

Ladon: Strengthening Confidential Computing by Decentralizing Trust in the Key Management Service via a Threshold KEM

Kiarash Sedghighadikolaei
University of South Florida
kiarashs@usf.edu

Thang Hoang
Virginia Tech
thanghoang@vt.edu

Attila A. Yavuz
University of South Florida
attilaayavuz@usf.edu

***Abstract*—The Key Management Service (KMS) plays a central role in confidential computing: it wraps the symmetric keys used to encrypt asset owners' sensitive data (*e.g.,* healthcare records, proprietary models), and later releases those keys to a trusted execution environment running the intended workload. Despite this critical role, current KMS deployments remain vulnerable to a single point of compromise: a breach of the KMS can expose the asset keys and, consequently, every asset they protect. To avoid this single point of compromise, a Post-Quantum (PQ) secure threshold Key Encapsulation Mechanism (KEM) can replace the traditional KMS, wherein the asset owner encapsulates the asset key under a single public key, while decapsulation is distributed across a committee of KMS servers. Unfortunately, existing PQ-secure threshold KEMs impose high communication overhead during asset-key decapsulation and produce large public keys and ciphertexts.**

In this work, we present Ladon, an efficient PQ-secure threshold KEM that addresses the limitations of prior work in communication during threshold decapsulation, public-key, and ciphertext sizes. Ladon builds on NIST's Module-Lattice Key Encapsulation Mechanism (ML-KEM) standard, whose underlying design principles enable more efficient parameter selection than other designs. These yield three prominent advantages: (i) Ladon's encapsulation and decapsulation are functionally equivalent to ML-KEM, inheriting its well-established security guarantees; (ii) Ladon's threshold protocol incurs lower communication overhead than the existing PQ-secure threshold KEMs, along with up to 8× smaller public keys and 5× smaller ciphertexts; and (iii) with Ladon, the asset owner encapsulates only once, regardless of the KMS committee size, making the threshold structure hidden to them. We also open-source a Rust implementation of Ladon.

1. Introduction

The Key Management Service (KMS) is central to confidential-computing deployments such as NVIDIA's Confidential Containers (CoCo) [1], Azure Confidential Computing [2], and AWS Nitro Enclaves [3]. An asset owner first encrypts sensitive data, such as confidential records or proprietary models, under a fresh symmetric asset key and asks the KMS to wrap that key under the KMS's own key [4]. The encrypted asset and wrapped asset key are then stored in the cloud. Later, when a workload inside a trusted execution environment (TEE) needs the asset, it sends a key-release request and hardware-rooted attestation to a key-release gateway [1], [5], [6]. After validating the attestation, the gateway forwards the request to the KMS, which unwraps the asset key and releases it into the TEE's protected memory for decryption and computation.

Current KMS implementations create a single point of trust: if the KMS is compromised, the wrapping key can be exposed, and therefore every asset key protected by it can be compromised. Asymmetric-key architectures reduce the attack surface during wrapping, since wrapping can be performed outside the KMS infrastructure [7] (*e.g.,* at the asset owner). However, the unwrapping key still remains with a single authority that controls every release. Hardware protections such as Hardware Security Modules (HSMs) do not fully remove this risk [8]: an HSM can ensure that the wrapping or decryption key never leaves the device, but it still exposes an unwrap interface to KMS-authorized software. If an adversary controls that software, it can submit valid unwrap requests and receive plaintext asset keys.

To remove this single point of trust, the unwrapping authority must also be distributed so that no single component can release a plaintext asset key on its own. A post-quantum (PQ)-secure threshold Key Encapsulation Mechanism (KEM) [9], [10], [11] provides this structure. The asset owner runs KEM encapsulation under a single public key to derive the asset key and an encapsulating ciphertext. Later, upon the TEE's request, a quorum of KMS committee members runs a distributed decryption protocol on the ciphertext and sends partial results to the TEE. The TEE securely combines these results and completes decapsulation to derive the asset key. Since no individual committee member holds the full decapsulation key, compromising one member does not expose asset keys. The PQ security further protects the encapsulations stored in the cloud against harvest-now-decrypt-later attacks [12], [13], similar to existing KMS use of quantum-resistant AES [13].

The design space for PQ-secure threshold KEMs remains limited. Current state-of-the-art constructions [14], [15] build PQ-secure threshold KEMs from threshold-friendly identity-based encryption. These schemes provide strong security, including indistinguishability under chosen-ciphertext attacks (IND-CCA) [16], meaning that an ad-

versary cannot learn the asset key even if it can observe decapsulation behavior on ciphertexts of its choice (*e.g.,* through implementation leakage). Their distributed decapsulation also tolerates malicious corruption of committee members that may deviate from the protocol. However, these guarantees come at a cost: large public keys and ciphertexts (*e.g.,* 450 KB in [14], reduced to 30 KB in [15]) and high online communication during key release. This overhead affects multiple parts of the distributed key-release path: ciphertexts are stored in the cloud and fetched by each KMS committee member on every release. In addition, IND-CCA-secure KEMs [14], [15], [17] require the public key during decapsulation, so it must also be available on the release path. Together, these costs increase both the cloud-storage footprint [18] and KMS response latency, which providers such as AWS identify as a deployment concern [13].

Other PQ-secure threshold KEMs [19], [20], [21], [22] target weaker security guarantees. These constructions, which are technically threshold versions of public-key encryption rather than full KEMs, do not provide IND-CCA security and are secure only against passive corruptions of committee members. Even under this weaker model, they have long faced practical limitations because they rely on techniques [23] that require significantly larger parameters to maintain both security and correctness. Recent works [19], [20] reduce this overhead, but they either rely on inefficient algebraic structures that still lead to large public keys and ciphertexts (*e.g.,* 35-110 KB), increasing storage overhead [18] and key-release latency [13], or propose extensions to more efficient settings whose security relies on conjectural assumptions not proven yet [20].

Confidential-computing deployments must defend against strong adversaries in practice [24], [25]. Can we replace the centralized KMS with a PQ-secure threshold KEM that provides IND-CCA security and malicious-committee security, while reducing public-key size, ciphertext size, and online communication compared with prior PQ-secure threshold KEMs?

1.1. Contributions of Ladon

We answer this question by introducing Ladon, a PQ-secure threshold KEM that decentralizes the KMS role in the confidential-computing key-release pipeline. Ladon builds on NIST's standardized Module-Lattice KEM (ML-KEM) [17], whose algebraic structure [26] enables compact public keys and ciphertexts, while inheriting its well-studied security guarantees.

To realize this design with ML-KEM, Ladon addresses a critical obstacle that prior PQ-secure threshold KEMs have avoided: directly distributing ML-KEM decapsulation across a KMS committee is inefficient because ML-KEM relies on the Fujisaki-Okamoto (FO) transform [27]. Unlike the BCHK transform [28] used by prior PQ-secure threshold KEMs [14], [15], which allows parts of decapsulation to be handled publicly without compromising security, FO requires decapsulation operations to be fully evaluated without revealing intermediate values; otherwise, IND-CCA security can be compromised. These operations include re-encryption, comparison, and symmetric-key derivation, all of which rely on hash and extendable-output functions (*e.g.,* SHAKE256 [29]) that are difficult to threshold efficiently and would require heavy multi-party computation (MPC) techniques (*e.g.,* garbled circuits [30]).

Ladon avoids this obstacle by exploiting a key property already present in confidential-computing deployments [1], [2]: the TEE is the endpoint of the key-release path. The TEE requests the asset key and ultimately uses it to decrypt the asset. This allows Ladon to split decapsulation across the existing trust boundary. The KMS committee performs only the threshold-friendly part of decapsulation, namely the linear decryption step, and sends partial results directly to the TEE. The TEE then combines these results inside protected memory and locally performs the threshold-unfriendly operations of the FO transform to derive the key. Under the threshold access structure, no corrupted committee member sees the reconstructed value, while the TEE's protected memory further shields the asset key. This design choice makes Ladon a threshold KEM-based solution specifically tailored to confidential computing deployments. By leveraging secure TEEs already present in these environments, Ladon achieves better performance than existing threshold KEM solutions [14], [15] in this setting, without aiming to serve as a general-purpose threshold KEM.

We summarize the main contributions of Ladon below.

