

PoP! Goes the VOLE: Shorter Proofs of Possession for KEM Certificates

Slim Bettaieb

Technology Innovation Institute (TII)
slim.bettaieb@tii.ae

Mukul Kulkarni

Technology Innovation Institute (TII)
Mukul.kulkarni@tii.ae

Alexandre Augusto Giron

Federal University of Technology - Paraná (UTFPR), Brazil
alexandregiron@utfpr.edu.br

Marco Palumbi

Technology Innovation Institute (TII)
Marco.Palumbi@tii.ae

Abstract

The shift to Post-Quantum Cryptography (PQC, a.k.a. quantum-resistant cryptography) is considered the immediate solution to the threat posed to public key cryptography/PKI by quantum computers. Due to the variety of PQC algorithms (and their characteristics), researchers have proposed different PQC migration strategies. For example, KEMTLS (Schwabe, Stebila, and Wiggers, CCS '20) replaces TLS handshake signing operations by Key Encapsulation Mechanisms (KEMs). However, KEMTLS requires KEM-based certificates (i.e., a certificate with a KEM public key), and the question of issuing KEM certificates on a large scale has received limited attention from the community. We address this research gap by comparing the performance of different Proof of Possession (PoP) approaches for KEMs and assessing their viability for practical adoption. To this end, we first propose and implement a non-interactive PoP for Hamming Quasi-Cyclic (HQC), a code-based post-quantum KEM selected by NIST for standardization. Our PoP is instantiated via the VOLE-in-the-Head (VOLEith) paradigm. Setting, reproducibility, and practicality as design goals, we integrate and benchmark our approach against PoPs for Kyber and FrodoKEM (from Güneysu et al., CCS '22) in the ACME protocol for automatic certificate issuance. Our results show that, in general, Kyber allows faster issuance of KEM certificates but at a cost of losing either compatibility with the protocol (necessitating significant alterations to proposed internet standards) or breaking compatibility/compliance with NIST standards. Notably, our PoP for HQC holds compliance, compatibility, and achieves smaller proof sizes. We also report integration choices and considerations, as well as network traffic analysis stemming from larger (KEM-based) Certificate Signing Requests (CSRs). Our testbed demonstrates the impact of large CSRs by simulating an unstable network. It shows that HQC-PoP's performance is competitive with the fastest options, thanks to its smaller proof sizes.

1 Introduction

The ability to identify entities on the Internet is a key mechanism for the growth of the digital economy. From financial transactions to e-government services, they all depend on the cryptographic techniques for authentication. However, the emergence of quantum computers poses a significant risk to the cryptographic systems widely used today, consequently affecting a wide set of Internet users [39].

Nowadays, the recommended solution to the quantum threat is to start using Post-Quantum Cryptography (PQC) [10]. PQC executes in today's computer architectures protecting users against attackers with access to quantum computers.

On the other hand, researchers have found potential difficulties when migrating traditional cryptography (in use today) to PQC. For instance, the cost of authenticating peers on the Internet when using PQC is significantly higher. Typically, the Web PKI infrastructure requires the transmission of two (or three) public keys and five (or six) digital signatures in the majority of today's Internet connections. Given the larger size of PQC signatures and keys, researchers debated whether this type of migration would effectively degrade the Internet performance (or user experience) [2, 34]. Nevertheless, such studies are needed, since they provide insights on the impacts of PQC before the arrival of quantum computers.

KEMTLS and KEM Certificates. Post-Quantum Key Encapsulation Mechanisms (KEMs), for example, are being used today as an initial step to the PQC migration [15]. A KEM allows a symmetric key to be shared securely between two parties. Additionally, KEMs are versatile for use as an authentication mechanism, as proposed by Schwabe, Stebila, and Wiggers [50]. Their proposal, called KEMTLS, integrates post-quantum KEMs into the TLS protocol [48] as an alternative PQC migration option. KEMTLS performance has been evaluated in different contexts, from high-speed connection networks to resource-constrained devices' communications [27]. A caveat is that KEMTLS peers require a

digital certificate with a KEM public key (i.e., a KEM certificate) to identify the peer in the network. Nevertheless, a KEM certificate can be issued by a Certificate Authority (CA) using digital signatures, thus showing the practicality of the KEMTLS proposal as a PQC migration alternative.

It is worth noting that KEMTLS is not the only use case of KEM-based certificates. Besides KEMTLS, other examples include S/MIME certificates for email addresses and IoT devices identified by serial numbers [45]. KEM-based certificates are part of the CMP protocol [13], and they can be used for encryption protocols such as CMS [30]. Before usage, however, the entity needs a certificate enrollment process in order to obtain the certificate.

Challenge 1: Validating KEM Certificates. Certificate Signing Requests (CSRs) are commonly used to demonstrate possession of the private key corresponding to the public key being certified. In the traditional setting, this is achieved by signing the CSR with the corresponding private key. The CA verifies this signature before issuing the certificate.

This mechanism cannot be directly used for KEM-based certificates. KEMs provide key generation, encapsulation, and decapsulation operations, but do not provide a signing operation that can be used to generate the CSR self-signature. Consequently, KEM certificate enrollment requires an alternative mechanism to demonstrate possession of the secret key corresponding to the KEM public key to be certified.

A Proof-of-Possession (PoP) provides such a mechanism. In our setting, the PoP proves knowledge of the corresponding KEM secret key and binds this proof to the public key and the contents of the CSR, mirroring the guarantee a conventional CSR self-signature provides today. Section 2.3 discusses the scope of this binding, while Section 3 describes our construction and its security properties.

The need for KEM PoP is also recognized in ongoing standardization efforts. An Internet-Draft under discussion in the IETF ACME Working Group proposes the pk-01 challenge for proving possession of KEM secret keys [24]. Proof-of-possession mechanisms for KEM certificates have also been studied academically [28].

Challenge 2: Integrating KEM Certificate Authentication in ACME Protocol. ACME [5] automates the certificate enrollment process. The client first completes a challenge to prove control of an identifier, such as a domain name, and then sends a CSR to the ACME server during the finalization step (Section 2.1). In the conventional certificate enrollment process, the CSR signature additionally proves possession of the private key corresponding to the public key to be certified. Note, however, that RFC 8555 does not specify an equivalent mechanism when the CSR contains a KEM public key. One option is an interactive key-confirmation approach for proving possession of the KEM secret key [24, 25]. Such approaches require additional protocol interactions and modi-

fications to the current ACME enrollment flow (Section 4.3.1). Alternatively, a non-interactive PoP can be included in the KEM-based CSR and verified during finalization, preserving the existing ACME message flow without introducing additional messages or endpoints. Beyond this integration challenge, KEM-based enrollment must also account for the larger PQC objects and the computational cost of KEM PoPs. These affect CSR sizes, processing times, and certificate issuance performance, especially under constrained or unstable network conditions.

1.1 Contributions/ Our results

In order to allow the issuance of KEM certificates, entities need a viable Proof-of-Possession (PoP) technique for KEMs. Moreover, it is important to assess the viability of using PoPs for PQC KEMs and their impacts on protocol performance. To enable validation of KEM-based certificates and support large scale certificate issuance,

- We propose a non-interactive PoP (based on [11]) for the secret key of HQC [23], enabling its integration into typical certificate issuance processes; while maintaining compatibility with proposed PQC and Internets standards. HQC PoP is based on VOLE correlations which have raised significant interest in design of PQ signatures, and therefore can be of independent interest.
- We benchmark four KEM-based certificate authentication approaches: our non-interactive HQC-PoP, the non-interactive Kyber and FrodoKEM PoPs based on Verifiable Generation [28], and an interactive challenge-response approach for ML-KEM [40].
- We provide the first (to the best of our knowledge) practical and reproducible integration study for issuance of KEM-based certificates in the ACME protocol, benchmarking sizes and performance.

While [11] identifies the secret key of HQC as an application of its modelings and estimates the resulting proof sizes, it gives neither the algorithmic description needed for an implementation nor any non-interactive or signature-style security treatment. Beyond the implementation and ACME integration listed above, we give the detailed algorithmic description specialized to HQC (Section 3) and a full sEUF-CMA security proof in the random oracle model for the resulting non-interactive HQC-PoP, binding the proof to the certificate signing request.

Our Techniques. A VOLE (Vector Oblivious Linear Evaluation) correlation of length ℓ over $\mathbb{F}_{2^\kappa}$ are tuples of random values $(\mathbf{u}, \mathbf{v}) \in \mathbb{F}_2^\ell \times \mathbb{F}_{2^\kappa}^\ell$, and $(\mathbf{q}, \Delta) \in \mathbb{F}_{2^\kappa}^\ell \times \mathbb{F}_{2^\kappa}$, such that

$$q_i = u_i \Delta + v_i \quad i \in [\ell].$$

The VOLE correlations are a crucial building block in the design of our proof-of-possession (PoP) scheme.¹ The prover (P) gets the values $(\mathbf{u}, \mathbf{v})$, and the verifier (V) gets $(\Delta, \mathbf{q})$. We will use the VOLE correlations as linearly homomorphic commitments. These commitments serve as an *information theoretically secure* commitment to P’s random value $\mathbf{u}$. Since, the uniform random mask $\mathbf{v}$ is unknown to V, it hides $\mathbf{u}$ perfectly, while a cheating prover must guess Δ in order to open the commitment to some $\mathbf{u}' \neq \mathbf{u}$, assuring the binding property. The linearly homomorphic property follows from the linearity of VOLE correlations. To prove the knowledge of secret key, P commits to its secret key (witness) using its VOLE correlation inputs. The verifier computes a public function of the witness (and other public inputs) that is verified using only its local VOLE correlation inputs $(\mathbf{q}, \Delta)$ in order to validate the proof.

P’s inputs $(u, v) \in \mathbb{F}_2 \times \mathbb{F}_{2^\kappa}$ can be used to describe a *linear* polynomial: $f_u := uX + v$. Similarly, the verifier’s input is $q = f_u(\Delta)$. That is, the prover’s VOLE inputs define a polynomial, where as verifier holds an evaluation point Δ and polynomial evaluation $f_u(\Delta)$ (which is same as q for well-formed VOLE correlation inputs). We denote such a equivalent polynomials held by P as $\llbracket u \rrbracket$, and their corresponding verifier’s input as q_u . In general, it is possible to perform operations like addition, multiplications, and generating linear combinations of polynomials derived from VOLE correlation inputs. For degree- d polynomial with leading coefficient s in a subfield of $\mathbb{F}_{2^\kappa}$, we call the corresponding share held by P as $\llbracket s \rrbracket^{(d)}$.

The core idea of building ZK-POK based on VOLEitH framework can be described as:

1. Prover constructs polynomial constraint (function) f , such that $f(w) = 0$ if and only if w is the secret witness that P claims to know (for example secret key for HQC).
2. P computes $f(w)$ as a leading term of some degree- d polynomial $\llbracket z \rrbracket^{(d)}$ using its shares $\llbracket u \rrbracket$.
3. Verifier checks if the leading term of f_z is equal to zero using its inputs q_u and Δ .

Our Experiments. All of our benchmarks are performed in a practical context for issuing digital certificates. In particular, we incorporate all four KEM-based certificate issuance approaches described above into an ACME [5] client and server implementation. We choose ACME since it has actively contributed to the TLS adoption over the years. In ACME, using a non-interactive PoP approach has the benefit of compatibility, requiring minimal changes to the protocol. For the second approach (the interactive PoP), we changed ACME using an Internet Draft ("I-D") specification [25], i.e., we integrated a challenge-response protocol for ACME. Table 1 compares available KEM-based approaches used in this work.

¹This is an efficient zero-knowledge proof of knowledge (ZK-PoK) protocol and throughout this work we will use the terms PoP and ZK-PoK interchangeably.

All of the approaches in this work target on the issuance of certificates for KEMTLS [50]. We focus on NIST’s security level 1 (for performance reasons and also since this choice seems most relevant in the context of wide adoption). This is reflected by the naming conventions like ML-KEM-512, Kyber512, HQC-1, and FrodoKEM-640, whereas ML-DSA-44 refers to use of PQ signatures in signing messages of ACME protocol (see 2.1 for details). One can achieve better proof sizes at the cost of worse performance [28] (Section 4).

Besides, our approach allows the PoP technique for HQC, to be used in different phases of the certificate lifecycle (e.g., issuance and revocation). This is in contrast to Güneysu et al. [28], as their approach can only be used once for the same key pair because proof generation happens at (by altering) the key generation phase. This is a significant shortcoming of PoPs for lattice based KEMs considered above, as such alterations to key generation break compatibility with NIST standards such as ML-KEM [40].

1.2 Related Work

Both academia and industry are concerned with the complexity of the PQC migration. For example, on the Internet, authentication operations often carry up to 2-3 public keys and 5-6 digital signatures [56]. In a complete PQC migration for that scenario, the computation complexity and increased sizes for PQC objects have to be accounted for to anticipate performance impacts.

Literature has benchmarks for the commonly used TLS protocol [48] with PQC [4, 47, 51, 53]. In TLS, those studies showed evidence of PQC large objects imposing slowdowns to the connections. Unreliable connections, i.e., subject to packet loss are more affected [53]. Such issues are among the reasons that is making the community propose alternative authentication methods, such as KEMTLS [50], Merkle-Tree Certificates (MTC) [9], among others [18].

It is worthy to note that KEMTLS showed some advantages when it was compared to a post-quantum version of TLS 1.3 in the embedded systems scenario [27]. However, KEMTLS requires provisioning methods for KEM-based certificates, which are not supported by standard ACME [5]. Knowing the contributions of ACME for the TLS adoption, analyzing ACME for KEM certificates automatic issuance would give a broader view on the KEMTLS adoption, considering viability, network conditions, computational costs, and sizes.

In order to allow ACME to issue KEM certificates, existing approaches are: adding an interactive PoP in the form of a challenge-response protocol [24, 25], but not compatible to existing enrollment process; or creating a non-interactive PoP for the desired KEM, however not all KEMs have non-interactive PoP available for benchmarking.

The proposal [24] is under active review within the IETF ACME working group [46], indicating growing industry interest in standardizing proof-of-possession for KEM certifi-

Table 1: Approaches for KEM-based certificate issuance integrated into our ACME testbed.

Protocol Type	PoP Type	Certificate algorithm parameter set	Compliance with NIST standards?	RFC 8555 compatible?	Unlimited PoP Usage?
Interactive	PoP by Challenge-response (based on an I-D [25])	ML-KEM-512-ML-DSA-44	✓	✗	✓
Non-interactive	PoP based on VOLE correlations (this work)	HQC-1-ML-DSA-44	✓	✓	✓
Non-interactive	PoP by Verifiable Generation [28]	Kyber512-ML-DSA-44	✗	✓	one-time per key
Non-interactive	PoP by Verifiable Generation [28]	FrodoKEM-640-ML-DSA-44	-	✓	one-time per key

cates. Our work addresses this gap by providing a new non-interactive PoP for HQC, enabling the comparison of different PoPs in a practical setup.

1.3 Text Organization

This text is organized as follows. Section 2 gives the required notations, explains the ACME protocol, defines HQC and VOLE correlations. Section 3 describes our PoP approach for HQC. Section 4 shows our testbed design and benchmark results. Lastly, Section 5 gives the conclusions and final recommendations.

2 Background

2.1 ACME protocol

RFC 8555 [5] specifies the Automatic Certificate Management Environment (ACME) protocol, standardized in March 2019 but in use since 2015 via Let’s Encrypt [36], together contributing significantly to the rise of secure Internet connections. ACME lets a client request a certificate from a CA after automatically proving control of the domain, then enable HTTPS; with Let’s Encrypt and other public CAs, issuance and renewal are free.

ACME is a client-server protocol with four main steps to obtain a certificate for a domain, typically with the ACME client and website software on the same machine and a CA controlling the ACME server. First, an ACME client (e.g., Certbot [17]) connects to the server over HTTPS/TLS [48], encrypting every subsequent HTTP request through that tunnel. Second, the client registers an account, generating a key pair used to sign (client) and verify (server) every HTTP POST request via the JWS standard [33]. Domain control is only proven at the third step, the “challenge”; HTTP-01, DNS-01, and TLS-ALPN-01 are example challenge types [37].

In the last step, “finalization”, the client, having proven domain control, sends an X.509 CSR (embedded in a JWS request) to the ACME server, using either a newly generated key pair or one created outside the protocol; the protocol keeps CSR keys (used by the to-be-certified website) distinct from account keys (used to sign JWS requests).

This paper focuses on the finalization step (Figure 1): the client sends a JWS request with the CSR, the server validates the signature and required authorizations, and the client then polls (via signed JWS requests) until issuance completes, downloading the certificate via a “POST-as-GET” request (an HTTP POST with an empty body).

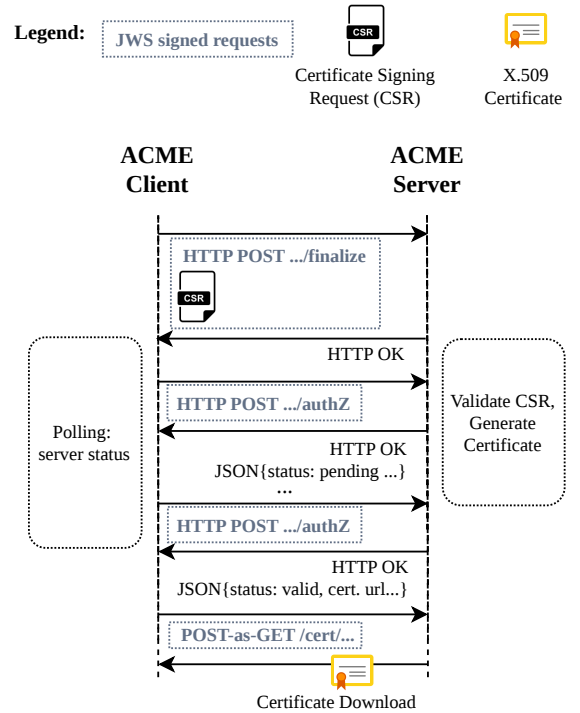


Figure 1: ACME Finalization step.

In the past, issuing certificates to domains on the Internet has been the main focus of ACME. On the other hand, the ACME Working Group at the IETF is active and evolving ACME for a broader range of applications. Examples include other protocols (e.g., OpenID Connect [38]), identifiers, and client certificates with device attestation [38, 55].

2.2 HQC and VOLE correlations

We begin by describing some notation and preliminaries used in the rest of the paper.

