) outputs a ciphertext and key (c, K), and Decaps(sk, c) outputs K or ⊥ if c is not a valid encapsulation. A KEM with implicit rejection does not output ⊥ when c is invalid, but instead outputs a random K.

FrodoKEM [18, 58] is a key encapsulation mechanism based on the (plain) learning-with-errors problem [63], built on a design by Lindner and Peikert [51] and using a variant of the Fujisaki–Okamoto (FO) transform [39, 43, 45] to achieve indistinguishability under chosen ciphertext attacks (IND-CCA). The FO transform is described in Appendix B.

For our purposes, it suffices to examine only the key generation algorithm, the relevant parts of which are shown in Figure 1. The key generation algorithm of FrodoKEM samples a public seed seed_A used to pseudorandomly generate a matrix A ∈ Z_q^{n \times n}. The algorithm also samples a secret seed seed_E used to pseudorandomly generate two secret n × π matrices S and E containing entries sampled from a distribution χ. The public key is the seed seed_A and the matrix B = AS + E; the secret key is S. The distribution χ is small relative to the modulus q, and we will sometimes refer to small values as a concise way to distinguish between integers sampled from χ and elements of Z_q. As a concrete example for FrodoKEM at the 128-bit security level, q = 2^{15} and β = 12; so χ yields samples in [0, ±2].

Kyber [17, 66] is a KEM based on the module LWE problem [49]. Very roughly speaking, Kyber can be thought of as similar to FrodoKEM, but where entries in the matrices A, B, S, E are polynomials in a ring R_q = Z_q[X]/(X^d + 1); see the Kyber specification for a more precise formulation [66]. As a concrete example of parameters for Kyber at the 128-bit security level, q ≈ 3329, d = 256, n = 2, and χ yields samples in [0, ±2].

3 DEFINING ZERO-KNOWLEDGE PROOFS OF POSSESSION

In this section, we define the syntax and security properties for non-interactive proof of possession, as well as give an example of how Certificate Signing Requests fit our framework.

Definition 1 (Key generation and proof of possession scheme). A key generation and proof of possession (KGPoP) scheme consists of:

- PoP.KG() :→ (pk, sk): A probabilistic key generation algorithm that outputs a public, secret key pair.
- PoP.PG(pk, sk, attrs) :→ π: A probabilistic proof generation algorithm that takes as input a public key, secret key,
and attributes $\text{attrs} \in \{0, 1\}^*$, and outputs a proof string $\pi$. (In our application, $\text{attrs}$ could be the body of a certificate signing request, for example.)

- $\text{PoP.Vf}(pk, \text{attrs}, \pi) \rightarrow \{0, 1\}$: A deterministic verification algorithm that takes as input a public key $pk$, attributes $\text{attrs}$, and proof $\pi$, and outputs 1 if the proof is valid, and 0 otherwise.

A more general formalization could be made with a setup procedure for a common reference string (CRS); since our constructions will not rely on a CRS, we opt to omit the generalization.

A key generation and proof of possession scheme should satisfy the obvious correctness property: for all attrs,

$$\Pr[\text{PoP.Vf}(pk, \text{attrs}, \pi) \Rightarrow 1 | (pk, sk) \leftarrow \text{PoP.KG}() \land \pi \leftarrow \text{PoP.PG}(pk, sk, \text{attrs})] = 1.$$

In our approach for proof-of-possession via verifiable generation, the key pair and the proof are generated at the same time. We call such a scheme a combined key generation and proof of possession scheme, and in this case the two algorithms PoP.KG and PoP.PG from Definition 1 (which we sometimes refer to as a “separable” scheme) are replaced with a single algorithm:

- $\text{cPoP.KPG}(\text{attrs}) \rightarrow (pk, sk, \pi)$: A probabilistic key and proof generation algorithm that takes as input attributes $\text{attrs} \in \{0, 1\}^*$, and outputs a public key $pk$, secret key $sk$, and proof string $\pi$.

One could build the combined $\text{cPoP.KPG}$ in the obvious way from separable KG and PG: generate the key pair $(pk, sk) \leftarrow \text{PoP.KG}()$, then the proof $\pi \leftarrow \text{PoP.PG}(pk, sk, \text{id})$, then return $(pk, sk, \pi)$. The corresponding correctness property applies.

Admittedly, combining proof generation with key generation limits the number of proofs generated, which may be unsatisfactory for some applications. For certificate issuance, a single proof suffices, since only a single certificate signing request is needed. (In practice, even if multiple certificates are requested over time for the same key, the same certificate signing request is typically used.) Furthermore, other certificate lifecycle events (such as revocation) can be handled by binding (into the proof attributes) an additional account key that authenticates other lifecycle events; see discussion in Section 7.

The main security property we want of a key generation and proof of possession scheme is unforgeable proofs of possession: it should be hard for an adversary to construct a valid proof $\pi$ for a public key without the corresponding secret key. However, we also need assurance that the PoP can be used safely in conjunction with the actual cryptographic primitive the corresponding keys are meant for. In particular, we have to ensure (a) that the PoP functionality does not undermine the security of the actual cryptographic primitive, and (b) use of the secret key in the actual cryptographic primitive does not undermine security of the PoP. In our case, the former will follow when the PoP is non-interactive zero knowledge; the latter will be shown by incorporating (constrained) use of the secret key into the unforgeability definition for PoPs. (See [24] for additional considerations on composition of PKIs with protocols.)

We state the formal definition of security for combined proofs of possession now, then present the details and rationale in Sections 3.1 and 3.2, and give an example of how these definitions apply to certificate signing requests in Section 3.4.

### Definition 2 (Security of Combined PoPs)

Let $\text{cPoP} = (\text{KPG}, \text{VF})$ be a combined key generation and proof of possession scheme. We say that $\text{cPoP}$ is a secure $\text{cPoP}$ scheme if it is:

- **Zero-knowledge**: no efficient adversary exists for the security experiment $\text{Exp}_{\text{cPoP}}^\text{ZF}$ (as shown in Figure 3) as defined in Definition 3.

- **Unforgeable**: no efficient adversary exists for the security experiment $\text{Exp}_{\text{cPoP}}^\text{UF}$ (as shown in Figure 2), for the required auxiliary secret key usage algorithm $\text{Aux}$.

- **Correct**: key pairs output by $\text{cPoP.KPG}$ are distributed identically to the $\text{KeyGen}$ function of the primitive that defines the key pair.

To highlight the difference between proof of possession and proof of knowledge or well-formedness: a proof of possession does not guarantee that a public key was generated honestly by someone who knows the secret key, but it does guarantee that an honestly generated public key cannot be claimed by someone else who does not know the secret key. In the context of PKI, a proof of possession does not protect against a malicious party getting a certificate for a possibly malformed public key, but does protect against a malicious party getting a certificate for someone else’s public key. This modelling also allows us to cover certificate signing requests, which are accepted in practice for demonstrating proof of possession but are not in general a proof of knowledge or well-formedness.

#### 3.1 Unforgeability

The unforgeability security experiment for a KGPOP scheme, shown in Figure 2, is defined with respect to an auxiliary secret key usage algorithm $\text{Aux}$, which models any usage of the secret key in a subsequent application:

- $\text{Aux}(sk, x) \rightarrow y$: A probabilistic or deterministic auxiliary secret key usage algorithm that takes as input a secret key $sk$ and input $x$, and produces an output $y$.

For example, in a PoP for a KEM key, $\text{Aux}$ would correspond to decapsulation; in a PoP for a signing key, $\text{Aux}$ would correspond to signature generation.

The unforgeability experiment $\text{Exp}_{\text{cPoP}}^\text{UF}$ in Figure 2 is analogous to weak existential unforgeability of a signature scheme under chosen message attacks. A version analogous to strong existential unforgeability could be had by checking that $(\text{attrs}', \pi')$ were not
the input/output of a query to PG. (While our verifiable generation construction for FrodoKEM and Kyber does satisfy the stronger notion, we include the weak unforgeability notion to accommodate proof-of-possession schemes such as certificate signing requests built from weakly unforgeable signature schemes such as ECDSA.)

For combined KGPOP schemes, we need a slightly different definition of proof unforgeability: since the proof generation is not separate from the key generation, the adversary cannot repeatedly obtain PoPs. The resulting simplified experiment is shown in Figure 2. It is analogous to weak existential unforgeability of a signature scheme under a key-only attack; as above, a strong existential unforgeability version can be had.

When a combined KGPOP scheme is constructed in the obvious way from a separable KGPOP scheme, security in the sense of the former immediately implies security in the sense of the latter.

### 3.2 Non-Interactive Zero Knowledge

Having defined security such that use of the key in an application does not affect security the PoP scheme, we now consider the other direction, and look at security of the application against attackers that have proofs created by the PoP scheme.

There certainly exist degenerate schemes where the proof of possession could undermine the intended application. Consider for example EdDSA [11] with the following proof of possession. Let \((pk, sk)\) be an ECC key pair. Let the PoP be \((\sigma, m)\) where \(\sigma\) is an ECDSA signature on \(m = \text{SHA512}(sk)\)\{attrs\}. Proof verification checks the ECDSA signature on the provided \(m\) as usual, without checking whether the first part of the message, \(\text{SHA512}(sk)\), has any connection to the public key; this is still a good proof of possession under the assumption that ECDSA is unforgeable. The intended use of the key is EDNSA signatures, for TLS authentication or any other purpose. The PoP is clearly unforgeable under weak unforgeability of ECDSA, but use of the key in EdDSA is completely broken since EdDSA uses \(\text{SHA512}(sk)\) as a PRF key to derive per-signature nonces. In isolation the two primitives are secure, but the composition is insecure: the PoP leaks information, namely \(\text{SHA512}(sk)\), that compromises the application.

The natural way to protect against such attacks is to require PoP schemes to be zero-knowledge: they should reveal nothing more than the fact that the party creating the proof knows \(sk\) associated to a given \(pk\). Our definition is nearly the same as honest verifier zero-knowledge (see e.g. [15, §19.1.1]) except that (i) we relax the definition to statistical zero-knowledge [41, Def. 4.3.4] and (ii) we have the \(\text{attrs}\) string that is allowed to be chosen arbitrarily, and is bound to the proof. Our definition also assumes the PoP is non-interactive, as motivated by our use case, so we note that the simulator will exist in an idealized model where Sim has some additional capability, such as the ability to choose system parameters, or simulate hash functions in the random oracle model (ROM). Otherwise, if Sim was efficiently implementable by any party the ZK property would exclude the unforgeability property.

**Definition 3.** Let \(\text{PoP} = (KG, PG, VF)\) be a key generation and proof of possession scheme. We say that PoP is zero-knowledge, if there exists an efficient probabilistic algorithm Sim, such that, for all possible outputs \((pk, sk)\) of KG, the output distribution of Sim\((pk, \text{attrs})\) is statistically close to the output distribution of

<table>
<thead>
<tr>
<th>Exp_{\text{cPoP,Sim}}^ZK (A)</th>
<th>Exp_{\text{PoP,Sim}}^ZK (A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: ((pk, sk) \leftarrow A())</td>
<td>1: (\text{attrs} \leftarrow A())</td>
</tr>
<tr>
<td>2: (b \leftarrow {0, 1})</td>
<td>2: ((pk, sk, x_0) \leftarrow \text{KPG}(\text{attrs}))</td>
</tr>
<tr>
<td>3: (R_{\text{PG}(pk, sk)}())</td>
<td>3: (x_1 \leftarrow \text{Sim}(pk, \text{attrs}))</td>
</tr>
<tr>
<td>4: (b = 0) then (b' \leftarrow A_{\text{Sim}(pk, \text{attrs})}())</td>
<td>4: (b \leftarrow {0, 1})</td>
</tr>
<tr>
<td>5: (b = 1) then (b' \leftarrow A_{\text{Sim}(pk, \text{attrs})}())</td>
<td>5: (b' \leftarrow A(x_0))</td>
</tr>
<tr>
<td>6: return (b = b')</td>
<td>6: return (b = b')</td>
</tr>
</tbody>
</table>

**Figure 3:** Security experiment for zero-knowledge of proofs of possession for a key generation scheme PoP (left) and a combined key generation and proof of possession scheme cPoP (right) with respect to simulator Sim.

