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 a 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 $(\mathbf{A}, \mathbf{B})$ where $\mathbf{B} = \mathbf{AS} + \mathbf{E} \bmod q$ for secret matrices $\mathbf{S}$ and $\mathbf{E}$ consisting of small entries. At the 128-bit security level, $q = 2^{15}$, "small" means in $[0, \pm 12]$, and $\mathbf{S}$ and $\mathbf{E}$ 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 construction yielding 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 $\approx 2^{36}$) 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 σ of entries among the secrets $\mathbf{S}$ and $\mathbf{E}$ (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 β 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 β 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

Table 1: Comparison of proof sizes and proof generation runtime for several parameter regimes

Scheme	**Technique**	**Regime 1**	**Regime 2**		Kyber512		Frodo640	
		Size	**Size**	**Time**	**Size**	**Time**	**Size**	**Time**
Proof of knowledge of secret key (and proof of verifiable decryption, denoted $^\diamond$)								
Stern-like [50]	ZKP from SIS	2.3 MB†	4.3 MB†					
[8]	MPCitH		4.1 MB	2.4 s			$\geq$ 8.42 MB‡	
[16]	ZKP from RLWE & RSIS	384 kB†						
[12]	Σ-prot. for permuted-kernel	233 kB	444 kB					
Ligero [4]	zkSNARK from PCPs	157 kB†	200 kB†					
Aurora [10]	zkSNARK for R1CS	72 kB†	71 kB†					
[36]	ZKP from MLWE & MSIS	47 kB, 61 kB$^\diamond$						
[55]	ZKP from MLWE & MSIS	47 kB*						
[56]	ZKP from MSIS & ext. MLWE	33 kB			43.6 kB$^\diamond$			
[54]	ZKP from MLWE & MSIS	14 kB			19.0 kB$^\diamond$			
Proof of verifiable generation								
Ours $(31, 26)$	MPCitH	251 kB	879 kB		52.9 kB	0.006 s	650 kB	0.12 s
Ours $(256, 16)$	MPCitH	155 kB	542 kB		33.4 kB	0.028 s	402 kB	0.63 s
Ours $(1626, 12)$	MPCitH	117 kB	407 kB		25.7 kB	0.109 s	302 kB	2.59 s
Ours $(65536, 8)$	MPCitH	79 kB	272 kB		17.8 kB	3.77 s	203 kB	85.6 s

Empty cell indicates estimates for parameter regime not available in the original paper or subsequent literature.
"Ours (N, τ)" denotes number N of MPC-in-the-head parties and number τ of MPC repetitions.
Regime 1: modulus $q \approx 2^{32}$, number of secret entries $\sigma = 2048$, ternary secrets $\{-1, 0, 1\}$, 128-bit security level.
Regime 2: modulus $q \approx 2^{61}$, number of secret entries $\sigma = 4096$, binary secrets $\{0, 1\}$, 128-bit security level.
Kyber512: modulus $q = 3329$, number of secret entries $\sigma = 1024$, secret $[0, \pm 2]$, 128-bit security level.
Frodo640: modulus $q = 2^{15}$, number of secret entries $\sigma = 10240$, secret $[0, \pm 12]$, 128-bit security level.
†: Estimates by Beullens [12]. ‡: Estimate by us, using edaBits [35] for comparisons.
*: Estimate by Lyubashevsky et al. [56]. $^\diamond$: For proof of verifiable decryption.
Runtime of "Ours" is a single-threaded implementation on Intel Core i7-8565U CPU running at up to 4.6 GHz, compiled with gcc 11.2.0.

non-interactive scheme. Any additional attributes needed by the certificate issuance process can be incorporated into the Fiat–Shamir challenge generation, similar to messages in a signature scheme.

The MPC approach is compelling for our relation, $\mathbf{B} = \mathbf{AS} + \mathbf{E}$ for public $\mathbf{A}$, since it is entirely linear in secret information ($\mathbf{S}$ and $\mathbf{E}$). Linear operations are done "locally" by the parties, and are free in terms of communication. The main challenge is to ensure that the values in $(\mathbf{S}, \mathbf{E})$ are small – and this is where we benefit by 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 τ times to prove that $\mathbf{B}$ 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 MPCitH-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 $\mathbf{B} = \mathbf{AS} + \mathbf{E}$. 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 ($q \approx 2^{61}$, $\sigma = 4096$, binary secrets), our system yields proof sizes for the same parameters at sizes as small as 272 kB and of 542 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 $\leq \beta = 12$). Applying the edaBits construction [35] for integer comparisons in MPC, we extrapolate that checking Frodo640's $\sigma = 10240$ secret values have absolute value at most $\beta = 12$ would lead to a proof of at least 8.42 MB (at just 80 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 $q \approx 2^{32}$, $\sigma = 2048$ samples, and ternary secrets. zkSNARKs achieve proof sizes of 72 kB (Aurora [10]) or 157 kB (Ligero [3]). Recent ZK proofs based on MLWE and MSIS achieve proof sizes of 47 kB [36], 33 kB [56], and 14 kB [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 ($q = 3329$, $\sigma = 1024$ samples, secrets $\leq \beta = 2$) 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 κ), that are stateful, i.e. if $\mathcal{A}$ appears twice in an security experiment, state may be shared between instances.

[1]Our C implementation of FrodoKEM and Kyber verifiable generation (with AVX2 optimizations for Kyber) can be downloaded at https://github.com/Chair-for-Security-Engineering/KEM-NIZKPoP , and is accompanied by scripts for calculating proof sizes.

```
FrodoKEM.KeyGen()
1 : seed_A ←$ {0,1}^{len_seedA}
2 : A ← Frodo.Gen(seed_A) ∈ $\mathbb{Z}_q^{n\times n}$
3 : seed_SE ←$ {0,1}^{len_seedSE}
4 : $(\mathbf{r}^{(0)}, \mathbf{r}^{(1)} \dots, \mathbf{r}^{(2n\bar{n}-1)}) \leftarrow$ SHAKE(0x5F||seed_SE, $2n\bar{n} \cdot \mathsf{len}_\chi$)
5 : $\mathbf{S}^T \leftarrow$ Frodo.SampleMatrix($(\mathbf{r}^{(0)}, \mathbf{r}^{(1)}, \dots, \mathbf{r}^{(n\bar{n}-1)}), \bar{n}, n, \chi$)
6 : $\mathbf{E} \leftarrow$ Frodo.SampleMatrix($(\mathbf{r}^{(n\bar{n})}, \mathbf{r}^{(n\bar{n}+1)}, \dots, \mathbf{r}^{(2n\bar{n}-1)}), n, \bar{n}, \chi$)
7 : $\mathbf{B} \leftarrow \mathbf{AS} + \mathbf{E}$
8 : return $(\mathsf{pk} = (\mathsf{seed_A}, \mathbf{B}), \mathsf{sk} = \mathbf{S}^T)$

Frodo.SampleMatrix($\mathbf{r}^{(0)}, \dots, \mathbf{r}^{(n_1 n_2 - 1)}$), where $\mathbf{r}^{(i)} \in \{0,1\}^{\mathsf{len}_\chi}$
1 : for $i = 0, \dots, n_1 - 1$ do
2 :   for $j = 0, \dots, n_2 - 1$ do
3 :     $\mathbf{E}_{i,j} \leftarrow$ Frodo.Sample($\mathbf{r}^{(i\cdot n_2 + j)}, \chi$)
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^κ) 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 $\perp$ if c is not a valid encapsulation. A KEM with *implicit rejection* does not output $\perp$ 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$_\mathbf{A}$ used to pseudorandomly generate a matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times n}$. The algorithm also samples a secret seed seed$_\mathbf{SE}$ used to pseudorandomly generate two secret $n \times \bar{n}$ matrices $\mathbf{S}$ and $\mathbf{E}$ containing entries sampled from a distribution χ. The public key is the seed seed$_\mathbf{A}$ and the matrix $\mathbf{B} = \mathbf{AS} + \mathbf{E}$; the secret key is $\mathbf{S}$. 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 $\mathbb{Z}_q$. As a concrete example for FrodoKEM at the 128-bit security level, $q = 2^{15}$ and $\beta = 12$, so χ yields samples in $[0, \pm 12]$.

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 $\mathbf{A}, \mathbf{B}, \mathbf{S}, \mathbf{E}$ are polynomials in a ring $R_q = \mathbb{Z}_q[X]/(X^d + 1)$; 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 $[0, \pm 2]$.

MPC-in-the-head. The MPC-in-the-head (MPCitH) paradigm [44] connects two fundamental primitives in cryptography: secure multiparty computation (MPC) and zero-knowledge proofs, showing that an MPC protocol for a functionality f can be be used to construct a ZK proof of x such that $y = f(x)$. Since MPC protocols exist (or can be designed) for most f, we can create ZK proofs for arbitrary circuits. Furthermore, this was shown to be efficient in [40], inspiring many constructions of signature schemes [7, 12, 13, 22, 28, 29, 31, 37, 46, 47, 69], as they need only assume that f is a one-way function (OWF) in the random oracle model, they potentially have post-quantum security. Conditions under which Fiat–Shamir-based signature schemes have security in the quantum random oracle model, are given in [33].

We give a high level explanation of MPCitH, somewhat specialized to our use, then give additional details in Section 4 and refer to the references given above for more details. We create an interactive proof protocol that $f(x) = y$ (for public f, y) between a prover P (with secret input x) and verifier V. P starts by additively secret sharing x among the N parties, as input to the MPC. Then P simulates the execution of the MPC protocol; parties record the messages they exchange (either directly or via broadcast), along with their input share to form their *view* of the protocol execution. The state of the computation remains secret shared until the last step, when the parties broadcast their shares of y so that it may be reconstructed. P then commits to all N views, and sends the commitment to V, who responds with a random subset of the N parties (of size $N - 1$). For these parties P will reveal their views – we refer to these parties as *opened*, and the remaining party as *unopened*. Given the input shares of the opened parties, and the messages sent by the unopened party, V can recompute all $N - 1$ views and check that they match those in P's commitment. The verifier accepts the proof if the output is y and the views match. After one repetition of this process, V is convinced that P knows x with probability $1/N$. As mentioned above, the MPC protocol for our f is very simple as the parties can add secret-shared values *locally*, i.e., without communication, and can also multiply secret-shared values by a public constant locally. Intuitively, the protocol is sound because V recomputes the circuit, and it is zero-knowledge if the protocol is $(N - 1)$-private, meaning that no information is leaked about x given the state of $N - 1$ parties.

Our protocol of Section 4 is a five-round protocol, where the first exchange acts as a setup phase (not to be confused with a preprocessing phase, as in [47]), that establishes the inputs to the MPC, and the rest of the protocol follows the sketch given above.

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() $\xrightarrow{\$}$ (pk, sk): A probabilistic key generation algorithm that outputs a public, secret key pair.
- PoP.PG(pk, sk, attrs) $\xrightarrow{\$}$ π: A probabilistic proof generation algorithm that takes as input a public key, secret key,

and attributes attrs $\in \{0,1\}^*$, and outputs a proof string π. (In our application, attrs could be the body of a certificate signing request, for example.)

- $\mathsf{PoP.Vf}(\mathsf{pk}, \mathsf{attrs}, \pi) \to \{0,1\}$: A deterministic verification algorithm that takes as input a public key pk, attributes attrs, and proof π, 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,

$$\Pr\left[\mathsf{PoP.Vf}(\mathsf{pk}, \mathsf{attrs}, \pi) \Rightarrow 1 \;\middle|\; \begin{array}{l}(\mathsf{pk}, \mathsf{sk}) \leftarrow\!\!\$\ \mathsf{PoP.KG}();\\ \pi \leftarrow\!\!\$\ \mathsf{PoP.PG}(\mathsf{pk}, \mathsf{sk}, \mathsf{attrs})\end{array}\right] = 1.$$

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:

- $\mathsf{cPoP.KPG}(\mathsf{attrs}) \xrightarrow{\$} (\mathsf{pk}, \mathsf{sk}, \pi)$: A probabilistic $\underline{k}$ey and $\underline{p}$roof $\underline{g}$eneration algorithm that takes as input attributes attrs $\in \{0,1\}^*$, and outputs a public key pk, secret key sk, and proof string π.

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

```
$\mathsf{Exp}^{\mathsf{UF}}_{\mathsf{PoP},\mathsf{Aux}}(\mathcal{A})$
1 : $(\mathsf{pk}, \mathsf{sk}) \leftarrow\!\!\$\ \mathsf{KG}()$
2 : $(\mathsf{attrs}', \pi') \leftarrow\!\!\$$
        $\mathcal{A}^{\mathsf{PG}(\mathsf{pk},\mathsf{sk},\cdot),\mathsf{Aux}(\mathsf{sk},\cdot)}(\mathsf{pk})$
3 : return $\mathsf{Vf}(\mathsf{pk}, \mathsf{attrs}', \pi')$
        $\wedge$ (attrs$'$ not queried to PG)
```