Let a positive integer $\lambda \in \mathbb{N}$ be the security parameter. For integers i, j let $[i, j]$ denote the set of integers k such that $i \leq k \leq j$ and let $[n]$ be a shorthand for $[0, n-1]$. For a vector $\mathbf{x}$ its Hamming weight is defined as the number of non-zero coordinates, and we denote the Hamming weight of $\mathbf{x}$ by $w_H(\mathbf{x})$. Let $\mathbb{F}_q$ denote the finite field of q elements where q is the power of a prime. If X is a finite set, let $x \xleftarrow{\$} X$ denote that x is chosen uniformly at random from X . Similarly, let $x \xleftarrow{\$, \theta} X$ denote that x is sampled pseudo-randomly from the set X based on the seed θ . Vectors are denoted by bold lower-case letters and matrices by bold capital letters (e.g., $\mathbf{v} = (v_i)_{i \in [n]} \in \mathbb{F}_q^n$ and $\mathbf{A} = (a_{ij})_{i \in [m], j \in [n]} \in \mathbb{F}_q^{m \times n}$). Let PRG denote a pseudo-random generator. We also denote random oracles instantiated using hash functions, by RO_i^j where i, j indicate the domain separators. For $\mathbf{v} \in \mathbb{F}_2^n$, let $\text{rot}(\mathbf{v})$ denote the circulant matrix generated by $\mathbf{v}$ over $\mathcal{R} = \mathbb{F}_2[X]/(X^n - 1)$, and let $\tilde{h}_{i,j}$ denote the (i, j) entry of $\text{rot}(\mathbf{h})^\top$ (i.e., $\mathbf{h} \cdot \mathbf{y} = \mathbf{y} \cdot \text{rot}(\mathbf{h})^\top$); see Appendix A.1 for the explicit matrix form.

Hamming Quasi-Cyclic (HQC). In what follows, we recall the notation used throughout the paper for the HQC public and secret keys, and state the Quasi-Cyclic Syndrome Decoding (QCSD) problem, on which the security of the HQC public key relies. The HQC key generation algorithm is recalled in Appendix A (Figures 8 and 9), and we refer the reader to [23] for the full specification. Informally, the HQC public key is a pair $(\mathbf{h}, \mathbf{s}) \in \mathbb{F}_2^n \times \mathbb{F}_2^n$, and the corresponding secret key is a pair $(\mathbf{x}, \mathbf{y}) \in \mathbb{F}_2^n \times \mathbb{F}_2^n$ with $w_H(\mathbf{x}) = w_H(\mathbf{y}) = \omega$ satisfying $\mathbf{s} = \mathbf{x} + \mathbf{h} \cdot \mathbf{y}$. The QCSD (search) problem is to recover $(\mathbf{x}, \mathbf{y})$ given $(\mathbf{h}, \mathbf{s})$. The full coding-theoretic definitions (linear codes, parity-check matrices, quasi-cyclic codes) and the formal statement of QCSD are given in Appendix A.1 (Definitions 1-4).

Overview of the VOLEitH framework. Recall that, a VOLE (Vector Oblivious Linear Evaluation) correlation of length ℓ over $\mathbb{F}_{2^\kappa}$ are tuples of random values $(\mathbf{u}, \mathbf{v}) \in \mathbb{F}_2^\ell \times \mathbb{F}_{2^\kappa}^\ell$, and $(\mathbf{q}, \Delta) \in \mathbb{F}_{2^\kappa}^\ell \times \mathbb{F}_{2^\kappa}$, such that

$$q_i = u_i \Delta + v_i \quad i \in [\ell].$$

For simplicity, one can assume that the VOLE correlation inputs $(\mathbf{u}, \mathbf{v}) \in \mathbb{F}_2^\ell \times \mathbb{F}_{2^\kappa}^\ell$, and $(\mathbf{q}, \Delta) \in \mathbb{F}_{2^\kappa}^\ell \times \mathbb{F}_{2^\kappa}$, such that $\mathbf{q} = \mathbf{u}\Delta + \mathbf{v}$ are generated by a trusted party (or an ideal functionality $\mathcal{F}_{\text{VOLE}}$). And it then sends $\mathbf{u}, \mathbf{v}$ to P, and $\mathbf{q}, \Delta$ to V privately. Similar to [1, 7, 8, 49, 59], our (interactive) PoP protocol is secure in the so-called $\mathcal{F}_{\text{VOLE}}$ -hybrid model, which assumes existence of such an ideal functionality. The interactive PoP is converted into a non-interactive PoP using the well established Fiat-Shamir transform [20].

In practice, P generates the VOLE correlations via puncturable PRFs instantiated with GGM trees [26]: it commits

to $N := 2^\kappa$ pseudorandom seeds and later reveals all but one, indexed by the verifier's challenge Δ . This gives a soundness error of $2^{-\kappa}$ per instance, so τ instances are run in parallel with $\tau\kappa \geq \lambda$ to reach the target security level. We refer to Appendix D for the full construction.

2.3 Threat Model

We adopt the threat model from RFC 8555 [5] (Sections 10.1 and 10.2), where an adversary cannot forge ACME account-key signatures or complete a domain-control challenge for an identifier it does not control. Since the CSR is included in the JWS-signed finalization request, this adversary cannot modify it after the request has been signed. We also consider a threat not modeled in RFC 8555: the compromise of a client-side component that constructs the CSR before the ACME request is signed. Such a component may replace a different KEM public key in the CSR, or otherwise modify or reuse the CSR content, without knowing the decapsulation key required to produce a valid proof under the new content. We therefore focus on the key-possession property of the KEM-based CSR.

In a conventional PKCS#10 CSR, the self-signature proves possession of the private key corresponding to the public key in the CSR and binds this proof to the CSR contents. This binding prevents tampering with the CSR's content after it is created, but it does not by itself prevent replay of an unmodified, validly-proved CSR, just as with signature-based CSRs today; providing request freshness is a general limitation of the CSR mechanism, orthogonal to the KEM-vs-signature question, and out of scope for this work [22]. This mechanism is not directly available for KEM keys, since KEMs do not produce signatures. Moreover, the ACME account-key signature authenticates the ACME request, while the domain-control challenge proves control of the identifier; neither provides proof of possession of the decapsulation key corresponding to the KEM public key. For KEMTLS [50], successful substitution with an adversary-controlled KEM key would allow the holder of the corresponding decapsulation key to impersonate the certified domain.

We therefore require proof of knowledge of the decapsulation key bound to both the encapsulation key and the specific CSR. Section 3 constructs such a proof for HQC using $\mu := \text{RO}_1(\text{ek} \parallel \text{CSR})$. Viewing the resulting proof as a signature over the CSR under the corresponding encapsulation key, we show that it achieves strong existential unforgeability under chosen-message attack (EUF-CMA) in the random oracle model (Theorem 3.1). This prevents a valid proof from being transferred or forged for a different CSR under the certified key.

We do not consider compromise of the component that generates both the KEM key pair and its PoP. Such a component could generate an adversary-controlled key pair along with a valid proof, which the PoP mechanism alone cannot prevent.

3 Proof Of Possession for HQC Secret Key

Our goal is to show that we (as a prover P), know a pair of secret vectors $(\mathbf{x}, \mathbf{y}) \in \mathbb{F}_2^n \times \mathbb{F}_2^n$ such that $w_H(\mathbf{x}) = w_H(\mathbf{y}) = \omega$, and $\mathbf{x} + \mathbf{h} \cdot \mathbf{y} - \mathbf{s} = \mathbf{0}$ where $(\mathbf{h}, \mathbf{s})$ is the public key for given HQC instance (the QCSD problem, Definition 4).² In [11], the authors presented following approach for building such a proof of knowledge.

The core idea builds in 3 stages: First, we create an efficient proof of knowledge to show that we know a secret vector $\mathbf{z}$ of length n with hamming weight $w_H(\mathbf{z}) = 1$. Such a vector is also called elementary vector. The next stage uses the fact that any vector $\mathbf{x}$ with $w_H(\mathbf{x}) = \omega$, can be described as a sum of ω elementary vectors. Therefore, in order to prove knowledge of such a vector $\mathbf{x}$, we will first decompose $\mathbf{x}$ into ω elementary vectors $\mathbf{z}_i$ for $i \in [\omega]$, prove that each $\mathbf{z}_i$ is an elementary vector using the proof from the first stage, and then additionally prove that $\mathbf{x}$ is computed by adding all $\mathbf{z}_i$ s. We can follow the same strategy described so far to also prove that we know a secret vector $\mathbf{y}$ such that $w_H(\mathbf{y}) = \omega$. Finally, we prove that the above mentioned $\mathbf{x}, \mathbf{y}$ also satisfy the linear equation $\mathbf{x} + \mathbf{h} \cdot \mathbf{y} - \mathbf{s} = \mathbf{0}$ for HQC public key $(\mathbf{h}, \mathbf{s})$. All of this together suffices as a proof of possession.

Proving Knowledge of Elementary Vectors An elementary vector $\mathbf{z} \in \mathbb{F}_2^n$ has a single non-zero coordinate, at some secret position $\text{pos} \in [n]$ (i.e., $z_{\text{pos}} = 1$ and $z_j = 0$ for $j \neq \text{pos}$). Writing $d = \lceil \log_2(n) \rceil$ and $\mathbf{w} := \mathcal{B}^{\text{pos}} \in \mathbb{F}_2^d$ for the binary decomposition of pos , each coordinate of $\mathbf{z}$ is the degree- d product

$$\forall j \in [n], z_j = \prod_{k=0}^{d-1} (\overline{\mathcal{B}_k^j} \oplus w_k), \quad (1)$$

which evaluates to 1 exactly at $j = \text{pos}$ and to 0 elsewhere. P therefore commits to pos by computing degree- d VOLE correlations $[\mathbf{z}]^{(d)}$ from d random VOLE correlations $([\mathbf{u}_k])_{k \in [d]}$, as shown in P.VOLE-EV (Appendix A), which also outputs the masked secret $\mathbf{t}$ derived from pos . This reveals nothing about pos , since V does not know the masks $(v_k)_{k \in [d]}$. From $\mathbf{t}$, V locally recomputes the matching VOLE correlations $\mathbf{q}_z$ (V.VOLE-EV, Appendix A) and accepts if $[\mathbf{z}]^{(d)}$ evaluated at Δ equals $\mathbf{q}_z$.³

Proving Knowledge of Hamming Weight A vector $\mathbf{z} \in \mathbb{F}_2^n$ with Hamming weight at most ω can be represented as the sum of ω elementary vectors, specified by positions $\text{PosArray} = (\text{pos}_0, \dots, \text{pos}_{\omega-1}) \in [n]^\omega$. Applying (1) to each $\mathbf{w}_i := \mathcal{B}^{\text{pos}_i}$ and summing the resulting elementary vectors, $z_j = \sum_{i=0}^{\omega-1} z_{i,j}$, gives $[\mathbf{z}]^{(d)}$ using $\omega \cdot d$ VOLE correlations, following the

²Note that, the HQC secret keys include encryption key ek , and random string σ however this part is not relevant for proof of possession so we only consider dk_{PKE} as secret key for our PoP.

³We omit this final check in our algorithms since this is deferred to another process in the actual algorithm for optimization reasons, see HQC-PoP.V.

same commit-and-verify pattern as above; see P.VOLE-HW and V.VOLE-HW (Appendix A).

Proving QCSD relation Using P.VOLE-HW, P commits to shares $[\mathbf{x}]^{(d)}$ and $[\mathbf{y}]^{(d)}$ (with masked vectors $\mathbf{t}_x, \mathbf{t}_y$), proving $w_H(\mathbf{x}) = w_H(\mathbf{y}) = \omega$. To additionally check the linear relation $\mathbf{x} + \mathbf{h} \cdot \mathbf{y} = \mathbf{s}$ for the HQC public key $(\mathbf{h}, \mathbf{s})$, we evaluate the equivalent polynomial equation $[\mathbf{x}]^{(d)} + \mathbf{h} \cdot [\mathbf{y}]^{(d)} - \mathbf{s} = \mathbf{0}$, unrolling $\mathbf{h} \cdot [\mathbf{y}]^{(d)}$ via the matrix-vector product $\sum_i \tilde{h}_{i,j} \cdot [\mathbf{y}_i]^{(d)}$ for $(i, j) \in [n] \times [n]$ (see Appendix A.1 for $\tilde{h}_{i,j}$). Since the result must land in the leading coefficient, $\mathbf{s}$ is multiplied by X^d before subtraction.

Putting it all together Convincing V naively would require sending $n (= 17669)$ degree- d ($d = 15$) polynomials f_j with coefficients in $\mathbb{F}_{2^\lambda}$, each with leading coefficient 0 when honestly generated. Since sending so many polynomials is inefficient, P and V instead take a random linear combination $f = \sum_j \alpha_j f_j$, with coefficients α_j provided by V as a challenge (for soundness against malicious provers), reducing the check to whether the single resulting polynomial f has leading coefficient zero. Since f is computed from the secret witness, sending it directly could leak information about that witness, so P masks it with f_{mask} via P.CheckZero (Appendix A). This whole process is described in P.Check-QCSD (Appendix A), which outputs the masked vector $\mathbf{t}$ and masked polynomial $[a]$ sent to V as the proof of possession.

Verification Verification (HQC-PoP.V.) mirrors the prover's steps: where P computes and outputs polynomials, V evaluates the corresponding polynomials at Δ using its local VOLE inputs $(\mathbf{q}, \Delta)$, together with the masked vector $\mathbf{t}$ and masked polynomial $[a]$ received from P. Algorithms V.VOLE-EV, V.VOLE-HW, V.CheckZero, and V.Check-QCSD (Appendix A) follow this approach.

Security The underlying interactive PoK is knowledge sound and zero-knowledge in the $\mathcal{F}_{\text{VOLE}}$ -hybrid model, following identically from the extractable-binding and multi-hiding properties of the VOLE commitments used in [1, 7, 8, 49, 59] (see [1, Section 7]).

Non Interactive HQC-PoP HQC-PoP.P. describes the non-interactive protocol we use. The interactive PoK divides into three rounds between P and V, converted to non-interactive via Fiat-Shamir [20]. Two rounds are problem-independent, ensuring the consistency and opening of P's VOLE correlation commitments; since these computations are generic to all VOLEitH schemes, we omit their details and refer to recent PQ signature specifications [1, 7].

We first hash the encapsulation key with the certificate signing request (CSR), then derive seed and salt from that digest μ and the decapsulation key, binding the proof to the HQC keys with multi-target resistance. (CSR carries the message

to be signed, e.g., domain names, IP addresses, and timestamps.) In the (problem-independent) first round we create $\ell := (2\omega \cdot d) + ((d-1)\lambda) + (\lambda + B)$ VOLE correlation inputs $\llbracket u_{i,k} \rrbracket$ from seed, salt (GGM-tree based, as in [1]), of which $\lambda + B$ ensure consistency for the later proof. The first round challenge ch_1 is derived from the commitments using a random oracle. The auxiliary values $\tilde{\mathbf{u}}$ and h_V are used in the VOLE-consistency checks and, together with ch_1 , form the auxiliary input to the subsequent problem-dependent round.

The second, problem-dependent round proves knowledge of the QCSD solution defined by the HQC instance: we compute $(\mathbf{t}, \llbracket a \rrbracket)$ (and ch_2) via **P.Check-QCSD** with secret input PosArray, the $(2\omega \cdot d) + ((d-1)\lambda)$ round-1 VOLE inputs, and auxiliary input $aux := (ch_1, \tilde{\mathbf{u}}, h_V)$.⁴ The third, problem-independent round computes the decommitment and opening information, updating aux accordingly, again as in [1].

HQC-PoP.P. (CSR, ek, dk)
Input: Public key (expanded) $ek := (\mathbf{h}, \mathbf{s})$, Secret key dk represented as PosArray an array of positions of non-zero entries of the secret vectors $\mathbf{x}, \mathbf{y}$ represented as 2ω positions $(pos_0, \dots, pos_{\omega-1}, pos_{\omega}, \dots, pos_{2\omega-1})$, and CSR to be issued. Output: Proof of possession proof.
▷ Initialization 1. $\mu := RO_1(ek CSR)$ 2. $\text{rnd} \xleftarrow{\\$} \{0, 1\}^{2\lambda}$ 3. $(\text{seed}, \text{salt}) := RO_3(dk \mu \text{rnd}) \in \{0, 1\}^\lambda \times \{0, 1\}^{2\lambda}$ ▷ Create VOLE correlations and commit to them Refer [1, Algorithm 4.34, lines 4-8] or [7, FAEST.Sign lines 6-11] to obtain values $(\mathbf{c}_i)_{i \in [\tau-1]}, ch_1, \tilde{\mathbf{u}}, h_V$. 4. $aux := (ch_1, \tilde{\mathbf{u}}, h_V)$ ▷ Committing to witness and QCSD proof 5. $(\mathbf{t}, \llbracket a \rrbracket, ch_2) :=$ P.Check-QCSD(PosArray, VOLE inputs, aux) ▷ Compute VOLE decommitments and opening for GGM tree (a.k.a VC) as explained in [1, Algorithm 4.34, lines 12-19] to obtain auxiliary values open required for verification, where, $\text{open} := ((\mathbf{c}_i)_{i \in [\tau-1]}, \tilde{\mathbf{u}}, \text{pdecom}, \text{com}_{e, \mathbf{i}^*[e]}, \text{ctr}, ch_3)$ 6. return proof := $(\mathbf{t}, \llbracket a \rrbracket, \text{open}, \text{salt})$

Figure 2: Non Interactive Proof Generation

⁴In the interactive version of **P.Check-QCSD**, ch_2 is provided by V and used to compute $\llbracket a \rrbracket$.

Verification parses open, recomputes the VOLE-consistency checks, runs **V.Check-QCSD**, and accepts if all checks pass; computations outside **V.Check-QCSD** follow [1].

Main Security Theorem Applying the Fiat-Shamir transform to this PoK, we obtain the following guarantee for our non-interactive HQC-PoP.