PG\((pk, sk, \text{attrs})\). More formally, PoP is zero-knowledge if no efficient adversary exists for \(\text{Exp}_{\text{PoP,Sim}}^ZK\) (as shown in Figure 3 (left)) with success probability different from 1/2. For combined KGPOP schemes, the corresponding experiment is shown in Figure 3 (right).

### 3.3 Handling Aux Queries for KEMs

Our unforgeability definition in Section 3.1 takes care to model use of the secret key in an application, so that this use does not allow an attacker to create a proof of possession for a user’s key after they use it. This is very broad and in general difficult to handle (as illustrated in the CSR example of Section 3.4). Fortunately for our KEMTLS scenario, we can show that unforgeability can be preserved when the KEM key is in use.

Full details of our approach are in Appendix B; we give the intuition here. For a KEM, we define the security notion of decapsulation simulatability. The decapsulation operation of a KEM is said to be simulatable if there exists a simulator Sim which takes a public key pk and ciphertext as input and has outputs that are indistinguishable from KEM.Decaps. From this, our security proof for unforgeability (Theorem 8) can simulate use of the KEM key (in the ROM) and answer Aux queries without using sk. We prove that KEMs constructed with the variant of the Fujisaki–Okamoto (FO) transform used in FrodoKEM and Kyber have this property. Our property is based on a result of [43] that implicitly defines the required simulator when proving that FO transform provides CCA security.

### 3.4 Example: Certificate Signing Requests

To build familiarity with these notions, we observe how a familiar PoP system for signature schemes, certificate signing requests (CSRs), can be modeled in this framework. CSRs are an example of a (separable) KGPOP scheme. Let \(\Sigma = (\mathcal{K}, \Sigma, \text{Sign}, \Sigma, \text{Vf})\) be a signature scheme. Define the KGPOP scheme \(\text{CSR}[\Sigma] = (\mathcal{K}, \text{PG}, \text{VF})\) with helper function CSR\text{fmt} as follows:

- **KG()**: Return \(\Sigma, \mathcal{K}()\).
- **PG\((pk, sk, \text{attrs})\)**: Return \(\Sigma, \text{Sign}(sk, \text{CSR\text{fmt}(pk, \text{attrs})})\).
- **VF\((pk, \text{attrs}, x)\)**: Return \(\Sigma, \text{Vf}(pk, \text{CSR\text{fmt}(pk, \text{attrs}, x))\).
- **CSR\text{fmt}(pk, \text{attrs})** → \(\{0, 1\}^*\): A deterministic CSR formatting function that generates the body of a certificate signing request from a public key \(pk\) and attributes \(\text{attrs}\), for example according to a standards document such as RFC 2986 [60]. We assume CSR\text{fmt} is collision-free.
It is straightforward to see that if \( \Sigma \) has existential unforgeability under chosen message attack, then it is hard to forge proofs of possession in \( \text{CSR}[\Sigma] \) – after all, PoPs are just signatures.

However, if we want to model that the secret key generated by \( \text{CSR}[\Sigma] \) will then be used as a signing key in some subsequent application, we have to make use of an auxiliary secret key usage algorithm. If the subsequent application (and hence the auxiliary secret key usage algorithm) allows an adversary to sign arbitrary messages, then the adversary could just ask it to sign a well-formatted CSR body. To prevent this, we must enforce some kind of domain separation. We model this by introducing a function \( \text{IsCSR} : \{0, 1\}^* \to \{0, 1\} \) that checks if a candidate message is a valid formatted CSR body, and refuses to sign if it is:

\[
\text{Aux}(sk, x): \text{If } \text{IsCSR}(x) = 1, \text{ then return } \perp, \\
\text{else return } \Sigma. \text{Sign}(sk, x).
\]

Assuming that \( \text{IsCSR}(x) = 1 \) for all \( x \leftarrow \text{CSRfmt}(pk, \text{attrs}) \), it is then straightforward to prove, via reduction, that \( \text{CSR}[\Sigma] \) has unforgeable proofs of possession (with respect to the given \( \text{Aux} \)) in the sense of Figure 2, assuming \( \Sigma \) is (weakly) existentially unforgeable under chosen message attack.

The domain separation between CSRs and application messages can be easy to satisfy in practice, though it still must be verified for each application. For example, consider digital signatures in the Web PKI: a server gets an X.509 certificate for its signing key by creating a CSR as per [60], and subsequently uses the signing key for authentication in the TLS 1.3 protocol [64]. In this case, a CSR body is a certain ASN.1 data structure, with various fields and formatting as specified in [60]; most notably, a CSR body is always an ASN.1 SEQUENCE object, which in BER or DER encoding means that the first byte will be hexadecimal \( 0x10 \) or hexadecimal \( 0x30 \). In TLS 1.3, signatures are generated over data structures with a different format as specified in [64, §4.4.3], specifically where the first byte will be hexadecimal \( 0x20 \). Assuming a server only uses its signing key in TLS, this immediately yields domain separation that can be modeled via CSRfmt and IsCSR.

4 NON-INTERACTIVE VERIFIABLE GENERATION FOR LATTICE-BASED KEMS

In this section we give our construction of a non-interactive, combined generation and proof of possession for lattice-based KEMs. We will focus on FrodoKEM as it is simpler to describe, but the approach and analysis also works for Kyber and other similar schemes. We start this section with a technical overview of our construction, then give full details of the construction for FrodoKEM, discuss parameter selection and provide a security analysis.

4.1 Technical Overview

Consider proving knowledge of a pair of secret matrices \( S, E \) whose entries are small and satisfy \( B = AS + E \bmod q \), where \( A \) and \( B \) are public matrices. Let \( P \) be the prover and \( V \) be the verifier. Let \( \sigma \) be the total number of entries in the secret matrices \( S \) and \( E \), integers sampled from a “small” interval according to a distribution \( \chi \). We describe the protocol as an interactive 5-round protocol between \( P \) and \( V \), as an MPCitH proof with \( N \) parties.

1. **Commit to small values.** \( P \) generates \( M > \sigma \) random small values, where \( M \) is a parameter. For each of the \( M \) values, \( P \) generates an \( N \)-party additive secret sharing (mod \( q \)). \( P \) commits to the shares and sends the commitment to \( V \).

2. **Challenge a subset of the small values to audit.** \( V \) chooses \( M' = \sigma \) of the small values to audit and sends this as a challenge to \( P \). The idea is that \( P \) has committed to more than \( \sigma \) values, and \( V \) will check a subset of them, to make sure they belong to the correct distribution.

3. **Generate keypair, simulate MPC, commit to views, and open the audited small values.** \( P \) responds with the opening of the commitment for each of the audited values. \( P \) uses the unaudited values to construct the secret key \((S, E)\). Note that the \( N \) parties have shares of \((S, E)\), and \( P \) has committed to the shares. \( P \) randomly generates a matrix \( A \) and computes the public key \( B = AS + E \). Given \( A \), the parties can compute shares of \( B \) using \( A \) and the sharings of \( S \) and \( E \) (no communication is required among the parties as this is a linear operation). Finally, \( P \) commits to the shares of \( B \) (the views of the parties), and sends \((A, B)\) to \( V \).

4. **Challenge the MPC simulation.** \( V \) selects \( N'-1 \) parties to audit. Here the verifier’s goal is to check that the MPC was executed correctly, namely that \( B \) was computed honestly.

5. **Open the MPC views and output the key pair and proof.** \( P \) reveals the state of the \( N'-1 \) audited parties and outputs the keypair computed in step 3.

6. **Verify final response.** \( V \) re-computes the \( N'-1 \) views and ensures they match the views committed to by \( P \). \( V \) also checks that the \( M' = \sigma \) opened values meet the range criteria given by \( \chi \). If both checks pass, \( V \) outputs accept; otherwise \( V \) rejects.

We will show that for an appropriate choice of \( M \) (relative to \( \sigma \) and the lattice parameters), \( V \) will be assured that the \( \sigma \) values in \((S, E)\) are from \( \chi \), except with probability \( 2^{-\kappa} \). However, the second challenge has only soundness \( 1/N \) and necessitates repeating the protocol \( r \) times in parallel. These parallel repetitions are not independent as they all use the same \( M \) values sampled from \( \chi \) (with independent sharings in each repetition).

To make this protocol into a non-interactive proof, \( V \)’s challenges are derived with a hash function (in the random oracle model (ROM) [9]), using the well-known Fiat–Shamir transform [38]. Going forward we will focus on the non-interactive case, since this is required by our proof-of-possession scenario.

When viewed as a proof of knowledge, combined KGPOP does not quite fit the established structure. The problem instance, or statement \( x \) (corresponding to matrices \( A \) and \( B \)) and witness \( w \) (corresponding to secrets \( S \) and \( E \)) are negotiated in the first part of the protocol, rather than fixed in advance. However, because of the protocol’s commit-and-open structure, in the ROM we can still extract \( P \)’s input including \( w \) (and we prove this in Lemma 7). We can also simulate transcripts in the ROM for any \( x \), to show that the protocol is statistical honest-verifier zero-knowledge (Lemma 5) in the ROM.
4.2 FrodoKEM KGPOP construction

We now provide a full description of the non-interactive verifiable generation protocol for FrodoKEM. The combined key generation and proof of possession protocol (cPoP.KPG) is given in Figure 4, and the verification operation (cPoP.VF) is given in Figure 5.

Notation. Recall that $q$ is the modulus used in FrodoKEM computations, $r$ is the number of parallel repetitions, and $N$ is the number of parties used in the MPC protocol. Let $\sigma$ denote the total number of small integers required for FrodoKEM private values. A bundle $B_\ell$ is a collection of $r$ sharings of a value $\nu$ between $N$ parties in the ring $\mathbb{Z}_q$. $\chi$ is the distribution used for secrets and errors.

The input to KPG is $\text{atts} \in \{0, 1\}^r$, the string encoding the attributes to be bound to the cPoP. We use $x^{(i)}$ to denote party $P_i$'s share of $x$, and $\{X\}$ to denote the set $\{1, \ldots, X\}$. In the MPC protocols simulated by the prover, each party can sample their share of a value $x$ from their random tape. This is sufficient when $x$ must be a uniform random value. To share a given value, $P$ provides a "delta value" computed as:

$$\Delta x = x - \sum_{i=1}^{N} x^{(i)}.$$ 

$P$ makes $\Delta x$ public, and it is added to $P_i$'s share to correct the sharing: $x^{(i)} = x^{(i)} + \Delta x$.