```
$\mathsf{Exp}^{\mathsf{UF}}_{\mathsf{cPoP},\mathsf{Aux}}(\mathcal{A})$
1 : $\mathsf{attrs} \leftarrow\!\!\$\ \mathcal{A}()$
2 : $(\mathsf{pk}, \mathsf{sk}, \pi) \leftarrow\!\!\$\ \mathsf{KPG}(\mathsf{attrs})$
3 : $(\mathsf{attrs}', \pi') \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{Aux}(\mathsf{sk},\cdot)}(\mathsf{pk}, \pi)$
4 : return $\mathsf{Vf}(\mathsf{pk}, \mathsf{attrs}', \pi')$
        $\wedge$ ($\mathsf{attrs} \neq \mathsf{attrs}'$)
```

Figure 2: Security experiment for proof unforgeability of a key generation and proof of possession scheme PoP = (KG, PG, Vf) (left) and a *combined* key generation and proof of possession scheme cPoP = (KPG, Vf) (right), with respect to auxiliary secret key usage algorithm Aux.

Definition 2 (Security of Combined PoPs). Let cPoP = (KPG, Vf) be a combined key generation and proof of possession scheme. We say that *cPoP* is a *secure cPoP scheme* if it is:

- *Zero-knowledge:* no efficient adversary exists for the security experiment $\mathsf{Exp}^{\mathsf{ZK}}_{\mathsf{PoP}}$ (as shown in Figure 3) as defined in Definition 3.
- *Unforgeable:* no efficient adversary exists for the security experiment $\mathsf{Exp}^{\mathsf{UF}}_{\mathsf{cPoP},\mathsf{Aux}}$ (as shown in Figure 2), for the required auxiliary secret key usage algorithm Aux.
- *Correct:* key pairs output by cPoP.KPG are distributed identically to the 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* Aux, which models any usage of the secret key in a subsequent application:

- $\mathsf{Aux}(\mathsf{sk}, x) \xrightarrow{\$} y$: 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, Aux would correspond to decapsulation; in a PoP for a signing key, Aux would correspond to signature generation.

The unforgeability experiment $\mathsf{Exp}^{\mathsf{UF}}_{\mathsf{PoP},\mathsf{Aux}}$ 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 (attrs', π') 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 $(\mathsf{pk}, \mathsf{sk})$ be an ECC key pair. Let the PoP be (σ, m) where σ is an ECDSA signature on $m = \mathsf{SHA512}(\mathsf{sk})\|\mathsf{attrs}$. Proof verification checks the ECDSA signature on the provided m as usual, without checking whether the first part of the message, 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 EdDSA 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 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 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 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 $\mathsf{PoP} = (\mathsf{KG}, \mathsf{PG}, \mathsf{Vf})$ 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 $(\mathsf{pk}, \mathsf{sk})$ of KG, the output distribution of $\mathsf{Sim}(\mathsf{pk}, \mathsf{attrs})$ is statistically close to the output distribution of

```
$\mathsf{Exp}^{\mathsf{ZK}}_{\mathsf{PoP},\mathsf{Sim}}(\mathcal{A})$
1 : $(\mathsf{pk}, \mathsf{sk}) \leftarrow\!\!\$\ \mathcal{A}()$
2 : $b \leftarrow\!\!\$\ \{0, 1\}$
3 : if $b = 0$ then $b' \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{PG}(\mathsf{pk},\mathsf{sk},\cdot)}()$
4 : if $b = 1$ then $b' \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{Sim}(\mathsf{pk},\cdot)}()$
5 : return $[\![ b = b' ]\!]$
```

```
$\mathsf{Exp}^{\mathsf{ZK}}_{\mathsf{cPoP},\mathsf{Sim}}(\mathcal{A})$
1 : $\mathsf{attrs} \leftarrow\!\!\$\ \mathcal{A}()$
2 : $(\mathsf{pk}, \mathsf{sk}, \pi_0) \leftarrow\!\!\$\ \mathsf{KPG}(\mathsf{attrs})$
3 : $\pi_1 \leftarrow\!\!\$\ \mathsf{Sim}(\mathsf{pk}, \mathsf{attrs})$
4 : $b \leftarrow\!\!\$\ \{0, 1\}$
5 : $b' \leftarrow\!\!\$\ \mathcal{A}(\pi_b)$
6 : return $[\![ b = b' ]\!]$
```

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.

$\mathsf{PG}(\mathsf{pk}, \mathsf{sk}, \mathsf{attrs})$. More formally, PoP is zero-knowledge if no efficient adversary exists for $\mathsf{Exp}^{\mathsf{ZK}}_{\mathsf{PoP},\mathsf{Sim}}$ (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 proof 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 $\Sigma = (\Sigma.\mathsf{KG}, \Sigma.\mathsf{Sign}, \Sigma.\mathsf{Vf})$ be a signature scheme. Define the KGPOP scheme $\mathsf{CSR}[\Sigma] = (\mathsf{KG}, \mathsf{PG}, \mathsf{Vf})$ with helper function CSRfmt as follows:

- $\mathsf{KG}()$: Return $\Sigma.\mathsf{KG}()$.
- $\mathsf{PG}(\mathsf{pk}, \mathsf{sk}, \mathsf{attrs})$: Return $\Sigma.\mathsf{Sign}(\mathsf{sk}, \mathsf{CSRfmt}(\mathsf{pk}, \mathsf{attrs}))$.
- $\mathsf{Vf}(\mathsf{pk}, \mathsf{attrs}, \pi)$: Return $\Sigma.\mathsf{Vf}(\mathsf{pk}, \mathsf{CSRfmt}(\mathsf{pk}, \mathsf{attrs}), \pi)$.
- $\mathsf{CSRfmt}(\mathsf{pk}, \mathsf{attrs}) \to \{0, 1\}^*$: A deterministic *CSR formatting* function that generates the body of a certificate signing request from a public key pk and attributes attrs, for example according to a standards document such as RFC 2986 [60]. We assume CSRfmt is collision-free.

It is straightforward to see that if Σ has existential unforgeability under chosen message attack, then it is hard to forge proofs of possession in $\mathsf{CSR}[\Sigma]$ – after all, PoPs are just signatures.

However, if we want to model that the secret key generated by $\mathsf{CSR}[\Sigma]$ 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 $\mathsf{IsCSR} : \{0,1\}^* \to \{0,1\}$ that checks if a candidate message is a valid formatted CSR body, and refuses to sign if it is:

$$\begin{aligned}\mathsf{Aux}(\mathsf{sk}, x)\text{: If } \mathsf{IsCSR}(x) = 1, \text{ then return } \bot,\\ \text{else return } \Sigma.\mathsf{Sign}(\mathsf{sk}, x).\end{aligned}$$

Assuming that $\mathsf{IsCSR}(x) = 1$ for all $x \leftarrow \mathsf{CSRfmt}(\mathsf{pk}, \mathsf{attrs})$, it is then straightforward to prove, via reduction, that $\mathsf{CSR}[\Sigma]$ has unforgeable proofs of possession (with respect to the given Aux) in the sense of Figure 2, assuming Σ 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 a 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 $\mathbf{S}, \mathbf{E}$ whose entries are small and satisfy $\mathbf{B} = \mathbf{AS} + \mathbf{E} \bmod q$, where $\mathbf{A}$ and $\mathbf{B}$ are public matrices. Let P be the prover and V be the verifier. Let σ be the total number of entries in the secret matrices $\mathbf{S}$ and $\mathbf{E}$, integers sampled from a "small" interval according to a distribution χ. 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 σ 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 $(\mathbf{S}, \mathbf{E})$. Note that the N parties have shares of $(\mathbf{S}, \mathbf{E})$, and P has committed to the shares. P randomly generates a matrix $\mathbf{A}$ and computes the public key $\mathbf{B} = \mathbf{AS} + \mathbf{E}$. Given $\mathbf{A}$, the parties can compute shares of $\mathbf{B}$ using $\mathbf{A}$ and the sharings of $\mathbf{S}$ and $\mathbf{E}$ (no communication is required among the parties as this is a linear operation). Finally, P commits to the shares of $\mathbf{B}$ (the views of the parties), and sends $(\mathbf{A}, \mathbf{B})$ 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 $\mathbf{B}$ 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 recomputes 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 χ. If both checks pass, V outputs accept; otherwise V rejects.

We will show that for an appropriate choice of M (relative to σ and the lattice parameters), V will be assured that the σ values in $(\mathbf{S}, \mathbf{E})$ are from χ, except with probability $2^{-\kappa}$. However, the second challenge has only soundness $1/N$ and necessitates repeating the protocol τ times in parallel. These parallel repetitions are not independent as they all use the same M values sampled from χ (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 $\mathbf{A}$ and $\mathbf{B}$) and witness w (corresponding to secrets $\mathbf{S}$ and $\mathbf{E}$) 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, τ is the number of parallel repetitions, and N is the number of parties used in the MPC protocol. Let σ denote the total number of small integers required for FrodoKEM private values. A *bundle* B_v is a collection of τ sharings of a value v between N parties in the ring $\mathbb{Z}_q$. χ is the distribution used for secrets and errors.

The input to KPG is attrs $\in \{0,1\}^*$, 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 Δx public, and it is added to P_1's share to correct the sharing: $x^{(1)} = x^{(1)} + \Delta x$.

As we will be working with an $N-1$ private MPC protocol with τ parallel repetitions, in repetition e, there is one party that is unopened, we denote their index by ${i_e}^*$, 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 $(\mathbf{S},\mathbf{E})$ from a list of σ values in a canonical way. Given a subset C of $\{1,\ldots,M\}$ of size σ, and values $\{v_1,\ldots,v_M\}$, let $(\mathbf{S},\mathbf{E}) = \mathsf{Arrange}(\{v_1,\ldots,v_M\},C)$ be the two matrices given by populating the entries of $\mathbf{S}$ and $\mathbf{E}$ from the elements v_i for $i \in C$, starting in the top-left of $\mathbf{S}$ and proceeding row-by-row from left to right, and then continuing with $\mathbf{E}$.

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 MPCitH 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 $x^{(i^*)} = x - \sum_{i \neq i^*}^{N} x^{(i)}$, 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 τ sharings of v_k: by revealing v_k, the verifier can recompute the missing shares and check the