Theorem 3.1 (EUF-CMA security of the HQC PoP). *Let $\mathcal{A}$ be a polynomial time adversary that breaks the EUF-CMA security of signature scheme derived from non-interactive HQC-PoP.P. Then there exists, another efficient adversary $\mathcal{B}$ that solves the QCSD problem Definition 4 with high probability (same as success probability of adv).*

Proof. The proof is given in appendix Section D. □

4 Implementation and Benchmarks

Our crypto API implementation is based on the following repositories:

- **KEM-NIZKPoP**: this repository (written in C) contains the implementations for PoP based on Verifiable Generation, which modifies Kyber and FrodoKEM Key Generation algorithms [14]. It enables a non-interactive certificate enrollment process for Kyber and FrodoKEM keys in ACME, but with some drawbacks (e.g., it cannot be used for revocation).
- **HQC-PoP (this work)**: this repository (packaged inside our testbed) contains the C implementation of the PoP based on VOLE correlations for HQC. It enables a non-interactive certificate enrollment process for HQC keys in ACME.
- **Liboqs-go** [44]: this repository is a Go binding for PQC algorithms of the Open Quantum Safe project. With liboqs-go, we are able to integrate Kyber (for the interactive approach) but also signature algorithms such as ML-DSA for signing JWS requests.

Although we need KEMs for certificates, ACME depends on signatures on the other parts of the protocol. This paper focuses on the finalization step of the ACME protocol, but for a realistic setup, we use PQC for the account keys, JWS requests, and TLS connection between the client and server. Additionally, our target is KEMTLS, so the KEM-based certificates have KEM public keys and a signature from the issuer. Therefore, PQC signatures are also necessary. For the KEM-based CSRs, we replace the signature using a zero-knowledge proof (for the non-interactive approach) or using random bytes (for the interactive approach). We detail the ACME integration in Section 4.2.

HQC-PoP.V. (CSR, ek, proof)
Input: Public key (expanded) $ek := (\mathbf{h}, \mathbf{s})$, proof of possession $proof := (\mathbf{t}, \llbracket a \rrbracket, open, salt)$, and certificate CSR to be validated. Output: Boolean indicating if the proof proof is accepted or rejected.
▷ Initialization 1. Parse $open := ((c_i)_{i \in [\tau-1]}, \tilde{\mathbf{u}}, pdecom, com_{e, i^*[e]}, ctr, ch_3)$ 2. $\mu := RO_1(ek \parallel CSR)$ ▷ Reconstruct VOLE correlations and check commitments along with consistency of VOLE correlations. Refer [1, Algorithm 4.35, lines 5-25] or [7, FAEST.Verify lines 6-15] to obtain values ch_1, and h_V. 3. $\overline{aux} := (\overline{ch_1}, \tilde{\mathbf{u}}, h_V)$ ▷ Checking QCSD proof 4. $(b, \overline{ch_2}) := V.Check\text{-}QCSD(\mathbf{t}, \llbracket a \rrbracket, \text{VOLE inputs}, \overline{aux})$ 5. $\overline{ch_3} := RO_2^3(\overline{ch_2} \parallel \llbracket a \rrbracket)$ ▷ Compute VOLE decommitments and opening for GGM tree (a.k.a VC) as explained in [1, Algorithm 4.34, lines 12-19] to obtain auxiliary values aux required for verification 6. return $b \wedge (ch_3 = \overline{ch_3})$

Figure 3: Non Interactive Verification

4.1 CPU Cycles

Before the integration into ACME implementations, we benchmark the KEM algorithms to evaluate performance and sizes through different parameters (for the non-interactive approaches). These "standalone" benchmarks were intended to give insights on the best choices for an integration, considering the parameters for the different KEM algorithms. Table 2 shows the CPU cycles count and sizes in bytes for different parameter sets.

One of our goals was experiment reproducibility; thus, we designed a container-based environment (Section 4.2). We tested the algorithms in a C4standard-2 Virtual Machine (VM) from the Google Cloud Provider (GCP) platform. However, due to the GCP's virtualization layer, we were not able to apply CPU frequency controls for this benchmark. Therefore, these results have limited precision. On the other hand, such cloud-based environment are common nowadays, giving practicality to our results.

From Table 2, we can see that HQC allows smaller proofs, and Kyber allows faster computations. We executed AVX2 implementations for all 3 KEMs and for HQC we additionally provide a multi-threaded version. As expected from the

VOLE-in-the-Head (or MPC-in-the-Head) framework, the choice of PoP parameters offers several trade-offs in terms of computational performance and sizes. This is observed as differences in orders of magnitude for different parameter sets used by the PoP, despite the limited precision of this benchmark. Typically, such trade-offs are presented by providing parameter sets that are, (a) "fast" and thus favor performance over sizes, (b) "short" that favor sizes over performance, or (c) that balance the trade-offs. The short regime imposes significant computational costs, especially for FrodoKEM and Kyber, but is also notable for HQC. While in case of HQC, the cost of VOLE generation is dominated significantly by the arithmetic computations involving thousands of polynomials over large finite fields, thus restricting the effectiveness of fast regime. Therefore, we choose the "balance" preset for our benchmarks in ACME protocol, reducing sizes at acceptable performance costs (when compared to the fast preset).

4.2 Integration into ACME

Figure 4 shows the software components used for our integration. For reproducibility, our implementation is self-contained based on Docker containers.⁵ Pebble [32] is a test server for ACME, and LEGO [19] is an ACME client, both written in Go. A first dependency is on "go-jose", an implementation for the standards from the IETF's JSON Object Signing and Encryption working group (JOSE), such as JWS [33]. All required algorithms are integrated into a modified version of the Go standard library ("go-std" for short).

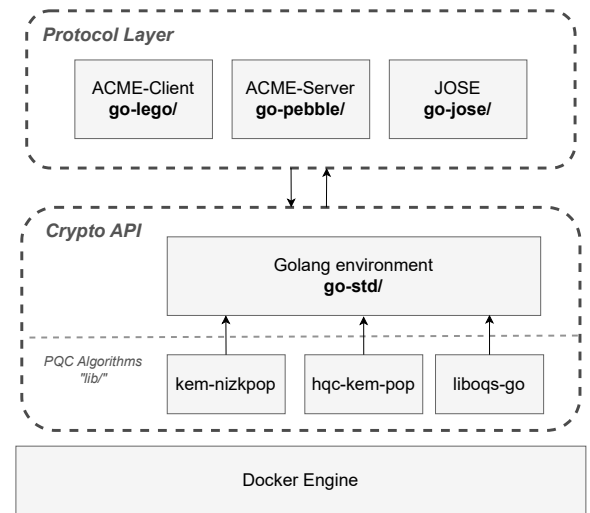


Figure 4: Overview of software components used for our integration.

⁵Available at <https://anonymous.4open.science/r/acme-repository-C437/>

Table 2: Performance and sizes of non-interactive approaches for Proof-of-Possession (PoP) in megacycles (10^6), rounded to two decimal places. We fix 500 iterations for each preset (except "short" preset), where for Kyber and FrodoKEM parameters (N, τ) for "short", "balanced", and "fast" regimes are set as (65536, 8), (31, 26), and (4, 64) respectively. And for HQC, the parameters ($N, \tau, T_{\text{open}}, w$) for "short", "balanced", and "fast" regimes are set as (65536, 8, 105, 4), (256, 16, 95, 4), and (256, 16, 115, 4) respectively. The corresponding results for Security Levels 3 and 5 are given in Appendix C (Table 5).

Non-interactive approach	Proof Generation (MCycles)		Proof Verification (MCycles)		Proof Size (bytes)
	Avg.	Median	Avg.	Median	
HQC-1-PoP-short (single-thread)	3903.84	3891.07	2482.06	2477.60	5928
HQC-1-PoP-short (multi-thread)	2732.74	2721.31	2478.55	2475.14	5928
HQC-1-PoP-balanced (single-thread)	1742.32	1734.16	328.86	328.87	9944
HQC-1-PoP-balanced (multi-thread)	577.99	566.35	328.86	328.85	9944
Kyber512-VG-short	11147.29	11148.45	21936.76	21939.00	20192
Kyber512-VG-balance	26.21	26.17	17.48	17.49	55364
Kyber512-VG-fast	9.03	9.02	5.61	5.60	129504
Frodo640-VG-short	165741.82	165712.55	156660.25	156696.60	222162
Frodo640-VG-balance	282.75	282.52	243.78	243.72	699163
Frodo640-VG-fast	99.30	99.58	74.56	74.55	1706050

For a realistic integration, we also need PQC signatures for go-std to be used in the ACME protocol (i.e., X.509, JWS requests, and TLS). For the experiments, we use ML-DSA-44 for all signatures because ML-DSA is the primary standard from the NIST process [43]. TLS key exchange is based on ML-KEM-768, which has already been used in modern browsers, but with the difference that we are not using it in hybrid mode for simplicity.

Since all PQC algorithms we use are developed in the C language, our integration relies on "go-bindings". These wrappers ease PQC integration into the protocol layer. Then, we create a KEM API (i.e., `KeyPair()`, `Encaps()`, `Decaps()`) in the crypto layer but with an additional `Sign` and `Verify` for the KEM PoP functions in order to ease integration into Golang's X.509 package. This allows us to create KEM-based CSRs and certificates with minimal changes to the go-std library structure. However, our integration faced some challenges and other design decisions, detailed in Section 4.3.

4.3 Challenges and Design Decisions

Normally, ACME developers do not expect large CSRs or certificate generation with costly computation. In this integration, however, the non-interactive approaches can render large CSRs or be significantly slower to produce. In addition, since ACME clients wait for servers to verify CSRs and issue certificates, clients will need to account for the time needed to generate or to verify proofs inside the KEM-based CSR. Therefore, we survey commonly used ACME implementations to find possible problems related to KEM-based CSR processing. Table 4 (Appendix B) shows size limits imposed by common ACME servers and waiting issuance timeouts controlled by ACME clients. In this work, we used the test server Pebble for our integration, which has not imposed any

limits for the JWS requests. It is worthy to note that this is one of the reasons why Pebble is not suitable for production environments, since attackers can try to flood endpoints without size limitations.

For example, in Boulder (v0.20251216.0) [31], the function `parseJWSRequest` checks if the request fits in 50000 bytes; otherwise throws "request body too large" without parsing the JWS request. Since CSRs are encoded inside JWS requests, large CSRs might need reassessment of such JWS size limits. The source code reference shows the variable (`maxRequestSize`) responsible for the JWS limit.

The integration of KEM-based certificates has to consider the cost to verify KEM PoPs to prevent timeout errors. Given the RESTful architecture of the ACME protocol, after sending the CSR to servers, ACME clients have to wait for validation and issuance. RFC 8555 leaves to developers the choice of acceptable timeouts. Thus, each client implementation defines its own policy. For example, LEGO [19] sets the default timeout to 30 seconds, which is, approximately, half the time needed to verify a FrodoKEM640 PoP under the "short" preset parameters. To prevent timeout in this case, LEGO (and others such as Certbot [17]) allows for the user to adjust the timeout value. Alternatively, servers may specify the "Retry-after" HTTP header, where some clients use it as a multiplier factor to extend or to define new intervals to query issuance status. Noteworthy, we noted some "hardcoded" exceptions that can be more difficult to detect or adapt (Table 4, Appendix B).

Given those integration challenges, we decided to use the "balance" preset for the tradeoff of size and performance in our protocol benchmarks. Our testbed is based on Pebble and LEGO, thus without limits for CSRs, and by choosing the "balance" preset, it will not trigger client timeouts. Therefore,

we select the following algorithms for the integration:

1. PoP based on Verifiable Generation [28]: we select Kyber and FrodoKEM based on the AVX2 available implementation. We choose the preset "balance" which means parameters (N, τ) (MPC-in-the-head parties and operations, respectively) as 31,26. This reduces the impact on the CSR size while not far from the best performance (Table 2).
2. PoP based on VOLE correlations (this work): we integrate HQC based on the AVX2 implementation for a fair comparison, in both single-threaded and multi-threaded configurations.
3. We also compare the above Non-interactive approaches to our interactive approach based on an I-D proposal [25]. In this case, we integrate only ML-KEM, i.e., the NIST standard version of Kyber (Section 4.3.1).
4. For completeness, we also integrated and provide experiments for the algorithms in security levels 3 and 5 (Appendix C).

4.3.1 The interactive approach

We have to modify the ACME protocol to add support for the interactive approach. In order to do so, at the finalization step, we first modify the CSR generation. The client generates and embeds a ML-KEM Public key for the CSR and adds 64 random bytes in the CSR's signature field. The random bytes are used so that the server would not break parsing a CSR with an empty signature field. The server will extract the public key from the CSR and, in the interactive approach, will perform an encapsulation as shown in the left part of Figure 5. The client proves control of the private key by decapsulating the ciphertext and sending the shared secret as confirmation to a new endpoint (*/KeyConfirm*) at the server. The server validates the shared secret, responding OK (HTTP 200) or Unauthorized (HTTP 401).

The */KeyConfirm* endpoint is not part of RFC 8555, which means that this approach is not compatible with the proposed standard for ACME. Nevertheless, we benchmark it for completeness (of the available approaches for KEMs).

4.3.2 The Non-interactive approach

Figure 5 also shows the non-interactive approach for ACME.

We slightly modify the finalization step to allow a KEM-based CSR to be sent by the client and validated at the server. No new messages or endpoints are needed for the protocol. Compatibility is preserved by extending only the CSR structure: we define a new OID for each KEM algorithm, we use the KEM public key as the Subject Public Key, and we replace the CSR's signature bitstring by the KEM-PoP proof data. Based on the OID, the server decides which verification

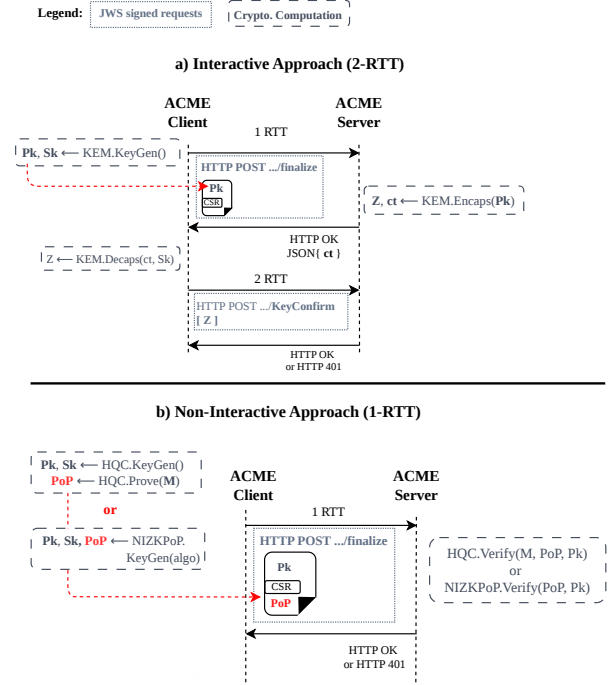


Figure 5: Interactive and Non-interactive approaches for KEM-based Certificate Enrollment. Note: the interactive approach is not compatible with RFC 8555.

routine to invoke, either a signature verifier or the corresponding KEM-PoP verifier.

As in Figure 4, we designed a separation for the crypto API and the application layer (whenever possible). For instance, the server application views the PoP verification step similarly to the X.509 signature verification process. In practice, however, there are some additional decisions such as how to manage single-use PoP proofs for the Kyber and FrodoKEM cases. For that particular case, we follow the guidelines in the original paper [28], which means that the proof is only used for the certificate issuance. For revocation and other management operations, the account keys (based on a signature algorithm) must be used. Noteworthy, our approach for HQC does not suffer from this limitation because it does not happen during key generation. Other aspects of the integration are omitted (for space), but they can be explored in our public repository.

It is important to note that regardless of the KEM-PoP approach, and since our primary target is enabling ACME to issue certificates for KEMTLS servers, we still need PQC signature algorithms. The KEM certificates issued in our experiments are signed using ML-DSA-44 from the liboqs repository. In summary, our integration uses:

- a KEM public key and a zero-knowledge proof to build X.509-like CSRs, identified by a KEM-specific OID,

where the main differences are the public key (from a KEM) and the signature field is replaced by the KEM-PoP in use; and

- a KEM public key and a ML-DSA-44 signature when issuing an X.509-like certificate, where the only difference is that the subject public key is a KEM public key.

We also use ML-DSA as the ACME account keys, i.e., to sign the JWS requests. For TLS, we used ML-DSA and ML-KEM for all connections between the ACME peers. We also specify the same NIST’s security level for all algorithms in our experiments. For instance, in security level 1, we use ML-DSA-44, ML-KEM-512 along with HQC-1, Kyber512, or FrodoKEM640.

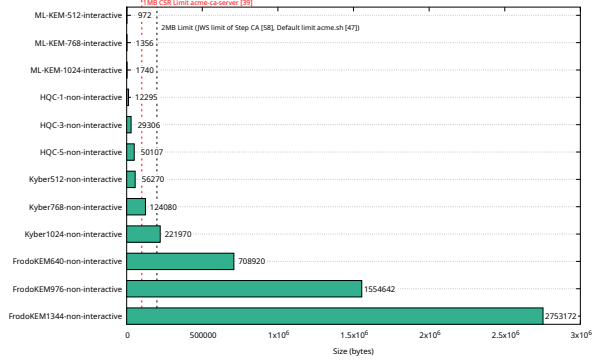
4.4 Object sizes

When executing our protocol implementation, we collect object sizes for KEM CSRs (Fig. 6a) and KEM certificates (Fig. 6b). The sizes are computed from the actual bytes sent in the protocol’s requests. The interactive protocol (i.e., with ML-KEM) does not include any PoP object in the CSR’s signature field (except for 64 random bytes), which results in a small CSR. HQC stands out when comparing the non-interactive approaches, with FrodoKEM being the worst case. On the other hand, Kyber allows a smaller KEM certificate since its public key is smaller than HQC and FrodoKEM. The size of the KEM certificate is also important since the KEM certificate is the output (of ACME) and, after installation, it will be transmitted in every KEMTLS connection. Note that, the certificate size solely depends on the size of KEM encapsulation keys and is unaffected by the choice of PoP parameters. Evaluating KEMTLS connections is beyond the scope of this work, but literature has benchmarks available [27, 50].