As we will be working with an $N - 1$ private MPC protocol with $r$ parallel repetitions, in repetition $e$, there is one party that is unopened, we denote their index by $i^*$, or simply $i^*$ when the repetition index is clear from context.

Helper functions. We use the following helper functions to simplify our presentation.

Arrange creates $(S, E)$ from a list of $\sigma$ values in a canonical way. Given a subset $C$ of $\{1, \ldots, M\}$ of size $\sigma$, and values $\{v_1, \ldots, v_M\}$, let $(S, E) = \text{Arrange}(\{v_1, \ldots, v_M\}, C)$ be the two matrices given by populating the entries of $S$ and $E$ from the elements $v_i$ for $i \in C$, starting in the top-left of $S$ and proceeding row-by-row from left to right, and then continuing with $E$.

Expand is a function that takes a hash digest, and expands it into a challenge of the correct form. The function ExpandTape() takes a seed and expands it into a random tape. These functions are implemented with an extendable output hash function, such as SHAKE [59]. The function Sample() reads random values from a random tape, converting the random bits to the required type, and keeping a pointer to the next bits to be read.

Optimizations. We use an optimization from [47] that is now standard in the MPCithH literature: $P$ derives per-party seeds with a binary tree construction, so that $N - 1$ seeds can be communicated to $V$ with $\lceil \log_2 N \rceil$ seeds. We use another optimization at a couple points of the protocol. If $x$ is a public value, and the prover has committed to shares of it, e.g., $h = H(x^{(1)}, \ldots, x^{(N)})$, the verifier may recompute $h$ from $x$ and $N - 1$ of the shares by recomputing the missing share $x^{(i)}$ as follows $x^{(i)} = x - \sum_{j \neq i} x^{(j)}$, since by definition $x = \sum_{i=1}^{N} x^{(i)}$. This saves the prover from having to communicate the unopened party's share. For example, in the bundles we audit, we commit to $r$ sharings of $v_k$: by revealing $v_k$, the verifier can recompute the missing shares and check the

\begin{figure}[h]
\centering
\includegraphics[width=\linewidth]{figure4.png}
\caption{Combined proof and key generation scheme, prover operations. Commit is a commitment scheme. See paragraph "Helper functions" for description of functions Arrange and Sample, and pseudorandom generators Expand and ExpandTape.}
\end{figure}

commitments. Since the $v_k$ are small values and the missing share is from $\mathbb{Z}_q$, this reduces proof size.

\textbf{Proof Size.} The size of the proof $\pi$ output by Figure 4 is

$$6k + 2r (2k + k \lceil \log_q (N) \rceil + \ell q M) + \ell q (M - \sigma)$$

(1)

bits, where the term $6k$ accounts for the salt and two challenges (each $2k$ bits long), $2k$ is the size of the per-party commitment, $k \lceil \log_q (N) \rceil$ is the number of bits required to communicate the seeds of the unopened parties in each repetition, $\ell q$ is $\lceil \log_q (q) \rceil$, and $\ell q$ is the number of bits required to represent a "small" value (sampled from the distribution $\chi$). In Section 4.3.1 we explain how
Suppose Bob is honest and has generated \( pk_B = (B, \text{seed}_B) \), but Alice is malicious and wants to create a PoP for \( pk_B \). Therefore Alice aims to demonstrate knowledge of a pair \((S', E')\) satisfying the relation \( B = AS' + E' \). If there are no size restrictions on the entries of \( S' \) or \( E' \), then it is easy to find \( S' \) and \( E' \) that are consistent with \( B \). If Alice’s malicious proof verifies, the verifier is convinced that most of the values in \((S', E')\) are small. But Lemma 4 shows that, since the honest secret key Bob generated has small (S, E), then it is unique (unconditionally, with overwhelming probability). Therefore, if Alice’s proof verifies, we must have \((S', E') = (S, E)\) and this implies that \((S', E')\) are in fact all small values.

The following lemma bounds the probability that, for a well-formed FrodoKEM key, there exists a second solution \((S', E')\) where every entry is small, or at most \( y \) entries total are not small.

**Lemma 4 (Uniqueness of small FrodoKEM solutions).** Let \( q = 2^D \), and let \( D, n, \pi \) be positive integers. Let \( A \leftarrow \mathbb{Z}_{2^{nq}}^{\pi} \) be selected uniformly at random, and let \( S, E \leftarrow \mathbb{Z}_{2^{nq}}^{\pi} \) have each entry be generated independently according to the FrodoKEM error distribution \( \chi_q \). Let \( \beta \) be the maximum value in the support of \( \chi_q \). Let \( B = AS + E \). Then the probability that there are values \((S', E') \in \mathbb{Z}_{2^{nq}}^{\pi} \) such that each entry of \((S', E')\) has absolute value at most \( \beta \), that \( B = AS' + E' \), and that \((S', E') \neq (S, E)\), is at most

\[
\Pr(1 - 2^{-n})(2\beta + 1)^n \cdot \sum_{t=0}^{D-1} \left( \frac{2\beta + 1}{2^t} \right)^n .
\]

Furthermore, if the size requirement is relaxed for at most \( y \) entries of \( S' \) and \( E' \), then this probability is at most

\[
\Pr(1 - 2^{-n})(2\beta + 1)^n \cdot q^{y} \cdot \sum_{t=0}^{D-1} \left( \frac{2\beta + 1}{2^t} \right)^n .
\]

**Proof.** Consider the first column of \( S \), the first row of \( A \), the first entry of the first row of \( B \), and the first entry of the first column of \( E \), denoted \( s, a, b, \) and \( e \) respectively. Let \( s' \in \mathbb{Z}_q^{\pi} \) with small coefficients (absolute value less than \( \beta \)). Let \( e' = b - (a, s') \). Then \( s' \) is a valid first column for a solution matrix \( S \) of the FrodoKEM relation if \( |e'| \) is less than or equal to \( \beta \). We can write \( e' = (a, s) + e \), where \( \overline{s} = s - s' \). For every \( \overline{s} \neq 0 \) (and hence \( s' \neq s \)), we will calculate the probability that \( |e'| \) is less than or equal to \( \beta \), and then bound the probability that there exists a second solution \( s' \) for the first column of the FrodoKEM relation using a union bound.

Note that if there is at least one odd entry \( s_j \) in \( s \), then \( (a, \overline{s}) \) is uniformly distributed on \( \mathbb{Z}_q \), since \( gcd(s_j, q) = 1 \), and so \( a_j \cdot s_j \) is uniformly distributed. Further, this means that \( (a, \overline{s}) + e \) is also uniformly distributed. Thus \( \Pr[|e'| \leq \beta] = \frac{2\beta + 1}{q} \).

If there are no odd entries in \( \overline{s} \), then this probability can be higher. This occurs for a fraction of possible values of size \( 2^{-n} \). Then, if there is at least one entry that is not a multiple of 4, \((a, \overline{s}) \) is distributed uniformly over even values in \( \mathbb{Z}_q \). This gives us that \( \Pr[|e'| \leq \beta] \leq 2 \cdot \frac{2\beta + 1}{2^t} \cdot \frac{1}{q} \). Continuing in this way for \( t \in \{0, ..., D - 1\} \), we get that for a proportion of possible values for \( \overline{s} \) of size \( 2^{-2n}(1 - 2^{-n}) \), we have \( \Pr[|e'| \leq \beta] \leq 2^t \cdot \frac{2\beta + 1}{2^t} \cdot \frac{1}{q} \).

The probability that the above relation holds for all \( n \) rows of the matrix \( A \) (using the corresponding other entries of the first column of \( E \)) is then at most \( \left( 2^t \cdot \frac{2\beta + 1}{2^t} \cdot \frac{1}{q} \right)^n \). Performing a union bound
over all \((2\beta + 1)^n\) possible values of \(\hat{s}\) gives the following upper bound on the probability that there exists a second solution \(s'\) for the first column:

\[
\sum_{i=0}^{D-1} 2^{-\kappa} (1 - 2^{-n})(2\beta + 1)^n \left(2^\kappa \cdot \frac{2\beta + 1}{2^\beta} \right) \left| q \right|^n = (1 - 2^{-n})(2\beta + 1)^n \cdot \sum_{i=0}^{D-1} \left(2^\kappa \cdot \frac{2\beta + 1}{2^\beta} \right) \left| q \right|^n.
\]

Repeating this argument for all \(\hat{\pi}\) columns of \(S\) (and corresponding columns of \(E\)) gives an upper bound of \(\hat{\pi}\) times the probability above, which is Equation (2).

Now consider what happens when we relax the size requirement for some values of \(s'\) and \(\epsilon'\). Specifically, if \(y_1\) entries of the first column of \(E\) do not need to meet the size constraint and \(y_2\) entries of \(s'\) do not need to meet the size constraint, then the above union bound instead becomes

\[
(1 - 2^{-n})(2\beta + 1)^{n-y_1} q^{y_1} \cdot \sum_{i=0}^{D-1} 2^{-\kappa} \left(2^\kappa \cdot \frac{2\beta + 1}{2^\beta} \right) \left| q \right|^n = (1 - 2^{-n})(2\beta + 1)^{n-y_1} q^{y_1} \cdot \sum_{i=0}^{D-1} 2^{-\kappa} \left(2^\kappa \cdot \frac{2\beta + 1}{2^\beta} \right) \left| q \right|^n.
\]

Let \(y = y_1 + y_2\). The choices of \(y_1\) and \(y_2\) maximizing this probability with a given \(y\) will have \(y_1 = 0\) and \(y_2 = y\). Again considering this result over all \(\hat{\pi}\) columns (taking a coarse upper bound by multiplying by \(\hat{\pi}\), even though not all of the columns can have \(y\) relaxed values simultaneously) of \(S\) gives us an upper bound of

\[
\hat{\pi}(1 - 2^{-n})(2\beta + 1)^{n-y} q^y \cdot \sum_{i=0}^{D-1} 2^{-\kappa} \left(2^\kappa \cdot \frac{2\beta + 1}{2^\beta} \right) \left| q \right|^n,
\]

which is Equation (3).

### 4.3.1 Parameter Selection

We explain how we use Lemma 4 with an example, consider parameters for Frodo640: \(n = 640\), \(\beta = 8\), \(\beta = 12\), and \(q = 2^{128}\). We choose \(y\) to be the largest integer so that, as per the relaxed bound in Lemma 4, the probability of a non-unique solution is less than \(2^{-128}\). The result is \(y = 340\). Hence we can relax up to 340 size constraints in an alternate solution, and the probability of the FrodoKEM solution \(S\) being non-unique is still less than \(2^{-128}\). We note that this probability holds unconditionally.

Consequently, we choose \(M\) in the FrodoKEM cPoP protocol so that the probability that 341 or more invalid bundles all fall in the \(\sigma\) unaudited values is less than \(2^{-128}\). If \(\gamma\) is the number of invalid bundles, then the probability of all \(\gamma\) invalid bundles being in the \(\sigma\) unaudited bundles is

\[
\left(\frac{M - \gamma}{M - \sigma}\right) \cdot \left(\frac{M}{M - \sigma}\right).
\]

For Frodo640, where \(\sigma = 10240\), the smallest value of \(M\) for which Equation (4) is smaller than \(2^{-128}\), with \(\gamma = 341\), is \(M = 10240 + 2993\). We give other values of \(M\) and \(y\) for FrodoKEM and Kyber in Table 2 (in Appendix D).