```
cPoP.KPG(attrs):
Phase 1: Commit to seeds and bundles
 1: Sample a random salt: salt $\leftarrow\$ \{0,1\}^{2\kappa}$
 2: Sample $v_k$ for $k \in [M]$ from the distribution $\chi$ as in FrodoKEM
 3: for each parallel repetition $e \in [\tau]$ do
 4:     Sample a root seed: $\mathsf{seed}_e \leftarrow\$ \{0,1\}^{\kappa}$
 5:     Derive $\mathsf{seed}_e^{(1)},\ldots,\mathsf{seed}_e^{(N)}$ from $\mathsf{seed}_e$ as leaves of a binary
        tree
 6:     for each party $i \in [N]$ do
 7:         Commit to seed: $\mathsf{com}_e^{(i)} \leftarrow \mathsf{Commit}(\mathsf{salt}, e, i, \mathsf{seed}_e^{(i)})$
 8:         Expand tape: $\mathsf{tape}_e^{(i)} \leftarrow \mathsf{ExpandTape}(\mathsf{salt}, e, i, \mathsf{seed}_e^{(i)})$
 9:         Compute shares: $(b_{1,e}^{(i)},\ldots,b_{M,e}^{(i)}) \leftarrow \mathsf{Sample}(\mathsf{tape}_e^{(i)})$
10:     for each $k \in [M]$ do
11:         Compute the offset: $\Delta b_{k,e} \leftarrow v_k - \sum_{i=1}^{N} b_{k,e}^{(i)} \bmod q$
12:         Update first party's share: $b_{k,e}^{(1)} \leftarrow b_{k,e}^{(1)} + \Delta b_{k,e} \bmod q$
13: Set $m_1 \leftarrow (\mathsf{com}_e^{(i)}, (\Delta b_{k,e})_{k\in[M]})_{i\in[N],e\in[\tau]}$
Phase 2: Challenge a subset of the bundles to audit
 1: Compute challenge hash: $h_1 \leftarrow H_1(\mathsf{salt}, m_1, \mathsf{attrs})$
 2: Compute $C \leftarrow \mathsf{Expand}(\mathsf{salt}, h_1)$ where $C \subset \{1,\ldots,M\}$, $|C| = \sigma$
Phase 3: Generate keypair, simulate MPC, commit to views, and open
the audited bundles
 1: Compute $(\mathbf{S},\mathbf{E}) \leftarrow \mathsf{Arrange}(\{v_1,\ldots,v_M\}, C)$
 2: Choose random $\mathsf{seed}_{\mathbf{A}}$, derive $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ as in FrodoKEM
 3: Compute public key: $\mathbf{B} \leftarrow \mathbf{AS} + \mathbf{E} \bmod q$
 4: for each parallel repetition $e$ do
 5:     for each party $i$ do
 6:         Compute $(\mathbf{S}_e^{(i)}, \mathbf{E}_e^{(i)}) \leftarrow \mathsf{Arrange}((b_{k,e}^{(i)})_{k\in[M]}, C)$
 7:         Compute $\mathbf{B}_e^{(i)} \leftarrow \mathbf{A}\mathbf{S}_e^{(i)} + \mathbf{E}_e^{(i)} \bmod q$
 8: Open audited bundles: set $X_e \leftarrow \{b_{k,e}^{(i)} : k \notin C, i \in [N]\}_{e\in[\tau]}$
 9: Set $m_2 \leftarrow (\mathbf{B}, (\mathbf{B}_e^{(1)},\ldots,\mathbf{B}_e^{(N)})_{e\in[\tau]}, (X_e)_{e\in[\tau]})$
Phase 4: Challenge the MPC simulation
 1: Compute challenge hash: $h_2 \leftarrow H_2(\mathsf{salt}, h_1, m_2)$
 2: Compute $({i_1}^*,\ldots,{i_\tau}^*) \leftarrow \mathsf{Expand}(\mathsf{salt}, h_2)$ where ${i_e}^* \in [N]$
Step 5: Open the MPC views and output the key pair and proof
 1: for each parallel repetition $e$ do
 2:     $\mathsf{seeds}_e \leftarrow$ nodes needed to compute $\{\mathsf{seed}_e^{(i)}, i \neq {i_e}^*\}$
 3: Output $\mathsf{pk} = (\mathbf{B}, \mathsf{seed}_{\mathbf{A}})$, $\mathsf{sk} = (\mathbf{S},\mathbf{E})$ and the proof $\pi$:
      $(h_1, h_2, \mathsf{salt}, (\mathsf{seeds}_e, \mathsf{com}_e^{({i_e}^*)}, (\Delta b_{k,e})_{k\in[M]},)_{e\in[\tau]}, (v_k)_{k\notin C})$
```

Figure 4: 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.

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

Proof Size. The size of the proof π output by Figure 4 is

$$6\kappa + \tau \cdot (2\kappa + \kappa\lceil \log_2(N)\rceil + \ell_q M) + \ell_\chi (M - \sigma) \qquad (1)$$

bits, where the term 6κ accounts for the salt and two challenges (each 2κ bits long), 2κ is the size of the per-party commitment, $\kappa\lceil \log_2(N)\rceil$ is the number of bits required to communicate the seeds of the unopened parties in each repetition, ℓ_q is $\lceil \log_2(q)\rceil$, and ℓ_χ is the number of bits required to represent a "small" value (sampled from the distribution χ). In Section 4.3.1 we explain how

```
cPoP.Vf(pk, $\pi$, attrs):
Parse pk and $\pi$, expand $h_1$ and $h_2$ as defined in Figure 4
Derive $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ from $\mathsf{seed}_\mathbf{A}$
for each parallel repetition $e$ do
    for each party $i \in [N]$, $i \neq i_e{}^*$ do
        Compute $\mathsf{com}'^{(i)}_e \leftarrow \mathsf{Commit}(\mathsf{salt}, e, i, \mathsf{seed}^{(i)}_e)$
        Expand tape: $\mathsf{tape}^{(i)}_e \leftarrow \mathsf{ExpandTape}(\mathsf{salt}, e, i, \mathsf{seed}^{(i)}_e)$
        Compute shares: $(b^{(i)}_{1,e}, \ldots, b^{(i)}_{M,e}) \leftarrow \mathsf{Sample}(\mathsf{tape}^{(i)}_e)$.
        if $i = 1$, for all $k \in [M]$, update $P_1$'s share:
            $b^{(1)}_{k,e} \leftarrow b^{(1)}_{k,e} + \Delta b_{k,e} \bmod q$
        Compute $(\mathbf{S}^{(i)}_e, \mathbf{E}^{(i)}_e) \leftarrow \mathsf{Arrange}((b^{(i)}_{k,e})_{k\in[M]}, C)$
        Compute $\mathbf{B}^{(i)}_e \leftarrow \mathbf{A}\mathbf{S}^{(i)}_e + \mathbf{E}^{(i)}_e \bmod q$
    Compute missing shares $\mathbf{B}^{(i^*)}_j \leftarrow \mathbf{B} - \sum_{i\neq i^*}^{N} \mathbf{B}^{(i)}_j \bmod q$
for $k \notin C$ do                                          ▷ Audit opened bundles
    Abort if $v_k$ does not meet the range criteria given by $\chi$
    for each parallel repetition $e$ do
        Recompute missing party's share:
            $b^{(i_e{}^*)}_{k,e} \leftarrow v_k - \sum_{i\in[N]\setminus i_e{}^*}^{N} b^{(i)}_{k,j} \bmod q$
        if $i_e{}^* = 1$, $b^{(i_e{}^*)}_{k,e} \leftarrow b^{(i_e{}^*)}_{k,e} + \Delta b_{k,e} \bmod q$
        Reconstruct $X_e$ from $b^{(i)}_{k,e}$ as in Figure 4
Compute $m_1$ and $m_2$ be as defined in Figure 4
Compute $h'_1 \leftarrow H_1(\mathsf{salt}, m_1, \mathsf{attrs})$ and $h'_2 \leftarrow H_2(\mathsf{salt}, h'_1, m_2)$
Output accept if $h'_1 = h_1$ and $h'_2 = h_2$, else output reject
```

Figure 5: Combined proof and key generation scheme, verifier operations.

to choose the parameters (N, τ, M, σ), and in Section 6 we give sizes and benchmarks from our implementation.

Note that the FrodoKEM secret key could be compactly represented by $\mathsf{seed}_{\mathbf{SE}}$ (though the FrodoKEM spec does not explicitly represent the key that way). For our cPoP, the secret key cannot directly be derived from a seed alone; one would instead have to derive all $(v_1, \ldots, v_M)$ from a seed $\mathsf{seed}'_{\mathbf{SE}}$, and also store C (to indicate which subset of the v_k to use) and then call Arrange to reconstruct $(\mathbf{S}, \mathbf{E})$.

4.3 Unique Secret Keys and Parameter Selection

We now explain how to choose parameters for our cPoP to provide κ-bit security. There are two places where a prover can cheat: by choosing values for $\mathbf{S}$ and $\mathbf{E}$ that do not have the correct distribution; or in the MPCitH executions. The first part of parameter selection is how to choose the number of bundles to audit in the first step of the protocol, and is new to our work. The second part addresses the choice of the MPCitH parameters, which is similar to many other MPCitH proof protocols in the literature.

First note that with the cut-and-choose mechanism used to ensure that values are small, a malicious prover can create a proof where one of the values is *not small* with high probability, unless M is exponential. For the practicality of our protocol, we want M to be as small as possible. In this section we will show that it is sufficient for the prover to convince the verifier that *most* of the secret values are small, and that this is possible for practical choices of M.

Suppose Bob is honest and has generated $\mathsf{pk}_B = (\mathbf{B}, \mathsf{seed}_\mathbf{A})$, but Alice is malicious and wants to create a PoP for pk_B. Therefore Alice aims to demonstrate knowledge of a pair $(\mathbf{S}', \mathbf{E}')$ satisfying the relation $\mathbf{B} = \mathbf{A}\mathbf{S}' + \mathbf{E}'$. If there are no size restrictions on the entries of $\mathbf{S}'$ or $\mathbf{E}'$, then it is easy to find $\mathbf{S}'$ and $\mathbf{E}'$ that are consistent with $\mathbf{B}$. If Alice's malicious proof verifies, the verifier is convinced that most of the values in $(\mathbf{S}', \mathbf{E}')$ are small. But Lemma 4 shows that, since the honest secret key Bob generated has small $(\mathbf{S}, \mathbf{E})$, then it is *unique* (unconditionally, with overwhelming probability). Therefore, if Alice's proof verifies, we must have $(\mathbf{S}', \mathbf{E}') = (\mathbf{S}, \mathbf{E})$ and this implies that $(\mathbf{S}', \mathbf{E}')$ are in fact all small values.

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

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

$$\overline{n}(1 - 2^{-n})(2\beta + 1)^n \cdot \sum_{t=0}^{D-1} \left(\left\lceil \frac{2\beta + 1}{2^t} \right\rceil / q \right)^n \quad . \tag{2}$$

Furthermore, if the size requirement is relaxed for at most γ entries of $\mathbf{S}'$ and $\mathbf{E}'$, then this probability is at most

$$\overline{n}(1 - 2^{-n})\,(2\beta + 1)^{n-\gamma}\, q^{\gamma - n} \cdot \sum_{t=0}^{D-1} \left\lceil \frac{2\beta + 1}{2^t} \right\rceil^n \tag{3}$$

Proof. Consider the first column of $\mathbf{S}$, the first row of $\mathbf{A}$, the first entry of the first row of $\mathbf{B}$, and the first entry of the first column of $\mathbf{E}$, denoted $\mathbf{s}$, $\mathbf{a}$, b, and e respectively. Let $\mathbf{s}' \in \mathbb{Z}_q^n$ with small coefficients (absolute value less than β). Let $e' = b - \langle \mathbf{a}, \mathbf{s}' \rangle$. Then $\mathbf{s}'$ is a valid first column for a solution matrix $\mathbf{S}$ of the FrodoKEM relation if $|e'|$ is less than or equal to β. We can write $e' = \langle \mathbf{a}, \bar{\mathbf{s}} \rangle + e$, where $\bar{\mathbf{s}} = \mathbf{s} - \mathbf{s}'$. For every $\bar{\mathbf{s}} \neq 0$ (and hence $\mathbf{s}' \neq \mathbf{s}$), we will calculate the probability that $|e'|$ is less than or equal to β, and then bound the probability that there exists a second solution $\mathbf{s}'$ 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 $\bar{\mathbf{s}}$, then $\langle \mathbf{a}, \bar{\mathbf{s}} \rangle$ 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 $\langle \mathbf{a}, \bar{\mathbf{s}} \rangle + e$ is also uniformly distributed. Thus $\Pr[|e'| \leq \beta] = \frac{2\beta+1}{q}$.

If there are no odd entries in $\bar{\mathbf{s}}$, 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, $\langle \mathbf{a}, \bar{\mathbf{s}} \rangle$ is distributed uniformly over even values in $\mathbb{Z}_q$. This gives us that $\Pr[|e'| \leq \beta] \leq 2 \cdot \left\lceil \frac{2\beta+1}{2} \right\rceil / q$. Continuing in this way for $t \in \{0, ..., D-1\}$, we get that for a proportion of possible values for $\bar{\mathbf{s}}$ of size $2^{-tn}(1 - 2^{-n})$, we have $\Pr[|e'| \leq \beta] \leq 2^t \cdot \left\lceil \frac{2\beta+1}{2^t} \right\rceil / q$.

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

$$\sum_{t=0}^{D-1} 2^{-tn}(1-2^{-n})(2\beta+1)^n \left(2^t \cdot \left\lceil \frac{2\beta+1}{2^t} \right\rceil /q\right)^n$$

$$= (1-2^{-n})(2\beta+1)^n \cdot \sum_{t=0}^{D-1} \left(\left\lceil \frac{2\beta+1}{2^t} \right\rceil /q\right)^n \quad .$$

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

Now we consider what happens when we relax the size requirement for some values of $\mathbf{s}'$ and e'. Specifically, if γ_1 entries of the first column of $\mathbf{E}$ do not need to meet the size constraint and γ_2 entries of $\mathbf{s}'$ do not need to meet the size constraint, then the above union bound instead becomes

$$(1-2^{-n})\,(2\beta+1)^{n-\gamma_2}\, q^{\gamma_2} \cdot \sum_{t=0}^{D-1} 2^{-tn} \left(2^t \left\lceil \frac{2\beta+1}{2^t} \right\rceil /q\right)^{n-\gamma_1}$$

$$= (1-2^{-n})\,(2\beta+1)^{n-\gamma_2}\, q^{\gamma_1+\gamma_2-n} \cdot \sum_{t=0}^{D-1} 2^{-t\gamma_1} \left\lceil \frac{2\beta+1}{2^t} \right\rceil^{n-\gamma_1} \quad .$$

Let $\gamma = \gamma_1 + \gamma_2$. The choices of γ_1 and γ_2 maximizing this probability with a given γ will have $\gamma_1 = 0$ and $\gamma_2 = \gamma$. Again considering this result over all $\bar{n}$ columns (taking a coarse upper bound by multiplying by $\bar{n}$, even though not all of the columns can have γ relaxed values simultaneously) of S gives us an upper bound of

$$\bar{n}(1-2^{-n})\,(2\beta+1)^{n-\gamma}\, q^{\gamma-n} \cdot \sum_{t=0}^{D-1} \left\lceil \frac{2\beta+1}{2^t} \right\rceil^{n} \quad ,$$

which is Equation (3). □

4.3.1 Parameter Selection. We explain how we use Lemma 4 with an example, consider parameters for Frodo640: $n = 640$, $\bar{n} = 8$, $\beta = 12$, and $q = 2^{15}$. We choose γ 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 $\gamma = 340$. Hence we can relax up to 340 size constraints in an alternate solution, and the probability of the FrodoKEM solution $\mathbf{S}$ 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 σ unaudited values is less than 2^{-128}. If γ is the number of invalid bundles, then the probability of all γ invalid bundles being in the σ unaudited bundles is

$$\binom{M-\gamma}{M-\sigma} \Big/ \binom{M}{M-\sigma} \quad . \tag{4}$$

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 γ for FrodoKEM and Kyber in Table 2 (in Appendix D).

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

4.4 Security Analysis

We prove two lemmas about 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-interactive 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 $\mathsf{Exp}^{\mathsf{ZK}}_{\mathsf{cPoP}}$ in Figure 3). Let S_{pk} be the set of all transcripts associated with a fixed public key pk:

$$S_{\mathsf{pk}} = \{(\pi, \mathsf{attrs}) : \mathsf{Vf}(\mathsf{pk}, \pi, \mathsf{attrs}) = 1\} \ .$$

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

The simulator. To describe the algorithm Sim we start by assuming it has both $(\mathsf{pk}, \mathsf{sk})$ 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 χ. Sim_1 indexes the bundles so that the values in sk are not audited by C: $\mathsf{sk} = \mathsf{Arrange}((v_k)_{k \in [M]}, C)$. Sim_1 then follows the protocol, but programs H_1 to output C for m_1. 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 $\mathsf{seed}_e^{(i^*)}$ 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 $\mathsf{seed}_e^{(i^*)}$ which happens with negligible probability when the number of hash queries is bounded.

Game 3. Sim_3 replaces $\mathsf{tape}_e^{(i^*)}$ with a uniformly random value. This is indistinguishable assuming ExpandTape is a secure PRG. Since we model ExpandTape as a random oracle that maps $(\mathsf{salt}, e, \mathsf{seed}_e^{(i^*)})$ to $\mathsf{tape}_e^{(i^*)}$, again the intuition is that games 2 and 3 are indistinguishable provided $\mathsf{seed}_e^{(i^*)}$ is not queried.

Game 4. Sim_4 replaces the commitments $\mathsf{com}_e^{(i^*)}$, with random values, i.e., without making a query to Commit. Sim_4 aborts if $\mathcal{A}$ queries x such that $\mathsf{Commit}(x)$ 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 Sim_4 is indistinguishable from Sim_3.

Game 5. Sim_5 no longer uses sk. Sim_5 chooses a random second challenge $(i_e{}^*)_{e\in[\tau]}$ in advance, and programs H_2 to output it for m_2. Sim_5 chooses random $\Delta b_{k,e}$ for $k \notin C$ (the unaudited values). Sim_5 computes shares of parties $P_i \neq P_{i^*}$ and solves for $\mathbf{B}_e^{(i^*)} = \mathbf{B} - \sum_{i\neq i^*} \mathbf{B}_e^{(i)}$. 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 $\mathsf{tape}_e^{(i^*)}$ is uniform as of Game 3), $\Delta b_{k,e}$ was uniformly random. The shares of the opened parties are computed honestly, and $\mathbf{B}_e^{(i^*)}$ is the same in both real and simulated transcripts.

Conclusion. Since Sim_5 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 MPCitH 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 MPCitH-based proofs, first described in [47, §3.1]. The basic idea is that because we've chosen our parameters correctly, if π verifies, then at least one of the τ 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 $\mathsf{com}_e^{(i^*)}$, which contains $\mathsf{seed}_e^{(i^*)}$. Then Ext has the shares of all N parties and can recover $(\mathbf{S}, \mathbf{E})$.

We now define straight-line extractability for cPoPs. Basically this says that for any proof and key pair (pk, π, attrs) output by $\mathcal{A}$ such that $\mathsf{cPoP.Vf}(\mathsf{pk}, \pi, \mathsf{attrs}) = 1$, there exists an extractor algorithm $\mathsf{Ext}_{\mathsf{Commit},H_1,H_2}(\mathsf{pk}, \pi, \mathsf{attrs})$ that recovers sk such that $R(\mathsf{pk}, \mathsf{sk}) = 1$, 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 $\varepsilon_{\mathsf{sle}}$ if there exists an efficient extractor Ext such that for all adversaries $\mathcal{A}$, we have the following:

$$\varepsilon_{\mathsf{sle}} \leq \Pr\left[R(\mathsf{pk}, \mathsf{sk}') \neq 1 \;\middle|\; \begin{array}{l} (\mathsf{pk}, \pi, \mathsf{attrs}) \leftarrow \mathcal{A}^H(1^\kappa) \\ \mathsf{cPoP.Vf}(\mathsf{pk}, \mathsf{attrs}, \pi) \Rightarrow 1 \\ \mathsf{sk}' \leftarrow \mathsf{Ext}_H(\mathsf{pk}, \pi, \mathsf{attrs}) \end{array} \right].$$