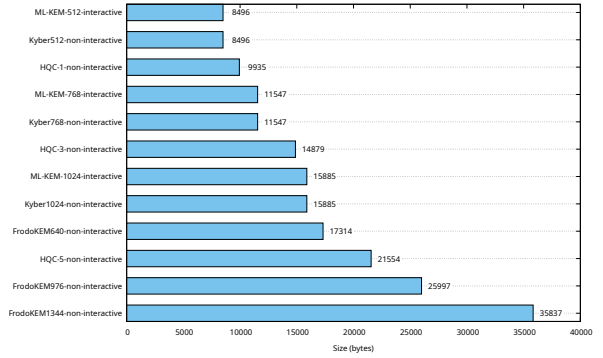
4.5 Protocol performance

We experiment with our implementation under different conditions. We set up two Virtual Machines (VMs) using Google Cloud Provider (GCP) model C4standard-8 (8 vCPUs / Intel Xeon Platinum 8581C @ 2.30 GHz base, 32 GB RAM). The client (VM 1) executes the ACME client, and it is located at region us-central1-a. The ACME server’s (VM 2) location is northamerica-northeast2-c and has 23.172 (min) to 23.861 ms (max) of Round Trip Time (RTT) to VM 1, measured using `ping -c 100 <hostIP>`.

Figures 7a to 7c show the performance results for the algorithms in Security Level 1 (other levels are in Appendix C). We execute the protocol for issuing 100 certificates (one at a time). For HQC multi-threaded version, we used 8 threads for all experiments to better explore our VM setup. Then, we compute percentiles over the issuance times (the Y-axis) for each algorithm tested and for each experiment. Given the variety of ACME challenges being used to prove control of a



(a) Sizes for KEM-based Certificate Signing Requests (CSRs). We include example limits from existing ACME implementations (Appendix B discusses them).



(b) Sizes of certificates issued by the ACME server

Figure 6: Sizes (in bytes) of KEM-based CSR (a) and certificates (b) obtained in the experiments, for all security levels. The certificate has an ML-DSA-44 signature, and the size includes issuer certificate, also using ML-DSA-44 for both the issuer’s certificate public key and signature.

domain and also to measure the impacts of our approach under different network conditions, we designed three experiments, detailed below.

Experiment 1. (The "simplified" scenario): we execute the protocol without the ACME challenge, i.e., without verification about control of the domain. This experiment gives us results independently of the ACME challenge (e.g., HTTP-01, DNS-01, and TLS-ALPN-01), so we can focus the analysis on the algorithm computations.

Experiment 1 shows a clear difference in the computations between algorithms. Overall, we observe almost no variation between the percentiles in Experiment 1, with Kyber using the interactive PoP approach being the fastest to issue certificates. However, the timings for Kyber in the non-interactive version were close, meaning that the choice of parameters did not harm Kyber’s performance significantly.

Experiment 2. ("Unstable network" scenario): we reproduced

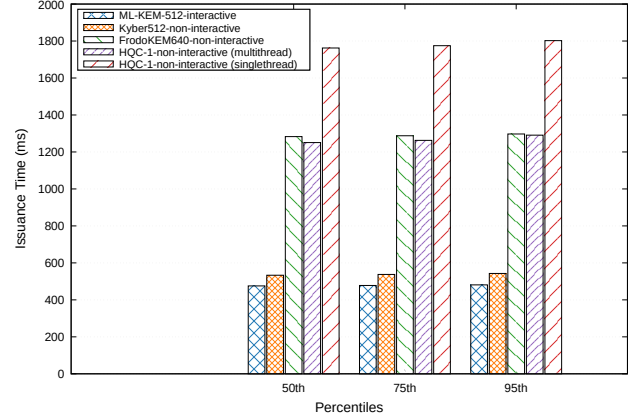
experiment 1 (but with ACME challenge enabled) but in a simulated unstable network. We use the Netem simulation framework to inject network parameters in the host's connections [29]. For insights on selecting concrete network parameters, we looked into the measurements from Wang et al. [54] and Xu et al. [58]. Wang et al. provided measurements focusing on wireless but also compared them with wired connections, in which their worst-case measurement showed latencies of 400 ms (in the 99th percentile). Xu et al. [58] measurements showed packet loss rates below 5%. On the other hand, packet retransmission rates, which include other causes than packet loss, are from 8 to 12% (4G) and 8% on average for 5G. From these measurements we set up for our Experiment 2 a "worst-case" unstable network using:

- 400ms of latency, based on Wang et al. [54] tail latency (200ms for each host); and
- 8% packet loss probability (4% for each host). We did set the loss above Xu et al. [58] for a worst-case and keeping the loss parameter near the real-world measurements.

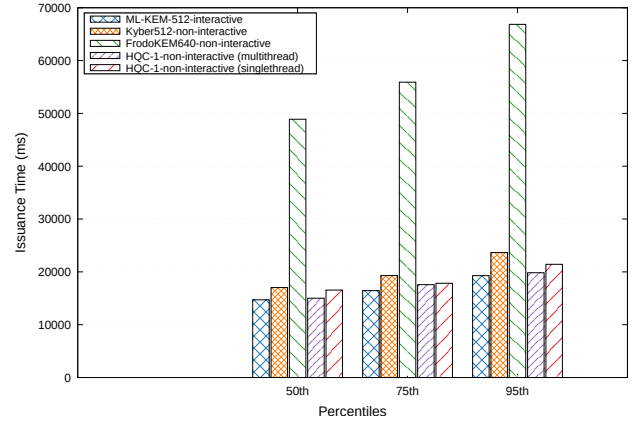
In contrast to Experiment 1, when considering unstable networks, slower computations are "concealed" by the network performance, in this case suffering latency and loss probabilities. All algorithms are affected, as the X-axis (in seconds) shows much slower certificate issuance times when compared to Experiment 1. In such an unstable scenario, sizes matter and that is why FrodoKEM is greatly affected. Having the largest CSR (Fig. 6a), packet loss happens more frequently, triggering retransmissions. Notably, HQC timings are now competitive because they are close to the fastest options (i.e., Kyber). This change happens due to the smaller CSRs when using HQC, easing transmission in a lossy environment. For a detailed network analysis of such events, please refer to Section 4.6.

Experiment 3. Full issuance protocol with HTTP-01 validation enabled. HTTP-01 is the most popular challenge used in practice [37]. We executed the protocol in a stable but geographically distributed network. This is our "realistic setting" for certificate issuance, assuming that the most ACME clients and servers execute in data centers connected in stable networks.

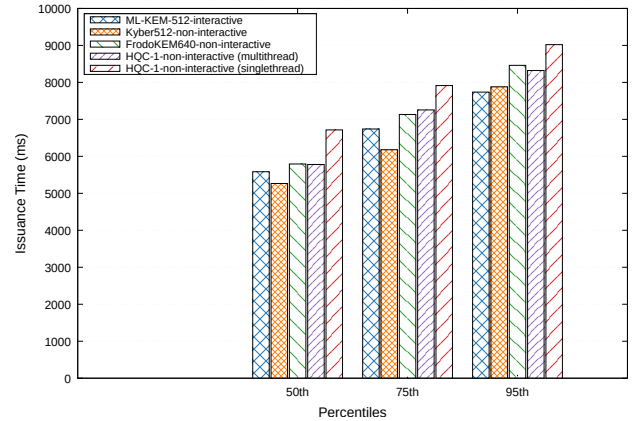
The results of the low-latency scenario (experiment 3) show a significant overlap in the timings between all algorithms tested. The time for domain validation (in this case by the HTTP-01 challenge) can be variable, and this "conceals" the differences between the cryptographic computations. For example, the interactive approach did not show much advantage over the non-interactive, as their timings are quite similar. Considering the issuance interval between 4s to 7.5s, most of HQC issuance timings sit intertwined with the other algorithm timings. Additionally, there is a significant overlap between the timings for Kyber and FrodoKEM. Only part of the HQC



(a) Experiment 1 (skip HTTP-01 validation)



(b) Experiment 2 (unstable network)



(c) Experiment 3 (realistic scenario, HTTP-01 enabled)

Figure 7: Certificate issuance timings for each experiment.

timings are non-overlapping (e.g., from 9.5s to ~11s). Depending on the domain validation approach, Experiment 3 results show that there is a significant chance of all algorithms resulting in a similar certificate issuance time, demonstrating their practical viability. Appendix C provides more statistics and additional experiments (e.g., for HQC-short and fast variants).

4.6 Network Traffic Analysis

To understand the impact of KEMs’ large CSRs, we captured client-server communication during Experiments 2 and 3, and compared network events for Experiment 2 (unstable network) via Table 3, using packet-loss indicators [21]: retransmissions, out-of-order packets, lost segments, congestion window full, and duplicate ACKs; all packet captures are available in the repository (Appendix C).

Table 3: Count of network events observed in Experiment 2 (unstable network) for each algorithm (except the multi-threaded version of HQC, since their sizes are the same). All approaches other than ML-KEM-512 use the non-interactive PoP (single-threaded for HQC).

TCP Event	ML-KEM-512 (Interactive)	HQC-1 (single-thread)	Kyber512	FrodoKEM640
Retransmission	273	281	268	455
Fast Retransmission	1	0	0	25
Spurious Retransmission	1	6	4	4
Out-Of-Order	49	84	261	2658
Lost Segment	158	179	306	2168
Window Full	0	0	4	11
Duplicated ACK	203	265	506	5294
TOTAL	685	815	1349	10615

Analyzing the TCP indicators in Experiment 2 shows that the sizes matter, especially due to packet loss probability. Given that FrodoKEM has the largest CSR, it increases the likelihood of packet loss, which is one of the reasons why FrodoKEM imposed nearly 7.8x and 13x more degradation events than Kyber and HQC, respectively (using the Total column from Table 3). HQC results were competitive in Experiment 2, and the primary reason is the reduced size, as not a single “TCP window full” event happened. Similarly, the interactive approach resulted in fewer TCP events due to its small sizes. In this case, the latency of Experiment 2 had more effect than sizes since the interactive approach adds a new message (request-response) not part of the standard ACME, i.e., the `key-confirm/` endpoint. Regarding Experiment 3 (stable network), all selected algorithms triggered a small number of network events (or zero). We omitted the results here, but we made available the capture files for completeness.

4.7 Other considerations

This work focuses on PoP integration and KEM-certificate issuance in ACME; the conventional signature-certificate case,

while interesting, is out of scope, as are other parts of the certificate lifecycle such as revocation, logging, and auditing (including logging into Certificate Transparency or Merkle-tree Certificates).

KEM PoP approaches are also under active discussion at the IETF [45]; a recent proposal [24] removes CSRs from the protocol entirely and enables KEM PoPs for issuance similarly to our interactive approach, but does not support KEM key revocation (Section 7.5 of the I-D), unlike our HQC-PoP. Although we did not implement their proposal, our interactive-approach results should extrapolate to it, since both rely on the same KEM operations.

NIST’s SP 800-227 [3] (Fig. 6) also discusses an enrollment approach based on symmetric encryption of the certificate under the encapsulated shared secret; we did not implement it due to its incompatibility with the current certificate transparency environment.

5 Conclusion

Certificate issuance for KEM keys has been a largely unexplored topic despite its growing relevance to the quantum threat facing public-key cryptography. This work proposed and assessed several ways of issuing KEM-based certificates, motivated by the need for a Proof-of-Possession (PoP) approach compatible with existing and proposed standards, preferably non-interactive. To this end, we built a novel non-interactive PoP for HQC using VOLE correlations and compared it against Kyber, FrodoKEM, and an interactive approach under different network conditions and automatic (ACME) issuance; the interactive approach is faster, since it avoids the computational cost of the non-interactive approaches’ zero-knowledge proofs.

Our experiments point to Kyber when issuance speed is the crucial requirement, a growing concern given the industry’s move towards short-lived certificates [12]. However, “Kyber non-interactive PoP” is not NIST-compliant, since Verifiable Generation alters key generation, deviating from the ML-KEM standard [40].

Large CSRs from the non-interactive approaches can cause real-world problems: unstable-network slowdowns in our experiments, and possible source-code refactoring (Table 4, Appendix B). HQC is thus the best non-interactive choice: shorter proofs, no key-generation changes needed (unlike Kyber/FrodoKEM’s Verifiable Generation, letting it comply with the upcoming NIST standard), and reuse across the certificate lifecycle (e.g., revocation) rather than single-use. ACME integration poses further challenges: interactive PoP breaks compatibility; non-interactive PoP is easier to adopt but not always CSR-size/timeout compatible (Table 4).

References

- [1] Najwa Aaraj, Slim Bettaieb, Loïc Bidoux, Alessandro Budroni, Victor Dyseryn, Andre Esser, Thibault Feneuil, Philippe Gaborit, Mukul Kulkarni, Victor Mateu, Marco Palumbi, Lucas Perin, Matthieu Rivain, Jean-Pierre Tillich, and Keita Xagawa. PERK. Technical report, 2025. available at <https://pqc-perk.org/assets/downloads/perk-v2.2.0.pdf>.
- [2] David Adrian. Post-quantum cryptography is too damn big. Available at: <https://dadrian.io/blog/posts/pqc-signatures-2024/>. Accessed on 05.04.2025., 2024.
- [3] Gorjan Alagic, Quynh Dang, Mark Davidson, Joshua Feegan, Dustin Moody, and Daniel Smith-Tone. Nist special publication (sp) 800-227, recommendations for key-encapsulation mechanisms. Technical report, National Institute of Standards and Technology, September 2025. URL: <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-227.pdf>, doi:<https://doi.org>.
- [4] Mila Anastasova and Panos Kampanakis. MLWE’s impact on web metrics, mTLS TTLB, and AWS service endpoint connections. Cryptology ePrint Archive, Paper 2025/2235, 2025. URL: <https://eprint.iacr.org/2025/2235>.
- [5] R. Barnes, J. Hoffman-Andrews, D. McCarney, and J. Kasten. Automatic Certificate Management Environment (ACME). RFC 8555, RFC Editor, March 2019.
- [6] Carsten Baum, Ward Beullens, Shibam Mukherjee, Emmanuela Orsini, Sebastian Ramacher, Christian Rechberger, Lawrence Roy, and Peter Scholl. One tree to rule them all: Optimizing GGM trees and OWFs for post-quantum signatures. In Kai-Min Chung and Yu Sasaki, editors, *ASIACRYPT 2024, Part I*, volume 15484 of *LNCS*, pages 463–493. Springer, Singapore, December 2024. doi:10.1007/978-981-96-0875-1_15.
- [7] Carsten Baum, Lennart Braun, Ward Beullens, Cyprien Delpech de Saint Guilhem, Michael Klooß, Christian Majenz, Shibam Mukherjee, Emmanuela Orsini, Sebastian Ramacher, Christian Rechberger, Lawrence Roy, and Peter Scholl. FAEST. Technical report, National Institute of Standards and Technology, 2024. available at <https://csrc.nist.gov/Projects/pqc-dig-sig/round-2-additional-signatures>.
- [8] Carsten Baum, Lennart Braun, Cyprien Delpech de Saint Guilhem, Michael Klooß, Emmanuela Orsini, Lawrence Roy, and Peter Scholl. Publicly verifiable zero-knowledge and post-quantum signatures from VOLE-in-the-head. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part V*, volume 14085 of *LNCS*, pages 581–615. Springer, Cham, August 2023. doi:10.1007/978-3-031-38554-4_19.
- [9] David Benjamin, Devon O’Brien, Bas Westerbaan, Luke Valenta, and Filippo Valsorda. Merkle Tree Certificates. Internet-Draft draft-davidben-tls-merkle-tree-certs-10, Internet Engineering Task Force, January 2026. Work in Progress. URL: <https://datatracker.ietf.org/doc/draft-davidben-tls-merkle-tree-certs/10/>.
- [10] Daniel J Bernstein and Tanja Lange. Post-quantum cryptography. *Nature*, 549(7671):188–194, 2017.
- [11] Slim Bettaieb, Loïc Bidoux, Philippe Gaborit, and Mukul Kulkarni. Modelings for generic PoK and applications: Shorter SD and PKP based signatures. Cryptology ePrint Archive, Report 2024/1668, 2024. URL: <https://eprint.iacr.org/2024/1668>.
- [12] Sectigo Blog. Understanding the risk scale: 6-month ssl/tls validity starts march 15, 2026. <https://www.sectigo.com/blog/200-day-ssl-certificate-expiration-risk>, Feb 19 2026. URL: <https://www.sectigo.com/blog/200-day-ssl-certificate-expiration-risk>.
- [13] Hendrik Brockhaus, David von Oheimb, Mike Ounsworth, and John Gray. Internet X.509 Public Key Infrastructure – Certificate Management Protocol (CMP). RFC 9810, July 2025. URL: <https://www.rfc-editor.org/info/rfc9810>, doi:10.17487/RFC9810.
- [14] Chair for Security Engineering. KEM-NIZKPoP repository. <https://github.com/Chair-for-Security-Engineering/KEM-NIZKPoP>, dec 29 2022. URL: <https://github.com/Chair-for-Security-Engineering/KEM-NIZKPoP>.
- [15] Cloudflare. Pqc support. Available at: <https://developers.cloudflare.com/ssl/post-quantum-cryptography/pqc-support/>, 2025.
- [16] Danny Backx. esp-32-acme repository. <https://sourceforge.net/p/esp32-acme-client/code/HEAD/tree/trunk/Acme.cpp#l277>, Apr 06 2025. URL: <https://sourceforge.net/p/esp32-acme-client/code/HEAD/tree/trunk/Acme.cpp#l277>.
- [17] Electronic Frontier Foundation. certbot v5.0.0 repository. <https://github.com/certbot/certbot/blob/v5.0.0/acme/src/acme/client.py#L172>,