The second group of parameters are chosen to prevent cheating in the MPCihH executions. Since the \(\tau\) executions (each with \(N\) parties) are independent, the probability of successful cheating is \(1/\tau\). For \(\kappa\)-bit security, we choose \(N\) and \(\tau\) so this is at most \(2^{-\kappa}\).

### 4.4 Security Analysis

We prove two lemmas regarding the protocol of Section 4.2, that roughly correspond to the two properties required for a secure cPoP scheme.

The first is honest verifier zero-knowledge, which corresponds to the ZK property of cPoPs (Definition 3). For this lemma we require that ExpandTape is a secure pseudorandom generator (PRG) as defined in, e.g., [15, Definition 3.1], implemented with a random oracle. As our implementation uses SHAKE [59] to implement ExpandTape this is a standard assumption. For the seed tree construction used to derive the per-party seeds, we (informally) must assume that after revealing \(N - 1\) of \(N\) seeds, the remaining seed is hidden to a computationally bounded adversary. This is shown to hold when the tree is constructed with a random oracle in [21, Section 6.3], and again our implementation uses SHAKE.

**Lemma 5.** The non-active cPoP protocol of Section 4.2 is zero-knowledge in the random oracle model.

**Proof.** (Sketch) We first recall the real distribution against which Sim’s output must be indistinguishable (from \(\text{Exp}_{\text{cPoP}}^{2\kappa}\)) in Figure 3. Let \(S_{pk}\) be the set of all transcripts associated with a fixed public key \(pk\):

\[
S_{pk} = \{(\pi, \text{attrs}) : \text{Vf}(pk, \pi, \text{attrs}) = 1\}.
\]

Then we will require that \((\text{Sim}(pk, \text{attrs}))_{\text{attrs}} \equiv c S_{pk}\), where \(\text{attrs}\) are drawn from the same distribution in both cases.

The simulator. To describe the algorithm Sim we start by assuming it has both (\(pk, sk\)) as input and runs the real protocol from Figure 4. This initial version is then modified in a series of hybrid arguments, each change maintaining the indistinguishability of transcripts, until we arrive at the final version of Sim that no longer uses \(sk\). The algorithm \(\mathcal{A}\) is any algorithm that attempts to distinguish real and simulated transcripts in the ROM.

**Game 1.** Sim\(_1\) chooses a random first challenge \(C\) before starting to generate the transcript, then chooses \(M - \sigma\) values \(v_k\) honestly from \(\chi\). Sim\(_1\) indexes the bundles so that the values in \(sk\) are not queried by \(\mathcal{C}\). Sim\(_1\) then follows the protocol, but programs \(H_1\) to output \(C\) for \(m_i\). The distribution of Sim\(_1\)’s output and the real distribution are identical: all values in the simulated transcript are computed honestly, for all \(C\).

**Game 2.** Sim\(_2\) replaces \(\text{seed}_e^{(i)}\) with a uniformly random value. This is indistinguishable assuming the tree construction is hiding. Intuitively, since the seed tree is a hash tree, the only way to distinguish games 1 and 2 is to query for \(\text{seed}_e^{(i)}\) which happens with negligible probability when the number of hash queries is bounded.

**Game 3.** Sim\(_3\) replaces \(\text{tape}_e^{(i)}\) with a uniformly random value. This is indistinguishable assuming ExpandTape is a secure PRG. Since we model ExpandTape as a random oracle that maps (salt, \(e\), \(\text{seed}_e^{(i)}\)) to \(\text{tape}_e^{(i)}\), again the intuition is that games 2 and 3 are indistinguishable provided \(\text{seed}_e^{(i)}\) is not queried.

**Game 4.** Sim\(_4\) replaces the commitments \(\text{com}_e^{(i)}\), with random values, i.e., without making a query to Commit. Sim\(_4\) aborts if \(\mathcal{A}\) queries \(x\) such that \(\text{Commit}(x)\) was output. Since Commit is a random oracle, finding a preimage of a random value happens
with negligible probability, therefore Sim aborts with negligible probability and the output of Sim₅ is indistinguishable from Sim₃.

**Game 5.** Sim₅ no longer uses sk. Sim₅ chooses a random second challenge \((\delta^5)^{i} \epsilon [r]\) in advance, and programs \(H_2\) to output it for \(m_2\). Sim₅ chooses random \(\Delta_{b,c}\) for \(k \neq C\) (the unaudited values). Sim₅ computes shares of parties \(P_i \neq P_r\) and solves for \(B_{c}^{(i)} = B - \sum_{i \neq r} B_{c}^{(i)}\). This amounts to cheating in the unopened party’s broadcast to make the MPC compute the correct public key. Here the distributions are identical: since the unopened party’s shares were already uniformly random (because the tape \(T^{(i)}\) is uniform as of Game 3), \(\Delta_{b,c}\) was uniformly random. The shares of the opened parties are computed honestly, and \(B_{c}^{(i)}\) is the same in both real and simulated transcripts.

**Conclusion.** Since Sim₅ no longer uses sk and its output is indistinguishable from the set of real transcripts for any pk, the lemma is proven. □

**Straight-line extractability.** Next we prove that proofs created by Section 4.2 are straight-line extractable in the ROM, meaning that we can extract the secret key from a successful prover. In the context of a cPoP, the adversary may create many proofs that verify, however, if they create one for an existing public key, our proof of cPoP security (Theorem 8) will use this property to recover the secret key, reducing cPoP-UF to key recovery (which is a search LWE problem for FrodoKEM).

Our proof is loosely based on the UF-KOA security proof for the Rainier signature scheme [31], and is in parts identical. (As Rainier is based on the Banquet scheme [7], its security analysis is in turn similar to Banquet’s.) Rainier is also based on a 5-round MPCiH proof, and the KOA security analysis implicitly defines an extractor in the ROM: when the adversary creates a forgery (a valid proof for a challenge public key), the proof extracts the secret key from the query history, in order to reduce KOA security to key recovery. This proof strategy is common for the analysis of MPCiH-based proofs, first described in [47, §3.1]. The basic idea is that because we’ve chosen our parameters correctly, if \(\pi\) verifies, then at least one of the \(r\) parallel repetitions was executed honestly. Let index \(e\) be one such repetition. From the query history, the extractor algorithm Ext can find the preimage of \(c_{e}^{(i)}\), which contains \(\text{seed}_{c}^{e, (i)}\). Then Ext has the shares of all \(N\) parties and can recover \((S, E)\).

We now define straight-line extractability for cPoPs. Basically this says that for any proof and key pair \((pk, \pi, \text{attrs})\) output by \(\mathcal{A}\) such that \(c_{\text{PoP}, \text{Vf}}(pk, \pi, \text{attrs}) = 1\), there exists an extractor algorithm \(\text{Ext}_{\text{commit},H_1,H_2}(pk, \pi, \text{attrs})\) that recovers sk such that \(R(pk, sk) = 1\), where \(R\) relates public and secret keys. Ext simulates the random oracles used by \(\mathcal{A}\) and fails with bounded probability.

**Definition 6.** Let \(R\) be the relation that relates secret keys and public keys in a KEM. A cPoP is straight-line extractable in the random oracle model with knowledge error \(\varepsilon_{\text{sle}}\) if there exists an efficient extractor Ext such that for all adversaries \(\mathcal{A}\), we have the following:

\[
\varepsilon_{\text{sle}} \leq \Pr[R(pk, sk') \neq 1 \mid (pk, \pi, \text{attrs}) \leftarrow \mathcal{A}^H(1^*) \quad \text{cPoP} \cdot \text{Vf}(pk, \pi, \text{attrs}, \pi) \Rightarrow 1 \quad sk' \leftarrow \text{Ext}_1(pk, \pi, \text{attrs})].
\]

\[H_2(q_c = (\text{salt}, e, i, \text{seed}_{c}^{(i)})) \quad \text{H}_2(q_2 = (\text{salt}, h_1, m_2))
\]

\[
\begin{align*}
1: & \quad x \leftarrow \{0, 1\}^{2k} \\
2: & \quad \text{if } x \in \text{Digests then abort} \\
3: & \quad x \rightarrow \text{Digests} \\
4: & \quad (q_c, x) \rightarrow Q_i \\
5: & \quad \text{return } x \\
6: & \quad \text{return } x
\end{align*}
\]

**Figure 6: Oracles \(H_c\) and \(H_2\) for extraction algorithm Ext in proof of Lemma 7.**

In this experiment \(H\) is a random oracle used by \(\mathcal{A}\) that is simulated by Ext.

**Lemma 7.** Let Commit, \(H_1\), and \(H_2\) be modeled as random oracles, Expand be modeled as a random function, and let cPoP be the cPoP scheme given in Figures 4 and 5 with parameters \((N, r)\). Let \(\mathcal{A}^\text{Commit}$,\(H_1, H_2\) be an adversary that makes a total of \(Q\) random oracle queries. Then cPoP is straight-line extractable with

\[
\varepsilon_{\text{sle}} \leq (rN + 1)Q^2 / 2^{2k} + 1/2^k
\]

when parameters are chosen as described in Section 4.3.1.

**Proof.** We give an algorithm Ext which simulates the random oracles used by adversary \(\mathcal{A}\) to compute a secret key corresponding to any pk associated to a valid proof \(\pi\) output by \(\mathcal{A}\). We note that for every query made by \(\mathcal{A}\), Ext simply samples a random value and outputs it – none of the outputs depend on other queries or outputs, i.e., there is no programming done by Ext, it only records information from the queries. Since Ext simulates \(H_c\) (in this proof we use \(H_c\) as shorthand for Commit), \(H_1\), and \(H_2\) perfectly, we can focus on analyzing the probability that extraction fails.

Algorithm Ext simulates the random oracles \(H_c, H_1,\) and \(H_2\) and maintains query lists \(Q_c, Q_1,\) and \(Q_2\). Ext also uses tables \(T_{sk}\) and \(T_{key}\) to store shares of the parties, and extracted secret keys recovered from \(\mathcal{A}\)’s RO queries. Ext also keep track of the outputs of all three random oracles with the set Digests. Our analysis will ignore calls to Expand, since they are only used to expand outputs from \(H_1\) and \(H_2\), and when Expand is a random function this is equivalent to increasing the output lengths of \(H_1\) and \(H_2\).

Ext answers random oracle queries from \(\mathcal{A}\) in the following way. When defining Ext’s RO implementations (given in Figures 6 and 7), without loss of generality we only consider queries that are correctly formed, and ignore duplicate queries.

- H_c: When \(\mathcal{A}\) queries the commitment random oracle, Ext stores the query to keep track of the seed corresponding to the commitment. See Figure 6.
- H_1: When \(\mathcal{A}\) queries with the commitments and \(\Delta\)-values for the secret key, Ext checks whether the commitments were output by its simulation of \(H_c\) for a specific party/repetition. If so, Ext reconstructs the shares for that party by following the steps used by the honest prover given in Figure 4. If Ext reconstructs the shares of all parties for any repetition, then it has recovered the secret key that \(\mathcal{A}\) used in that execution. See Figure 7.
Lemma 7. Figure 7: Oracle $H_1$ for extraction algorithm Ext in proof of

- $H_2$: No extraction takes place during this random oracle simulation, but we do some bookkeeping to quantify the probability of collisions. See Figure 6.