```
$H_c(q_c = (\mathsf{salt}, e, i, \mathsf{seed}_e^{(i)}))$
1 : $x \leftarrow\!\!\$ \{0,1\}^{2\kappa}$
2 : if $x \in$ Digests then abort
3 : $x \rightarrow$ Digests
4 : $(q_c, x) \rightarrow Q_c$
5 : return $x$
```

```
$H_2(q_2 = (\mathsf{salt}, h_1, m_2))$
1 : $h_1 \rightarrow$ Digests
2 : $x \leftarrow\!\!\$ \{0,1\}^{2\kappa}$
3 : if $x \in$ Digests then abort
4 : $x \rightarrow$ Digests
5 : $(q_2, x) \rightarrow Q_2$
6 : return $x$
```

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, τ). Let $\mathcal{A}^{\mathsf{Commit},H_1,H_2}$ be an adversary that makes a total of Q random oracle queries. Then* cPoP *is straight-line extractable with*

$$\varepsilon_{\mathsf{sle}} \leq \frac{(\tau N + 1)Q^2}{2^{2\kappa}} + 1/2^\kappa$$

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 π 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 $\mathcal{T}_{\mathsf{sh}}$ and $\mathcal{T}_{\mathsf{key}}$ 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 Δ-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.

```
$H_1(q_1 = ((\mathsf{com}_e^{(i)}, h_e^{(i)}, (\Delta b_{k,e})_{k\in[M]})_{i\in[N],e\in[\tau]}))$
1 :  $x \leftarrow\!\!\$\ \{0,1\}^{2\kappa}$
2 :  if $x \in$ Digests then abort
3 :  $x \to$ Digests
4 :  $(q_1, x) \to Q_1$
5 :  for $e \in [\tau], i \in [N]$ do
6 :    $\mathsf{com}_e^{(i)} \to$ Digests
7 :  $C = \mathsf{Expand}(x)$
     / If the opening of any of the $\mathsf{com}_e^{(i)}$ is known,
     / compute and record the associated shares
8 :  if $(\mathsf{salt}, e, i, \mathsf{seed}_e^{(i)}, \mathsf{com}_e^{(i)}) \in Q_c$ then
9 :    Compute $(b_{k,e}^{(i)}) \leftarrow \mathsf{Expand}(\mathsf{salt}, e, i, \mathsf{seed}_e^{(i)})$
10 :   if $i = 1$ then adjust shares with $\Delta b_{k,e}$
11 :   Compute $(\mathbf{S}_e^{(i)}, \mathbf{E}_e^{(i)}) \leftarrow \mathsf{Arrange}((b_{k,e}^{(i)})_{k\in[M]}, C)$
12 :   Store $(\mathbf{S}_e^{(i)}, \mathbf{E}_e^{(i)})$ in table $\mathcal{T}_{\mathsf{sh}}[q_1, e, i]$
     / If we have all shares, store the corresponding key
13 : foreach $e \in [\tau]$ do
14 :   if $\mathcal{T}_{\mathsf{sh}}[q_1, e, i] \neq \emptyset\ \forall i \in [N]$ then
15 :     $\mathcal{T}_{\mathsf{key}}[q_1, e] \leftarrow \left(\sum \mathbf{S}_e^{(i)}, \sum \mathbf{E}_e^{(i)}\right)$
16 : return $x$
```

Figure 7: Oracle H_1 for extraction algorithm Ext **in proof of Lemma 7.**

- 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 $\mathcal{A}$ completes and outputs $(\mathsf{pk}, \pi, \mathsf{attrs})$, Ext checks the $\mathcal{T}_{\mathsf{key}}$ table for any entry sk_e such that $R(sk_e, \mathsf{pk}) = 1$. If a match is found, Ext outputs sk_e as the secret key corresponding to pk. Otherwise, Ext outputs $\perp$.

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

$$\begin{aligned}\Pr[\mathcal{A}\text{ wins}] =& \Pr[\mathcal{A}\text{ wins} \wedge \text{Ext aborts}]\\ &+ \Pr[\mathcal{A}\text{ wins} \wedge \text{Ext outputs } \perp]\\ \leq& \Pr[\text{Ext aborts}] + \Pr[\mathcal{A}\text{ wins} \mid \text{Ext outputs } \perp] \quad (5)\end{aligned}$$

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

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

$$\begin{aligned}\Pr[\text{Ext aborts}] &\leq (Q_{\mathsf{com}} + Q_1 + Q_2)\cdot\frac{\max|\text{Digests}|}{2^{2\kappa}}\\ &= (Q_{\mathsf{com}} + Q_1 + Q_2)\cdot\frac{Q_{\mathsf{com}} + (\tau N + 1)Q_1 + 2Q_2}{2^{2\kappa}}\\ &\leq \frac{(\tau N + 1)(Q_{\mathsf{com}} + Q_1 + Q_2)^2}{2^{2\kappa}}. \qquad (6)\end{aligned}$$

Now for the main quantity: the probability that $\mathcal{A}$ wins the experiment given that Ext fails, i.e., Ext outputs $\perp$ since no matching sk was found in $\mathcal{T}_{\mathsf{key}}$. It can be checked that Ext always succeeds when $\mathcal{A}$ generates π honestly, therefore for Ext to fail $\mathcal{A}$ must cheat. We divide the two ways $\mathcal{A}$ 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 query $q_1 \in Q_1$, and its corresponding answer $h_1 = C \subset [M]$, $|C| = \sigma$, let $G_1(q_1, h_1)$ be the set of indices $e \in [\tau]$ of "good executions" where both $\mathcal{T}_{\mathsf{key}}[q_1, e] = (\mathbf{S}, \mathbf{E})$ is non-empty and and it holds that all audited values are small, i.e.,

$$\forall\ k \in C, v_k \text{ meets the range criteria given by } \chi \qquad (7)$$

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 $\perp$ but $\mathcal{A}$ wins, there must be more than γ values v_k, $k \notin C$, i.e., not audited by the verifier, that do not satisfy the range condition given by χ. As analyzed in Equation (4) of Section 4.3, this happens with probability not more than

$$p_1 = \binom{M-\gamma}{M-\sigma}\Big/\binom{M}{M-\sigma}\ , \qquad (8)$$

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 $e \in [\tau]$ are in $G_1(q_1, h_1)$ with probability p_1.

Cheating in the second phase. Each second round query $q_2 = (\mathsf{salt}, h_1, m_2)$ that $\mathcal{A}$ 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 $e \in [\tau] \setminus G_1(q_1, h_1)$, either verification failed, in which case $\mathcal{A}$ 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 $h_2 \in [N]^\tau$ is distributed uniformly at random, the probability that this happens for all repetitions e is $p_2 = 1/N^\tau$.

Finally, conditioning on Ext outputting $\perp$

$$\Pr[\mathcal{A}\text{ wins} \mid \text{Ext outputs } \perp] \leq p_1 + p_2 \qquad (9)$$

where p_1 and p_2 are as defined above.

Conclusion. Bringing Equation (5), Equation (6) and Equation (9) together, we obtain the following.

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

Setting $Q = Q_{\text{com}} + Q_1 + Q_2$, and setting parameters such that $p_1 + p_2 \le 2^{-\kappa}$ gives the required bound. □

Note that when sk is not unique for a given pk the extractor of Lemma 7 might not output the same sk used by $\mathcal{A}$. However, when pk is honestly generated, the analysis of Section 4.3 ensures that 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 (as defined in [41, Section 2.2]) if is it hard to recover sk from 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* cPoP *construction of Section 4.2 is a secure cPoP in the random oracle model, when* KEM.Decaps *is the auxiliary secret key usage algorithm.*

Proof. The cPoP 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 $\text{Exp}^{\text{ZK}}_{\text{cPoP}}$ (Figure 3), $\mathcal{A}$'s distinguishing advantage is negligible, bounded by ε_{ZK}. Note that in combined key generation and PoP schemes, $\mathcal{A}$ gets only one proof per key pair.

The correctness property is immediate if, in the cPoP 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 $\mathcal{A}$ be a cPoP attacker in the security experiment $\text{Exp}^{\text{UF}}_{\text{cPoP,Aux}}$ given in Figure 2. We construct $\mathcal{B}$, an algorithm that uses $\mathcal{A}$ as a subroutine in the ROM to invert KEM.KeyGen. Therefore $\mathcal{B}$ must implement the random oracles H_1, H_2, and Commit for $\mathcal{A}$. Algorithm $\mathcal{B}$ is initialized with a public key pk output by KEM.KeyGen, and must output sk such that $R(\text{pk}, \text{sk}) = 1$. Algorithm $\mathcal{A}$ is initialized with nothing, and starts by outputting attrs.

We describe a sequence of games, where $\mathcal{B}$ starts by implementing $\text{Exp}^{\text{UF}}_{\text{cPoP,Aux}}(\mathcal{A})$ as given in Figure 2 (this is Game 0), then in the last game $\mathcal{B}$ inverts KEM.KeyGen if $\mathcal{A}$ succeeds.

In Game 1, $\mathcal{B}$ replaces calls to Aux, which are KEM.Decaps queries, with simulated Decaps queries, using the simulator given by KEM-SIM security of KEM. Thus $\text{G}_1 - \text{G}_0 \le \varepsilon_{\text{KEM-SIM}}$.

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

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

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

Conclusion. We have shown that the success probability for all adversaries $\mathcal{A}$ against the cPoP scheme is bounded by

$$2\varepsilon_{\text{ZK}} + q\varepsilon_{\text{KEM-SIM}} + \varepsilon_{\text{OW-KeyGen}} + \varepsilon_{\text{sle}} ,$$

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

Theorem 8 shows that the KEM does not undermine the security of the cPoP. In Appendix C we consider the other direction and show that adding the cPoP 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 cPoP.

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* 2^n*-th cyclotomic polynomial and* q *is prime. Let* $\mathbf{A} \in R_q^{\ell \times \ell}$ *be selected uniformly at random, and let* $\mathbf{s} \in R_q^{\ell}$ *and* $\mathbf{e} \in R_q^{\ell}$ *be sampled from a centered binomial distribution as in the Kyber specification, so that the absolute values of the coefficients of the polynomials in* $\mathbf{s}$ *and* $\mathbf{e}$ *are at most* β. *Let* $\mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e}$. *Then the probability that there are values* $\mathbf{s}', \mathbf{e}' \in R_q^{\ell}$, *with coefficients whose absolute values are at most* β *such that* $\mathbf{b} = \mathbf{A}\mathbf{s}' + \mathbf{e}'$ *and*

$(\mathbf{s}', \mathbf{e}') \neq (\mathbf{s}, \mathbf{e})$ *is at most*

$$(2\beta+1)^{\ell n}\left(\frac{2\beta+1}{q}\right)^{\ell n} .$$

Furthermore, if the size requirement is relaxed for some γ coefficients of $\mathbf{s}'$ and $\mathbf{e}'$, then the probability is

$$(2\beta+1)^{\ell n}\left(\frac{2\beta+1}{q}\right)^{\ell n-\gamma} .$$

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 $\mathbf{s}' \in R_q^\ell$ with small coefficients and $\mathbf{e}' = \mathbf{b} - \mathbf{A}\mathbf{s}'$. Then $\mathbf{s}'$ is a valid solution only if $\mathbf{e}'$ also has all small coefficients. We can write $\mathbf{e}' = \mathbf{A}\bar{\mathbf{s}} + \mathbf{e}$, where $\bar{\mathbf{s}} = \mathbf{s} - \mathbf{s}'$. For every $\bar{\mathbf{s}} \neq 0$ (and hence $\mathbf{s}' \neq \mathbf{s}$), we will calculate the probability that $\mathbf{e}'$ 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 $\mathbf{A}\bar{\mathbf{s}}$ is a vector of uniformly random polynomials, and so is $\mathbf{e}' = \mathbf{A}\bar{\mathbf{s}} + \mathbf{e}$.

The probability that every coefficient of $\mathbf{e}'$ is small is then $\left(\frac{2\beta+1}{q}\right)^{\ell n}$. Performing a union bound over all $(2\beta+1)^{\ell n}$ possible $\bar{\mathbf{s}}$ gives a probability of $(2\beta+1)^{\ell n}\left(\frac{2\beta+1}{q}\right)^{\ell n}$ that there is another solution $\mathbf{s}'$.

If we relax the size constraint of γ_1 coefficients in $\mathbf{s}'$ and γ_2 coefficients in $\mathbf{e}'$, then the probability is $(2\beta+1)^{\ell n-\gamma_1} q^{k_1}\left(\frac{2\beta+1}{q}\right)^{\ell n-\gamma_2}$. Setting $\gamma = \gamma_1 + \gamma_2$, this is $(2\beta+1)^{\ell n}\left(\frac{2\beta+1}{q}\right)^{\ell n-\gamma}$, 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, σ 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.