- dec 29 2025. URL: <https://github.com/certbot/certbot/blob/v5.0.0/acme/src/acme/client.py#L172>.
- [18] Internet Engineering Task Force. State of protocols and pqc. <https://github.com/ietf-wg-pquip/state-of-protocols-and-pqc>, Jun 26 2025. URL: <https://github.com/ietf-wg-pquip/state-of-protocols-and-pqc>.
- [19] Ludovic Fernandez. go-acme lego v4.30.1 repository. <https://github.com/go-acme/lego/blob/v4.30.1/cmd/flags.go#L239>, dec 29 2025. URL: <https://github.com/go-acme/lego/blob/v4.30.1/cmd/flags.go#L239>.
- [20] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Berlin, Heidelberg, August 1987. doi:10.1007/3-540-47721-7_12.
- [21] Wireshark Foundation. Wireshark documentation: TCP Analysis, 2026. https://www.wireshark.org/docs/wsug_html_chunked/ChAdvTCPAnalysis.html. Accessed: 2026-01-27.
- [22] Aaron Gable. IETF Mail Archive. Available at: <https://mailarchive.ietf.org/arch/msg/acme/QMz7wnTzakF5ISCWluZZZREVTEY/>, Aug 8 2023.
- [23] Philippe Gaborit, Carlos Aguilar-Melchor, Nicolas Aragon, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Edoardo Persichetti, Gilles Zémor, Jurjen Bos, Arnaud Dion, Jérôme Lacan, Jean-Marc Robert, Pascal Véron, Paulo L. Barreto, Santosh Ghosh, Shay Gueron, Tim Güneysu, Rafael Misoczki, Jan Richter-Brokmann, Nicolas Sendrier, Jean-Pierre Tillich, and Valentin Vasseur. Hamming quasi-cyclic (HQC). https://pqc-hqc.org/doc/hqc_specifications_2025_08_22.pdf, August 2025. Official specifications of HQC, dated 22 August 2025. Accessed: 6 May 2026.
- [24] Feng Geng, Panyu Wu, Liang Xia, and Xin Chen. Automated Certificate Management Environment (ACME) Extension for Public Key Challenges. Internet-Draft draft-geng-acme-public-key-07, Internet Engineering Task Force, June 2026. Work in Progress. URL: <https://datatracker.ietf.org/doc/draft-geng-acme-public-key/07/>.
- [25] Alexandre Augusto Giron. ACME PQC Algorithm Negotiation. Internet-Draft draft-giron-acme-pqcnegotiation-03, Internet Engineering Task Force, February 2024. Work in Progress. URL: <https://datatracker.ietf.org/doc/draft-giron-acme-pqcnegotiation/03/>.
- [26] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions (extended abstract). In *25th FOCS*, pages 464–479. IEEE Computer Society Press, October 1984. doi:10.1109/SFCS.1984.715949.
- [27] Ruben Gonzalez and Thom Wiggers. Kemtls vs. post-quantum tls: Performance on embedded systems. In *International Conference on Security, Privacy, and Applied Cryptography Engineering*, pages 99–117. Springer, 2022.
- [28] Tim Güneysu, Philip W. Hodges, Georg Land, Mike Ounsworth, Douglas Stebila, and Greg Zaverucha. Proof-of-possession for KEM certificates using verifiable generation. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 1337–1351. ACM Press, November 2022. doi:10.1145/3548606.3560560.
- [29] Stephen Hemminger. Linux network emulator. Online, 2011. <https://www.linux.org/docs/man8/tc-netem.html>.
- [30] Russ Housley and Sean Turner. Use of the RSA-KEM Algorithm in the Cryptographic Message Syntax (CMS). RFC 9690, February 2025. URL: <https://www.rfc-editor.org/info/rfc9690>, doi:10.17487/RFC9690.
- [31] Internet Security Research Group. letsencrypt/boulder repository v0.20251216.0. <https://github.com/letsencrypt/boulder/blob/v0.20251216.0/wfe2/verify.go#L414>, dec 24 2025. URL: <https://github.com/letsencrypt/boulder/blob/v0.20251216.0/wfe2/verify.go#L414>.
- [32] Internet Security Research Group. letsencrypt/pebble repository. <https://github.com/letsencrypt/pebble/>, dec 29 2025. URL: <https://github.com/letsencrypt/pebble/>.
- [33] M. Jones, J. Bradley, and N. Sakimura. JSON Web Signature (JWS). RFC 7515, RFC Editor, May 2015. <http://www.rfc-editor.org/rfc/rfc7515.txt>. URL: <http://www.rfc-editor.org/rfc/rfc7515.txt>.
- [34] Panos Kampanakis and Will Childs-Klein. The impact of data-heavy, post-quantum TLS 1.3 on the time-to-last-byte of real-world connections. Cryptology ePrint Archive, Report 2024/176, 2024. URL: <https://eprint.iacr.org/2024/176>.

- [35] Knrdl. knrdl/acme-ca-server repository. <https://github.com/knrdl/acme-ca-server/blob/v0.6.2/app/acme/order/router.py#L31>, dec 29 2025. URL: <https://github.com/knrdl/acme-ca-server/blob/v0.6.2/app/acme/order/router.py#L31>.
- [36] Let's Encrypt: 10 Years of Let's Encrypt Certificates. Available at: <https://letsencrypt.org/2025/12/09/10-years>. Accessed on: 26/12/2025. URL: <https://letsencrypt.org/2025/12/09/10-years>.
- [37] Challenge Types. Available at: <https://letsencrypt.org/docs/challenge-types/>. Accessed on: 26/12/2025. URL: <https://letsencrypt.org/docs/challenge-types/>.
- [38] Giuseppe De Marco, Brandon Pitman, Tim Goghegan, David Cook, and J.C. Jones. Automatic Certificate Management Environment (ACME) with OpenID Federation 1.0. Internet-Draft draft-ietf-acme-openid-federation-00, Internet Engineering Task Force, December 2025. Work in Progress. URL: <https://datatracker.ietf.org/doc/draft-ietf-acme-openid-federation/00/>.
- [39] Michele Mosca and Marco Piani. Quantum threat timeline report. Available at: <https://globalriskinstitute.org/publication/2024-quantum-threat-timeline-report/>. Accessed on 18.05.2025., 2024.
- [40] National Institute of Standards and Technology (NIST). Module-Lattice-Based Key-Encapsulation Mechanism Standard. NIST Federal Information Processing Standards Publication (FIPS) 203, aug 2024. doi:10.6028/NIST.FIPS.203.
- [41] Neil Pang. acme.sh v3.0.5 repository. <https://github.com/acmesh-official/acme.sh/blob/v3.0.5/acme.sh#L5069>, dec 29 2025. URL: <https://github.com/acmesh-official/acme.sh/blob/v3.0.5/acme.sh#L5069>.
- [42] Nicola Di Lieto. uacme repository. <https://github.com/ndilieto/uacme/blob/master/uacme.c#L1134>, dec 29 2025. URL: <https://github.com/ndilieto/uacme/blob/master/uacme.c#L1134>.
- [43] NIST. PQC Standardization Process: Announcing four candidates to be standardized, plus fourth round candidates, 2022. <https://csrc.nist.gov/News/2022/pqc-candidates-to-be-standardized-and-round-4>. Accessed: 2022-11-02.
- [44] Open Quantum Safe Project. liboqs-go repository. <https://github.com/open-quantum-safe/liboqs-go>, mar 27 2024. URL: <https://github.com/open-quantum-safe/liboqs-go>.
- [45] Mike Ounsworth. IETF Mail Archive. Available at: <https://mailarchive.ietf.org/arch/msg/acme/2AfchjcDc-rSbpdZH9Y3UA4Bcac/>, Apr 27 2026. URL: <https://mailarchive.ietf.org/arch/msg/acme/2AfchjcDc-rSbpdZH9Y3UA4Bcac/>.
- [46] Mike Ounsworth. IETF Mail Archive. Available at: <https://mailarchive.ietf.org/arch/msg/acme/qu3nEZFo9Yemtvr5JazArHKFKEM/>, Jul 3 2026.
- [47] Christian Paquin, Douglas Stebila, and Goutam Tamvada. Benchmarking post-quantum cryptography in tls. In Jintai Ding and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography*, pages 72–91, Cham, 2020. Springer International Publishing.
- [48] E. Rescorla. The transport layer security (tls) protocol version 1.3. RFC 8446, RFC Editor, August 2018.
- [49] Lawrence Roy. SoftSpokenOT: Quieter OT extension from small-field silent VOLE in the minicrypt model. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part I*, volume 13507 of *LNCS*, pages 657–687. Springer, Cham, August 2022. doi:10.1007/978-3-031-15802-5_23.
- [50] Peter Schwabe, Douglas Stebila, and Thom Wiggers. Post-quantum tls without handshake signatures. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, CCS '20*, page 1461–1480, New York, NY, USA, 2020. ACM. doi:10.1145/3372297.3423350.
- [51] Dimitrios Sikeridis, Panos Kampanakis, and Michael Devetsikiotis. Assessing the overhead of post-quantum cryptography in tls 1.3 and ssh. In *Proceedings of the 16th International Conference on Emerging Networking EXperiments and Technologies, CoNEXT '20*, page 149–156, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3386367.3431305.
- [52] Smallstep CA. Certificates repository v0.29.0. <https://github.com/smallstep/certificates/blob/v0.29.0/scep/api/api.go#L185>, dec 29 2025. URL: <https://github.com/smallstep/certificates/blob/v0.29.0/scep/api/api.go#L185>.
- [53] Konstantina Souvatzidaki and Konstantinos Limniotis. Post-quantum key exchange in tls 1.3: Further analysis on performance of new cryptographic standards. *Cryptography*, 9(4), 2025. URL: <https://www.mdpi.com/2410-387X/9/4/73>, doi:10.3390/cryptography9040073.

- [54] Bo Wang, Xingxing Yang, Zili Meng, Yaning Guo, Chen Sun, Justine Sherry, Hongqiang Harry Liu, Mingwei Xu, and Jianping Wu. Zhuge: Toward consistent low latency with minimal control loop delay. *IEEE Transactions on Networking*, 33(2):624–639, 2025. doi:10.1109/TNET.2024.3502822.
- [55] Brandon Weeks, Ganesh Mallaya, and Sven Rajala. Automated Certificate Management Environment (ACME) Device Attestation Extension. Internet-Draft draft-ietf-acme-device-attest-00, Internet Engineering Task Force, December 2025. Work in Progress. URL: <https://datatracker.ietf.org/doc/draft-ietf-acme-device-attest/00/>.
- [56] Bas Westerbaan. Sizing up post-quantum signatures. Available at: <https://blog.cloudflare.com/sizing-up-post-quantum-signatures/>, 2021.
- [57] Wouter Tinus. win-acme v2.2.9 repository. <https://github.com/win-acme/win-acme/blob/2.2.9/src/main.lib/Clients/ScriptClient.cs#L108>, dec 29 2025. URL: <https://github.com/win-acme/win-acme/blob/2.2.9/src/main.lib/Clients/ScriptClient.cs#L108>.
- [58] Dongzhu Xu, Anfu Zhou, Xinyu Zhang, Guixian Wang, Xi Liu, Congkai An, Yiming Shi, Liang Liu, and Huadong Ma. Understanding operational 5g: A first measurement study on its coverage, performance and energy consumption. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '20, page 479–494, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3387514.3405882.
- [59] Kang Yang, Pratik Sarkar, Chenkai Weng, and Xiao Wang. QuickSilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 2986–3001. ACM Press, November 2021. doi:10.1145/3460120.3484556.

Open Science

We support Open Science principles and make our anonymous code available for reproducibility. Our code is publicly available at <https://anonymous.4open.science/r/acme-repository-C437/>.

Ethical Considerations

No human or other subjects were involved in our experiments and to the best of our knowledge there are no ethical consid-

erations of this work.

A Supplementary Algorithms and Figures

This section gathers the additional figures used throughout the paper. HQC is built from an IND-CPA public-key encryption scheme, HQC.PKE, via the salted Fujisaki–Okamoto transform with implicit rejection; in particular, HQC.KEM’s key generation algorithm is essentially the same as that of HQC.PKE. Figures 8 and 9 give these two procedures, respectively.

The remaining algorithms support the Proof-of-Possession construction described in Section 3. In particular, Figures 10 and 11 describe the prover and verifier procedures used to prove knowledge of an elementary vector, while Figures 12 and 13 give the corresponding procedures for proving the Hamming-weight condition. Finally, Figures 14 and 15 describe the masking and verification steps used in the check-zero procedure, and Figures 16 and 17 describe the prover and verifier procedures used to check the QCSD relation.

A.1 HQC Background Definitions

We recall here the coding-theoretic background and the full formal statements underlying the notation used in Section 2.2 and Section 3; see [23] for the complete specification.

The $\text{rot}(\cdot)$ operator. Let $\mathcal{R} = \mathbb{F}_2[X]/(X^n - 1)$ and let $\mathbf{v} \in \mathbb{F}_2^n$. We define $\text{rot}(\mathbf{v})$ as the circulant matrix generated by $\mathbf{v}$, given by:

$$\text{rot}(\mathbf{v}) = \begin{pmatrix} v_0 & v_{n-1} & \dots & v_1 \\ v_1 & v_0 & \dots & v_2 \\ \vdots & \vdots & \ddots & \vdots \\ v_{n-1} & v_{n-2} & \dots & v_0 \end{pmatrix} \in \mathbb{F}_2^{n \times n}.$$

We write $\tilde{h}_{i,j}$ for the (i, j) entry of $\text{rot}(\mathbf{h})^\top$, i.e., $\mathbf{h} \cdot \mathbf{y} = \mathbf{y} \cdot \text{rot}(\mathbf{h})^\top$.

Definition 1 (Linear Codes [23]). *A linear code C of dimension k and length n (denoted as $[n, k]$) is a k -dimensional subspace of $\mathcal{R}$. Elements of C are called codewords.*

Definition 2 (Parity-Check Matrix [23]). *For a $[n, k]$ code C , the parity check matrix $\mathbf{H} \in \mathbb{F}_2^{(n-k) \times n}$ is a matrix such that for every codeword $\mathbf{v} \in C$, $\mathbf{H}\mathbf{v} = \mathbf{0}$.*

Definition 3 (Quasi-Cyclic Codes of index 2 (Informal) [23]). *Let C be a $[2n, n]$ linear code, where each codeword $\mathbf{c} \in C$ is expressed as $\mathbf{c} := (\mathbf{c}_0, \mathbf{c}_1)$ with $\mathbf{c}_i \in \mathcal{R}$. Then, C is called Quasi-Cyclic (QC) if for any $\mathbf{c} := (\mathbf{c}_0, \mathbf{c}_1) \in C$, it holds that $(X \cdot \mathbf{c}_0, X \cdot \mathbf{c}_1)$ is also in C .*

A Quasi-Cyclic (QC) code C is called systematic if its parity check matrix $\mathbf{H}$ has following structure.

$$\mathbf{H} = [\mathbf{I}_n \quad \mathbf{A}]$$

where $\mathbf{A}$ is a circulant $n \times n$ matrix.

Definition 4 (QCS problem (Informal) [23]). Let n and ω be positive integers. Let $\mathbf{H} := [\mathbf{I}_n \quad \text{rot}(\mathbf{h})] \in \mathbb{F}_2^{n \times 2n}$ parity-check matrix of a systematic code C sampled uniformly at random. Also let $\mathbf{x}, \mathbf{y}$ be uniform random vectors in $\mathbb{F}_2^n$ conditioned on $w_H(\mathbf{x}) = w_H(\mathbf{y}) = \omega$. The QCS (search) problem is to recover $\mathbf{x}, \mathbf{y}$, given $(\mathbf{H}, \mathbf{s})$ where $\mathbf{s} := \mathbf{x} + \mathbf{h} \cdot \mathbf{y}$.

HQC-PKE.Keygen(seed)

Input: Seed seed.

Output: Encryption key ek_{PKE} , decryption key dk_{PKE} .

- ▷ Compute dk_{PKE} and ek_{PKE} seeds
- 1. $(\text{seed}_{\text{dk}}, \text{seed}_{\text{ek}}) \leftarrow \text{I}(\text{seed})$
- ▷ Compute decryption key dk_{PKE}
- 2. $\text{ctx}_{\text{dk}} \leftarrow \text{XOF.Init}(\text{seed}_{\text{dk}})$
- 3. $(\text{ctx}_{\text{dk}}, \mathbf{y}) \leftarrow \text{SampleFixedWeightVect}(\text{ctx}_{\text{dk}}, \mathcal{R}_\omega)$
- 4. $(\text{ctx}_{\text{dk}}, \mathbf{x}) \leftarrow \text{SampleFixedWeightVect}(\text{ctx}_{\text{dk}}, \mathcal{R}_\omega)$
- 5. $\text{dk}_{\text{PKE}} \leftarrow \text{seed}_{\text{dk}}$
- ▷ Compute encryption key ek_{PKE}
- 6. $\text{ctx}_{\text{ek}} \leftarrow \text{XOF.Init}(\text{seed}_{\text{ek}})$
- 7. $(\text{ctx}_{\text{ek}}, \mathbf{h}) \leftarrow \text{SampleVect}(\text{ctx}_{\text{ek}}, \mathcal{R})$
- 8. $\mathbf{s} \leftarrow \mathbf{x} + \mathbf{h} \cdot \mathbf{y}$
- 9. $\text{ek}_{\text{PKE}} \leftarrow (\text{seed}_{\text{ek}}, \mathbf{s})$
- 10. **return** $(\text{ek}_{\text{PKE}}, \text{dk}_{\text{PKE}})$

Figure 8: HQC PKE Key Generation Algorithm [23]

HQC-KEM.Keygen()

Input: None.

Output: Encapsulation key ek , decapsulation key dk .