• **Size-efficient Public Key and Encapsulation.** Ladon inherits ML-KEM's algebraic structure [26], which enables efficient parameter selection and compact public keys and ciphertexts at fixed security levels compared with prior works [14], [15], [20]. We instantiate Ladon at both 128-bit and 256-bit security. At 128-bit security, Ladon achieves a public-key size of 5.66 KB and a ciphertext size of 5.97 KB. Compared with [14], this reduces ciphertext size by $75\times$ and public-key size by $8.8\times$. Compared with [15], it reduces ciphertext size by $5\times$ and public-key size by $1.2\times$.

• **Efficient Distributed KMS-committee Protocol.** Ladon builds on recent thresholding techniques [31] that avoid heavy MPC tools such as garbled circuits [30], [32] by using simple arithmetic-friendly secret-sharing-based methods, in particular double sharing [33]. Since Ladon applies these techniques over a different algebraic structure than [31], we provide a full correctness analysis and security proof. For example, at 128-bit security, against the strongest-security competitors, and for a committee of three members, Ladon reduces end-to-end communication by $27.3\times$ compared with [14] and by $2.85\times$ compared with [15].

• **Security against Passive and Active Committee Corruptions.** Ladon provides two variants of its distributed protocol. The first is secure against passive corruption, in which corrupted KMS committee members follow the protocol, but the adversary learns their internal state. The second, stronger variant targets active corruption, in which corrupted members may deviate arbitrarily from the protocol. We provide full simulation-based security proofs for both variants.

• **Open-source Artifact.** We provide an open-source Rust implementation of Ladon, including both variants in two separate crates. For each variant, we provide a demo of the underlying KEM, a full demo of the threshold key-release pipeline, and benchmark scripts for each implemented operation in the new setting. We also provide the security-analysis scripts used to evaluate the underlying KEM and threshold protocol, which support our parameter selection. Our artifact is available at:

https://github.com/kiarashsedghigh/Ladon

2. Building Blocks

Ladon builds on two components: a PQ-secure KEM, which lets two entities establish a shared secret key, and secret sharing, which Ladon uses to distribute the KEM decapsulation procedure.

2.1. Key Encapsulation Mechanism

The Module-Lattice Key Encapsulation Mechanism (ML-KEM) [17], specified in NIST FIPS 203, is a NIST-standardized PQ-secure KEM based on the Module Learning With Errors (MLWE) assumption [26] (see Appendix A). It achieves IND-CCA security [16] by applying the Fujisaki-Okamoto (FO) transform [27] to its underlying public-key encryption scheme. We briefly summarize the ML-KEM operations below; full details are deferred to Appendix B.

ML-KEM operates over the polynomial ring $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^N+1)$ with modulus q. It uses two hash functions G and H, a key-derivation function KDF, and the Centered Binomial Distribution (CBD) to sample short elements.

In key generation, the recipient samples a public matrix $\mathbf{A}$ and short secret and error vectors $\mathbf{s}, \mathbf{e}$, publishes the public key $\mathbf{t} := \mathbf{A}\mathbf{s} + \mathbf{e}$, and keeps $\mathbf{s}$ secret. To encapsulate under $\mathbf{t}$, the sender samples a fresh seed m and uses $G(m \,\|\, H(\mathbf{t}))$ to derive a pre-key $\bar{K}$ and an encryption-randomness seed, where $\|$ denotes concatenation. This seed is used to sample short vectors and element $(\mathbf{r}, \mathbf{e}_1, e_2)$ and produce the ciphertext $c := (\mathbf{u}, v)$, where $\mathbf{u} := \mathbf{A}^T\mathbf{r} + \mathbf{e}_1$ and $v := \mathbf{t}^T\mathbf{r} + e_2 + \lfloor q/2 \rceil \cdot m$, where $\lfloor . \rceil$ denotes rounding to the nearest integer. The shared key is then set as $K := \mathsf{KDF}(\bar{K} \,\|\, c)$. To decapsulate, the recipient uses $\mathbf{s}$ to recover m' from $v - \mathbf{s}^T\mathbf{u}$, then uses $G(m' \,\|\, H(\mathbf{t}))$ to obtain the pre-key $\bar{K}'$ and the corresponding encryption-randomness seed. This seed is used to re-encrypt m' and obtain a candidate ciphertext c'. If c' matches c, the recipient outputs $\mathsf{KDF}(\bar{K}' \,\|\, c)$; otherwise, it outputs a pseudorandom value, following the FO transform [27].

ML-KEM also reduces communication by compressing the ciphertext (coefficient-wise) before transmission and decompressing it during decapsulation. If d_b bits are kept per coefficient x, we denote compression by $\mathsf{Compress}_q(x, d_b) := \lfloor (2^{d_b}/q) \cdot x \rceil$ and decompression by $\mathsf{Decompress}_q(x, d_b) := \lfloor (q/2^{d_b}) \cdot x \rceil$.

2.2. Secret Sharing

A linear secret sharing scheme splits a secret x into shares $[\![x]\!]_1, \ldots, [\![x]\!]_n$ held by n parties. The secret can be recovered from a qualified subset of shares using a reconstruction procedure, and linear operations on shares can be computed locally without interaction, such as addition or multiplication by public constants. Shares may be defined over a prime field $\mathbb{F}_q$ or over the ring $\mathbb{Z}_{2^k}$; when the modulus matters, we write $[\![x]\!]^q$ or $[\![x]\!]^{2^k}$.

Shamir secret sharing. A (t, n)-Shamir sharing [34] represents a secret $x \in \mathbb{F}_q$ as the constant term of a random degree-$(t-1)$ polynomial f. Party i receives the share $[\![x]\!]_i = f(\alpha_i)$ at a distinct nonzero $\alpha_i \in \mathbb{F}_q$. In the honest-majority setting [35], any t or more parties can recover x using Lagrange interpolation: $x = \sum_i L_i \cdot [\![x]\!]_i$, where $L_i = \prod_{j \neq i} \frac{\alpha_j}{\alpha_j - \alpha_i}$ is the Lagrange coefficient for party i.

Additive secret sharing. A secret $x \in \mathbb{Z}_m$ is additively shared as $[\![x]\!]_i \in \mathbb{Z}_m$ such that $x = \sum_i [\![x]\!]_i \pmod m$. In this setting (dishonest-majority [35]), reconstructing x requires all n shares.

Additive shares can also be authenticated using information-theoretic Message Authentication Codes (MACs). Following the SPDZ framework [36], each party i holds a triple $[\![x]\!]_i = (x_i, m_i, \alpha_i)$, where x_i is share of the value, m_i is its share of the MAC tag, and α_i is its share of a global MAC key α. These shares satisfy $\sum_i m_i = \alpha \cdot x$, where $\alpha = \sum_i \alpha_i$. While SPDZ is defined over a prime field, SPDZ$_{2^k}$ [37] extends the same idea to rings $\mathbb{Z}_{2^k}$ by defining shares and MACs over the larger ring $\mathbb{Z}_{2^{k+\lambda_s}}$, where λ_s is the statistical-security parameter and the global MAC key α lies in $\mathbb{Z}_{2^{\lambda_s}}$.

We also use batch MAC checking [37]. Instead of verifying the MAC tag of each opened value during the protocol, the protocol defers these checks and verifies them together in a single batch at the end. This is discussed more in Section 4.

Double sharing. A *double sharing* [38] keeps the same value shared under two different moduli. For moduli μ and μ' with μ' dividing μ ($\mu' \mid \mu$), a double sharing of $d \in \mathbb{Z}_{\mu'}$, written $[\![d]\!]^{\mu',\mu}$, is a tuple consisting of two linear secret sharings of d: one over $\mathbb{Z}_\mu$ and one over $\mathbb{Z}_{\mu'}$. In the ring R_q, a polynomial is double-shared by double-sharing each of its N coefficients separately.

3. System and Threat Models

This section presents the system model for Ladon in the confidential-computing key-release setting introduced in Section 1. Ladon replaces the centralized KMS role with a PQ-secure threshold KEM and focuses on three core entities: (i) an *asset owner*, (ii) a *KMS committee* that replaces the centralized KMS, and (iii) a *trusted execution environment*. We describe these components, explain how Ladon fits into common deployment models, and define the security threats that Ladon is designed to withstand.