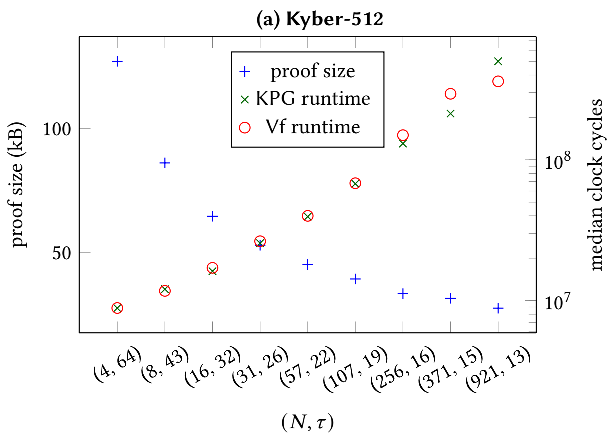
Parameter sizes. Recall that we must choose (N, τ) such that $N^\tau \geq 2^\kappa$ for κ-bit security, and note that there are many valid pairs for each κ. As with all MPCitH proofs with flexible choice of N, we have a speed-size tradeoff: larger N gives shorter proofs that are slower to create and 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).

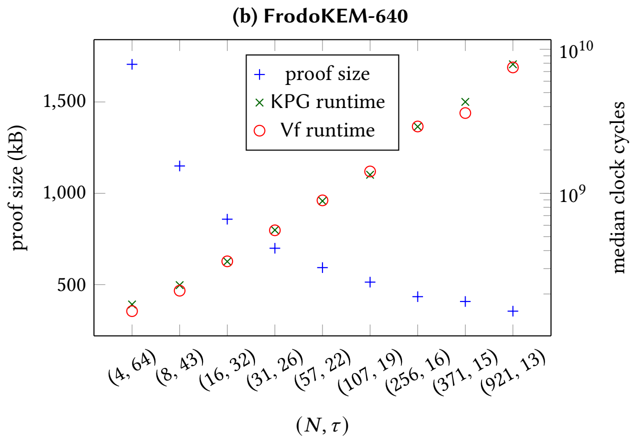
We implemented our construction for all FrodoKEM and Kyber parameter sets by extending their existing public C code. For Kyber, we use the already highly optimized AVX2 code base, whereas the FrodoKEM code base has no hand-crafted AVX2 optimized code, but rather exploits the compiler's ability to auto-vectorize 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 i_e^* whether $\lfloor \log_2 N \rfloor$ or $\lceil \log_2 N \rceil$ nodes from the seed tree must be sent. Thus, the proof size might vary by $\pm\kappa\tau$ 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 $N = 4, \tau = 64$ using gprof 2.38. For Kyber512 approximately 51% of the time is spent hashing (out of which 74% is parallelized)

Figure 8: KPG and Vf performance and proof size for 128-bit security





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 $\{b_{e,k}^{(i_e^*)}\}_{k \notin C}$. 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 $\mathrm{com}_e^{(i)}$ in KPG and $\mathrm{com}_e^{(i)}$ 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 $N\tau$ 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 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 NLnet 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

[1] Carlisle Adams, Stephen Farrell, Tomi Kause, and Tero Mononen. 2005. RFC 4210: Internet X.509 public key infrastructure certificate management protocol (CMP).
[2] Carlisle Adams and Steve Lloyd. 1999. *Understanding public-key infrastructure: concepts, standards, and deployment considerations.* Sams Publishing.
[3] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. 2017. Ligero: Lightweight Sublinear Arguments Without a Trusted Setup. In *ACM CCS 2017*, Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM Press, 2087–2104. https://doi.org/10.1145/3133956.3134104

[4] Frederico Araujo, Kevin W. Hamlen, Sebastian Biedermann, and Stefan Katzenbeisser. 2014. From Patches to Honey-Patches: Lightweight Attacker Misdirection, Deception, and Disinformation. In *ACM CCS 2014*, Gail-Joon Ahn, Moti Yung, and Ninghui Li (Eds.). ACM Press, 942–953. https://doi.org/10.1145/2660267.2660329

[5] N Asokan, Valtteri Niemi, and Pekka Laitinen. 2003. On the usefulness of proof-of-possession. In *Proceedings of the 2nd Annual PKI Research Workshop*. 122–127.

[6] R. Barnes, J. Hoffman-Andrews, D. McCarney, and J. Kasten. 2019. RFC 8555: Automatic Certificate Management Environment (ACME).

[7] Carsten Baum, Cyprien de Saint Guilhem, Daniel Kales, Emmanuela Orsini, Peter Scholl, and Greg Zaverucha. 2021. Banquet: Short and Fast Signatures from AES. In *PKC 2021, Part I (LNCS, Vol. 12710)*, Juan Garay (Ed.). Springer, Heidelberg, 266–297. https://doi.org/10.1007/978-3-030-75245-3_11

[8] Carsten Baum and Ariel Nof. 2020. Concretely-Efficient Zero-Knowledge Arguments for Arithmetic Circuits and Their Application to Lattice-Based Cryptography. In *PKC 2020, Part I (LNCS, Vol. 12110)*, Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas (Eds.). Springer, Heidelberg, 495–526. https://doi.org/10.1007/978-3-030-45374-9_17

[9] Mihir Bellare and Phillip Rogaway. 1993. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In *ACM CCS 93*, Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby (Eds.). ACM Press, 62–73. https://doi.org/10.1145/168588.168596

[10] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. 2019. Aurora: Transparent Succinct Arguments for R1CS. In *EUROCRYPT 2019, Part I (LNCS, Vol. 11476)*, Yuval Ishai and Vincent Rijmen (Eds.). Springer, Heidelberg, 103–128. https://doi.org/10.1007/978-3-030-17653-2_4

[11] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. 2012. High-speed high-security signatures. *Journal of Cryptographic Engineering* 2, 2 (Sept. 2012), 77–89. https://doi.org/10.1007/s13389-012-0027-1

[12] Ward Beullens. 2020. Sigma Protocols for MQ, PKP and SIS, and Fishy Signature Schemes. In *EUROCRYPT 2020, Part III (LNCS, Vol. 12107)*, Anne Canteaut and Yuval Ishai (Eds.). Springer, Heidelberg, 183–211. https://doi.org/10.1007/978-3-030-45727-3_7

[13] Ward Beullens and Cyprien de Saint Guilhem. 2020. LegRoast: Efficient Post-quantum Signatures from the Legendre PRF. In *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020*, Jintai Ding and Jean-Pierre Tillich (Eds.). Springer, Heidelberg, 130–150. https://doi.org/10.1007/978-3-030-44223-1_8

[14] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. 2012. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *ITCS 2012*, Shafi Goldwasser (Ed.). ACM, 326–349. https://doi.org/10.1145/2090236.2090263

[15] Dan Boneh and Victor Shoup. 2020. A Graduate Course in Applied Cryptography. Available at https://crypto.stanford.edu/~dabo/cryptobook/.

[16] Jonathan Bootle, Vadim Lyubashevsky, and Gregor Seiler. 2019. Algebraic Techniques for Short(er) Exact Lattice-Based Zero-Knowledge Proofs. In *CRYPTO 2019, Part I (LNCS, Vol. 11692)*, Alexandra Boldyreva and Daniele Micciancio (Eds.). Springer, Heidelberg, 176–202. https://doi.org/10.1007/978-3-030-26948-7_7

[17] Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. 2018. CRYSTALS–Kyber: A CCA-Secure Module-Lattice-Based KEM. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*. 353–367. https://doi.org/10.1109/EuroSP.2018.00032

[18] Joppe W. Bos, Craig Costello, Léo Ducas, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Ananth Raghunathan, and Douglas Stebila. 2016. Frodo: Take off the Ring! Practical, Quantum-Secure Key Exchange from LWE. In *ACM CCS 2016*, Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi (Eds.). ACM Press, 1006–1018. https://doi.org/10.1145/2976749.2978425

[19] Cecilia Boschini, Jan Camenisch, Max Ovsiankin, and Nicholas Spooner. 2020. Efficient Post-quantum SNARKs for RSIS and RLWE and Their Applications to Privacy. In *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020*, Jintai Ding and Jean-Pierre Tillich (Eds.). Springer, Heidelberg, 247–267. https://doi.org/10.1007/978-3-030-44223-1_14

[20] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. 2018. Bulletproofs: Short Proofs for Confidential Transactions and More. In *2018 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 315–334. https://doi.org/10.1109/SP.2018.00020

[21] Melissa Chase, David Derler, Steven Goldfeder, Jonathan Katz, Vladimir Kolesnikov, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, Xiao Wang, and Greg Zaverucha. 2020. The Picnic Signature Scheme Design Document (version 2.2). https://github.com/microsoft/Picnic/tree/master/spec

[22] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. 2017. Post-Quantum Zero-Knowledge and Signatures from Symmetric-Key Primitives. In *ACM CCS 2017*, Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM Press, 1825–1842. https://doi.org/10.1145/3133956.3133997

[23] Cong Chen, Oussama Danba, Jeffrey Hoffstein, Andreas Hulsing, Joost Rijneveld, John M. Schanck, Peter Schwabe, William Whyte, Zhenfei Zhang, Tsunekazu Saito, Takashi Yamakawa, and Keita Xagawa. 2020. *NTRU*. Technical Report. National Institute of Standards and Technology. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

[24] Vincent Cheval, Véronique Cortier, and Bogdan Warinschi. 2017. Secure Composition of PKIs with Public Key Protocols. In *CSF 2017 Computer Security Foundations Symposium*, Boris Köpf and Steve Chong (Eds.). IEEE Computer Society Press, 144–158. https://doi.org/10.1109/CSF.2017.28

[25] Vincent Cheval, Véronique Cortier, and Bogdan Warinschi. 2017. Secure composition of PKIs with public key protocols. In *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*. IEEE, 144–158.

[26] Santosh Chokhani, W Ford, R Sabett, C Merrill, and S Wu. 2003. RFC 3647: Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework.

[27] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, Frederik Vercauteren, Jose Maria Bermudo Mera, Michiel Van Beirendonck, and Andrea Basso. 2020. *SABER*. Technical Report. National Institute of Standards and Technology. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

[28] Cyprien de Saint Guilhem, Lauren De Meyer, Emmanuela Orsini, and Nigel P. Smart. 2019. BBQ: Using AES in Picnic Signatures. In *SAC 2019 (LNCS, Vol. 11959)*, Kenneth G. Paterson and Douglas Stebila (Eds.). Springer, Heidelberg, 669–692. https://doi.org/10.1007/978-3-030-38471-5_27

[29] Cyprien de Saint Guilhem, Emmanuela Orsini, and Titouan Tanguy. 2021. Limbo: Efficient Zero-knowledge MPCitH-based Arguments. In *ACM CCS 2021*, Giovanni Vigna and Elaine Shi (Eds.). ACM Press, 3022–3036. https://doi.org/10.1145/3460120.3484595

[30] Rafa´el del Pino, Vadim Lyubashevsky, and Gregor Seiler. 2019. Short Discrete Log Proofs for FHE and Ring-LWE Ciphertexts. In *PKC 2019, Part I (LNCS, Vol. 11442)*, Dongdai Lin and Kazue Sako (Eds.). Springer, Heidelberg, 344–373. https://doi.org/10.1007/978-3-030-17253-4_12

[31] Christoph Dobraunig, Daniel Kales, Christian Rechberger, Markus Schofnegger, and Greg Zaverucha. 2021. Shorter Signatures Based on Tailor-Made Minimalist Symmetric-Key Crypto. In *ACM CCS 2022*, Cas Cremers and Elaine Shi (Eds.). ACM Press. Available at https://eprint.iacr.org/2021/692.

[32] Jelle Don, Serge Fehr, and Christian Majenz. 2020. The Measure-and-Reprogram Technique 2.0: Multi-round Fiat-Shamir and More. In *CRYPTO 2020, Part III (LNCS, Vol. 12172)*, Daniele Micciancio and Thomas Ristenpart (Eds.). Springer, Heidelberg, 602–631. https://doi.org/10.1007/978-3-030-56877-1_21

[33] Jelle Don, Serge Fehr, Christian Majenz, and Christian Schaffner. 2019. Security of the Fiat-Shamir Transformation in the Quantum Random-Oracle Model. In *CRYPTO 2019, Part II (LNCS, Vol. 11693)*, Alexandra Boldyreva and Daniele Micciancio (Eds.). Springer, Heidelberg, 356–383. https://doi.org/10.1007/978-3-030-26951-7_13

[34] Jelle Don, Serge Fehr, Christian Majenz, and Christian Schaffner. 2022. Efficient NIZKs and Signatures from Commit-and-Open Protocols in the QROM. Cryptology ePrint Archive, Report 2022/270. https://eprint.iacr.org/2022/270.