- ▷ Sample seed and randomness σ
- 1. $\text{seed} \leftarrow \text{PRG.GetBytes}(|\text{seed}|)$
- 2. $\sigma \leftarrow \text{PRG.GetBytes}(k)$
- ▷ Compute HQC-PKE keypair $(\text{ek}_{\text{PKE}}, \text{dk}_{\text{PKE}})$
- 3. $(\text{ek}_{\text{PKE}}, \text{dk}_{\text{PKE}}) \leftarrow \text{HQC-PKE.Keygen}(\text{seed})$
- ▷ Compute HQC-KEM keypair (ek, dk)
- 4. $\text{ek} \leftarrow \text{ek}_{\text{PKE}}$
- 5. $\text{dk} \leftarrow (\text{dk}_{\text{PKE}}, \text{ek}, \sigma)$
- 6. **return** (ek, dk)

Figure 9: HQC Key Generation Algorithm [23]

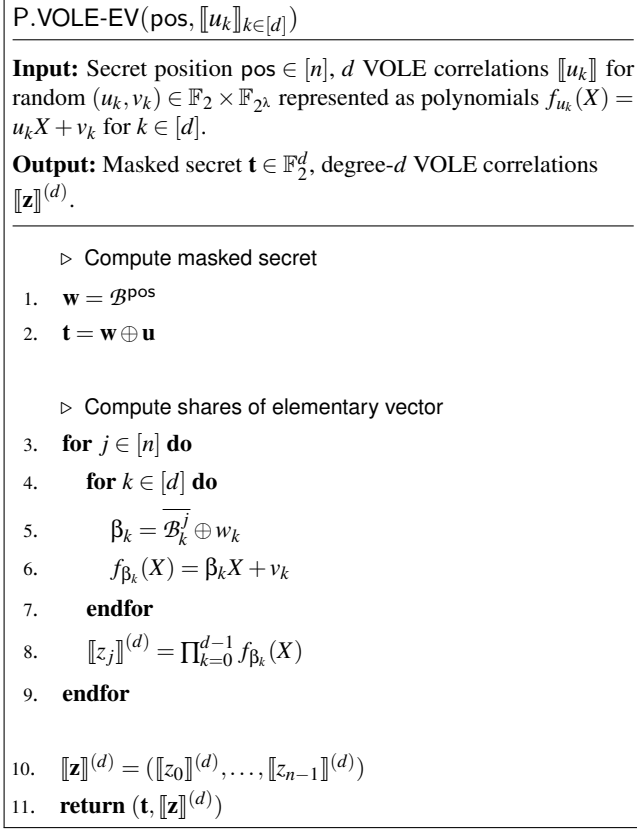


Figure 10: Proof Generation: PoK Secret Elementary Vector

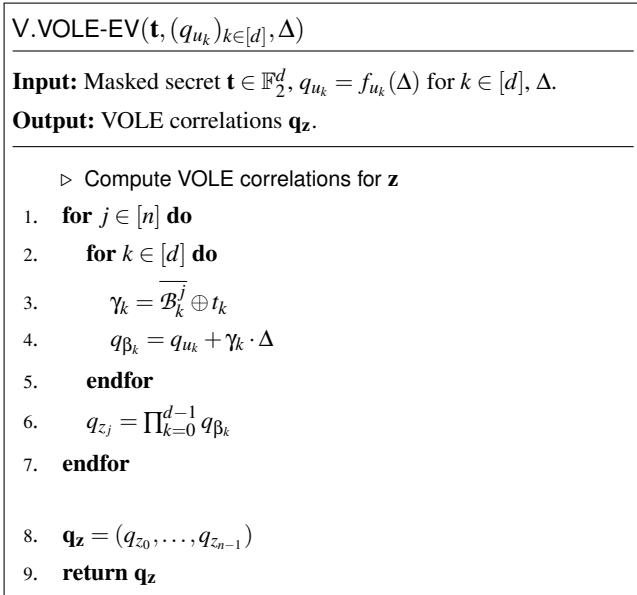


Figure 11: Verification: PoK Secret Elementary Vector

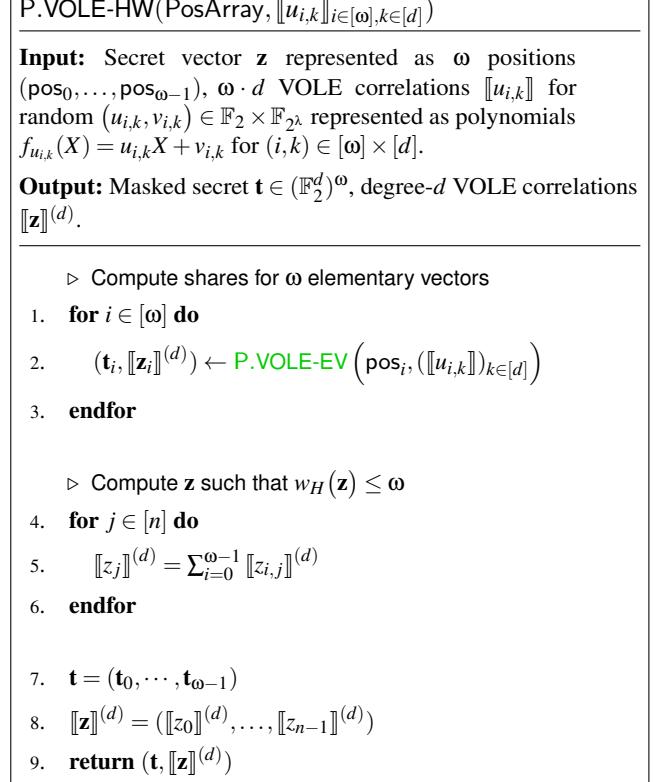


Figure 12: Proof Generation: PoK Hamming Weight

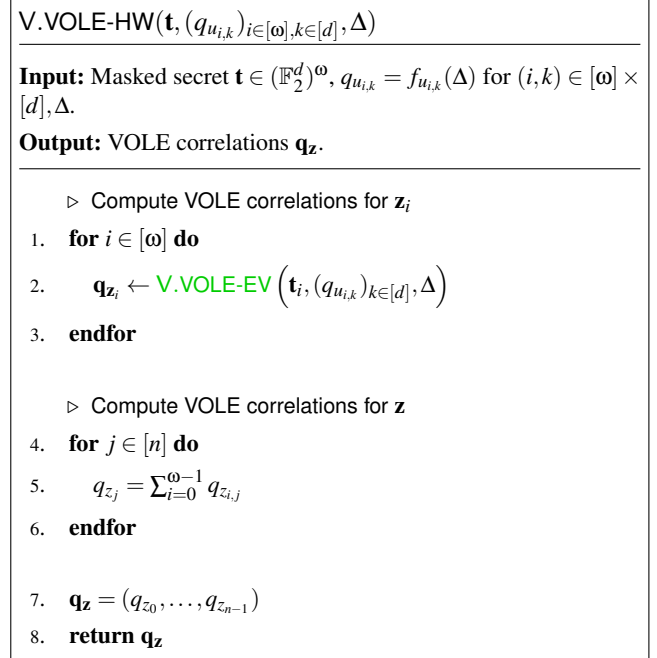


Figure 13: Verification: PoK Hamming Weight

P.CheckZero($f, \llbracket u_{i,k} \rrbracket_{i \in [d-1], k \in [\lambda]}$)

Input: Secret degree- d polynomial f , $(d-1) \cdot \lambda$ VOLE correlations $\llbracket u_{i,k} \rrbracket$ for random $(u_{i,k}, v_{i,k}) \in \mathbb{F}_2 \times \mathbb{F}_{2^\lambda}$ represented as polynomials $f_{u_{i,k}}(X) = u_{i,k}X + v_{i,k}$ for $(i,k) \in [d-1] \times [\lambda]$.

Output: Masked polynomial $\llbracket a \rrbracket$, such that for an honest prover, the leading coefficient (coefficient of X^d term will be equal to 0).

▷ Generating VOLE correlations in $\mathbb{F}_{2^\lambda} \times \mathbb{F}_{2^\lambda}$

1. **for** $i \in [d-1]$:
 - ▷ $\{\gamma_\lambda^k\}_{k=0}^{\lambda-1}$ is the power basis of $\mathbb{F}_{2^\lambda}$ with coefficients in $\mathbb{F}_2$.
2. $u'_i := \sum_{k=0}^{\lambda-1} u_{i,k} \gamma_\lambda^k$
3. $v'_i := \sum_{k=0}^{\lambda-1} v_{i,k} \gamma_\lambda^k$
4. $f_{u'_i}(X) := u'_i X + v'_i$ ▷ (u'_i, v'_i) in $\mathbb{F}_{2^\lambda} \times \mathbb{F}_{2^\lambda}$.
5. **endfor**
6. $f_{\text{mask}}(X) := \sum_{i=0}^{d-2} f_{u'_i}(X) \cdot X^i$
7. $\llbracket a \rrbracket := f_w(X) + f_{\text{mask}}(X)$
8. **return** $\llbracket a \rrbracket$

Figure 14: Proof Generation: Checking whether leading coefficient of f is zero or not.

V.CheckZero($\llbracket a \rrbracket, q_f, (q_{u_{i,k}})_{(i,k) \in [d-1] \times [\lambda]}, \Delta$)

Input: Masked polynomial $\llbracket a \rrbracket$, VOLE correlation inputs q_f , $q_{u_{i,k}} = f_{u_{i,k}}(\Delta)$ for $(i,k) \in [d-1] \times [\lambda]$, and $\Delta \in \mathbb{F}_{2^\lambda}$.

Output: Boolean indicating if leading coefficient (coefficient of X^d) of some degree- d polynomial $f(X)$ (for which verifier already holds $q_f \in \mathbb{F}_{2^\lambda}$) is equal to zero or not.

1. **if** degree of $\llbracket a \rrbracket \neq d-1$:
2. **return** $\perp$
3. **else** :
 - ▷ Generating $q'_{u_i} \in \mathbb{F}_{2^\lambda}$ for $i \in [d-1]$
4. **for** $i \in [d-1]$:
 - ▷ $\{\gamma_\lambda^k\}_{k=0}^{\lambda-1}$ is the power basis of $\mathbb{F}_{2^\lambda}$.
5. $q'_{u_i} := \sum_{k=0}^{\lambda-1} q_{u_{i,k}} \cdot \gamma_\lambda^k$
6. **endfor**
- ▷ Parse $\llbracket a \rrbracket$ as $\llbracket a \rrbracket := a_{d-1}X^{d-1} + \dots + a_1X + a_0$
7. $\bar{q} := \sum_{i=0}^{d-1} a_i \cdot \Delta^i$
8. $q = q_f + \sum_{i=0}^{d-2} q'_{u_i} \cdot \Delta^i$
9. $b := (q \stackrel{?}{=} \bar{q})$
10. **return** b

Figure 15: Verification: Checking whether leading coefficient of f is zero or not.

P.Check-QCSD
 (PosArray, $\llbracket u_{i,k} \rrbracket_{i \in [2\omega], k \in [d]}, \llbracket u_{i',k'} \rrbracket_{i' \in [d-1], k' \in [\lambda]}, \text{aux}$)

Input: Secret vectors $\mathbf{x}, \mathbf{y}$ represented as 2ω positions $(\text{pos}_0, \dots, \text{pos}_{\omega-1}, \text{pos}_{\omega}, \dots, \text{pos}_{2\omega-1})$, $2\omega \cdot d$ VOLE correlations $\llbracket u_{i,k} \rrbracket$ for random $(u_{i,k}, v_{i,k}) \in \mathbb{F}_2 \times \mathbb{F}_{2^\lambda}$ represented as polynomials $f_{u_{i,k}}(X) = u_{i,k}X + v_{i,k}$ for $(i, k) \in [2\omega] \times [d]$, $(d-1) \cdot \lambda$ VOLE correlations $\llbracket u_{i',k'} \rrbracket$ for random $(u_{i',k'}, v_{i',k'}) \in \mathbb{F}_2 \times \mathbb{F}_{2^\lambda}$ represented as polynomials $f_{u_{i',k'}}(X) = u_{i',k'}X + v_{i',k'}$ for $(i', k') \in [d-1] \times [\lambda]$, and auxiliary information aux .

Output: Proof consisting masked vector $\mathbf{t}$, and masked polynomial $\llbracket a \rrbracket$.

- ▷ Compute shares of $\mathbf{x}$ from $\mathbf{P}_\mathbf{x} := \text{PosArray}[0 : \omega - 1]$
1. $(\mathbf{t}_\mathbf{x}, \llbracket \mathbf{x} \rrbracket^{(d)}) \leftarrow \text{P.VOLE-HW}(\mathbf{P}_\mathbf{x}, (\llbracket u_{i,k} \rrbracket)_{i \in [\omega], k \in [d]})$
- ▷ Compute shares $\mathbf{y}$ from $\mathbf{P}_\mathbf{y} := \text{PosArray}[\omega : 2\omega - 1]$
2. $(\mathbf{t}_\mathbf{y}, \llbracket \mathbf{y} \rrbracket^{(d)}) \leftarrow \text{P.VOLE-HW}(\mathbf{P}_\mathbf{y}, (\llbracket u_{\omega+i,k} \rrbracket)_{i \in [\omega], k \in [d]})$
3. $\mathbf{t} := (\mathbf{t}_\mathbf{x}, \mathbf{t}_\mathbf{y})$
- ▷ Compute $\mathbf{x} + \mathbf{h} \cdot \mathbf{y} - \mathbf{s}$
4. **for** $j \in [n]$ **do**
5. $f_j(X) = \llbracket \mathbf{x}_j \rrbracket^{(d)} + \sum_{i=0}^{n-1} \tilde{h}_{i,j} \cdot \llbracket \mathbf{y}_i \rrbracket^{(d)} - \mathbf{s}_j \cdot X^d$
6. **endfor**
- ▷ Merge polynomials and run CheckZero
7. $\text{ch}_2 := \text{RO}_2^2(\text{aux} || \mathbf{t})$
8. $\alpha \xleftarrow{\$} \text{ch}_2 \in \mathbb{F}_{2^\lambda}^n$ ▷ ch_2 must be sent by V in interactive protocol.
9. $f(X) = \sum_{i=0}^{n-1} \alpha_i \cdot f_i(X)$
10. $\llbracket a \rrbracket \leftarrow \text{P.CheckZero}(f, \llbracket u_{i',k'} \rrbracket_{i' \in [d-1], k' \in [\lambda]})$
11. $\text{proof} = (\mathbf{t}, \llbracket a \rrbracket, \text{ch}_2)$
12. **return** proof

Figure 16: Proof Generation: PoK QCSD Solution

B ACME implementations and limitations

Table 4 presents a summary of ACME client and server implementations and its limitations regarding CSR sizes and CSR validation timeouts. Some ACME server implementations have size limits, such as Boulder (as mentioned in Section 4.3, and Small Step CA implementation (v0.29.0). The latter sets 2 MB in the `maxPayloadSize` variable before parsing JWS requests [52]. Although the reference code found for `step-ca` is applicable to the SCEP protocol, it remains a possible problem for integrating large CSRs.

lic keys and signatures (at compile time), which is one possible problem when generating a large KEM-based CSR. The `esp32-acme-client` [16] also supports MbedTLS. System resource limits can obviously be a problem, and for `acme.sh` [41] CSRs depends on `ARG_MAX`. In common Linux distributions this is set to 2 MB.

V.Check-QCSD $(\mathbf{t}, \llbracket a \rrbracket, (q_{u_{i,k}})_{(i,k) \in [2\omega] \times [d]}, (q_{u_{i',k'}})_{(i',k') \in [d-1] \times [\lambda]}, \Delta, \text{aux})$
Input: Masked secret $\mathbf{t} \in (\mathbb{F}_2^d)^{2\omega}$, masked polynomial $\llbracket a \rrbracket$ VOLE correlation inputs $q_{u_{i,k}} = f_{u_{i,k}}(\Delta)$ for $(i,k) \in [2\omega] \times [d]$, $q_{u_{i',k'}} = f_{u_{i',k'}}(\Delta)$ for $(i',k') \in [d-1] \times [\lambda]$, and auxiliary information <code>aux</code> . Output: Boolean indicating if the proof is accepted or rejected.
▷ Compute VOLE correlations for $\mathbf{x}$ from $\mathbf{t}_x$ 1. $\mathbf{q}_x \leftarrow \text{V.VOLE-HW}(\mathbf{t}_x, (q_{u_{i,k}})_{i \in [\omega], k \in [d]}, \Delta)$ ▷ Compute VOLE correlations for $\mathbf{y}$ from $\mathbf{t}_y$ 2. $\mathbf{q}_y \leftarrow \text{V.VOLE-HW}(\mathbf{t}_y, (q_{u_{\omega+i,k}})_{i \in [\omega], k \in [d]})$ 3. $\mathbf{q} := (\mathbf{q}_x, \mathbf{q}_y) \in \mathbb{F}_{2^\lambda}^{2n}$ ▷ Computing q_{f_j} values 4. for $j \in [n]$ do 5. $q_{f_j} = q_j + \sum_{i=0}^{n-1} \tilde{h}_{i,j} \cdot q_{n+i} - \mathbf{s}_j \cdot \Delta^d$ 6. endfor ▷ Merge polynomials and run CheckZero 7. $\text{ch}_2 := \text{RO}_2^2(\text{aux} \mathbf{t})$ ▷ In interactive protocol ch_2 is same as the one sent to P. 8. $\alpha \xleftarrow{\\$, \text{ch}_2} \in \mathbb{F}_{2^\lambda}^n$ 9. $q_f = \sum_{i=0}^{n-1} \alpha_i \cdot q_{f_i}$ 10. $b \leftarrow \text{V.CheckZero}(\llbracket a \rrbracket, q_f, (q_{u_{i',k'}})_{(i',k') \in [d-1] \times [\lambda]}, \Delta)$ 11. return (b, ch_2)

Figure 17: Verification: PoK QCSD Solution

In regard to the ACME clients, the size limitations are not specified by the implementation but in the crypto library, if any. Starting with `uacme` [42], it can be configured to use OpenSSL, GnuTLS, or MbedTLS as the crypto library for generating CSR. MbedTLS defines a maximum size for pub-

Table 4: Survey of ACME implementations and their size limits and timeouts. Note: Some underlying crypto libraries may define maximum signature sizes (e.g., MbedTLS); win-acme leaves 600s timeout for the whole protocol.

Implementation name	Type	Programming language	Possible impacts of large CSRs	ACME finalization Timeout (default, in seconds)
Boulder [31]	Server	Go	50000 bytes limit for reading JWS requests.	N/A
Step CA [52]	Server	Go	2MB limit for reading JWS requests	N/A
Pebble [32]	Server	Go	No explicit size limits found	N/A
acme-ca-server [35]	Server	Python	Maximum CSR size of 1MB	N/A
Certbot [17]	Client	Python	No explicit size limits found	Timeout of 90 seconds (configurable)
LEGO [19]	Client	Go	No explicit size limits found	Timeout of 30 seconds (configurable)
uacme [42]	Client	C	No explicit size limits found*	Timeout hardcoded to 2560s
esp32-acme-client [16]	Client	C++	No explicit size limits found*	Timeout not defined
win acme [57]	Client	C#	No explicit size limits found	Timeout hardcoded to 600s**
acme.sh [41]	Client	Shell script	Default limit to 2MB	Timeout hardcoded to 30s (multiplied by a factor)

C Additional Experiments and Aggregated data

This Appendix provides additional experiments for NIST’s security levels 3 and 5. Tables 5 and 7 provides benchmarks for the cryptographic primitives and after integrating in the ACME protocol, respectively. While Table 5 is complimentary to Table 2, Table 7 summarizes all protocol experiments in one Table, for comparison. The experiment results, logs, raw data, and packet captures are publicly available.⁶

C.1 Cycles for L3 and L5

Table 5 gives the Security Level 3 and Security Level 5 counterpart of Table 2 (Section 4), which reports Security Level 1 only in the main body. All settings (iteration counts, parameter presets, and hardware) are identical; see Table 2’s caption for the parameter definitions.

C.2 HQC parallelization analysis

Table 6 provides benchmarks for the two parameter sets of HQC (short, balanced, accordingly to Section 4). Balanced option provides a faster proving operation up to 8 threads, exploring our setup (Section 4). The verify operation was not parallelized.