After $A$ completes and outputs $(pk, \pi, attrs)$, Ext checks the $T_{\text{key}}$ table for any entry $sk_k$ such that $R(sk_k, pk) = 1$. If a match is found, Ext outputs $sk_k$ as the secret key corresponding to $pk$. Otherwise, Ext outputs ⊥.

Extraction Failure Probability. We now analyze the probability that $A$ wins the straight-line extraction experiment. Assuming $A$ outputs $(pk, \pi, attrs)$ such $cPoP.Vf(pk, \pi, attrs) = 1$, this can happen in one of two ways: Ext can abort, or fail to output $sk$. By the law of total probability:

$$\Pr[A \text{ wins}] = \Pr[A \text{ wins } \land \text{ Ext aborts}] + \Pr[A \text{ wins } \land \text{ Ext outputs } \perp]$$

$$\leq \Pr[\text{Ext aborts}] + \Pr[A \text{ wins } | \text{ Ext outputs } \perp]$$

(5)

Noting also that Ext is defined to output ⊥ if it extracts $sk' \neq \perp$ such that $R(pk', sk) \neq 1$.

The probability that Ext aborts is the probability of a collision among the hash functions. Let $Q_{\text{com}}$, $Q_1$, and $Q_2$ denote the number of queries made by $A$ to $H_k$, $H_1$, and $H_2$, respectively. Every time Ext samples a random $x$ to output, it might abort. More precisely,

$$\Pr[\text{Ext aborts}] \leq (Q_{\text{com}} + Q_1 + Q_2) \cdot \frac{\max \{\text{Digests}\}}{2^{2k}}$$

$$= (Q_{\text{com}} + Q_1 + Q_2) \cdot \frac{Q_{\text{com}} + (rN + 1)Q_1 + 2Q_2}{2^{2k}}$$

$$\leq \frac{(rN + 1)(Q_{\text{com}} + Q_1 + Q_2)^2}{2^{2k}}.$$  

(6)

Now for the main quantity: the probability that $A$ wins the experiment given that Ext fails, i.e., Ext outputs ⊥ since no matching $sk$ was found in $T_{\text{key}}$. It can be checked that Ext always succeeds when $A$ generates $x$ honestly, therefore for Ext to fail $A$ must cheat. We divide the two ways $A$ can cheat, either by cheating in the first part of the protocol, the "setup phase" (i.e., Phase 1-2 of Figure 4) during which the prover commits to $M$ bundles, and the verifier audits $M - \sigma$ of them, or the second phase (i.e., Phase 3-4 of Figure 4) when the prover simulates the MPC and the verifier audits $N - 1$ of $N$ parties.

Cheating in the setup phase. For any $q_1 \in Q_1$, and its corresponding answer $h_1 = C \subseteq [M]$, $|C| = \sigma$, let $G_1(q_1, h_1)$ be the set of indices $i \in [\sigma]$ of "good executions" where both $T_{\text{key}}[q_1, i] = (S, E)$ is non-empty and and it holds that all audited values are small, i.e.,

$$\forall k \in C, v_k \text{ meets the range criteria given by } \chi$$

(7)

If there does not exist such a $q_1$, let $G_1(q_1, h_1) = \emptyset$.

For any such good execution $e \in G_1(q_1, h_1)$, since Ext outputs ⊥ but $A$ wins, there must be more than $\gamma$ values $v_k, k \notin C$, i.e., not audited by the verifier, that do not satisfy the range condition given by $\chi$. As analyzed in Equation (4) of Section 4.3, this happens with probability no more than

$$p_1 = \left(\frac{M - \gamma}{M - \sigma}\right) \left(\frac{M}{M - \sigma}\right)$$

(8)

given that $h_1$ is distributed uniformly at random (which holds assuming $H_1$ and Expand are random functions). As the response $h_1$ is uniform, and there is one $C$ per query (i.e., one set of $M$ bundles per proof), all $e \in \mathcal{E}$ are in $G_1(q_1, h_1)$ with probability $p_1$.

Cheating in the second phase. Each second round query $q_2 = (salt, h_1, m_2)$ that $A$ makes to $H_2$ can only be used in a valid proof if there exists a corresponding query $(q_1, h_1) \in Q_1$. Then for each repetition $e \in [\sigma] \setminus G_1(q_1, h_1)$, either verification failed, in which case $A$ couldn’t have won, or the verification passed, despite Equation (7) not being satisfied. This implies that exactly one of the parties must have cheated. At least one cheater is required for verification to pass, but as $N - 1$ parties are opened, verification would fail if more than one party cheated.

Since $h_2 \in [N]^T$ is distributed uniformly at random, the probability that this happens for all repetitions $e$ is $p_2 = 1/N^T$.

Finally, conditioning on Ext outputting ⊥

$$\Pr[A \text{ wins } | \text{ Ext outputs } \perp] \leq p_1 + p_2$$

(9)

where $p_1$ and $p_2$ are as defined above.
Conclusion. Bringing Equation (5), Equation (6) and Equation (9) together, we obtain the following.

\[ \Pr[A \text{ wins}] \leq \frac{(\tau N + 1)(Q_{\text{com}} + Q_1 + Q_2)^2}{2^k} + p_1 + p_2 \]

Setting \( Q = Q_{\text{com}} + Q_1 + Q_2 \), and setting parameters such that \( p_1 + p_2 \leq 2^{-k} \) gives the required bound. \( \square \)

Note that when \( \text{sk} \) is not unique for a given \( \text{pk} \) the extractor of Lemma 7 might not output the same \( \text{sk} \) used by \( A \). However, when \( \text{pk} \) is honestly generated, the analysis of Section 4.3 ensures that \( \text{sk} \) is unique and Lemma 7 ensures that Ext will extract it.

We now come to the main theorem of this section, that uses the previous two lemmas to prove that our new PoP is secure. We also require that the KEM be KEM-SIM secure; we define this property in Appendix B and prove that it holds for KEMs constructed with the FO transform (including FrodoKEM and Kyber) in Theorem 10. We say that the key generation function of a KEM is a one-way function with probability \( \epsilon \) if is it hard to recover \( \text{sk} \) from \( \text{pk} \), such that \( R(\text{sk}, \text{pk}) = 1 \). Note that IND-CPA security of the KEM implies one-wayness of the key generation function.

Theorem 8. If KEM.KeyGen is a one-way function, and KEM is KEM-SIM secure, then the PoP construction of Section 4.2 is a secure PoP in the random oracle model, when KEM.Decaps is the auxiliary secret key usage algorithm.

Proof. The PoP security definition Definition 2 has three parts: zero-knowledge, correctness, and unforgeability.

The ZK property (Definition 3) follows immediately from our ZK lemma, when using the simulator in Lemma 5 in Exp_{POP} (Figure 3), \( A \)'s distinguishing advantage is negligible, bounded by \( \epsilon_{\text{ZK}} \). Note that in combined key generation and PoP schemes, \( A \) gets only one proof per key pair.

The correctness property is immediate if, in the PoP proof, \( P \) samples \((v_1, \ldots, v_M)\) using the same procedure as in KEM.KeyGen, which is the case for our construction in Figure 4.

Now for the UF property (Definition 2). Let \( A \) be a PoP attacker in the security experiment Exp_{PoP,UF} given in Figure 2. We construct \( B \), an algorithm that uses \( A \) as a subroutine in the ROM to invert KEM.KeyGen. Therefore \( B \) must implement the random oracles \( H_1, H_2 \), and Commit for \( A \). Algorithm \( B \) is initialized with a public key \( \text{pk} \) output by KEM.KeyGen, and must output such that \( R(\text{pk}, \text{sk}) = 1 \). Algorithm \( A \) is initialized with nothing, and starts by outputting attrs.

We describe a sequence of games, where \( B \) starts by implementing Exp_{PoP,UF} (\( A \)) as given in Figure 2 (this is Game 0), then in the last game \( B \) inverts KEM.KeyGen if \( A \) succeeds.

In Game 1, \( B \) replaces calls to Aux, which are KEM.Decaps queries, with simulated Decaps queries, using the simulator given by KEM-SIM security of KEM. Thus \( G_1 - G_0 \leq \epsilon_{\text{KEM,sim}} \).

In Game 2, \( B \) no longer computes \( (\text{pk}, \text{sk}, \pi) \leftarrow \text{PoP,KPG}(\text{attrs}) \); instead \( B \) computes \( (\text{pk}, \text{sk}) \leftarrow \text{KEM.KeyGen}() \), then uses the ZK Sim(\( \text{pk}, \text{attrs} \)) from Lemma 5 to simulate \( \pi \). As \( \text{pk} \) is identically distributed in PoP and KEM.KeyGen, this part of the change is identical in both games. The proof \( \pi \) is indistinguishable except with probability \( \epsilon_{\text{ZK}} \).

In Game 3, \( B \) no longer calls KEM.KeyGen in order to generate \( (\text{pk}, \text{sk}) \). As of Game 2, only \( \text{pk} \) is needed, and \( B \) gets a challenge \( \text{pk} \) in the OWF security game for KEM.KeyGen.

Now suppose \( A \) wins in Game 3, i.e., outputs \((\pi', \text{attrs}')\) such that \( \text{PoP,VF}(\text{pk}, \pi', \text{attrs'}) = 1 \). \( B \) uses the straight-line extractor Ext of Lemma 7 to extract \( \text{sk}' \) from \( A \) such that \( R(\text{pk}, \text{sk}') = 1 \), and outputs \( \text{sk}' \) in the OWF security game for KEM.KeyGen. Therefore, when \( A \) wins Exp_{PoP,UF}, \( B \) inverts KEM.KeyGen.

Conclusion. We have shown that the success probability for all adversaries \( A \) against the PoP scheme is bounded by

\[ 2\epsilon_{\text{ZK}} + q\epsilon_{\text{KEM,sim}} + \epsilon_{\text{OW,KEM-gen}} + \epsilon_{\text{ZK}} \]

where \( q \) is the number of Aux queries made by \( A \). The quantity \( \epsilon_{\text{ZK}} \) was shown to be negligible in Lemma 5, \( \epsilon_{\text{KEM,sim}} \) is given in Theorem 10 for a class of KEMs constructed with the FO transform (including FrodoKEM and Kyber), and \( \epsilon_{\text{ZK}} \) is given for our construction in Lemma 7.

Theorem 8 shows that the KEM does not undermine the security of the PoP. In Appendix C we consider the other direction and show that adding the PoP does not undermine security of the KEM. We show IND-CCA and KEM-SIM security are preserved when KeyGen is replaced with the combined key and proof generation function and the proof is output with the public key. Security largely follows from the zero-knowledge property of the PoP.

4.4.1 Resistance to quantum attacks. In the above analysis, we have assumed a classical attacker. A proof in the quantum random oracle model (QROM) would provide additional assurance against quantum attacks. Recent work [34] gives a way to prove straight line (or online) extractability of commit-and-open NIZK proofs in the QROM. Since our PoP is a commit-and-open protocol, and our extraction algorithm is simply reading the RO query histories, the techniques seem directly applicable to prove a QROM analog of Lemma 7. Our ZK result (Lemma 5), relies crucially on programming the random oracle so that Sim can know the challenge in advance. This was considered in [32, 33] in the context of signatures schemes constructed from 3- and 5-round ID schemes using the Fiat–Shamir transform. Since our combined generation and verification construction is not based on an ID scheme, these generic results cannot be immediately applied, however they appear to be the closest in the the QROM literature.