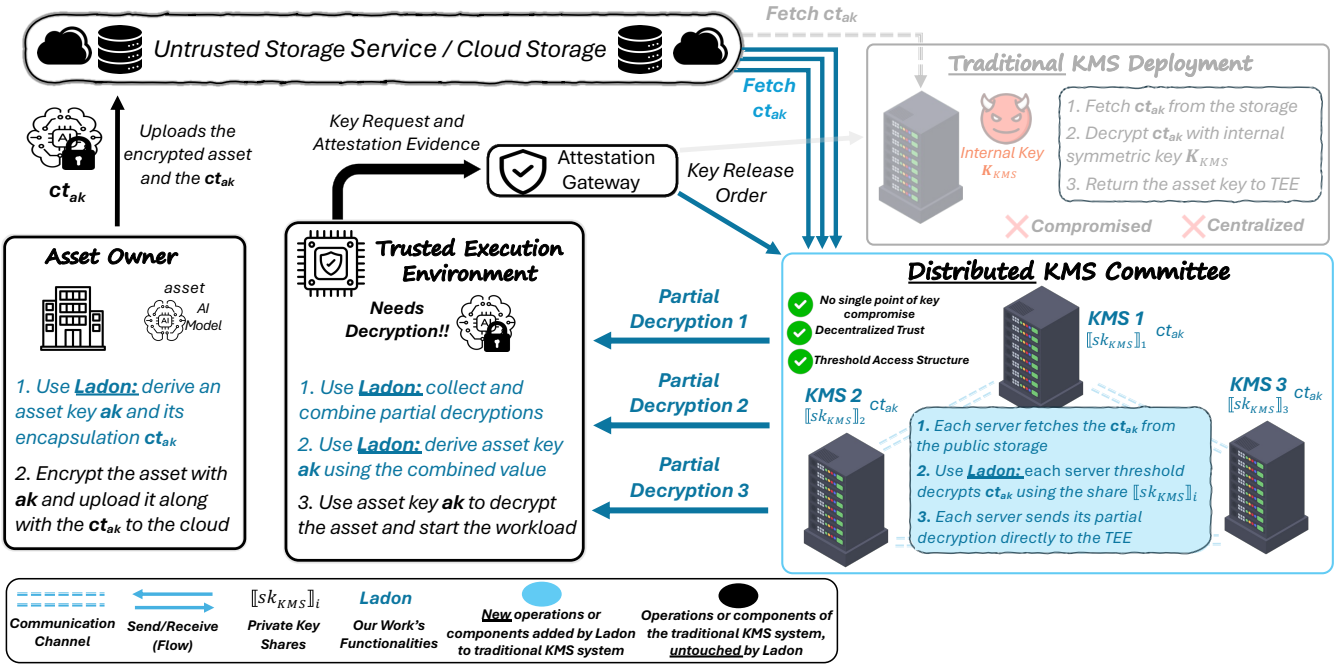

Figure 1: Ladon's decentralized KMS workflow, showing how Ladon integrates into prior deployment architectures. We use **Black** to denote operations that remain unchanged and **Blue** to denote the new operations Ladon introduces.

3.1. System Components

At a high level, Ladon modifies the existing KMS workflow by replacing centralized key release with a PQ-secure KEM whose decapsulation procedure is distributed. In particular, the asset owner uses the public encapsulation key to derive an asset key and an encapsulating ciphertext. Later, when a confidential workload within the TEE requests the asset key, the KMS committee runs a threshold protocol on the encapsulating ciphertext and sends partial results to the TEE, which can be used to further derive the asset key within the protected memory.

We define the core entities of Ladon and their operations below. Additionally, Figure 1 illustrates how Ladon fits within prior deployment models by highlighting the changes introduced when Ladon is used.

KMS Committee. A KMS committee consists of n independently operated members $\mathcal{KMS}_1, \ldots, \mathcal{KMS}_n$. Ladon defines two main operations involving the KMS committee:

- *KMS Committee Provision*: at deployment, a deployment authority generates the committee's KEM key pair $(\mathsf{pk}_{\mathcal{KMS}}, \mathsf{sk}_{\mathcal{KMS}})$, publishes the public key $\mathsf{pk}_{\mathcal{KMS}}$, and secret-shares the private key $\mathsf{sk}_{\mathcal{KMS}}$ among the committee members over authenticated and confidential channels. Consistent with established cloud provisioning practices [39], the authority's role is limited to this *one-time* setup, with no involvement in subsequent operations.
- *KMS Distributed Decryption*: upon receiving a key-release request whose hardware-rooted attestation has already been verified (*e.g.,* by key-release gateways [1], [5], [6]), the committee runs the threshold decryption protocol on the encapsulating ciphertext and sends partial results to the TEE for local derivations of the requested asset key.

Asset Owner. An asset owner ($\mathcal{AO}$) possesses a sensitive asset, such as confidential records or a proprietary model. The asset owner performs the following operations:

- *Owner Asset Key Derivation*: the owner runs Ladon's underlying KEM encapsulation procedure under the KMS committee's public key ($\mathsf{pk}_{\mathcal{KMS}}$) and obtains a fresh asset key (ak) together with an encapsulating ciphertext ($\mathsf{ct}_{\mathsf{ak}}$).

The owner uses the derived asset key ak to encrypt the bulk asset (*e.g.,* using AES-GCM [40]) and uploads both the asset ciphertext and $\mathsf{ct}_{\mathsf{ak}}$ to cloud storage for later retrieval by a confidential workload.

Trusted Execution Environment. A trusted execution environment ($\mathcal{TEE}$) is a hardware-backed runtime that executes the workload consuming the asset. At runtime, the $\mathcal{TEE}$ produces a hardware-rooted attestation of its identity and runtime state, which is sent along with its key-release request for a particular asset key.

- *TEE Asset Key Derivation*: after receiving responses from the KMS committee members (running the threshold decryption protocol), the $\mathcal{TEE}$ combines them locally to derive the requested asset key ak.

Real deployments include additional entities beyond the three described above, such as a key-release gateway [1], an attestation service, and the infrastructure provider operating the host hardware. The three entities described above are common to confidential-computing key-release deployments [1], [2] and are the ones most directly involved in addressing

KMS centralization. Therefore, Ladon fits into the existing key-release path without changing the other entities' roles.

3.2. Threat Model

We consider a quantum adversary focused on the asset-key release path, whose goal is to recover the asset key ak and break its confidentiality. The adversary can access all cloud-stored contents, including the encapsulating ciphertext $\mathsf{ct}_{\mathsf{ak}}$ and the bulk-encrypted asset, and may retain them indefinitely for a harvest-now-decrypt-later attack [12].

Communication Assurances. Following established cloud KMS security practices [41], [42], [43], Ladon employs TLS with PKI-based certificate authentication for communication between the TEE and individual KMS committee nodes. PKI lets the TEE verify each node's identity and prevent impersonation, while authenticated TLS ensures the confidentiality and integrity of exchanged messages, protecting against man-in-the-middle attacks.

Deployment Authority. Ladon assumes that the deployment authority is fully trusted during the *one-time* system initialization, following conventional cloud KMS provisioning practices [39], where a tenant or administrator performs the initial setup. Specifically, the authority is trusted to generate the KMS committee's key pair, securely distribute private-key shares among committee members, and publish the corresponding public key. After distributing the shares, the authority securely erases the original plaintext secret key to prevent its exposure if the authority is later compromised.

Corruption of KMS Committee. Ladon considers two types of adversaries that can corrupt the KMS committee. The first is a passive adversary: corrupted members follow the specified protocol, but the adversary learns their full internal state, including their shares of $\mathsf{sk}_{\mathcal{KMS}}$ and all messages exchanged during KMS Distributed Decryption. The second is a malicious adversary: corrupted members may additionally deviate arbitrarily from the protocol and send malformed messages. The adversary may try to learn information about the secrets held by non-corrupted KMS committee members during protocol execution, with the goal of recovering the decapsulation secret.

Integrity and Confidentiality of the TEE. We assume the $\mathcal{TEE}$'s hardware-rooted isolation holds against software-level adversaries, including a privileged operating system, hypervisor, and co-tenants on the same host. The adversary cannot inspect or modify the $\mathcal{TEE}$'s protected memory and cannot forge its remote attestation.

Attestation Gateway. Ladon assumes the attestation gateway in the system (see Figure 1) is always trusted, similar to real-world deployments such as NVIDIA's CoCo architecture [1]. The KMS committee nodes rely on the gateway's authorization decisions to proceed with the key release, without independently verifying the attestation evidence.

4. Our Proposed Ladon

This section presents the core operations of Ladon defined in Section 3: (i) KMS Committee Provision, (ii) Owner Asset Key Derivation, (iii) KMS Distributed Decryption, and (iv) TEE Asset Key Derivation. Together, these operations instantiate a PQ-secure threshold KEM that replaces the centralized KMS with a distributed committee. We describe each operation at the protocol-flow level and defer implementation and concrete instantiations to Section 6.