C.3 Protocol Performance for L3 and L5

Figures 18 and 19 shows the performance for issuance of KEM certificates following the same methodology presented in Section 4, but for NIST’s security levels 3 and 5. We can observe faster issuances using ML-KEM and Kyber, at the cost of loosing compatibility to the ACME’s RFC or, in the non-interactive setting, compliance to the NIST standards. We noted that FrodoKEM at level 5 suffer heavy losses due to

its large proof, which extrapolated our timeout setting for the LEGO implementation.

C.4 Aggregated protocol performance statistics

Table 7 summarizes issuance times for all experiment data, providing average and standard deviation as complimentary statistics to Figures 7, 18 and 19.

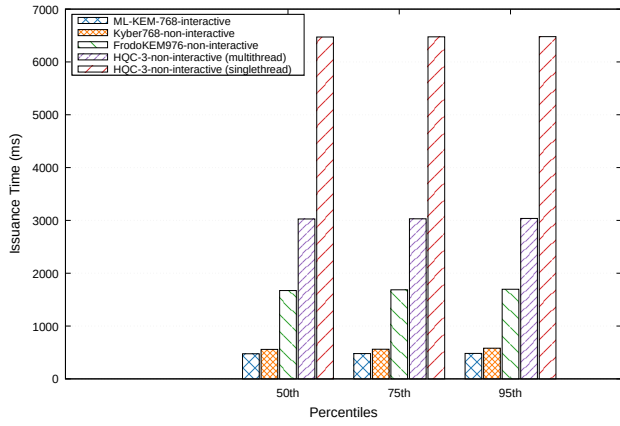
⁶<https://drive.proton.me/urls/SNZD6FBTRM#4eWOpQRZLSJO>

Table 5: Performance and sizes of non-interactive approaches for Proof-of-Possession (PoP) in megacycles (10^6), rounded to two decimal places, for NIST Security Levels 3 and 5. Parameter definitions and experimental settings are as in Table 2.

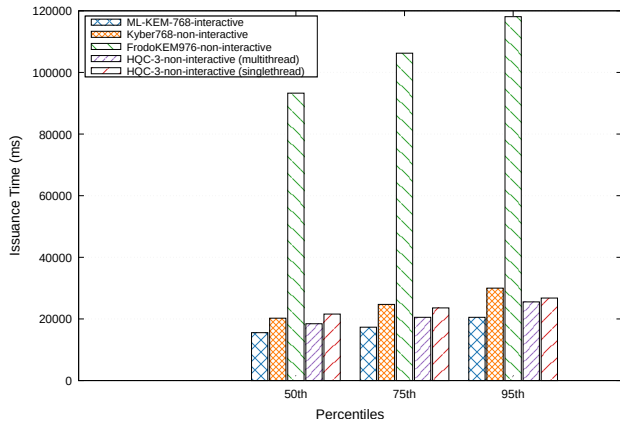
Non-interactive approach	Proof Generation (MCycles)		Proof Verification (MCycles)		Proof Size (bytes)
	Avg.	Median	Avg.	Median	
HQC-3-PoP-short (single-thread)	11650.50	11642.91	2446.47	2446.36	18008
HQC-3-PoP-short (multi-thread)	3736.20	3732.27	2444.57	2444.22	18008
HQC-3-PoP-balanced (single-thread)	11294.88	11283.87	2059.01	2058.93	24680
HQC-3-PoP-balanced (multi-thread)	3356.63	3352.15	2060.18	2060.19	24680
Kyber768-VG-short	16771.94	16795.22	16273.78	16213.02	29964
Kyber768-VG-balance	56.74	56.76	37.03	37.02	122771
Kyber768-VG-fast	13.18	13.16	8.16	8.16	191916
Frodo976-VG-short	371067.16	370988.37	333322.74	332887.87	326170
Frodo976-VG-balance	893.70	893.17	803.49	803.54	1538884
Frodo976-VG-fast	206.17	206.12	158.38	158.36	2511178
HQC-5-PoP-short (single-thread)	46675.42	46675.48	18536.59	18536.22	25768
HQC-5-PoP-short (multi-thread)	20199.50	20194.05	18520.55	18519.50	25768
HQC-5-PoP-balanced (single-thread)	36229.06	36149.04	8676.10	8676.19	42760
HQC-5-PoP-balanced (multi-thread)	10396.64	10384.58	8677.71	8678.57	42760
Kyber1024-VG-short	23230.56	23223.04	22010.75	21989.56	40330
Kyber1024-VG-balance	100.71	100.74	66.18	66.20	220180
Kyber1024-VG-fast	17.56	17.55	10.85	10.84	258618
Frodo1344-VG-short	614255.76	613283.59	580955.78	580999.17	433636
Frodo1344-VG-balance	2197.25	2197.14	1991.86	1991.78	2731558
Frodo1344-VG-fast	362.91	362.73	271.43	271.43	3347652

Table 6: HQC parallelization benchmark.

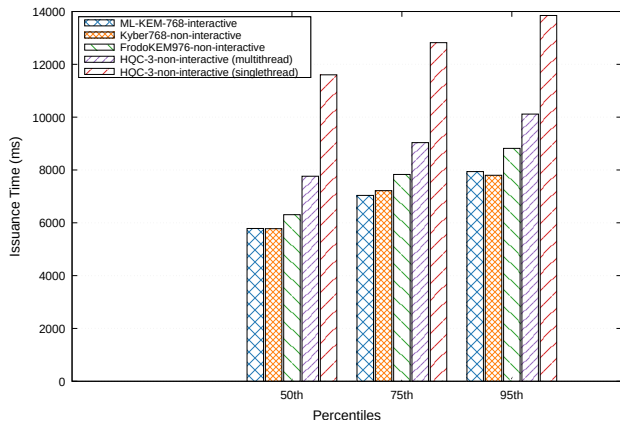
Algorithm name	(Average cycles times 10^6)				
	1 thread	2 threads	4 threads	8 threads	16 threads
HQC-1-pop-prove (balanced)	1765	1005	621	585	684
HQC-3-pop-prove (balanced)	11269	6059	3461	3354	3743
HQC-5-pop-prove (balanced)	36503	19416	10837	10402	11371
HQC-1-pop-prove (short)	3904	3152	2765	2733	2832
HQC-3-pop-prove (short)	11651	6435	3839	3736	4122
HQC-5-pop-prove (short)	46675	29470	20824	20199	20232



(a) Level 3 - Experiment 4 (skip HTTP-01 validation)

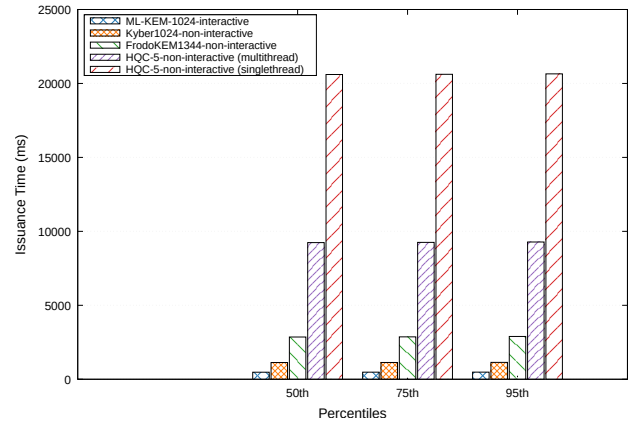


(b) Level 3 - Experiment 5 (unstable network)

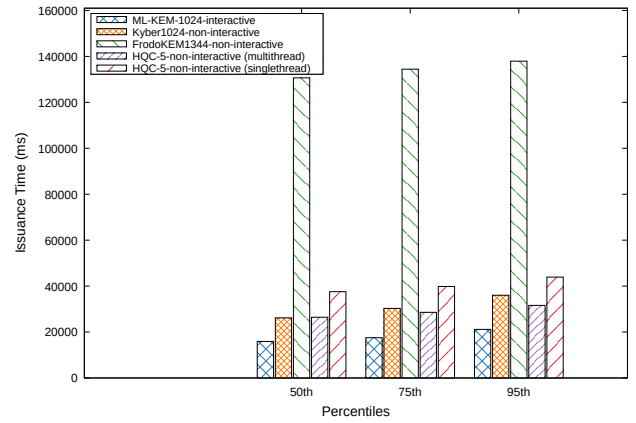


(c) Level 3 - Experiment 6 (realistic scenario, HTTP-01 enabled)

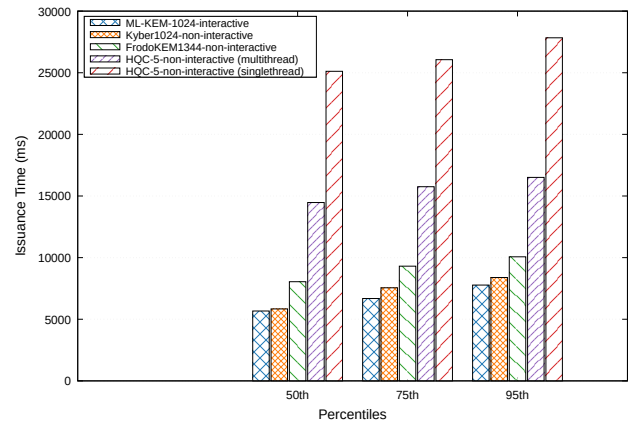
Figure 18: Certificate issuance timings for each experiment (Security Level 3).



(a) Level 5 - Experiment 7 (skip HTTP-01 validation)



(b) Level 5 - Experiment 8 (unstable network). PS: we observe a failure rate of 89% in FrodoKEM due to our timeout of 130 seconds.



(c) Level 5 - Experiment 9 (realistic scenario, HTTP-01 enabled)

Figure 19: Certificate issuance timings for each experiment (Security Level 5).

Table 7: Statistics about Issuance times (in ms) for the different algorithms. The experiment ID numbers correspond to each scenario and security level: 1-3 for security level 1, 4-6 for level 3, and 7-9 for level 5. Note: all statistics are computed after issuance of 100 certificates, except for FrodoKEM1344 in the unstable networks which had a failure rate of 89% due to the 130s timeout.

ID	Algorithm/Parameter Set	Avg.	Std. Deviation	P95%
1	ML-KEM-512-Interactive	475.04	4.86	481.06
	Kyber-512-non-interactive	528.17	11.46	542.81
	FrodoKEM640-non-interactive	1280.80	12.21	1297.20
	HQC-1-multithread-non-interactive	1255.85	17.34	1290.91
	HQC-1-single-thread-non-interactive	1768.08	17.59	1802.15
2	ML-KEM-512-Interactive	14826.66	2623.33	19285.32
	Kyber-512-non-interactive	17308.82	3608.85	23656.83
	FrodoKEM640-non-interactive	47543.83	13079.15	66856.04
	HQC-1-multithread-non-interactive	15361.33	2958.32	19832.36
	HQC-1-single-thread-non-interactive	16275.08	2896.75	21430.60
3	ML-KEM-512-Interactive	5619.18	1373.30	7737.61
	Kyber-512-non-interactive	5273.55	1320.51	7879.69
	FrodoKEM640-non-interactive	6066.42	1406.50	8461.14
	HQC-1-multithread-non-interactive	6040.41	1395.31	8321.49
	HQC-1-single-thread-non-interactive	6656.45	1429.85	9021.37
4	ML-KEM-768-Interactive	475.62	3.78	481.95
	Kyber-768-non-interactive	560.36	9.21	580.70
	FrodoKEM976-non-interactive	1676.72	12.53	1696.54
	HQC-3-multithread-non-interactive	3027.34	5.97	3037.16
	HQC-3-single-thread-non-interactive	6471.55	5.06	6478.77
5	ML-KEM-768-Interactive	15688.13	2692.27	20528.98
	Kyber-768-non-interactive	21414.83	4870.53	29999.13
	FrodoKEM976-non-interactive	93887.14	15780.00	118098.70
	HQC-3-multithread-non-interactive	18988.02	3408.94	25532.59
	HQC-3-single-thread-non-interactive	21675.64	2959.55	26781.08
6	ML-KEM-768-Interactive	5659.53	1494.84	7938.92
	Kyber-768-non-interactive	5790.07	1475.91	7799.76
	FrodoKEM976-non-interactive	6527.51	1489.33	8818.02
	HQC-3-multithread-non-interactive	7855.38	1423.03	10116.37
	HQC-3-single-thread-non-interactive	11620.33	1473.55	13849.45
7	ML-KEM-1024-Interactive	476.84	3.49	481.87
	Kyber-1024-non-interactive	1131.53	8.50	1141.53
	FrodoKEM1344-non-interactive	2857.76	19.27	2893.99
	HQC-5-multithread-non-interactive	9243.14	29.80	9280.13
	HQC-5-single-thread-non-interactive	20604.66	26.96	20647.76
8	ML-KEM-1024-Interactive	15837.28	3118.95	21149.78
	Kyber-1024-non-interactive	26115.87	5798.17	36012.98
	FrodoKEM1344-non-interactive*	129861.98	6074.82	137939.22
	HQC-5-multithread-non-interactive	26551.77	3422.18	31590.24
	HQC-5-single-thread-non-interactive	37773.18	3246.74	43922.18
9	ML-KEM-1024-Interactive	5520.23	1506.68	7773.73
	Kyber-1024-non-interactive	6112.08	1519.22	8387.89
	FrodoKEM1344-non-interactive	8014.07	1440.54	10066.48
	HQC-5-multithread-non-interactive	14439.33	1349.89	16508.53
	HQC-5-single-thread-non-interactive	25318.44	1321.34	27835.33

D Security Proof

As described in Section 2.3, we do not alter the protocol so the ACME's original threat model (RFC 8555, Section 10.1, 10.2) is applicable to our results. We only replace CSR self-signatures with non-interactive PoP. Since Fiat-Shamir based non-interactive PoP can serve as signatures (with CSR content as message), it suffices to show that our non-interactive PoPs (signatures) provide strong existential unforgeability under chosen message attack (EUF-CMA) guarantees in the random oracle model. This signature scheme is similar to NIST signature candidates in the MPC/VOLE-in-the-head category (SDitH, MQOM, and FAEST).

We begin by providing a high-level overview of creating VOLE correlations in practice. These correlations can be seen as VOLE commitments and therefore we then state some important lemmas related to the security properties of the created VOLE commitments.

VOLE Correlations via GGM Trees

In practice, P generates the VOLE correlations with the help of puncturable pseudorandom functions (puncturable PRF), which are instantiated from pseudorandom generators (PRG) using GGM trees [26].⁷ First, P computes a binary tree from a uniform random master seed $seed$ with the help of length-doubling PRG (say PRG_1) to obtain, $N := 2^\kappa$ pseudorandom strings (say $\mathbf{r}'_0, \mathbf{r}'_1, \dots, \mathbf{r}'_{N-1}$). These N strings will be used to generate the required VOLE correlations. Let $\mathbf{r}_i = PRG_2(\mathbf{r}'_i)$ be ℓ -bit pseudorandom strings for $i \in [N]$. The VOLE correlation inputs $(\mathbf{u}, \mathbf{v})$ are obtained as,

$$\mathbf{u} = \sum_{i=0}^{N-1} \mathbf{r}_i, \quad \mathbf{v} = \sum_{i=0}^{N-1} i \cdot \mathbf{r}_i. \quad (2)$$

To obtain the verifier's inputs, V chooses $\Delta \in [N]$ uniformly at random and receives all the strings $\mathbf{r}'_i$ for $i \in [N] \setminus \Delta$ from P . It can then compute

$$\mathbf{q} = \sum_{i=0}^{N-1} (\Delta - i) \cdot \mathbf{r}_i. \quad (3)$$

For security, it is required that the above computation is binding for the prover. To achieve this, P commits to the N strings $\mathbf{r}'_0, \mathbf{r}'_1, \dots, \mathbf{r}'_{N-1}$, as well as all the computations in the proof that rely on $(\mathbf{u}, \mathbf{v})$ to ensure soundness. Note that V should reveal Δ (to receive all-but-one random strings) *only after* P has committed to $\mathbf{r}'_0, \mathbf{r}'_1, \dots, \mathbf{r}'_{N-1}$ and proof computations. Otherwise, knowing Δ in advance trivially leads to successful cheating (breaking soundness of PoP) by malicious prover. P can reveal all pseudorandom strings except $\mathbf{r}'_\Delta$ by revealing the siblings on the path from the root of GGM tree to the leaf

⁷This kind of constructions are sometimes referred as *all-but-one vector commitment schemes* in literature.

node $\mathbf{r}'_\Delta$. The verifier V can then reconstruct all strings $\mathbf{r}'_i$ for $i \in [N]$, except $\mathbf{r}'_\Delta$ from the $O(\log_2(N))$ intermediate values sent by P .

The soundness error of the PoP (based on the binding property of VOLE correlations mentioned above) is $2^{-\kappa}$ (probability of malicious prover guessing Δ). Therefore, to achieve the desired security level, it is required to create many parallel instances of VOLE correlations in $\mathbb{F}_{2^\kappa}$ and then to concatenate those together to produce a single VOLE correlation instance in exponentially large field $\mathbb{F}_{2^\lambda}$. The repetition and VOLE-Field parameters $\tau \in \mathbb{N}$ and $\kappa \in \mathbb{N}$ respectively, are selected such that $\lambda \leq \tau\kappa$. Rather than building individual GGM-tree for each of the τ repetitions, we rely on forming a single big tree, as described in [6] and other subsequent optimizations, to save communication costs (which affects to proof sizes).

Security Properties of VOLE commitments

We begin by stating some useful lemmas that we require in our security proof. Since we use the same algorithms as [1, 7], for creating and committing to VOLE correlations, we do not prove these lemmas and refer the readers to [1, 7] for the proofs.

The following lemma provides bounds for breaking for collision-resistance and one-wayness properties of random oracles.

Lemma D.1 (Random oracle graph). [7, Lemma 9.18], [1, Lemma 7.1]

Let $RO: X \rightarrow \mathcal{Y}$ be the random oracle. We consider the following random oracle graph game between a challenger and an adversary:

1. The challenger initializes two sets $\mathcal{V}$ and $\mathcal{E}$ by $\emptyset$ and runs the adversary.
2. The adversary can query the random oracle RO in the following two ways:
 - The adversary queries $y \in \mathcal{Y}$ to the random oracle RO ;
 - If $y \notin \mathcal{V}$, then the challenger adds node y to $\mathcal{V}$.
 - The adversary queries $x \in X$ to the random oracle RO ; Let $RO(x) = y$.
 - If $y \in \mathcal{V}$ but there is no edge $(x', y) \in \mathcal{E}$, then the adversary wins (because it breaks onewayness).
 - If an edge exists $(x', y) \in \mathcal{E}$ with $x' \neq x$, then the adversary wins (because it finds a collision).
 - Else, the challenger adds nodes x and y to $\mathcal{V}$ and an edge $e = (x, y)$ from x to y to $\mathcal{E}$.