5 APPLICATION TO KYBER

Now we consider proof of possession via verifiable generation for Kyber instead of FrodoKEM. Kyber’s structure is similar to FrodoKEM, but Kyber is based on module LWE instead of plain LWE, so we need a new version of Lemma 4 to choose parameters.

Lemma 9 (Uniqueness of small Kyber solutions). Let \( R_q = \mathbb{Z}_q[X]/(X^n + 1) \), where \( X^n + 1 \) is the \( n \)-th cyclotomic polynomial and \( q \) is prime. Let \( A \in R_q^{q+2} \) be selected uniformly at random, and let \( s \in R_q^q \) and \( e \in R_q^2 \) be sampled from a centered binomial distribution as in the Kyber specification, so that the absolute values of the coefficients of the polynomials in \( s \) and \( e \) are at most \( \beta \). Let \( b = As + e \). Then the probability that there are values \( s', e' \in R_q^2 \), with coefficients whose absolute values are at most \( \beta \) such that \( b = As' + e' \) and
(s', e') ≠ (s, e) is at most
\[(2β + 1)^t \left( \frac{2β + 1}{q} \right)^{tn}.\]

Furthermore, if the size requirement is relaxed for some γ coefficients of s' and e', then the probability is
\[(2β + 1)^t \left( \frac{2β + 1}{q} \right)^{tn-γ}.\]

**Proof.** This proof proceeds similarly to that of Lemma 4, although because \(q\) is prime, the situation is somewhat simpler. First, note that because \(X^n + 1\) is a cyclotomic polynomial, it is irreducible, and hence, noting that \(q\) is prime, \(R_q\) is a field.

Let \(s' \in R_q^n\) with small coefficients and \(e' = b - As'\). Then \(s'\) is a valid solution only if \(e'\) also has all small coefficients. We can write \(e' = As + e\), where \(s = s' - s\). For every \(s \neq 0\) (and hence \(s' \neq s\)), we will calculate the probability that \(e'\) has small coefficients.

Note that since \(R_q\) is a field, any given non-zero polynomial \(p\) has an inverse, and hence multiplication by any fixed \(p\) polynomial is a bijection from \(R_q\) to itself. Therefore multiplying \(p\) by a uniformly random polynomial yields a uniformly random polynomial. Since summing uniformly random polynomials yields a uniformly random polynomial, we can conclude that \(As\) is a vector of uniformly random polynomials, and so is \(e' = As + e\).

The probability that every coefficient of \(e'\) is small is then \(\left( \frac{2β + 1}{q} \right)^{tn}\). Performing a union bound over all \((2β + 1)^t\) possible \(s\) gives a probability of \((2β + 1)^t \left( \frac{2β + 1}{q} \right)^{tn}\) that there is another solution \(s'\).

If we relax the size constraint of \(γ_1\) coefficients in \(s'\) and \(γ_2\) coefficients in \(e'\), then the probability is \((2β + 1)^{tn-γ_1}q^{γ_1} \left( \frac{2β + 1}{q} \right)^{tn-γ_2}\).

Setting \(γ = γ_1 + γ_2\), this is \((2β + 1)^{tn} \left( \frac{2β + 1}{q} \right)^{tn-γ}, as desired.□

To generate a Kyber key pair along with a proof of possession, we can use a slightly modified version of our cPoP from Section 4. The differences will be that the secret values are sampled differently, \(e\) is different, and vectors are replaced by polynomials where appropriate. We then choose parameters as in Section 4.3.1, where the main difference is the choice of \(M\), calculated using Lemma 9 instead of Lemma 4. Parameters \((N, τ)\) are the same for Kyber, and the values of \((M, σ)\) are given in Table 2.

**6 IMPLEMENTATION AND EVALUATION**

We now discuss performance of the key and proof generation and proof verification, and resulting proof sizes, depending on the number of parties \(N\) and the number of parallel executions \(τ\).

**Parameter sizes.** Recall that we must choose \((N, τ)\) such that \(N^τ \geq 2^κ\) for \(κ\)-bit security, and note that there are many valid pairs for each \(κ\). As with all MPCitH proofs with flexible choice of \(N\), we have a speed-size tradeoff: larger \(N\) gives shorter proofs that are slower to create and verify (as there are more parties to simulate), while smaller \(N\) has larger and faster proofs. The curve is very steep; a small change in size results in a large change in speed (we plot these curves for our implementation in Figure 8).

We implemented our construction for all FrodoKEM and Kyber parameter sets by extending their existing public C code. For Kyber, we use the already highly optimized AVX2 code base, whereas the FrodoKEM code base has no hand-crafted AVX2 optimized code, but rather exploits the compiler’s ability to auto-vectorize code. The benchmarks have been conducted on an Intel Core i7-8565U CPU running at up to 4.6 GHz, compiled with gcc 11.2.0, and with the CPU scaling governor set to performance. Figure 8 shows the median cycle counts for KPG and Vf as well as the median proof size, all over 100 executions. To obtain reliable median cycle counts for Vf, we measure the median cycle count of 100 executions of KPG and Vf. Each of these median values is then subtracted by the respective median cycle count of KPG.

The proof size varies slightly for non-power-of-two \(N\), as it depends on the values \(ι_κ\) whether \(\lceil \log_2 N \rceil\) or \(\lfloor \log_2 N \rfloor\) nodes from the seed tree must be sent. Thus, the proof size might vary by \(2κτ\) bits around the given median value.

**Profile of execution.** To get a better understanding of the proportions of single operations, we additionally profiled 1000 iterations of combined runs of KPG and Vf for the implementation of Kyber512 with \(N = 4, τ = 64\) using gprof 2.38. For Kyber512 approximately 51% of the time is spent hashing (out of which 74% is parallelized).
and 26% on sampling uniform randomness mod $q$, which is partially parallelized. Arithmetic operations take 7% of the overall time. Furthermore, 16% is spent on Arrange, which is not parallelized.

**Optimizations.** For our implementations, we use a number of optimizations. In Kyber, we use dedicated AVX2 intrinsics to vectorize the computation of $\Delta b_{k,e}$ and the computation of the missing shares $\{b_{i,k}^{G}\}_{k\in S}$. For the polynomial arithmetic, we use the already existing optimized functions. Apart from that, the main goal is to speed up the costly hashing operations. We instantiate all hash and expansion functions with SHAKE-128 for $\kappa = 128$ and SHAKE-256 for $\kappa \in \{192, 256\}$ and apply domain separation where necessary. We use the existing AVX2-enabled 4-times parallel hashing where possible: to compute the commitments $\text{com}_e^{(i)}$ in KPG and $\text{com}_e^{(i)}$ in Vf, respectively; and to expand the seeds both in KPG and Vf. Additionally, we hash the values $\mathbf{B}_e^{(i)}$ in parallel, and then add their hashes to $m_2$ rather than the shares themselves. Similarly, we hash the audited bundles $X_e$ for each party in parallel, and then add their hashes to $m_2$ rather than the audited bundles themselves. The latter two adjustments also help us reduce the memory footprint during verification, where we need to reconstruct one share per parallel execution. Instead of buffering $N - 1$ matrices $\mathbf{B}_e^{(i)}$ for FrodoKEM (or polynomial vectors for Kyber) and $N - 1$ audited bundles per parallel execution, we only need to buffer $2(N - 1)$ hash values until we can compute the two missing hashes.

In KPG, storing all $N_r$ shares (each with $M$ values) until Phase 3 is the potentially most memory-consuming operation. We tackle this by not storing the shares after the offsets $\Delta b_{k,e}$ are computed incrementally. Because of this, however, it is necessary to re-sample the shares for Phase 3 of KPG. On the other hand, we achieve that our implementations do not exceed 8 MB RAM usage for any of the parameter choices in Figure 8 as well as for the higher parameter sets in Appendix D. If we just store the shares and omit re-sampling, the cycle count for KPG is reduced by 30% for Kyber512 with $N = 4$, $\tau = 64$.

More details on performance and proof size at higher security levels can be found in Appendix D.

7 CONCLUSION AND DISCUSSION

From our construction, we can achieve a proof of possession for KEM keys that can be used in a non-interactive certificate enrollment process, similar to Certificate Signing Requests.

It is the case that our combined proof and key generation method means only a single proof can be generated. To authenticate other events in the certificate lifecycle, we cannot use a proof from the same verifiable generation approach (since proof generation happens concurrently with key generation), but fortunately certificate lifecycle protocols like ACME already allow for revocation events to be signed by a separate account key [6, §7.6]. Furthermore, by using the attributes field we can bind a signature public key to the PoP and use it for subsequent PoPs (or revocation), as a work-around.

It is not required to know the authority or to obtain a nonce from them in advance, since this would break the desirable non-interactive property of CSRs. For example, the ACME Protocol [6] for certificate enrolment does not require that CSR contents are bound to the protocol or CA, and ACME doesn’t include any provisions preventing CSR replays (which is often considered a feature, e.g. where the ACME client is run outside of the production network and does not have access to the private key).

Our construction allows arbitrary attributes to be bound to the key generation, which can be used to incorporate the standard fields from a CSR or more, binding the key generation to its intended use.

Compared to existing techniques to prove knowledge of $\text{LWE}$ secrets, our MPCitH-based approach for proof of possession via verifiable generation can be configured to both run in reasonable time and achieve reasonably low proof sizes; see Table 1 for some comparisons. Additionally, the flexibility in parameter choice allows adjustments to meet use case requirements. For CSRs, which are usually generated infrequently, a runtime on the order of a second might be acceptable, which helps to keep communication cost low while still providing reasonable user experience. On the other side, when a larger proof is acceptable, it is possible to run KPG and Vf in less than 10 ms for Kyber512.

Furthermore, our construction scales well for higher parameter sets with $\kappa \in \{192, 256\}$. To the best of our knowledge, we are the first to give a practical proof of possession with parameters and an implementation for lattice secrets for security levels above 128 bits.

We have demonstrated our technique for FrodoKEM and Kyber. For the remaining NIST Round 3 lattice-based KEMs, SABER [27] and NTRU [23], our approach may be applied in some form, since MPCitH is so flexible. However, doing so efficiently may require some additional research, due to differences in the constructions. SABER is close in structure to FrodoKEM and Kyber, but its public key computation includes a right-shift that removes low-order bits from the public key, which is not conducive to efficient implementation in our MPCitH approach. NTRU has more differences in its structure, and key generation includes an inversion operation in the ring which can be implemented with an MPCitH proof, albeit with a different MPC protocol (e.g., see [7] for approaches to implementing the AES S-box, which is a field inversion). To follow our approach, one would also need to prove analogs of Lemmas 4 and 9 about uniqueness of small solutions.

ACKNOWLEDGMENTS

D.S. was supported by Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery grants RGPIN-2016-05146 and RGPIN-2022-03187 and an NNet Assure grant for the project “Standardizing KEMTLS”. T.G. and G.L. were supported by the German Research Foundation under Germany’s Excellence Strategy – EXC 2092 CASA – 390781972, through the H2020 project PROMETHEUS (grant agreement ID 780701), and by the Federal Ministry of Education and Research of Germany through the QuantumRISC (16KIS1038) and PQC4Med (16KIS1044) projects.