[35] Daniel Escudero, Satrajit Ghosh, Marcel Keller, Rahul Rachuri, and Peter Scholl. 2020. Improved Primitives for MPC over Mixed Arithmetic-Binary Circuits. In *CRYPTO 2020, Part II (LNCS, Vol. 12171)*, Daniele Micciancio and Thomas Ristenpart (Eds.). Springer, Heidelberg, 823–852. https://doi.org/10.1007/978-3-030-56880-1_29

[36] Muhammed F. Esgin, Ngoc Khanh Nguyen, and Gregor Seiler. 2020. Practical Exact Proofs from Lattices: New Techniques to Exploit Fully-Splitting Rings. In *ASIACRYPT 2020, Part II (LNCS, Vol. 12492)*, Shiho Moriai and Huaxiong Wang (Eds.). Springer, Heidelberg, 259–288. https://doi.org/10.1007/978-3-030-64834-3_9

[37] Thibauld Feneuil, Antoine Joux, and Matthieu Rivain. 2022. Syndrome Decoding in the Head: Shorter Signatures from Zero-Knowledge Proofs. Cryptology ePrint Archive, Report 2022/188. https://eprint.iacr.org/2022/188.

[38] Amos Fiat and Adi Shamir. 1987. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *CRYPTO'86 (LNCS, Vol. 263)*, Andrew M. Odlyzko (Ed.). Springer, Heidelberg, 186–194. https://doi.org/10.1007/3-540-47721-7_12

[39] Eiichiro Fujisaki and Tatsuaki Okamoto. 1999. Secure Integration of Asymmetric and Symmetric Encryption Schemes. In *CRYPTO'99 (LNCS, Vol. 1666)*, Michael J. Wiener (Ed.). Springer, Heidelberg, 537–554. https://doi.org/10.1007/3-540-48405-1_34

[40] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. 2016. ZKBoo: Faster Zero-Knowledge for Boolean Circuits. In *USENIX Security 2016*, Thorsten Holz and Stefan Savage (Eds.). USENIX Association, 1069–1083.

[41] Oded Goldreich. 2007. *Foundations of Cryptography: Volume 1, Basic Tools*. Cambridge University Press.

[42] Peter Gutmann. 2020. RFC 8894: Simple Certificate Enrolment Protocol.

[43] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. 2017. A Modular Analysis of the Fujisaki-Okamoto Transformation. In *TCC 2017, Part I (LNCS, Vol. 10677)*,

Yael Kalai and Leonid Reyzin (Eds.). Springer, Heidelberg, 341–371. https://doi.org/10.1007/978-3-319-70500-2_12
[44] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. 2009. Zero-Knowledge Proofs from Secure Multiparty Computation. *SIAM J. Comput.* 39, 3 (2009), 1121–1152. https://doi.org/10.1137/080725398
[45] Haodong Jiang, Zhenfeng Zhang, Long Chen, Hong Wang, and Zhi Ma. 2018. IND-CCA-Secure Key Encapsulation Mechanism in the Quantum Random Oracle Model, Revisited. In *CRYPTO 2018, Part III (LNCS, Vol. 10993)*, Hovav Shacham and Alexandra Boldyreva (Eds.). Springer, Heidelberg, 96–125. https://doi.org/10.1007/978-3-319-96878-0_4
[46] Daniel Kales and Greg Zaverucha. 2020. Improving the Performance of the Picnic Signature Scheme. *IACR TCHES* 2020, 4 (2020), 154–188. https://doi.org/10.13154/tches.v2020.i4.154-188 https://tches.iacr.org/index.php/TCHES/article/view/8680.
[47] Jonathan Katz, Vladimir Kolesnikov, and Xiao Wang. 2018. Improved Non-Interactive Zero Knowledge with Applications to Post-Quantum Signatures. In *ACM CCS 2018*, David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang (Eds.). ACM Press, 525–537. https://doi.org/10.1145/3243734.3243805
[48] Akinori Kawachi, Keisuke Tanaka, and Keita Xagawa. 2008. Concurrently Secure Identification Schemes Based on the Worst-Case Hardness of Lattice Problems. In *ASIACRYPT 2008 (LNCS, Vol. 5350)*, Josef Pieprzyk (Ed.). Springer, Heidelberg, 372–389. https://doi.org/10.1007/978-3-540-89255-7_23
[49] Adeline Langlois and Damien Stehlé. 2015. Worst-case to average-case reductions for module lattices. *Designs, Codes and Cryptography* 75 (2015), 565–599. Issue 3. https://doi.org/10.1007/s10623-014-9938-4
[50] Benoît Libert, San Ling, Khoa Nguyen, and Huaxiong Wang. 2018. Lattice-Based Zero-Knowledge Arguments for Integer Relations. In *CRYPTO 2018, Part II (LNCS, Vol. 10992)*, Hovav Shacham and Alexandra Boldyreva (Eds.). Springer, Heidelberg, 700–732. https://doi.org/10.1007/978-3-319-96881-0_24
[51] Richard Lindner and Chris Peikert. 2011. Better Key Sizes (and Attacks) for LWE-Based Encryption. In *CT-RSA 2011 (LNCS, Vol. 6558)*, Aggelos Kiayias (Ed.). Springer, Heidelberg, 319–339. https://doi.org/10.1007/978-3-642-19074-2_21
[52] San Ling, Khoa Nguyen, Damien Stehlé, and Huaxiong Wang. 2013. Improved Zero-Knowledge Proofs of Knowledge for the ISIS Problem, and Applications. In *PKC 2013 (LNCS, Vol. 7778)*, Kaoru Kurosawa and Goichiro Hanaoka (Eds.). Springer, Heidelberg, 107–124. https://doi.org/10.1007/978-3-642-36362-7_8
[53] Vadim Lyubashevsky. 2008. Lattice-Based Identification Schemes Secure Under Active Attacks. In *PKC 2008 (LNCS, Vol. 4939)*, Ronald Cramer (Ed.). Springer, Heidelberg, 162–179. https://doi.org/10.1007/978-3-540-78440-1_10
[54] Vadim Lyubashevsky, Ngoc Khanh Nguyen, and Maxime Plancon. 2022. Efficient Lattice-Based Blind Signatures via Gaussian One-Time Signatures. In *PKC 2022, Part II (LNCS, Vol. 13178)*, Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe (Eds.). Springer, Heidelberg, 498–527. https://doi.org/10.1007/978-3-030-97131-1_17
[55] Vadim Lyubashevsky, Ngoc Khanh Nguyen, and Gregor Seiler. 2020. Practical Lattice-Based Zero-Knowledge Proofs for Integer Relations. In *ACM CCS 2020*, Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna (Eds.). ACM Press, 1051–1070. https://doi.org/10.1145/3372297.3417894
[56] Vadim Lyubashevsky, Ngoc Khanh Nguyen, and Gregor Seiler. 2021. Shorter Lattice-Based Zero-Knowledge Proofs via One-Time Commitments. In *PKC 2021, Part I (LNCS, Vol. 12710)*, Juan Garay (Ed.). Springer, Heidelberg, 215–241. https://doi.org/10.1007/978-3-030-75245-3_9
[57] Daniele Micciancio and Salil P. Vadhan. 2003. Statistical Zero-Knowledge Proofs with Efficient Provers: Lattice Problems and More. In *CRYPTO 2003 (LNCS, Vol. 2729)*, Dan Boneh (Ed.). Springer, Heidelberg, 282–298. https://doi.org/10.1007/978-3-540-45146-4_17
[58] Michael Naehrig, Erdem Alkim, Joppe Bos, Léo Ducas, Karen Easterbrook, Brian LaMacchia, Patrick Longa, Ilya Mironov, Valeria Nikolaenko, Christopher Peikert, Ananth Raghunathan, and Douglas Stebila. 2020. *FrodoKEM*. Technical Report. National Institute of Standards and Technology. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.
[59] NIST. 2015. SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. National Institute of Standards and Technology (NIST), FIPS PUB 202, U.S. Department of Commerce.
[60] Magnus Nystrom and Burt Kaliski. 2000. RFC 2986: PKCS #10: Certification Request Syntax Specification Version 1.7.
[61] Thomas Poppelmann, Erdem Alkim, Roberto Avanzi, Joppe Bos, Léo Ducas, Antonio de la Piedra, Peter Schwabe, Douglas Stebila, Martin R. Albrecht, Emmanuela Orsini, Valery Osheter, Kenneth G. Paterson, Guy Peer, and Nigel P. Smart. 2019. *NewHope*. Technical Report. National Institute of Standards and Technology. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions.
[62] Max Pritikin, Peter Yee, and Dan Harkins. 2013. RFC 7030: Enrollment over secure transport.
[63] Oded Regev. 2003. New lattice based cryptographic constructions. In *35th ACM STOC*. ACM Press, 407–416. https://doi.org/10.1145/780542.780603
[64] Eric Rescorla. 2018. RFC 8446: The Transport Layer Security (TLS) Protocol Version 1.3.
[65] Jim Schaad, Blake Ramsdell, and Sean Turner. 2019. RFC 8551: Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 4.0 Message Specification.
[66] Peter Schwabe, Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, and Damien Stehlé. 2020. *CRYSTALS-KYBER*. Technical Report. National Institute of Standards and Technology. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.
[67] Peter Schwabe, Douglas Stebila, and Thom Wiggers. 2020. Post-Quantum TLS Without Handshake Signatures. In *ACM CCS 2020*, Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna (Eds.). ACM Press, 1461–1480. https://doi.org/10.1145/3372297.3423350
[68] Jacques Stern. 2006. A New Paradigm for Public Key Identification. *IEEE Trans. Inf. Theor.* 42, 6 (sep 2006), 1757–1768. https://doi.org/10.1109/18.556672
[69] Greg Zaverucha, Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, Jonathan Katz, Xiao Wang, Vladmir Kolesnikov, and Daniel Kales. 2020. *Picnic*. Technical Report. National Institute of Standards and Technology. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

A ON THE ROLE OF PROOFS OF POSSESSION

The overarching role of a PKI is to bind a cryptographic public key to an identity – typically some form of name – by way of digital certificates and this is the primary role of the Certification Authority (CA) which issues certificates [2]. It stands to reason that prior to issuing a certificate, the CA must verify that the requester is the entity named in the certificate request, and that they own the requested public key; that is, that they possess the corresponding private key.

As the security of a PKI and the applications that rely on it is complex to define with many subtleties, it is not always obvious what security value proofs of possession at certification time (PoPs) provide [5]. The arguments for the necessity of PoP at certification time generally follow that without it, it becomes possible for an entity to obtain a certificate containing a public key for which they do not possess the corresponding private key. While the holder of such a certificate can not directly use it (as they do not have the corresponding private key), there are several scenarios in which the existence of such certificates undermine the security of PKI.

Strong proof of possession checks at certification time guard against various forms of "sloppy application protocol" implementations [5]. Consider, for example, non-interactive protocols such as S/MIME [65] where the sender must rely on the accuracy of the recipient's encryption certificate as the only guarantee of confidentiality between the sender and the recipient. If the corresponding private key is in fact owned by a different entity than the one named in the certificate, then the sender actually has no guarantees about who may be able to read the encrypted message.

This can be escalated into an attack, described by Cheval et al. [25] and referred to here as a forwarding attack, which works against a naively-implemented sign-then-encrypt protocol where the signer's certificate is carried externally to the signed-then-encrypted content. Imagine Alice signs a message and then encrypts it using the encryption public key in Bob's certificate. Eve intercepts the message but is unable to read it due to the encryption. Assume Eve can get a 'malicious' certificate with her name and Alice's public key Cert("Eve", $\mathrm{pk}_{\mathrm{Alice}}$) – the issuance of which would be prevented by a PoP check by the issuing CA. Depending on the details of the protocol, Eve may be able to supply her malicious certificate with the message in such a way that Bob will decrypt the message and validate the inner signature against Eve's certificate,