4.1. KMS Committee Provision

KMS Committee Provision, presented in Algorithm 1, is the setup operation that establishes the KMS committee. A deployment authority runs this operation before any asset is wrapped and then plays no further role.

Internally, this operation generates a fresh KEM key pair ($\mathsf{pk}_{\mathcal{KMS}}$, $\mathsf{sk}_{\mathcal{KMS}}$) using the standard ML-KEM key-generation procedure and splits the secret decapsulation key into shares $[\![\mathsf{sk}_{\mathcal{KMS}}]\!]_i$ under a linear secret-sharing scheme. The deployment authority publishes the public encapsulation key $\mathsf{pk}_{\mathcal{KMS}}$ for use by the asset owner(s) and delivers each share to its corresponding committee member over an authenticated secure communication channel (*e.g.,* TLS).

Algorithm 1 KMS Committee Provision

Input: Number of $\mathcal{KMS}$ committee members n.
Output: $\mathcal{KMS}$ committee public key $\mathsf{pk}_{\mathcal{KMS}}$ and shares of the committee secret key.

1: Sample a matrix $\mathbf{A}$ over $\mathcal{R}_q^{k\times k}$.
2: Sample (CBD) short vectors $\mathbf{s}, \mathbf{e} \in \mathcal{R}_{\eta_1}^k$ with coefficients drawn from $[-\eta_1, \eta_1]$.
3: Compute the public key vector $\mathbf{t} := \mathbf{A}\mathbf{s} + \mathbf{e}$.
4: Set the committee public key $\mathsf{pk}_{\mathcal{KMS}} := (\mathbf{A}, \mathbf{t})$ and the committee secret key $\mathsf{sk}_{\mathcal{KMS}} := \mathbf{s}$.
5: Split $\mathsf{sk}_{\mathcal{KMS}}$ into n shares $([\![\mathsf{sk}_{\mathcal{KMS}}]\!]_1, \ldots, [\![\mathsf{sk}_{\mathcal{KMS}}]\!]_n)$ under a linear secret-sharing scheme.
6: Deliver each share $[\![\mathsf{sk}_{\mathcal{KMS}}]\!]_i$ to its corresponding member, $\mathcal{KMS}_i$, over an authenticated, confidential channel.

4.2. Owner Asset Key Derivation

Ladon's Owner Asset Key Derivation operation, presented in Algorithm 2, is the encapsulation procedure that the asset owner runs locally under the KMS committee's public key $\mathsf{pk}_{\mathcal{KMS}}$. This operation is independent of the committee's threshold structure and the number of committee members, making the distributed trust architecture hidden to the asset owner(s). By running this operation, the owner obtains an asset key ak and an encapsulating ciphertext $\mathsf{ct}_{\mathsf{ak}}$. The owner then uses ak as the symmetric key to encrypt the bulk asset (*e.g.,* via AES-GCM), and uploads both the asset ciphertext and $\mathsf{ct}_{\mathsf{ak}}$ to cloud storage.

Internally, this step follows the ML-KEM design (see Section 2). The asset owner samples a fresh seed m and uses it to derive both a pre-key and an encryption-randomness seed for producing the encapsulating ciphertext. The asset key ak is then derived by applying a key-derivation function to the pre-key and the ciphertext. Deriving the encryption

randomness deterministically from m is essential for IND-CCA security, since decapsulation must use the same randomness to check whether the ciphertext is valid.

Algorithm 2 Owner Asset Key Derivation

Input: Committee public key $\mathsf{pk}_{\mathcal{KMS}} = (\mathbf{A}, \mathbf{t})$.
Output: Asset key ak and encapsulating ciphertext $\mathsf{ct}_{\mathsf{ak}}$.

1: Sample a random seed m.
2: Derive pre-key $\bar{K}$ and randomness seed r from the hash evaluation $G(m \,\|\, H(\mathsf{pk}_{\mathcal{KMS}}))$.
3: Sample (CBD) short vector $\mathbf{r} \in \mathcal{R}_{\eta_1}^k$ with coefficients in $[-\eta_1, \eta_1]$, and sample vector $\mathbf{e}_1 \in \mathcal{R}_{\eta_2}^k$ and e_2 with coefficients in $[-\eta_2, \eta_2]$ using seed r.
4: Compute vector $\mathbf{u} := \mathsf{Compress}_q(\mathbf{A}^T\mathbf{r} + \mathbf{e}_1,\ d_u)$.
5: Compute $v := \mathsf{Compress}_q\big(\mathbf{t}^T\mathbf{r} + e_2 + \lfloor q/2 \rceil \cdot m,\ d_v\big)$.
6: Set the encapsulating ciphertext as $\mathsf{ct}_{\mathsf{ak}} := (\mathbf{u}, v)$.
7: Derive the asset key as $\mathsf{ak} := \mathsf{KDF}(\bar{K} \,\|\, H(\mathsf{ct}_{\mathsf{ak}}))$.

4.3. KMS Distributed Decryption

KMS Distributed Decryption, shown in Figure 2, is the threshold protocol that the KMS committee runs on the encapsulating ciphertext $\mathsf{ct}_{\mathsf{ak}}$. It is triggered when the $\mathcal{TEE}$ needs a specific asset key: the $\mathcal{TEE}$ sends a request with hardware-rooted attestation, which an attestation gateway verifies and forwards to each committee member. The committee members retrieve $\mathsf{ct}_{\mathsf{ak}}$ from cloud storage, and each uses its share $[\![\mathsf{sk}_{\mathcal{KMS}}]\!]_i$ in a joint computation to produce a partial decryption result. Each member then sends its partial result directly to the $\mathcal{TEE}$, which combines them and completes asset-key derivation locally.

Recall from Section 2 that the underlying decryption step is linear in the secret key: $v - \mathsf{sk}_{\mathcal{KMS}}^T\mathbf{u}$ produces $\mu \cdot m + e$, where $\mu = \lfloor q/2 \rceil$ and e is a small accumulated noise term. The goal of KMS Distributed Decryption is to remove e so that the committee members obtain shares of m. To do this, the committee isolates e from $\mu \cdot m + e$ and subtracts it using a double-shared random polynomial mask d [31]. At a high level, the idea is simple. The committee first reduces its shares of $\mu \cdot m + e$ modulo μ to isolate the noise term e. It then opens the masked noise $e + d$ using the shares of d modulo μ; since d is random, this opening reveals nothing about e. Finally, each member subtracts $e + d$ from $\mu \cdot m + e$ and cancels the mask using its share of d under the second modulus q, leaving a share of $\mu \cdot m$. When $q/2$ is not an integer, Ladon uses modulus switching in Step 2 (with new q' and μ') to facilitate the isolation of the noise term e. We discuss the full correctness argument in Section 5.1.

The current Shamir-based distributed decryption protocol tolerates server failures or unavailability. Key release proceeds as long as at least $t + 1$ of the n servers remain available, and present servers can reshare the secret.

4.4. TEE Asset Key Derivation

Ladon's TEE Asset Key Derivation operation, presented in Algorithm 3, is run by the $\mathcal{TEE}$ after it receives partial decryption results from the KMS committee. This operation runs inside the $\mathcal{TEE}$'s protected memory, where the $\mathcal{TEE}$ combines the partial results, recovers the seed m, and derives the asset key ak to further access the asset in plaintext.

KMS Distributed Decryption Protocol

Input of each KMS committee member: the encapsulating ciphertext $\mathsf{ct}_{\mathsf{ak}} := (\mathbf{u}, v)$, share $[\![\mathsf{sk}_{\mathcal{KMS}}]\!]_i$ and ℓ double-shared polynomials $[\![d_1]\!]_i^{\mu',q'}, \ldots, [\![d_\ell]\!]_i^{\mu',q'}$ where $q' < q$ is an even modulus and $\mu' = q'/2$
Output: each $\mathcal{KMS}_i$ outputs a partial decryption of $\mathsf{ct}_{\mathsf{ak}}$, sent to the $\mathcal{TEE}$.

Each $\mathcal{KMS}_i$ converts their linear key share into an additive share and participates in the following protocol.