If the adversary makes at most q queries to RO, then the adversary's advantage is at most $q^2/|\mathcal{V}|$.

Since the size of the range ($|\mathcal{V}|$) of the hash functions such as SHA-3 or SHAKE modelled as random oracles in our constructions is exponential in the security parameter λ , from now onward we assume that for a PPT attacker $\mathcal{A}$, its winning probability in the random oracle game is $\text{negl}(\lambda)$.

As mentioned in [HQC-PoP.P.](#), we create and commit to VOLE correlations in identical manner as shown in [1, Algorithm 4.34, lines 4-8]. Specifically, we also implement and use the algorithms VOLECommit and VOLEHash described as [1, Algorithm 4.5] and [1, Algorithm 4.8] respectively. Following two lemmas state the extractable-binding and multi-hiding properties of VOLE commitments created using these algorithms. We refer the readers to [1] for proofs of the lemmas [Theorem D.2](#), and [Theorem D.3](#)

Lemma D.2 (Extractable Binding (Informally restated)). [1] *Let $\mathcal{A}$ be a PPT attacker that makes polynomial number of queries to random oracles RO, used in VOLECommit and output a commitment value h_{com} along with its valid opening. Then, there exists a straight-line extractor Ext, such that Ext extracts VOLE correlation values $\mathbf{u}, \mathbf{V}$ consistent with h_{com} , with probability $1 - \text{negl}(\lambda)$.*

For precise statement and proof of the lemma proving the extractable-binding property, please refer to [1, Lemma 7.2, p.74] and its proof.

Lemma D.3 (Multi Hiding (Informally restated)). [1] *Let $\mathcal{A}$ be a PPT attacker that makes polynomial number of queries to random oracle RO, and secure pseudo-random generator PRG used in VOLECommit. There exists an efficient simulator which simulates view of $\mathcal{A}$ (that includes VOLE correlation values $\mathbf{u}$), such that the attacker's advantage when trying to distinguish between the view comprising of outputs of VOLECommit and the simulated view is $\text{negl}(\lambda)$.*

For precise statement and proof of the lemma proving the multi-hiding property, please refer to [1, Lemma 7.3, p.76] and its proof.

Analogues of lemmas proving the extractable-binding and multi-hiding properties of VOLE commitments are also proven in [7, Lemmas 9.15, 9.20].

D.1 Digital signature schemes

We will next revisit security definitions related to digital signatures (recall that non-interactive PoP is a signature) and then prove the strong existential unforgeability under chosen message attack (sEUF-CMA).

Definition 5 (Signature Scheme). *A signature scheme is a tuple of three PPT algorithms $\Sigma = (\text{KGen}, \text{Sign}, \text{Vf})$ that performs:*

- $\text{KGen}(1^\lambda)$: The key generation algorithm on input security parameter, outputs a pair of keys (vk, sk) . The key vk is the public key used for verification and sk is the private (secret) signing key.
- $\text{Sign}(\text{sk}, \text{m})$: The signing algorithm takes as input a secret signing key sk along with a message m and outputs a signature $\sigma \leftarrow \text{Sign}(\text{sk}, \text{m})$.
- $\text{Vf}(\text{vk}, \text{m}, \sigma)$: The deterministic verification algorithm on input a verification key vk , a message m , and a signature σ ; outputs a bit $b := \text{Vf}(\text{vk}, \text{m}, \sigma)$, with indicating if the signature-message pair is valid ($b = 1$) or not ($b = 0$)

Definition 6 (EUF-CMA Security and EUF-NMA Security). *A signature scheme $\Sigma = (\text{KGen}, \text{Sign}, \text{Vf})$ is existentially unforgeable under chosen-message attacks (EUF-CMA secure) if, for all PPT adversaries $\mathcal{A}$ there is a negligible function $\text{negl}(\cdot)$ such that,*

$$\Pr[\text{Expt}_{\Sigma, \mathcal{A}}^{\text{euf-cma}}(\lambda) = 1] \leq \text{negl}(\lambda),$$

where the security game $\text{Expt}_{\Sigma, \mathcal{A}}^{\text{euf-cma}}(\lambda)$ is defined below. In addition, if we consider the game where the signing oracle OSign is removed, then the signature scheme is said to be existentially unforgeable under no-message attacks (EUF-NMA secure).

Security game $\text{Expt}_{\Sigma, \mathcal{A}}^{\text{euf-cma}}(\lambda)$:

- First generate, the key pair $(\text{vk}, \text{sk}) \leftarrow \text{KGen}(1^\lambda)$ and send it to the attacker $\mathcal{A}$.
- The attacker can adaptively query a signing oracle OSign that signs messages of $\mathcal{A}$'s choice with the signing key sk . Let Q be the set that keeps track of attacker's message queries.
- The attacker outputs forgery (m^*, σ^*) , and wins if both of the following conditions are satisfied: (a) $\text{Vf}(\text{vk}, \text{m}^*, \sigma^*)$ outputs 1, and (b) $\text{m}^* \notin Q$. Output 1 if attacker wins.

Definition 7 (SUF-CMA Security). *A signature scheme $\Sigma = (\text{KGen}, \text{Sign}, \text{Vf})$ is strongly existentially unforgeable under chosen-message attacks (SUF-CMA secure) if, for all PPT adversaries $\mathcal{A}$ there is a negligible function $\text{negl}(\cdot)$ such that,*

$$\Pr[\text{Expt}_{\Sigma, \mathcal{A}}^{\text{suf-cma}}(\lambda) = 1] \leq \text{negl}(\lambda),$$

where the security game $\text{Expt}_{\Sigma, \mathcal{A}}^{\text{suf-cma}}(\lambda)$ is defined below.

Security game $\text{Expt}_{\Sigma, \mathcal{A}}^{\text{suf-cma}}(\lambda)$:

- First generate, the key pair $(\text{vk}, \text{sk}) \leftarrow \text{KGen}(1^\lambda)$ and send it to the attacker $\mathcal{A}$.

- The attacker can adaptively query a signing oracle OSign that signs messages of $\mathcal{A}$'s choice with the signing key sk . Let $\mathcal{Q}$ be the set that keeps track of attacker's message queries along with the signature output by the signing oracle.
- The attacker outputs forgery (m^*, σ^*) , and wins if both of the following conditions are satisfied: (a) $\forall f(\text{vk}, m^*, \sigma^*)$ outputs 1, and (b) $(m^*, \sigma^*) \notin \mathcal{Q}$. Output 1 if attacker wins.

Security proofs for the EUF-NMA security.

The proofs in this section are heavily inspired and mainly follow the proofs in specifications of PERK [1], and FAEST [7]. We present the proofs here mainly for the purpose of making the paper self-contained, without exposing the full details and precise formulae for attackers success probabilities.

Theorem D.4 (EUF-NMA security in the ROM). *Let $\mathcal{A}$ be a polynomial time adversary that breaks the EUF-NMA security of signature scheme derived from non-interactive HQC-PoP.P. Then there exists, another efficient adversary $\mathcal{B}$ that solves the QCSD problem Definition 4 with high probability (same as success probability of adv).*

Proof of Theorem D.4. We will use $\Pr[W_i]$ to denote the attacker's advantage (success probability) in game G_i . Similarly we will also use $\Pr[\text{Fail}_i]$ to indicate the attacker's success probability of winning game G_i by winning the random oracle graph game (breaking the collision-resistance or one-wayness of the random oracles).

G_0 : The original EUF-NMA game. We have

$$\Pr[W_0] = \text{Adv}_{\mathcal{A}}^{\text{euf-nma}}.$$

We note that the random oracles and commitments in the PoP (as well as VOLE related functions) denoted as RO_i^j are implemented by lazy sampling and the lists, $\mathcal{E}_i^j$, where i, j are used to indicate the domain separation between various random oracle instances. Looking ahead $\mathcal{E}_i^j$ will be edge sets of the random oracle graphs.

G_1 : In this game, we replace “(seed, salt) := $\text{RO}_3(\text{dk}||\mu||\text{rnd})$ ” in line 3 of HQC-PoP.P. with “(seed, salt) $\xleftarrow{\$} \{0, 1\}^{\lambda+2\lambda}$ ”. Due to security of random oracle we have that,

$$|\Pr[W_1] - \Pr[W_0]| \leq \text{negl}(\lambda).$$

G_2 : In this game, we consider attacker's chances of winning the random oracle graph game by making polynomial number of queries to RO_i^j . We will therefore also consider the vertex sets $\mathcal{V}_i^j$. If $\mathcal{A}$ wins any of the random oracle graph game, then we abort.

Recall that $\Pr[\text{Fail}_i]$ indicates the attacker's success probability of the event that it wins the random oracle graph game. Since we did not change any winning conditions for the attacker,

$$\Pr[W_1] = \Pr[W_2 \mid \overline{\text{Fail}_2}] \cdot \Pr[\overline{\text{Fail}_2}] + \Pr[W_2 \mid \text{Fail}_2] \cdot \Pr[\text{Fail}_2]$$

We note that if Fail_2 does not occur, then G_2 is equivalent to G_1 .

$$\Pr[W_1] \leq \Pr[W_2 \mid \overline{\text{Fail}_2}] + \Pr[\text{Fail}_2]$$

G_3 : We next use the extractor Ext from Theorem D.2 to obtain $(\mathbf{u}, \mathbf{V})$ values, after line 3 of HQC-PoP.P. instead of using VOLECommit [1, Algorithm 4.34, line 4]. Since Ext fails with only negligible probability, we have

$$\Pr[W_2 \mid \overline{\text{Fail}_2}] \leq \Pr[W_3 \mid \overline{\text{Fail}_3}] + \Pr[\text{Fail}_3] + \text{negl}(\lambda)$$

G_4 : We next replace challenge ch_1 with random value after line 3 of HQC-PoP.P. instead of using [1, Algorithm 4.34, line 5] to get it from RO_2^1 . and if the query is extractable, then it runs the VOLE consistency checks using VOLEHash (after line 3 of HQC-PoP.P. using [1, Algorithm 4.34, line 6,7]) the extracted values $(\mathbf{u}, \mathbf{V})$. Following [7, Lemma 9.34, Claim 1] and [8, Thm.2] this consistency check fails with negligible probability assuming the universality of VOLEHash . Therefore,

$$\Pr[W_3 \mid \overline{\text{Fail}_3}] \leq \Pr[W_4 \mid \overline{\text{Fail}_4}] + \text{negl}(\lambda)$$

G_5 : We next modify RO_2^2 to force the chain of hash values. On query $\text{aux} := (\text{ch}_1, \tilde{\mathbf{u}}, h_V)$ to RO_2^2 in line 7 of P.Check-QCSD, we do as follows:

- If ch_1 has no preimage, that is, if $\text{ch}_1 \notin \mathcal{V}_2^1$, then add ch_1 to $\mathcal{V}_2^1$. (Recall, ch_1 is computed after line 3 of HQC-PoP.P. using [1, Algorithm 4.34, line 5])
- If h_V has no preimage, that is, if $h_V \notin \mathcal{V}_1$, then add h_V to $\mathcal{V}_1$. (Recall, h_V is computed after line 3 of HQC-PoP.P. using [1, Algorithm 4.34, line 8])

Hence, $\Pr[W_4 \mid \overline{\text{Fail}_4}] = \Pr[W_5 \mid \overline{\text{Fail}_5}]$.

G_6 : Next, we modify output of RO_2^2 : On every new query $\text{aux} := (\text{ch}_1, \tilde{\mathbf{u}}, h_V)$ to RO_2^2 , if the h_{com} used to compute ch_1 is extractable and VOLE-consistent, then do as follows:

1. Sample ch_2 uniformly at random. If ch_2 is already queried to RO_4 , then it aborts.
2. If the ZK soundness check below fails, then abort.

- (a) Let $\alpha \xleftarrow{\$ \text{ch}_2} \mathbb{F}_{2^\lambda}^n$. (as in line 8 of P.Check-QCSD)

- (b) Compute $\mathbf{p} \in \mathbb{F}_{2^\lambda}^n$, which is a vector consisting of the degree d coefficients of $f_i(X)$ for $i = 0, \dots, n-1$ constructed from the extracted witness.
- (c) If $\mathbf{p} \neq \mathbf{0}$ but $\sum_{i=0}^{n-1} \alpha_i p_i = 0$, then the output “fail”. Otherwise, output “success”.
3. Otherwise, look up a preimage $\tilde{\mathbf{V}}$ of h_V . If a preimage does not exist, then abort.
4. Otherwise, return ch_2 .

Probability of finding a collision at step 1 is $\text{negl}(\lambda)$, If α is uniformly at random, then the probability that the check in step 2 fails is also $\text{negl}(\lambda)$ (due to exponential size of $\mathbb{F}_{2^\lambda}$). Hence,

$$\Pr[W_5 \mid \overline{\text{Fail}}_5] \leq \Pr[W_6 \mid \overline{\text{Fail}}_6] + \text{negl}(\lambda)$$

G₇: We next modify RO_2^3 as follows: On query $(\text{ch}_2, \llbracket a \rrbracket, \text{ctr})$ to RO_2^3 after line 5 of **HQC-PoP.P.** using [1, Algorithm 4.34, line 14], we do as follows:

- If ch_2 has no preimage, that is, if $\text{ch}_2 \notin \mathcal{V}_2^2$, then add ch_2 to $\mathcal{V}_2^2$.

We have $\Pr[W_6 \mid \overline{\text{Fail}}_6] = \Pr[W_7 \mid \overline{\text{Fail}}_7]$.

G₈: We modify the handling RO_2^3 (after line 5 of **HQC-PoP.P.** using [1, Algorithm 4.34, line 14]) on a query $(\text{ch}_2, \llbracket a \rrbracket, \text{ctr})$ as:

1. If ch_2 has no preimage, then abort. Otherwise, let $\text{aux} := (\text{ch}_1, \tilde{\mathbf{u}}, h_V)$, $\mathbf{t}$ be the preimage of ch_2 .
2. If h_V has no preimage, then abort.
3. Otherwise, extract $\mathbf{u}$ and $\mathbf{V}$, and define the witness $\mathbf{w}$.
4. Sample $\text{ch}_3 \xleftarrow{\$} \{0, 1\}^{\tau_0 \kappa_0 + \tau_1 \kappa_1 + w}$.
5. If $\mathbf{w}$ doesn't satisfy the constraints and the bad event occurs, then abort the game. The bad event is the event that, for uniformly random $\text{ch}_3 \in \{0, 1\}^{\tau_0 \kappa_0 + \tau_1 \kappa_1 + w}$ in step 3, a) ch_3 passes the verification test (T_{open} and 0^w check) and b) for uniformly random $\text{ch}_3 \in \{0, 1\}^{\tau_0 \kappa_0 + \tau_1 \kappa_1 + w}$, we have $\Delta = \Delta'$, where Δ is computed from ch_3 whereas Δ' is computed from the query $\llbracket a \rrbracket$ by the adversary.

Both the probabilities of a) and b) are $\text{negl}(\lambda)$ due to the size of the range from where ch_3 is sampled and size of $\mathbb{F}_{2^\lambda}$. Therefore,

$$\Pr[W_7 \mid \overline{\text{Fail}}_7] \leq \Pr[W_8 \mid \overline{\text{Fail}}_8] + \text{negl}(\lambda)$$

We finally discuss the probability $\Pr[\text{Fail}_8]$ and the reduction to the QCSD problem.

Evaluation of $\Pr[\text{Fail}_8]$.

The total advantage of the random oracle games is bounded by summing each of the $\Pr[\text{Fail}_8]$ in previous games due to union bound which all occur with $\text{negl}(\lambda)$ probability for polynomial time adversaries. Hence, $\Pr[\text{Fail}_8]$ is negligible.

Reduction to QCSD

We construct a reduction algorithm $\mathcal{B}$ using $\mathcal{A}$ in the final game G_8 conditioned on that $\overline{\text{Fail}}_8$. The reduction algorithm $\mathcal{B}$ that solves the QCSD problem (Definition 4) is defined as follows:

1. On given a random QCSD instance with public values $(\mathbf{H}, \mathbf{s})$ it should output $\mathbf{x}$ and $\mathbf{y}$ with $w_H(\mathbf{x}) = w_H(\mathbf{y}) = \omega$ such that $\mathbf{s} := \mathbf{x} + \mathbf{h} \cdot \mathbf{y}$, where $\mathbf{H} := [\mathbf{I}_n \quad \text{rot}(\mathbf{h})]$.
2. It sets $\text{vk} = (\mathbf{H}, \mathbf{s})$ and runs $\mathcal{B}$ in G_8 .
3. Finally, as $\mathcal{A}$ outputs (m^*, σ^*) as a forgery. If $\mathcal{A}$ wins and Fail_8 does not occur, then $\mathcal{B}$ extracts the witness $\mathbf{x}$ and $\mathbf{y}$ from the random oracle queries, and outputs $(\mathbf{x}, \mathbf{y})$ as solution to QCSD problem.

Since the simulation of $\mathcal{B}$ is perfect, if $\mathcal{A}$ wins the game, then $\mathcal{B}$ can extract the witness $(\mathbf{x}, \mathbf{y})$ from $\mathcal{A}$'s queries to the random oracles. Thus, we have

$$\Pr[W_8 \mid \overline{\text{Fail}}_8] \leq \text{AdvOW}_{\mathcal{B}}^{\text{QCSD}}.$$

□

From EUF-NMA to EUF-CMA security.

Once the EUF-NMA security is established then the proof for showing EUF-CMA security is exactly the same as in PERK specification [1]. Therefore, we only state the informal theorem statement below and refer the interested readers to [1] for the precise statement and proof.

Theorem D.5 (EUF-CMA security in the ROM). *Let $\mathcal{A}$ be a PPT adversary that breaks the EUF-CMA security of signature scheme derived from non-interactive HQC-PoP.P. Then there exists, another efficient adversary $\mathcal{B}$ that breaks the EUF-NMA security of signature scheme derived from non-interactive HQC-PoP.P.*

Proof. The proof of Theorem D.5 is identical to the proof of [1, Theorem 7.2, p.81]. □