REFERENCES


[63] Max Pritikin, Peter Yee, and Dan Harkins. 2013. RFC 7038: Enrollment over Transport key exchange protocol version 1.0


Given a KEM, we define the security notion of decapsulation simulatability. The decapsulation operation of a KEM is said to be simulatable if there exists a simulator $\text{Sim}$, such that $\text{Sim}$ takes a public key $pk$ and ciphertext $c$ as inputs and returns a plaintext $m$ in polynomial time. Decapsulation simulatability is necessary to ensure that the same ciphertext is re-created during the re-encryption step of decapsulation (a property called rigidity in [43]).

**B.2 Decapsulation Simulatability**

For a KEM, we define the security notion of decapsulation simulatability. The decapsulation operation of a KEM is said to be simulatable if there exists a simulator $\text{Sim}$, such that $\text{Sim}$ takes a public key $pk$ and ciphertext $c$ as inputs and returns a plaintext $m$ in polynomial time. Decapsulation simulatability is necessary to ensure that the same ciphertext is re-created during the re-encryption step of decapsulation (a property called rigidity in [43]).

In Figure 9, we give the specific details of the variant of the FO transform we study. This version is called "FO with implicit rejection" and denoted $\text{FO}^⊥$ in [43], and underlies the KEM constructions of Kyber [66] and Frodo [58]. In [43] the transform is separated into two parts: first any IND-CPA secure scheme PKE is transformed to $\text{PKE}_1$ by derandomizing encryption by generating the randomness by hashing the message, and adding the re-encryption check in decapsulation. The second part uses $\text{PKE}_1$ to construct the KEM with implicit rejection. Our presentation in Figure 9 combines these two parts. In practice, Kyber and Frodo also derandomize the KEM by deriving the random coins used for $\text{PKE.Enc}$ from $m$. This is necessary to ensure that the same ciphertext is re-created during the re-encryption step of decapsulation (a property called rigidity in [43]).

In [43], the authors prove that if $\text{PKE}$ is IND-CPA secure, then $\text{KEM} = \text{FO}^⊥[\text{PKE}]$ is IND-CCA secure. This reduction must answer Decaps queries from the KEM IND-CCA attacker, and as $sk'$ is generated as part of the IND-CPA game, the reduction has only pk. Therefore this CCA proof also implicitly proves KEM-SIM security, giving the result in Theorem 10, which we have re-proven more directly here. A second requirement of KEM-SIM secure with a game; we say that a KEM is KEM-SIM secure if no adversary wins this game with probability significantly better than 1/2. Of course if such a simulator were efficiently implementable by any party the KEM would not be secure, so it only makes sense in an idealized model where $\text{Sim}$ has some additional capability, such as the ability to choose system parameters, or simulate hash functions in the random oracle model (ROM). Our focus will be on the ROM setting, and our definition makes this explicit by adding the random oracle $H$ and simulated version $H_{\text{Sim}}$ as inputs to the adversary.

Figure 9: The Fujisaki–Okamoto transform (variant $\text{FO}^⊥$ from [43]) creates the KEM (KeyGen, Encaps, Decaps) from the PKE (KeyGen, Enc, Dec). The functions $G$ and $H$ are hash functions modeled as random oracles.

```
f_{\text{fo}} \leftarrow \text{FO.KeyGen}(1^\kappa)
1: m \leftarrow \text{PKE.KeyGen}(1^\kappa)
2: s \leftarrow \{0, 1\}^\kappa
3: sk \leftarrow (s, sk')
4: return (pk', sk)
```

Figure 10: KEM-SIM security game in the random oracle model.

```
\text{Game KEM-SIM}
1: (pk, sk) \leftarrow \text{KEM.KeyGen}(1^\kappa)
2: b \leftarrow \{0, 1\}
3: if b = 0 then b' \leftarrow A^{\text{Decaps}(sk')}_{H^⊥}(pk)
4: if b = 1 then b' \leftarrow A_{H_{\text{Sim}}}(pk, sk')
5: return \{b = b'\}
```
Theorem 10. Let PKE be an IND-CPA secure public key encryption scheme that is δ-correct, and let KEM = FO^4[PKE] be the KEM obtained by applying the Fujisaki–Okamoto transform given in Figure 9. Then KEM is IND-SIM secure in the random oracle model.

Concretely, for any KEM-SIM and IND-CPA adversaries, we have
\[ \text{Adv}_{\text{KEM-SIM}}^{\text{KEM}} \leq 2q_\text{H}/2^k + q_G \cdot \delta + q_G \text{Adv}_{\text{PKE}}^{\text{OW-CPA}} \]
where \( q_G \) and \( q_\text{H} \) are the number of queries to the random oracles \( G \) and \( H \) (respectively), and \( \kappa \) is the security parameter.

Proof. The proof is an application of Theorem 3.2 and a modified version of Theorem 3.4 from [43]. First note that in Figure 9 we apply the transform \( T \) from [43], so our presentation of the FO transform is equivalent to FO^4 of [43] applied to PKE = T[PKE, G]. (Recall that PKE is the derandomized version of PKE with a re-encryption check during decryption.) Then we can apply Theorem 3.2 of [43] to show that PKE is OW-CPA secure. Briefly, this means that PKE is a one-way function of the message, even under plaintext checking attacks. In this type of attack, the adversary has a plaintext checking oracle, that tests if ciphertext \( c \) has plaintext \( m \). Intuitively, after derandomization, anyone can implement a plaintext checking oracle as

\[ \text{Pco}(c, m) \text{ Plaintext checking oracle for PKE} \]

\[
\begin{align*}
1 & : \text{if } c = \text{PKE}_1.\text{Enc}(pk, m) \text{ then return } 1 \\
2 & : \text{else return 0}
\end{align*}
\]

Our use of Theorem 3.2 of [43] is why we require that PKE be δ-correct.

Now we prove that KEM is KEM-SIM secure. We essentially use part of the IND-CCA security proof of KEM in Theorem 3.4 of [43] as it already simulates Decaps queries; we use the same approach to implement the Sim algorithm in the KEM-SIM game. Let \( B \) be an attacker in the OW-CPA game of PKE; \( B \) implements the random oracle \( H \) and the KEM-SIM game for adversary \( A \). We proceed by a series of games, transitioning from Decaps(sk, ·) in Game 0 to Sim(pk, ·) in Game 3, and quantify \( A \)'s success probability for a bounded number of hash queries \( q_H \) (note that in our accounting each Decaps query incurs a H query). We use the shorthand \( G_i \) to denote the probability that \( A \) outputs 1 in game \( i \).

Game G0. In game G0, \( B \) implements \( H : \{0, 1\}^* \rightarrow \{0, 1\}^* \) by lazy sampling with the list \( L_H \) (keeping track of previous queries \((c, m)\) and corresponding output \( K \)) as in the left side of Figure 11.

Figure 11: Oracles used in games G0 and G1 in proof of Theorem 10.

Since \( B \) implements the KEM-SIM game exactly as in Figure 10, and Sim is the same as Decaps, \( A \)'s advantage is exactly 1/2.

Game G1. In game G1, we change Decaps so that \( B \) no longer makes use of the secret key component \( s \) during Decaps as shown in the right side of Figure 11: we replace \( H(c|m) \) with \( H'(c) \) where \( H' \) is a private random oracle internal to \( B \). \( B \) also returns \( H'(c) \) for \( H(c|m) \) in the Decaps oracle in the event that \( m' = s \). Since \( s \) is chosen uniformly from \( \{0, 1\}^k \), \( G_1 - G_0 \leq q_H/2^k \).

Game G2. In game G2 we change \( B \) to no longer make use of the secret key component \( sk' \), by changing Decaps and H as shown in Figure 12. In G2 the list \( L_D \) keeps track of \((c, K)\) such that \( \text{Decaps}(c) = K \), and either \( H \) was queried on \((c, m)\) or \( H \) was queried on \( c \).

Figure 12: Oracles used in games G2 and G3 in proof of Theorem 10.

We now argue that \( G_2 = G_1 \). Consider a fixed ciphertext \( c \), with \( m' = \text{PKE}_1.\text{Dec}(sk', c) \).

- Case 1: \( m' \in \{\bot, s\} \). We argue that such a \( c \) cannot be added to \( L_D \) via an H-query. Querying \( m' = \bot \) is not allowed, and if \( m' = s \) is queried, \( G_2 \) aborts. Therefore, when such a \( c \) is queried to Decaps, it is not in \( L_D \), so a random key \( K \) is output, as in \( G_1 \).
- Case 2: \( m' \not\in \{\bot, s\} \). For this case we must show that \( G_2 \)'s implementation of H and Decaps appear consistent to \( A \) (as they were in \( G_1 \)). There are two sub cases, depending on the order that \( A \) queries H and Decaps.
  - First is the "natural" order, i.e., the order that occurs when following the code of KEM.Encaps then KEM.Decaps from Figure 9. \( A \) starts with the query \( H(c|m') \), and we have \( \text{Pco}(c, m') = 1 \) so \((c, K)\) with a random \( K \) is added to \( L_D \), defining \( \text{Decaps}(c) = K = H(c|m') \) as in \( G_1 \).
  - Now for the reverse order, \( A \) queries \( \text{Decaps}(c) \), and there is no entry in \( L_D \), so a random \( K \) is output and \((c, K)\) is logged in \( L_D \). When \( H(c|m') \) is subsequently queried, again \( \text{Pco}(c, m') = 1 \) and this time \( c \) is found in \( L_D \), so that the correct value of \( K \) is returned, and we have \( H(c|m') = \text{Decaps}(c) \).

Since \( A \)'s view is identical in games 1 and 2, we have \( G_2 = G_1 \).
Game $G_3$. Finally, in game 3 we modify $\mathcal{B}$ to stop using $s$ for H queries; we simply delete Line 4 as shown in Figure 12. Since $s$ is not used elsewhere, and is chosen uniformly at random from $\{0, 1\}^*$, the probability of an abort is negligible and the change will go unnoticed by $\mathcal{A}$. More precisely, we have $G_3 - G_2 = q_{H}/2^k$. In Game 3 $\mathcal{B}$ no longer uses sk, and so $\text{Sim}_{pk}$ and $\text{H}_{\text{Sim}}$ are given by their $G_3$ versions in Figure 12.

In the KEM-SIM game, $\mathcal{A}$’s advantage corresponds to $G_3 - G_0$. Collecting the probabilities, we have that $\mathcal{A}$’s concrete advantage is bounded by $2q_{H}/2^k + \text{Adv}_{\text{Adv}_{\text{PKE}}}(\mathcal{B})$. The bound on $\text{Adv}_{\text{Adv}_{\text{PKE}}}(\mathcal{B})$ given by [43, Theorem 3.2] is

$$\text{Adv}_{\text{Adv}_{\text{PKE}}}(\mathcal{B}) \leq (q_G + q_P)\delta + (q_G + q_P)\text{Adv}_{\text{Adv}_{\text{PKE}}}$$

where $q_G$ is the number of queries to the random oracle G and $q_P$ is the number of plaintext checking queries. Since the plaintext checking queries are simple re-encryption checks, they do not require an oracle query, only a query to the hash function $G$. Therefore, the bound can be simplified by having $q_G$ account for both plaintext checking and G-queries (as was done in the theorem statement).

We do not know if in general IND-CCA security of a KEM implies KEM-SIM security, however this seems plausible based on the KEMs we considered.

As FrodoKEM and Kyber are constructed from $\delta$-correct2 IND-CPA encryption schemes with the FO$^4$ transform, they also enjoy KEM-SIM security.