1) **Linear decryption (local).** $\mathcal{KMS}_i$ locally computes $[\![w]\!]_i^q := v - [\![\mathsf{sk}_{\mathcal{KMS}}^T]\!]_i\mathbf{u} \pmod q$.
2) **Modulus switch (local).** $\mathcal{KMS}_i$ rescales the share $[\![w]\!]_i^q$ from $\mathbb{Z}_q$ down to $\mathbb{Z}_{q'}$ by computing $[\![w']\!]_i^{q'} := \lfloor (q'/q) \cdot [\![w]\!]_i^q \rceil \pmod{q'}$, obtaining a share of $\mu' \cdot m + e'$, where e' is the total accumulated noise from the linear decryption and the rescaling step.
3) **Isolate the error (local).** $\mathcal{KMS}_i$ reduces its share modulo μ' by computing $[\![e']\!]_i^{\mu'} := [\![w']\!]_i^{q'} \bmod \mu'$, obtaining a share of the accumulated noise e' alone.
4) **Mask the error and open (interactive).** For each $j \in \{1, \ldots, \ell\}$, $\mathcal{KMS}_i$ computes $[\![\tilde{e}_j]\!]_i^{\mu'} := [\![e']\!]_i^{\mu'} + [\![d_j]\!]_i^{\mu'}$, and the committee jointly opens the masks of the error $\tilde{e}_1, \ldots, \tilde{e}_\ell$ in the clear to all members.
5) **Obtain share of the seed (local).** Each $\mathcal{KMS}_i$ computes $[\![(\mu' \cdot m)_j]\!]_i^{q'} := [\![w']\!]_i^{q'} - \big(\tilde{e}_j - [\![d_j]\!]_i^{q'}\big)$, for each $j \in \{1, \ldots, \ell\}$, obtaining ℓ shares of $(\mu' \cdot m)_j$ over $\mathbb{Z}_{q'}$, which it securely sends to the $\mathcal{TEE}$ as its partial decryption.

Figure 2: Ladon's KMS Distributed Decryption protocol

Each partial decryption contains shares of ℓ polynomials, each intended to reconstruct to $\mu' \cdot m$. We discuss the role of ℓ in Section 5.1. The $\mathcal{TEE}$ first combines the corresponding shares across committee members, obtaining ℓ candidate values. It then takes a coefficient-wise majority across these candidates to obtain a single value $\mu' \cdot m$, from which it extracts the seed m. Once the seed is recovered, the $\mathcal{TEE}$ completes decapsulation as in ML-KEM, which we discussed in Section 2.

4.5. Active-Security Variant of Ladon

The KMS Distributed Decryption protocol presented in the previous section achieves passive security. For active security, Ladon must ensure that secret and intermediate shared values remain valid, or authenticated, so that corrupted members cannot deviate undetected. The main obstacle is the modulus-switch step: rescaling shares by q'/q

Algorithm 3 TEE Asset Key Derivation

Input: Encapsulating ciphertext $\mathsf{ct}_{\mathsf{ak}}$, committee public key $\mathsf{pk}_{\mathcal{KMS}}$, and the partial decryptions from the committee.
Output: Asset key ak.

1: Recover ℓ instances $(\mu' \cdot m)_1, \ldots, (\mu' \cdot m)_\ell$ by combining (adding) their corresponding shares.
2: Take the coefficient-wise majority of $(\mu' \cdot m)_1, \ldots, (\mu' \cdot m)_\ell$ to obtain a single $\mu' \cdot m$.
3: Recover seed m by dividing each coefficient of $\mu' \cdot m$ by μ' and rounding to the nearest integer (0 or 1).
4: Obtain pre-key $\bar{K}'$ and encryption-randomness seed r' from $G(m \,\|\, H(\mathsf{pk}_{\mathcal{KMS}}))$.
5: Re-encrypt m under $\mathsf{pk}_{\mathcal{KMS}}$ with randomness r' to obtain ciphertext $\mathsf{ct}'_{\mathsf{ak}}$.
6: If $\mathsf{ct}_{\mathsf{ak}}$ and $\mathsf{ct}'_{\mathsf{ak}}$ are equal derive the asset key as $\mathsf{ak} := \mathsf{KDF}(\bar{K}' \,\|\, H(\mathsf{ct}_{\mathsf{ak}}))$. Otherwise, sample a fresh seed z and set $\mathsf{ak} := \mathsf{KDF}(z \,\|\, H(\mathsf{ct}_{\mathsf{ak}}))$.

involves rounding, which is not an exact linear operation. Checking this rounding directly without revealing values would introduce significant overhead. To avoid this issue, Ladon instantiates the active-secure variant over the ring $\mathbb{Z}_{2^k}$, with $q = 2^k$ fixed from KMS Committee Provision onward. This is because the plaintext modulus is 2, and hence, no modulus switching is necessary to further isolate the noise term. In this setting, additive shares are authenticated using information-theoretic MACs, as described in Section 2; linear operations on the shares remain unchanged and are applied in the same way to the MAC shares.

In both the passive and active-secure variants of Ladon, the Owner Asset Key Derivation operation performed by the asset owner and the TEE Asset Key Derivation operation performed by the $\mathcal{TEE}$ remain algorithmically unchanged from the KEM perspective. We discuss the correctness and security of both variants in Section 5.

Avoiding MAC-Key Leakage Vulnerability. A subtle issue arises in the active-secure, dishonest-majority setting when the MAC-check protocol is executed concurrently or across multiple threads. As shown by Kyster *et al.* [44], an adversary running an active attack against one thread can extract the global MAC key α and reuse it in another thread to forge valid openings. Naively running a MAC check for each opened polynomial during partial decryption would expose this attack vector, since the committee opens ℓ polynomials in parallel. Instead, Ladon defers all checks to the final stage of the protocol and aggregates them: rather than running ℓ separate MAC checks, the committee collects the coefficients of all ℓ opened polynomials into a single aggregated value and performs one MAC check.

Concretely, this aggregation uses public random values known as agreed-upon random values (AGVs). Let coeff_{ij} denote the j-th coefficient of the i-th opened polynomial, for $i \in [\ell]$ and $j \in [N]$, and let agv_{ij} denote the AGV associated with coeff_{ij}. Each KMS committee member $\mathcal{KMS}_k$ computes $\sum_{i=1}^{\ell} \sum_{j=1}^{N} \mathsf{agv}_{ij} [\![\mathsf{mac}(\mathsf{coeff}_{ij})]\!]_k - \left(\sum_{i=1}^{\ell} \sum_{j=1}^{N} \mathsf{agv}_{ij}, \mathsf{coeff}_{ij}\right) [\![\alpha]\!]_k$, where $[\![\mathsf{mac}(\mathsf{coeff}_{ij})]\!]_k$ is the MAC share corresponding to coeff_{ij} held by $\mathcal{KMS}_k$, and $[\![\alpha]\!]_k$ is its share of the global MAC key. Each KMS committee member then broadcasts the computed value and verifies that the sum of all values is zero. If the check passes, all partial openings during the protocol execution are valid.

Compared to the passively secure variant that tolerated server unavailability, Ladon's actively secure variant offers stronger security guarantees, using MACs to detect malicious behavior and abort key release upon detection.

4.6. Key Resharing, Rotation, and Revocation

Long-term deployments require key-management operations [45], [46] to limit exposure from compromised shares, replace aging or compromised keys, and retire keys that should no longer be used. Ladon supports these through key-share resharing, key rotation, and key revocation.

Proactive resharing can refresh the KMS committee's private-key shares after each key-release round without changing the underlying private key, reducing the risk of key exposure from accumulated share compromises. Key rotation, which generates a new KMS key pair, can be supported without re-encrypting assets using a key hierarchy. The asset owner uses the encapsulated key as a key-encryption key (KEK) to wrap a random asset key ak with AES-GCM, storing the wrapped key alongside the encapsulation ciphertext and encrypted asset. During key release, Ladon recovers the KEK through threshold decryption and the FO transform within the TEE, then unwraps ak to decrypt the asset. Upon rotation, only ak is rewrapped under a new KEK encapsulated using the new $\mathsf{pk}_{\mathcal{KMS}}$. This requires no changes to Ladon, adding only an AES-GCM ciphertext (approximately 44 bytes) to storage and one AES-GCM decryption within the TEE during key release. Key revocation can also be supported without changing Ladon's design. Once a key is revoked, the KMS committee switches to newly provisioned keys and no longer uses the revoked private key shares.