```
KEM.KeyGen(1^κ)
1 :  $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{PKE.KeyGen}(1^\kappa)$
2 :  $s \leftarrow\$ \{0,1\}^\kappa$
3 :  $\mathsf{sk} \leftarrow (s, \mathsf{sk}')$
4 :  return $(\mathsf{pk}', \mathsf{sk})$

KEM.Encaps(pk)
1 :  $m \leftarrow\$ \{0,1\}^\kappa$
2 :  $c \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, m; \mathsf{G}(m))$
3 :  $K \leftarrow \mathsf{H}(c \| m)$
4 :  return $(c, K)$

KEM.Decaps(sk, c)
1 :  $m' \leftarrow \mathsf{PKE.Dec}(\mathsf{sk}, c)$
2 :  if $c = \mathsf{PKE.Enc}(\mathsf{pk}, m'; \mathsf{G}(m'))$
3 :      return $\mathsf{H}(c \| m')$
4 :  else
5 :      return $\mathsf{H}(c \| s)$
```

Figure 9: The Fujisaki–Okamoto transform (variant $\mathsf{FO}^{\not\perp}$ 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.

thus assuming that Eve is the originator of the message. This could lead to a number of attacks, including information disclosure if Bob replies directly to Eve and includes the content of Alice's message, for example if Bob's reply contains the original message.

PoP checks at certification time also guard against some forms of implementation errors at enrollment time; for example when a certificate enrollment requires a certificate signing request (CSR) containing the requested name and attribute information, then it remains integrity-protected during transmission from the key-holder to the CA, preventing accidental or malicious modification and providing assurance that the name and attributes remain associated with the intended key pair.

Finally, while not directly a security argument, since PoP checks – particularly CSRs – form the basis of many modern certificate enrollment protocols, extending the existing PoP mechanisms to cover KEM keys would allow for continued use of existing protocols and avoids the engineering effort – and associated security bugs – that would come from designing completely new enrollment mechanisms (both processes and protocols) for KEM keys.

B DECAPSULATION SIMULATABILITY OF THE FUJISAKI–OKAMOTO TRANSFORM

In this section we define the security notion of *decapsulation simulatability* for KEMs, and prove that KEMs constructed with the FO transform satisfy this notion.

As the FO transform takes a PKE as input, we recall it briefly. A *public key encryption* (PKE) scheme is a triple of algorithms PKE = (KeyGen, Enc, Dec). Key generation is used to generate a key pair, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\kappa)$. We can encrypt message a message m and randomness r (chosen from the appropriate spaces) with $c = \mathsf{Enc}(\mathsf{pk}, m; r)$ and decrypt a ciphertext c with $\mathsf{Dec}(\mathsf{sk}, c)$. Decryption may output $\perp$ to indicate that the ciphertext is invalid and decryption has failed. Encryption is probabilistic unless otherwise noted.

B.1 The FO Transform

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 $\mathsf{FO}^{\not\perp}$ 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 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 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 PKE by deriving the random coins used for 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]).

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 Sim, such that Sim takes a public key pk and ciphertext as input and has outputs that are indistinguishable from KEM.Decaps. In Figure 10 we formally define the KEM-SIM security 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 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 $\mathsf{H}_{\mathsf{Sim}}$ as inputs to the adversary.

```
Game KEM-SIM
1 :  $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KEM.KeyGen}(1^\kappa)$
2 :  $b \leftarrow\$ \{0,1\}$
3 :  if $b = 0$ then $b' \leftarrow \mathcal{A}^{\mathsf{Decaps}(\mathsf{sk},\cdot),\mathsf{H}(\cdot)}(\mathsf{pk})$
4 :  if $b = 1$ then $b' \leftarrow \mathcal{A}^{\mathsf{Sim}_{\mathsf{pk}}(\cdot),\mathsf{H}_{\mathsf{Sim}}(\cdot)}(\mathsf{pk})$
5 :  return $[\![b = b']\!]$
```

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

In [43], the authors prove that if PKE is IND-CPA secure, then $\mathsf{KEM} = \mathsf{FO}^{\not\perp}[\mathsf{PKE}]$ 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 PKE is that it is δ-correct. We leave the formal definition to [43], but intuitively when δ is negligible no efficient adversary can find a message m such that $\mathsf{Dec}(sk, \mathsf{Enc}(pk, m)) \neq m$, even when given (sk, pk).

An analogous result in the QROM is given in [43, §4.3.2] and, like the ROM proof, implicitly proves KEM-SIM security. Therefore, KEM-SIM security of FO-based KEMs can also be shown in the QROM (but with worse concrete bounds).

Theorem 10. *Let* PKE *be an IND-CPA secure public key encryption scheme that is δ-correct, and let* KEM = $\mathsf{FO}^{\not\perp}[\mathsf{PKE}]$ *be the* KEM *obtained by applying the Fujisaki–Okamoto transform given in Figure 9. Then* KEM *is KEM-SIM secure in the random oracle model.*

Concretely, for any KEM-SIM and IND-CPA adversaries, we have

$$Adv^{KEM\text{-}SIM}_{\mathsf{KEM}} \leq 2q_{\mathsf{H}}/2^{\kappa} + q_{\mathsf{G}} \cdot \delta + q_{\mathsf{G}} Adv^{OW\text{-}CPA}_{\mathsf{PKE}}$$

where q_{G} and q_{H} are the number of queries to the random oracles G *and* H *(respectively), and κ 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 $\mathsf{FO}^{\not\perp}$ of [43] applied to $\mathsf{PKE}_1 = \mathsf{T}[\mathsf{PKE}, \mathsf{G}]$ (Recall that PKE_1 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_1 is OW-PCA secure. Briefly, this means that PKE_1.Enc 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

```
Pco(c, m) Plaintext checking oracle for PKE_1
1 : if $c = \mathsf{PKE}_1.\mathsf{Enc}(\mathsf{pk}, m)$ then return 1
2 : else return 0
```

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 $\mathcal{B}$ be an attacker in the OW-PCA game of PKE_1; $\mathcal{B}$ implements the random oracle H and the KEM-SIM game for adversary $\mathcal{A}$. We proceed by a series of games, transitioning from $\mathsf{Decaps}(\mathsf{sk}, \cdot)$ in Game 0 to $\mathsf{Sim}_{\mathsf{pk}}(\cdot)$ in Game 3, and quantify $\mathcal{A}$'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 $\mathcal{A}$ outputs 1 in game G_i.

Game G_0. In game G_0, $\mathcal{B}$ implements $\mathsf{H} : \{0,1\}^* \to \{0,1\}^{\ell_{\mathsf{H}}}$ by lazy sampling with the list $\mathcal{L}_{\mathsf{H}}$ (keeping track of previous queries (c, m) and corresponding output K) as in the left side of Figure 11.

```
H(c||m) in G_0 and G_1
1 : if $\exists\ (c, m, K) \in \mathcal{L}_{\mathsf{H}}$ return $K$
2 : $K \leftarrow\!\!\$\ \{0,1\}^{\ell_{\mathsf{H}}}$
3 : Add $(c, m, K)$ to $\mathcal{L}_{\mathsf{H}}$
4 : return $K$

Decaps(c) in G_1
1 : $m' \leftarrow \mathsf{PKE.Dec}(\mathsf{sk}, c)$
2 : if $m' = s$ return $\mathsf{H}'(c)$
3 : if $c = \mathsf{PKE.Enc}(\mathsf{pk}, m'; \mathsf{G}(m'))$
4 :     return $\mathsf{H}(c\|m')$
5 : else
6 :     return $\mathsf{H}'(c)$
```

Figure 11: Oracles used used in games G_0 and G_1 in proof of Theorem 10.

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

Game G_1. In game G_1, we change Decaps so that $\mathcal{B}$ no longer makes use of the secret key component s during Decaps as shown in the right side of Figure 11: we replace $\mathsf{H}(c\|s)$ with $\mathsf{H}'(c)$ where H' is a private random oracle internal to $\mathcal{B}$. $\mathcal{B}$ also returns $\mathsf{H}'(c)$ for $\mathsf{H}(c\|m)$ in the Decaps oracle in the event that $m' = s$. Since s is chosen uniformly from $\{0,1\}^{\kappa}$, $\mathsf{G}_1 - \mathsf{G}_0 \leq q_{\mathsf{H}}/2^{\kappa}$.

Game G_2. In game G_2 we change $\mathcal{B}$ to no longer make use of the secret key component sk', by changing Decaps and H as shown in Figure 12. In G_2 the list $\mathcal{L}_D$ keeps track of (c, K) such that $\mathsf{Decaps}(c) = K$, and either H was queried on (c, m) or H was queried on c.