**Corollary 1.** FrodoKEM and Kyber are KEM-SIM secure.

### C COMPOSITION SECURITY OF PROOFS OF POSSESSION AND KEMS

In Section 4.3 we considered the composition security of our cPoP protocol and KEMs. Theorem 8 considers one direction, showing that the KEM does not undermine the security of the cPoP: when the Decaps function is used as the auxiliary function, the cPoP remains secure. Here we consider the other direction and show that adding the cPoP does not undermine security of the KEM.

We must show that, given an IND-CCA-secure KEM, if we then modify KeyGen to additionally output a proof of possession, the augmented KEM is also IND-CCA-secure. A first requirement is that the cPoP be ZK, otherwise it could leak information about the KEM secret key. But even so, there may still be a bad interaction between the KEM and the cPoP. For instance, consider a (contrived) KEM that outputs the secret key when the Decaps algorithm is given an input that is a valid cPoP. Such a KEM could be IND-CCA-secure without cPoPs, since creating a valid cPoP is difficult if the cPoP is unforgeable, but once a cPoP is made public, IND-CCA security is immediately broken. For this latter composition issue, we rely on the random oracle model to separate the KEM and cPoP. The random oracles used to realize the cPoP will, by assumption, be independent of the random oracles used to realize the KEM (so in the contrived example above, it becomes impossible for Decaps to verify the cPoP). The assumption of random oracle independence can be met in practice by domain separation, where each hash function is prefixed by a primitive-specific value.

3For FrodoKEM, $\delta$-correctness is analyzed in the specification [58, §2.2.7], and for Kyber all parameter sets are chosen to ensure $\delta < 2^{-128}$, see the specification [66, §1.3].

The next two definitions make precise what it means to use a cPoP with a KEM, and for the modified KEM to remain secure.

**Definition 11.** Let $KEM = (\text{KeyGen}, \text{Encaps}, \text{Decaps})$ be a key encapsulation mechanism. Let $KEM^+ = (\text{KeyGen}^+, \text{Encaps}, \text{Decaps})$ be an associated KEM that integrates a key generation and proof of possession scheme (Definition 1). We call $KEM^+$ a key encapsulation mechanism with verifiable generation. The key generation function $\text{KeyGen}^+$ takes as input the string $\text{attrs}$, and outputs $(pk, sk, \pi)$ where $(pk, sk)$ is the KEM keypair, and $\pi$ is a proof of possession for sk with respect to pk and the attributes $\text{attrs}$. Note that this requires $\text{KeyGen}^+$ to output $(pk, sk)$ that are identically distributed to KeyGen.

**Definition 12.** A key encapsulation mechanism with verifiable generation, $KEM^+$, is IND-CCA-secure if no efficient adversary wins the IND-CCA$^+$ security game for KEMs. The game IND-CCA$^+$ is identical to IND-CCA, except that the $\text{attrs}$ string required for $\text{KeyGen}^+$ is chosen by the adversary at the start of the experiment.

Analogous to Definition 12 we can define what it means for $KEM^+$ to be KEM-SIM secure.

**Definition 13.** A key encapsulation mechanism with verifiable generation, $KEM^+$, is KEM-SIM-secure if no efficient adversary wins the KEM-SIM$^+$ security game for KEMs. The game KEM-SIM$^+$ is identical to KEM-SIM (Figure 10), except that the $\text{attrs}$ string required for $\text{KeyGen}^+$ is chosen by the adversary at the start of the experiment.

We now prove that FrodoKEM remains IND-CCA- and KEM-SIM-secure when combined with our verifiable generation scheme; the proof may easily be adapted to Kyber. As noted above, we model all random oracles separately.

**Theorem 14.** Let KEM be the FrodoKEM key encapsulation mechanism, and $KEM^+$ be the associated KEM with verifiable generation, using our cPoP protocol of Section 4.2 with random oracles $\text{Commit}, \text{ExpandTape}, H_1, H_2$ independent from those used in KEM. Then $KEM^+$ is IND-CCA$^+$-secure and KEM-SIM$^+$-secure in the random oracle model.

**Proof.** We start with IND-CCA$^+$, and give the reduction in detail; the reduction for KEM-SIM$^+$ is nearly identical.

First we note that $KEM = (\text{KeyGen}, \text{Encaps}, \text{Decaps})$ is IND-CCA-secure [58]. In Theorem 8 we have shown that the protocol of Section 4.2 is a secure cPoP (Definition 2). By the correctness property of the cPoP, the outputs $(pk, sk)$ of $\text{KeyGen}^+$ are identically distributed to those of $\text{KeyGen}$.

We now reduce the IND-CCA$^+$ security of $KEM^+$ to the IND-CCA security of KEM in the ROM. Let $C$ be the challenger for the IND-CCA security game of KEM, played with our reduction algorithm $B$. Algorithm $B$ is attacking the IND-CCA security of KEM, using algorithm $\mathcal{A}$ as a subroutine. Algorithm $\mathcal{A}$ is an IND-CCA$^+$ attacker for $KEM^+$. 
Table 2: Parameter selection for FrodoKEM and Kyber based on Section 4.3.1. At security level $\kappa$, to ensure with high probability $\sigma$ secret key entries at most $\gamma$ of which may not be small, generate $M$ bundles and audit $M - \sigma$ of them.

<table>
<thead>
<tr>
<th>Security level $\kappa$</th>
<th>FrodoKEM $\sigma$</th>
<th>$M$</th>
<th>$\gamma$</th>
<th>Kyber $\sigma$</th>
<th>$M$</th>
<th>$\gamma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>10240</td>
<td>13233</td>
<td>341</td>
<td>1024</td>
<td>1280</td>
<td>336</td>
</tr>
<tr>
<td>192</td>
<td>15616</td>
<td>19485</td>
<td>591</td>
<td>1536</td>
<td>1870</td>
<td>558</td>
</tr>
<tr>
<td>256</td>
<td>21504</td>
<td>25986</td>
<td>919</td>
<td>2048</td>
<td>2493</td>
<td>744</td>
</tr>
</tbody>
</table>

Message flows. Algorithm $B$ is initialized by $C$ with a public key $pk$ output by $\text{KEM.KeyGen}$. We allow $A$ to output the attributes string $\text{attrs}$, following the IND-CCA* game. Since the $\text{cPoP}$ scheme is ZK (see Definition 3 and Lemma 5), $B$ uses $\text{Sim}(pk, \text{attrs})$ to simulate $\pi$, a proof of possession for $sk$ associated with $pk$. ($\text{Sim}$ uses the random oracles to simulate transcripts.) All other messages from $A$ in the IND-CCA* game are simply passed through by $B$ to $C$.

Random oracles. There are multiple independent random oracles in use during the reduction. First, FrodoKEM uses three ROs ($G_1$, $G_2$, and $F$) during $\text{KEM.Encaps}$ and $\text{KEM.Decaps}$ and $\text{KEM.KeyGen}$. These are implemented by $C$, and $B$ is given oracle access to them. Since KEM* uses these ROs in exactly the same way as in KEM, when $A$ makes queries to these to ROs, $B$ simply proxies them to the corresponding oracles exposed by $C$. Our $\text{cPoP}$ uses four ROs ($\text{Commit}$, $\text{ExpandTape}$, $H_1$, $H_2$) during KEM*. $\text{KeyGen}$. These are all simulated by $B$ and exposed to $A$, in particular note that $\text{Sim}$ depends on these RO to simulate transcripts.

Success Probability. We now argue that when $A$ is successful in the IND-CCA* game, $B$ is successful in the IND-CCA game. First we argue that $B$’s simulation of $\pi$ is indistinguishable. We proceed with a hybrid argument. In game $G_0$, $B$ does not use $\text{Sim}$; it calls $\text{cPoP.KPG}$ and generates $(sk, pk, \pi)$ honestly. Then in $G_1$, $B$ no longer uses $sk$ but instead uses the $pk$ value from $C$, and the simulator to create $\pi$. By correctness of $\text{cPoP.KPG}$, $pk$ is identically distributed in both games, and by the zero-knowledge property $\pi$ is distributed statistically close in both games (the distributions differ by at most a negligible distance $\epsilon_{\pi}$). In particular this means the distribution of $\pi$ is independent of $sk$.

The last thing to note is that $\pi$ cannot be verified without access to the random oracles used by $\text{cPoP}$ ($\text{Commit}$, $\text{ExpandTape}$, $H_1$, $H_2$). This is easy to see by inspection of Figure 5, as the last step of verification compares two hash digests. It follows that the output distribution of KEM. Decaps cannot depend on the validity of a well-formed proof string $\pi$. Since the simulation is indistinguishable to $A$ and KEM. Decaps behaves identically in the real and simulated games, $B$ succeeds in the IND-CCA game when $A$ succeeds in the IND-CCA* game.

KEM-SIM Security. By our analysis in Appendix B, FrodoKEM is KEM-SIM-secure (Theorem 10 and Corollary 1). Using the same setup as in the IND-CCA case, we can use a KEM-SIM* attacker $A$ to construct a KEM-SIM attacker $B$. The same main arguments apply: replacing the proof $\pi$ with a simulated proof is indistinguishable, and the simulated $\pi$ is may not be verified without access to the independent ROs that instantiate the proof system. □

D ADDITIONAL DETAILS OF PARAMETERS, PERFORMANCE AND COMPARISON

Table 2 shows secure choices of $M$ for all security levels of FrodoKEM and Kyber depending on $\kappa$ according to Equation (4).

Table 3 and Figures 13 and 14 show runtime performance and proof sizes for our approach applied to FrodoKEM and Kyber at the 192- and 256-bit security levels.
Figure 13: KPG and Vf performance and proof size for the 192-bit security level

(a) Kyber-768, \( \kappa = 192 \)

(b) FrodoKEM-976, \( \kappa = 192 \)

Figure 14: KPG and Vf performance and proof size for the 256-bit security level

(a) Kyber-1024, \( \kappa = 256 \)

(b) FrodoKEM-1344, \( \kappa = 256 \)

Table 3: KPG and Vf performance and proof size for FrodoKEM and Kyber at 3 security levels

<table>
<thead>
<tr>
<th>( N )</th>
<th>( \kappa )</th>
<th>Level 1 (( \kappa = 128 ))</th>
<th>Level 3 (( \kappa = 192 ))</th>
<th>Level 5 (( \kappa = 256 ))</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>KPG time</td>
<td>Vf time</td>
<td>KPG time</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \times 10^6 ) cycles</td>
<td>( \times 10^6 ) cycles</td>
<td>( \times 10^6 ) cycles</td>
</tr>
<tr>
<td>FrodoKEM</td>
<td>4</td>
<td>1594.1</td>
<td>169.2</td>
<td>152.0</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>1072.3</td>
<td>230.4</td>
<td>210.8</td>
</tr>
<tr>
<td></td>
<td>31</td>
<td>650.0</td>
<td>555.8</td>
<td>553.4</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>401.5</td>
<td>2901.3</td>
<td>2909.8</td>
</tr>
<tr>
<td>Kyber</td>
<td>4</td>
<td>127.2</td>
<td>8.8</td>
<td>8.9</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>86.2</td>
<td>12.0</td>
<td>11.7</td>
</tr>
<tr>
<td></td>
<td>31</td>
<td>53.0</td>
<td>25.5</td>
<td>26.4</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>33.4</td>
<td>130.5</td>
<td>149.5</td>
</tr>
</tbody>
</table>

Using same \( r \) corresponding to the \( N \) as given in Figures 8, 13 and 14.