5. Analysis of Ladon

We analyze Ladon along two dimensions. First, we show correctness: the distributed protocol allows the $\mathcal{TEE}$ to recover the seed originally encrypted by the asset owner and, in turn, derive ak. Second, we show that Ladon is secure in three senses: (i) the encapsulations produced by Ladon are secure against quantum adversaries; (ii) if the original ciphertext is adaptively corrupted, the underlying KEM still preserves its strong security notion, namely IND-CCA; and (iii) the passive and active variants of Ladon's threshold protocol are secure against their corresponding adversaries. We discuss each security guarantee in turn and defer formal details to Appendix D.

5.1. Correctness of Asset Key Recovery

Recall that recovering the asset key is a two-step process. We analyze these two steps separately as follows:

Distributed Decryption Correctness. The first step, the threshold decryption from Section 4.3, succeeds when the joint opening avoids a modular wrap-around: the committee opens $e' + d \bmod \mu'$, where $d \in [-\mu'/2, \mu'/2)$ is a fresh double-shared mask and e' is the per-coefficient noise accumulated from the ciphertext and threshold operations, with $|e'|$ denoting its maximum absolute value across all coefficients. Recovery succeeds only when $e' + d$ does not wrap around modulo μ', so that, in the next step, subtracting the shares of d under the second modulus q' cancels the mask and isolates the noise. The per-execution success probability is bounded by $p_{\mathsf{dec\text{-}succ}} \geq 1 - N \cdot |e'|/\mu'$ across the N coefficients of the polynomial e'. We provide a detailed correctness analysis of Ladon's variants in Appendix C.

Following the analysis of [31], Ladon amplifies this success probability by running ℓ parallel openings of the masked noise, with the $\mathcal{TEE}$ taking the coefficient-wise majority during the TEE Asset Key Derivation operation. This raises the overall success probability to at least $\sum_{i=\lfloor \ell/2 \rfloor + 1}^{\ell} \binom{\ell}{i} \cdot p_{\mathsf{dec\text{-}succ}}^{i} \cdot (1 - p_{\mathsf{dec\text{-}succ}})^{\ell - i}$, which approaches 1 exponentially in ℓ whenever $p_{\mathsf{dec\text{-}succ}} > 1/2$.

Asset Key Extraction. Once the seed is correctly recovered, the $\mathcal{TEE}$ follows the ML-KEM's decapsulation to derive ak: it derives the encryption randomness from the seed, re-encrypts under the committee public key, and derives the final key based on whether the resulting ciphertext matches $\mathsf{ct}_{\mathsf{ak}}$. This step is faithfully executed entirely inside the $\mathcal{TEE}$'s protected memory.

5.2. Security Analysis

Ladon's security rests on two pillars. The first is the computational hardness of the underlying KEM, which protects both the asset key ak and the committee's secret key $\mathsf{sk}_{\mathcal{KMS}}$ against adversaries, including quantum ones, that observe only public values [17]. The second is the threshold decryption protocol, which ensures that running KMS Distributed Decryption over shares of $\mathsf{sk}_{\mathcal{KMS}}$ reveals no more than a centralized execution would, even when a subset of committee members is corrupted.

Underlying KEM. Ladon's KEM functionality is algorithmically identical to ML-KEM. As a result, Ladon inherits ML-KEM's full computational security guarantees against quantum adversaries. Recovering ak from the encapsulating ciphertext $\mathsf{ct}_{\mathsf{ak}}$, or recovering $\mathsf{sk}_{\mathcal{KMS}}$ from the public key $\mathsf{pk}_{\mathcal{KMS}}$, is as hard as solving the underlying MLWE problem [26]. In Section 7, we discuss parameter selection so that both tasks remain computationally hard.

Furthermore, since Ladon's decapsulation follows ML-KEM and incorporates the FO transform, it inherits IND-CCA security, a strong adversarial notion that guarantees that the adversary cannot learn anything about the asset key even if it can observe the behavior of the decapsulation procedure on ciphertexts of its choice.

If the TEE's security is compromised, an adversary may access intermediate values during key release, including the plaintext output of threshold decryption, potentially undermining the IND-CCA security guarantees of the FO transform [27], [47]. In contrast, general-purpose threshold KEMs [14], [15] use the BCHK transform [28], which allows certain decapsulation operations to be performed publicly without compromising security. However, Ladon targets confidential computing infrastructure, where TEE security is explicitly assumed in its threat model.

Threshold Decryption Protocol. Ladon's operation of KMS Distributed Decryption replaces centralized decapsulation with a joint computation on shares of $\mathsf{sk}_{\mathcal{KMS}}$.

The security argument follows the standard simulation-based approach. We show that the joint view of corrupted members during KMS Distributed Decryption is statistically indistinguishable from their view in a centralized execution, where a trusted party applies $\mathsf{sk}_{\mathcal{KMS}}$ to $\mathsf{ct}_{\mathsf{ak}}$ and sends the result to the $\mathcal{TEE}$. The statistical gap between the two executions is bounded by Ladon's decryption-failure probability (Section 5.1). In the active-security setting, the same indistinguishability holds, with the additional guarantee that any deviation by a corrupted member is detected except with probability $2^{-\lambda_s + \log(\lambda_s + 1)}$ for the statistical-security parameter λ_s, in which case the protocol aborts.

We present the full security proof for Ladon's variants, along with the ideal-functionality definition and simulator construction, in Appendix D.

6. Implementation of Ladon

We implemented Ladon in Rust, covering both the passive and active-secure variants. The implementation builds the entire threshold machinery from scratch and integrates it directly with the underlying KEM arithmetic. For the KEM functionality, Ladon builds on an existing Rust ML-KEM library[1] to implement the KEM-related operations used in KMS Committee Provision, Owner Asset Key Derivation, and TEE Asset Key Derivation.

Symmetric Primitives and Seeds. Ladon instantiates the symmetric primitives following the ML-KEM specification using the SHA3 family [29]: H is SHA3-256, G is SHA3-512, KDF is SHAKE256, and the matrix and short-vector samplers (CBD-based) are driven by SHAKE-based XOF streams [29]. To keep public keys compact, the public matrix $\mathbf{A}$ is not stored; instead, Ladon stores only a 32-byte seed and regenerates $\mathbf{A}$ on demand by reseeding the XOF. This further reduces the public-key size in Ladon.

KEM Arithmetic. For KEM arithmetic over the polynomial ring, Ladon considers two settings. The passive variant is parameterized over an NTT-friendly prime modulus (detailed in Section 7), so Ladon uses the standard negacyclic NTT to convert polynomial multiplications into coefficient-wise products in the evaluation domain.

The active variant requires a power-of-two modulus (Section 4.5), so the NTT no longer applies. Instead, Ladon expresses each polynomial multiplication as a dense matrix-vector product: the multiplicand is unrolled into an $N \times N$

1. https://github.com/conorpo/kemkem.git

negacyclic matrix and multiplied by the coefficient vector of the other operand [48]. The implementation uses Rust's native `i128` type, which is supported on modern 64-bit CPUs and provides enough headroom to accumulate length-N inner products over coefficients of $k + \lambda_s$ bits without intermediate reduction. It also uses the `Array2` dense-array type from the `ndarray` crate for cache-friendly, SIMD-amenable multiplications.

Linear Secret Sharing Instantiation. Ladon instantiates the linear secret sharing used in Section 4 in two ways. For the passive variant, Ladon uses Shamir secret sharing with $n = 2t + 1$ over the prime field $\mathbb{Z}_q$. For the active variant, Ladon uses additive secret sharing over $\mathbb{Z}_{2^k}$, with the committee secret-key shares and their corresponding MAC shares lifted to $\mathbb{Z}_{2^{k+\lambda_s}}$ for the statistical security parameter λ_s. We refer to Section 4.5 for a discussion of the modulus-switching obstacle, which limits certain instantiations (*e.g.,* Shamir secret sharing) in the active-security setting.

7. Evaluation

In this section, we evaluate Ladon from several perspectives. First, we evaluate the performance of its underlying KEM operations, including the storage affected by private and public-key and ciphertext sizes. Second, we evaluate the distributed threshold protocol in both variants: the passive and active variants. For the threshold protocol, we measure per-member performance, communication overhead, round complexity, and storage for each KMS committee member.

7.1. Experimental Setup

Software. We implemented Ladon fully using the Rust programming language (1.94 toolchain).

Hardware. All experiments were run on an Ubuntu 26.04 machine, equipped with an Intel Core i9-11900K CPU, 64 GB RAM, and a 1 TB SSD.