```
H(c||m) in G_2 and G_3
1 : if $\exists\ (c, m, K) \in \mathcal{L}_{\mathsf{H}}$
2 :     return $K$
3 : $K \leftarrow\!\!\$\ \{0,1\}^{\ell_{\mathsf{H}}}$
4 : $\boxed{\text{if m = s then abort}}$    / Only in $G_2$
5 : if $\mathrm{Pco}(c, m) = 1$ then
6 :     if $\exists\ K' s.t. (c, K') \in \mathcal{L}_D$
7 :         $K \leftarrow K'$
8 :     else
9 :         Add $(c, K)$ to $\mathcal{L}_D$
10 : Add $(c, m, K)$ to $\mathcal{L}_{\mathsf{H}}$
11 : return $K$

Decaps(c) in G_2 and G_3
1 : if $\exists\ (c, K) \in \mathcal{L}_D$
2 :     return $K$
3 : else
4 :     $K \leftarrow\!\!\$\ \{0,1\}^{\ell_{\mathsf{H}}}$
5 :     Add $(c, K)$ to $\mathcal{L}_D$
6 :     return $K$
```

Figure 12: Oracles used used in games G_2 and G_3 in proof of Theorem 10.

We now argue that $\mathsf{G}_2 = \mathsf{G}_1$. Consider a fixed ciphertext c, with $m' = \mathsf{PKE}_1.\mathsf{Dec}(\mathsf{sk}, c)$.

- Case 1: $m' \in \{\perp, s\}$. We argue that such a c cannot be added to $\mathcal{L}_D$ via an H-query. Querying $m' = \perp$ 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 $\mathcal{L}_D$, so a random key K is output, as in G_1.
- Case 2: $m' \notin \{\perp, s\}$. For this case we must show that G_2's implementation of H and Decaps appear consistent to $\mathcal{A}$ (as they were in G_1). There are two sub cases, depending on the order that $\mathcal{A}$ 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. $\mathcal{A}$ starts with the query $\mathsf{H}(c\|m')$, and we have $\mathrm{Pco}(c, m') = 1$ so (c, K) with a random K is added to $\mathcal{L}_D$, defining $\mathsf{Decaps}(c) = K = H(c\|m')$ as in G_1.
  - Now for the reverse order. $\mathcal{A}$ queries $\mathsf{Decaps}(c)$, and there is no entry in $\mathcal{L}_D$, so a random K is output and (c, K) is logged in $\mathcal{L}_D$. When $\mathsf{H}(c\|m')$ is subsequently queried, again $\mathrm{Pco}(c, m') = 1$ and this time c is found in $\mathcal{L}_D$, so that the correct value of K is returned, and we have $\mathsf{H}(c\|m') = \mathsf{Decaps}(c)$.

Since $\mathcal{A}$'s view is identical in games 1 and 2, we have $\mathsf{G}_2 = \mathsf{G}_1$.

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 $\{0,1\}^\kappa$, the probability of an abort is negligible and the change will go unnoticed by $\mathcal{A}$. More precisely, we have $\mathsf{G}_3 - \mathsf{G}_2 = q_\mathsf{H}/2^\kappa$. In Game 3 $\mathcal{B}$ no longer uses sk, and so Sim_pk and H_Sim are given by their G_3 versions in Figure 12.

In the KEM-SIM game, $\mathcal{A}$'s advantage corresponds to $\mathsf{G}_3 - \mathsf{G}_0$. Collecting the probabilities, we have that $\mathcal{A}$'s concrete advantage is bounded by $2q_\mathsf{H}/2^\kappa + \mathsf{Adv}^{\text{OW-PCA}}_{\mathsf{PKE}_1}(\mathcal{B})$. The bound on $\mathsf{Adv}^{\text{OW-PCA}}_{\mathsf{PKE}_1}(\mathcal{B})$ given by [43, Theorem 3.2] is

$$\mathsf{Adv}^{\text{OW-PCA}}_{\mathsf{PKE}_1}(\mathcal{B}) \le (q_\mathsf{G} + q_P)\delta + (q_\mathsf{G} + q_P)\mathsf{Adv}^{\text{OW-CPA}}_{\mathsf{PKE}}$$

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 δ-correct[2] IND-CPA encryption schemes with the $\mathsf{FO}^{\not\perp}$ 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.

[2]For FrodoKEM, δ-correctness is analyzed in the specification [58, §2.2.7], and for Kyber all parameter sets are chosen to ensure $\delta < 2^{140}$, see the specification [66, §1.5].

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 $\mathsf{KEM} = (\mathsf{KeyGen}, \mathsf{Encaps}, \mathsf{Decaps})$ be a key encapsulation mechanism. Let $\mathsf{KEM}^+ = (\mathsf{KeyGen}^+, \mathsf{Encaps}, \mathsf{Decaps})$ 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 KeyGen^+ takes as input the string attrs, and outputs $(\mathsf{pk}, \mathsf{sk}, \pi)$ where $(\mathsf{pk}, \mathsf{sk})$ is the KEM keypair, and π is a proof of possession for sk with respect to pk and the attributes attrs. Note that this requires KeyGen^+ to output $(\mathsf{pk}, \mathsf{sk})$ 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 attrs string required for 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 attrs string required for 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 Commit, 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 $\mathsf{KEM} = (\mathsf{KeyGen}, \mathsf{Encaps}, \mathsf{Decaps})$ 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 $(\mathsf{pk}, \mathsf{sk})$ of KeyGen^+ are identically distributed to those of 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 $\mathcal{B}$. Algorithm $\mathcal{B}$ 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 κ, to ensure with high probability σ secret key entries at most γ of which may not be small, generate M bundles and audit $M - \sigma$ of them.

Security	FrodoKEM			Kyber		
level κ	σ	M	γ	σ	M	γ
128	10240	13233	341	1024	1280	336
192	15616	19485	591	1536	1870	558
256	21504	25986	919	2048	2493	744

Message flows. Algorithm $\mathcal{B}$ is initialized by C with a public key pk output by KEM.KeyGen. We allow $\mathcal{A}$ to output the attributes string attrs, following the IND-CCA$^+$ game. Since the cPoP scheme is ZK (see Definition 3 and Lemma 5), $\mathcal{B}$ uses Sim(pk, attrs) to simulate π, a proof of possession for sk associated with pk. (Sim uses the random oracles to simulate transcripts.) All other messages from $\mathcal{A}$ in the IND-CCA$^+$ game are simply passed through by $\mathcal{B}$ 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 KEM.Encaps and KEM.Decaps and KEM.KeyGen. These are implemented by C, and $\mathcal{B}$ is given oracle access to them. Since KEM$^+$ uses these ROs in exactly the same way as in KEM, when $\mathcal{A}$ makes queries to these to ROs, $\mathcal{B}$ simply proxies them to the corresponding oracles exposed by C. Our cPoP uses four ROs (Commit, ExpandTape, H_1, H_2) during KEM$^+$.KeyGen. These are all simulated by $\mathcal{B}$ and exposed to $\mathcal{A}$; in particular note that Sim depends on these RO to simulate transcripts.

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

The last thing to note is that π cannot be verified without access to the random oracles used by cPoP (Commit, 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 π. Since the simulation is indistinguishable to $\mathcal{A}$ and KEM.Decaps behaves identically in the real and simulated games, $\mathcal{B}$ succeeds in the IND-CCA game when $\mathcal{A}$ 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 $\mathcal{A}$ to construct a KEM-SIM attacker $\mathcal{B}$. The same main arguments apply: replacing the proof π with a simulated proof is indistinguishable, and the simulated π is may not be verified without access to the independent ROs that instantiate the proof system. □

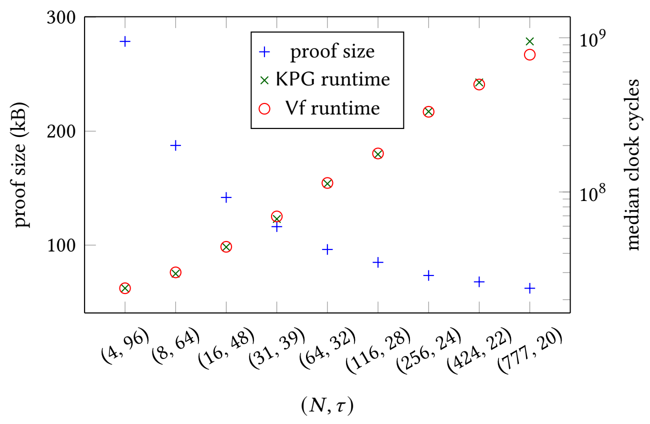
D ADDITIONAL DETAILS OF PARAMETERS, PERFORMANCE AND COMPARISON

Table 2 shows secure choices of M for all security levels of FrodoKEM and Kyber depending on κ 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, $\kappa = 192$

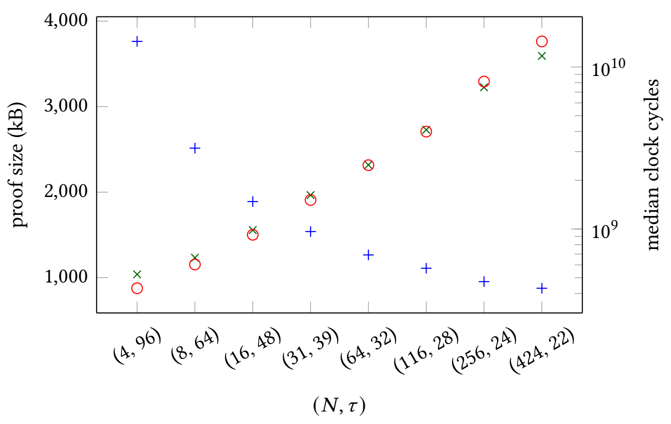
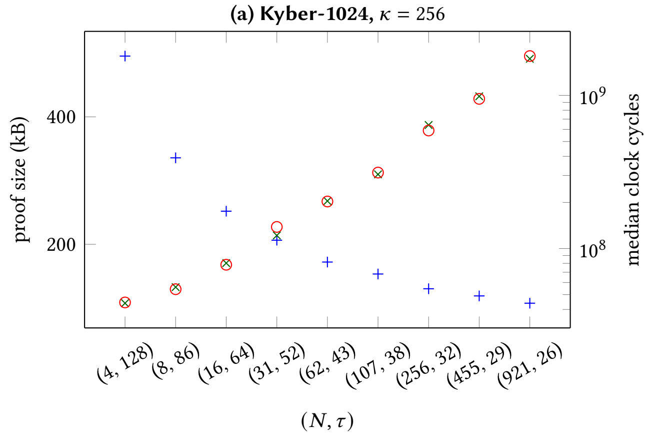


Figure 14: KPG and Vf performance and proof size for the 256-bit security level

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



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

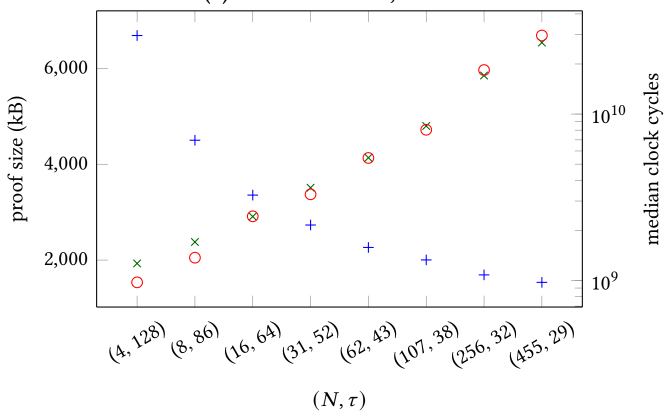


Table 3: KPG and Vf performance and proof size for FrodoKEM and Kyber at 3 security levels

	Level 1 ($\kappa = 128$)			**Level 3** ($\kappa = 192$)			**Level 5** ($\kappa = 256$)		
N	**proof size** kB	**KPG time** $\times 10^6$ cycles	**Vf time** $\times 10^6$ cycles	**proof size** kB	**KPG time** $\times 10^6$ cycles	**Vf time** $\times 10^6$ cycles	**proof size** kB	**KPG time** $\times 10^6$ cycles	**Vf time** $\times 10^6$ cycles
FrodoKEM									
4	1594.1	169.2	152.0	3752.9	524.8	430.7	6671.2	1265.2	973.8
8	1072.3	230.4	210.8	2504.3	664.7	603.9	4485.8	1702.6	1369.6
31	650.0	555.8	553.4	1538.9	1618.9	1509.8	2716.6	3609.5	3292.5
256	401.5	2901.3	2909.8	943.6	7490.0	8129.9	1675.8	17053.9	18393.5
Kyber									
4	127.2	8.8	8.9	278.8	23.7	23.7	495.4	44.0	44.5
8	86.2	12.0	11.7	187.5	29.5	30.1	335.7	56.0	54.3
31	53.0	25.5	26.4	116.2	66.8	69.4	206.5	121.8	138.4
256	33.4	130.5	149.5	73.3	332.4	331.3	130.3	640.9	589.9

Using same τ corresponding to the N as given in Figures 8, 13 and 14.