Network. We emulate two network-latency settings [49]: a low-latency setting with 1 ms one-way latency, modeling committee members within a single cloud region across availability zones, and a high-latency setting with 15 ms one-way latency, modeling members distributed across regions within a continent. Bandwidth is fixed at 1 Gbps.

Parameter Selection. Ladon's parameter selection has two parts. The first selects the parameters of the underlying KEM, which determine the computational security of the encapsulating ciphertext and of secret-key recovery. We instantiate Ladon at two main security levels [50]: 128-bit security and 256-bit security, as shown in Table 1.

Concrete security estimates are computed using the CRYSTALS security estimator[2] and the Lattice Estimator[3], the same tools used to derive ML-KEM's parameters.

The second part selects the threshold-protocol parameters, which are reported concretely in later sections: the success-amplification parameter ℓ and the statistical-security parameter λ_s. This selection follows the correctness analysis in Section 5.1. To increase the success probability of threshold decryption, the ratio between the accumulated noise e' and the modulus μ' must be small; otherwise, success must be amplified through more repetitions, that is, a larger ℓ, which increases communication overhead. The parameters in Table 1 are chosen so that the accumulated noise remains small throughout the computation; detailed bounds on the total noise are given in Appendix C. For the same reason, Ladon does not aggressively compress the ciphertext, particularly in the choice of d_v compared with ML-KEM [17], since stronger compression adds noise and reduces the success probability of the threshold protocol.

TABLE 1: Ladon parameter sets[†].

Security	k	η_1	η_2	d_u	d_v
128	6	2	2	28	23
256	10	2	2	28	26

[†] We use $N = 256$, which allows encrypting 256-bit seeds. For the active variant, we set $q = 2^{30}$. For the passive variant, we use $q = 687659009$, a 30-bit prime with primitive root 174453379. In the distributed protocol, we set $q' = 2^{\lfloor \log q \rfloor}$. These parameters ensure that the underlying KEM's decryption-failure probability ranges from 2^{-131} to 2^{-107}.

Comparison with State of the Art. In the following sections, we compare the main processing stages of Ladon with prior works. This comparison highlights the strengths and weaknesses of these schemes when adapted to confidential computing to address vulnerabilities arising from centralized KMS compromise. The results therefore do not constitute a general comparison of standalone threshold KEMs. For a fair comparison, we compare the strongest variant of Ladon, namely the active-secure variant, against prior active-secure PQ-secure threshold KEMs by Lapiha *et al*. [14] (LP) and Boudgoust *et al*. [15] (BDLP), including the robust variant BDLP-R [15], at both 128-bit and 256-bit security. We also compare the passive variant of Ladon against the passive-secure constructions of Pilvi [19] and Micciancio *et al*. [20] (MS), whose thresholded KEMs do not provide IND-CCA security, while Ladon still inherits IND-CCA security from its underlying KEM. We note that [19], [20] provide parameters only for the 128-bit security level.

Regarding computational performance, none of our counterpart schemes [14], [15], [19], [20] provide publicly available open-source implementations. Therefore, the performance numbers reported for these schemes are estimates derived from our own evaluations, conducted as conservatively as possible to ensure a reasonably fair comparison.

7.2. Asset Owner: Wrapping Cost

Recall that the asset owner runs this operation to derive a fresh asset key and its encapsulating ciphertext. The main owner-side costs are encapsulation time, the size of the public key $\mathsf{pk}_{\mathcal{KMS}}$ used for wrapping, and the size of $\mathsf{ct}_{\mathsf{ak}}$ uploaded with the encrypted asset.

2. https://github.com/pq-crystals/security-estimates
3. https://github.com/malb/lattice-estimator

Table 2 shows the active-secure comparison. The original LP construction has very large owner-side overhead at 128-bit security, requiring a modulus of around 100 bits. Compared with LP, Ladon reduces ciphertext size by $75\times$ and public-key size by $8.8\times$. Compared with BDLP, the strongest prior non-robust baseline, Ladon reduces ciphertext size by 5-6$\times$ and public-key size by 1.2-1.4$\times$ across 128- and 256-bit security. Compared with BDLP-R, the strongest robust prior baseline, Ladon reduces ciphertext size by 5.2-6.6$\times$ and public-key size by 1.4-1.7$\times$ across the same security levels. We note that these savings are achieved even though Ladon does not rely on aggressive ciphertext compression, as we discussed before.

TABLE 2: Comparison of Ladon's Owner Asset Key Derivation operation in **Active-secure** settings.

Work$_{\lambda_c}$	**Computation (ms)**	**Ciphertext** $\mathsf{ct}_{\mathsf{ak}}$ **(KB)**	**Public key** $\mathsf{pk}_{\mathcal{KMS}}$ **(KB)**
LP_{128}	21.36	450	50
BDLP_{128}	1.16	29.4	6.53
BDLP-R_{128}	1.23	31.18	8.03
Ladon_{128}	16.04	**5.97**	**5.66**
BDLP_{256}	2.54	58.72	13.56
BDLP-R_{256}	2.77	63.25	16.06
Ladon_{256}	43.49	**9.57**	**9.41**

* Computational security level λ_c is shown as a superscript. Counterparts' computations are estimated.

In terms of computation, Ladon's active variant, as discussed in Section 6, works over a power-of-two modulus and therefore cannot use the NTT. Instead, it falls back to negacyclic matrix multiplication, which first converts each polynomial into a matrix, a costly step, and then uses fast matrix multiplication to recover much of the lost performance. Therefore, in this setting, competitors that can use the NTT achieve better computational performance.

Table 3 shows the passive-secure comparison. Ladon's public-key and ciphertext sizes are identical across its passive and active variants because the KEM parameters match and both moduli are 30-bit. Against these baselines, ciphertext reduction is the main advantage: at 128-bit security, Ladon reduces ciphertext size by 6-10$\times$ compared with Pilvi and by $29\times$ compared with MS.

In terms of encapsulation performance, Ladon outperforms MS because MS relies on algebraic structures that do not support the NTT. Compared with Pilvi, although Pilvi is built on NTT-supporting algebraic structures similar to Ladon, its parameters use a large 89-115 bit modulus, which makes the computations costly even with NTT. As a result, Ladon achieves better computational performance.

7.3. KMS Committee: Threshold Decryption Communication and Storage

Each KMS committee member stores a share of $\mathsf{sk}_{\mathcal{KMS}}$ for the duration of the deployment, participates in every KMS Distributed Decryption, and consumes preprocessed material generated during an offline phase. We discuss the preprocessing stage more in Appendix E.

TABLE 3: Comparison of Ladon's Owner Asset Key Derivation operation in **Passive-secure** settings.

Work$_{\lambda_c}$	**Computation (ms)**	**Ciphertext** $\mathsf{ct}_{\mathsf{ak}}$ **(KB)**	**Public key** $\mathsf{pk}_{\mathcal{KMS}}$ **(KB)**
Pilvi^{*}_{128}	24-38	35.8-57.1	68.9-110.7
MS_{128}	5.08	170.27	1.36
Ladon_{128}	**0.158**	**5.97**	**5.66**
Ladon_{256}	**0.351**	**9.57**	**9.41**

* Pilvi uses different parameter sets for different threshold structures, scaling up to 32 servers, so we report the resulting time and size range. Counterparts' computations are estimated.

In this section, we evaluate two per-member costs: (i) long-term storage and (ii) per-release online end-to-end (E2E) communication, measured from fetching the ciphertext from cloud storage to sending the partial decryptions to the $\mathcal{TEE}$. The strongest active-secure competitors, namely LP, BDLP, and BDLP-R, report a threshold-failure probability of only 2^{-30}. For a fair comparison, we evaluate Ladon at two failure rates per security level: one matched to 2^{-30} and one tightened for deployments requiring a stronger guarantee (*e.g.,* failure probabilities of 2^{-131}-2^{-128}).

TABLE 4: Per-KMS storage and online E2E communication overhead in **Active-secure** settings.

Work$_{\lambda_c}$	†**Storage (KB)**	**E2E Communication**			**Security**‡ **Guarantee**
		#Rnds*	**Threshold Failure**	**Per-KMS (KB)**	
LP_{128}	100	4	2^{-30}	$852.12{+}50.03n$	SA
BDLP_{128}	$25\tilde{n}$	4	2^{-30}	$66.87 + 12.53n$	SA
BDLP-R_{128}	$25\tilde{n}$	4	2^{-30}	$68.65 + 12.53n$	R
Ladon_{128}	54.91	3	$\mathbf{2^{-30}}$	$\mathbf{6.12{+}10.16}\boldsymbol{n}$	SA
Ladon_{128}	187.59	3	$\mathbf{2^{-129}}$	$\mathbf{6.80{+}62.45}\boldsymbol{n}$	SA
BDLP_{256}	$50\tilde{n}$	4	2^{-30}	$133.66{+}25.06n$	SA
BDLP-R_{256}	$50\tilde{n}$	4	2^{-30}	$138.19{+}25.06n$	R
Ladon_{256}	89.09	3	$\mathbf{2^{-30}}$	$\mathbf{9.77{+}14.23}\boldsymbol{n}$	SA
Ladon_{256}	233.83	3	$\mathbf{2^{-131}}$	$\mathbf{10.52{+}71.7}\boldsymbol{n}$	SA

† BDLP uses Vandermonde secret sharing [51], [52], where each member stores up to $\tilde{n} = \log_2 n + 1$ shares. Ladon's storage includes the secret share, corresponding MAC shares, the double-shared masks, and the random AGVs discussed in Section 4.5 for the batch MAC checking.

* **Rnds** denotes the number of communication rounds from fetching the ciphertext to sending the partial decryptions.

For the success amplification parameter ℓ, Ladon achieves failure probability 2^{-30} at $\ell = 5$ and 2^{-129} at $\ell = 27$ for 128-bit security. At 256-bit security, it achieves failure probability 2^{-30} at $\ell = 7$ and 2^{-131} at $\ell = 31$. For active security, we set $\lambda_s = 36$ for failure rates below 2^{-40} and $\lambda_s = 46$ for failure rate 2^{-129}.

‡ R denotes extraction-share robustness, allowing invalid-share rejection and identification of misbehaving parties. SA denotes security with abort [35].

Table 4 shows the active-secure comparison. At the matched 2^{-30} threshold-failure level, Ladon achieves lower

end-to-end communication than all prior schemes. For example, in 128-bit security and for committee sizes of $n = 3$ or $n = 4$, LP imposes prohibitive per-member bandwidth, exceeding 1 MB; Ladon reduces this cost by 22-27×. Compared with the improved BDLP, Ladon reduces per-member bandwidth by 2.5-3×, with BDLP-R incurring essentially the same E2E communication overhead as BDLP. In 256-bit security, the aforementioned gaps increase to 3.5-4×.

TABLE 5: Per-KMS storage and online E2E communication overhead in **Passive-secure** settings.

Work$_{\lambda_c}$	†**Storage (KB)**	**E2E Communication**		
		#Rnds	**Threshold Failure**	**Per-KMS (KB)**
Pilvi$_{128}$	33-53	2	2^{-128}	38.55 - 60.67
MS$_{128}$	1.33	2	2^{-128}	170.53
Ladon$_{128}$	62.76	3	2^{-131}	$\mathbf{6.94 + 28.09n}$
Ladon$_{256}$	66.52	3	2^{-131}	$\mathbf{10.53 + 28.09n}$
Ladon$_{128}$	18.52	3	2^{-34}	$6.19 + 6.34n$
Ladon$_{256}$	22.27	3	2^{-34}	$9.78 + 6.34n$

Ladon's parameters support threshold structures up to $t = 512$ without affecting its success probability, far above what practical deployments require, and well beyond [19], which supports only up to 32 servers.
† In the passive variant, the storage for Ladon only consists of the secret key share and the double-shared masks.
For the success amplification parameter ℓ, Ladon achieves failure probability 2^{-34} at $\ell = 7$ and 2^{-131} at $\ell = 31$ for both security levels.

Table 5 shows the passive-secure comparison. For a fair comparison with prior schemes that report threshold-failure probability in the security parameter (2^{-128}), we compare them with Ladon configured to achieve a negligible threshold-failure probability, *i.e.,* a failure probability of 2^{-131}. In this setting, Ladon reduces per-member bandwidth by 1.4-1.9× compared with MS (for example, for a three-member committee), but has higher communication than Pilvi because the larger success-amplification parameter ℓ increases the number of transmitted partial decryptions. For completeness, we also report Ladon with a relaxed failure probability of 2^{-34}, in which Ladon achieves lower E2E communication. For example, for a committee size of $n = 3$, it reduces per-member bandwidth by 6.8× compared with MS and by 1.5× compared with Pilvi.

7.4. KMS Committee: Threshold Decryption Computation and Online Communication Latency

Local computation in the key-release pipeline consists of the work performed by each KMS committee member and by the $\mathcal{TEE}$ to derive the key. The TEE-side computation is the only component whose cost depends on the committee size, since the $\mathcal{TEE}$ must reconstruct the secret shares before completing decapsulation.

In Table 6, we report the total local computation and compare it with prior CCA-secure PQ-secure threshold KEMs. Prior schemes work over fields with NTT-friendly moduli, which they can exploit for KEM arithmetic. In contrast, as discussed in Section 7.2, Ladon's active variant works over a power-of-two modulus and therefore cannot use the NTT. The robust variant BDLP-R additionally performs share verification, which contributes more computational overhead than BDLP.

TABLE 6: KMS-member local computation and TEE computation in **Active-secure** settings.

Work$_{\lambda_c}$	**KMS and TEE Computation (ms)**			
	n=4	n=8	n=16	n=32
LP$_{128}$	53.79	54.3	60.31	63.29
BDLP$_{128}$	2.41	2.49	2.57	2.71
BDLP-R$_{128}$	5.78	8.32	15.62	29.72
Ladon$_{128}$	16.65	17.75	18.34	19.24
BDLP$_{256}$	5.26	5.32	6.14	6.45
BDLP-R$_{256}$	13.26	20.03	34.28	62.56
Ladon$_{256}$	45.19	46.23	47.13	47.91

Counterparts' computations are estimated.

For passive security, the trend is reversed, as shown in Table 7. MS relies on algebraic structures that do not support the NTT, while Pilvi uses a large 89-115 bit modulus, as discussed earlier. In contrast, the passive variant of Ladon employs an NTT-friendly prime modulus, resulting in faster KEM arithmetic in this setting. Nevertheless, it is important to note that Ladon implements a complete KEM, whereas MS and Pilvi do not. In addition, Ladon performs reencryption, comparison, and symmetric-key derivation operations, which contribute to its higher runtime compared to MS.

TABLE 7: KMS-member local computation and TEE computation in **Passive-secure** settings.

Work$_{\lambda_c}$	**KMS and TEE Computation (ms)**			
	*t=4	t=8	t=16	t=32
Pilvi$_{128}$	1.95	5.83	10.92	19.67
MS$_{128}$	0.013	0.028	0.053	0.107
Ladon$_{128}$	**0.308**	**0.335**	**0.413**	**0.572**
Ladon$_{256}$	**0.503**	**0.542**	**0.619**	**0.792**

† As discussed in Section 6, we set $n = 2t+1$. Counterparts' computations are estimated.

Threshold Communication Latency. Turning to network latency, Ladon and all compared schemes [14], [15], [19], [20] share two common communication steps: receiving the ciphertext from the cloud and sending partial decryptions to the TEE. Beyond these steps, Ladon and the schemes of [14], [15] are multi-round threshold protocols that add further broadcast rounds: one for Ladon and two for [14], [15]. Table 8 evaluates this overhead across low- and high-latency networks for different committee sizes.

TABLE 8: Communication latency of broadcast rounds.

Work$_{\lambda_c}$	Low Latency (ms)			High Latency (ms)		
	n=4	n=8	n=16	n=4	n=8	n=16
LP$_{128}$	2.53	4.51	8.74	29.07	33.92	37.79
BDLP-R$_{128}$	1.46	2.18	3.52	29.96	30.42	31.51
Ladon$_{128}$	1.51	2.06	3.24	15.94	16.39	17.27
BDLP-R$_{256}$	1.89	2.96	5.28	30.27	31.38	34.3
Ladon$_{256}$	1.63	2.3	3.8	15.96	16.54	17.84

Both BDLP and BDLP-R have similar broadcast complexities.

7.5. Deployment Authority: Provisioning

The deployment authority runs KMS Committee Provision once at deployment time to generate the committee key pair and distribute shares of the secret key to KMS committee members. After this step, its role is complete. Since the secret key is used during provisioning to produce the shares, we report its size in this section.

Table 9 shows the deployment-authority cost in the active-secure setting. With 128-bit security, Ladon's secret key is $179\times$ smaller than LP's and $45\times$ smaller than BDLP's. At 256-bit security, Ladon maintains a similar $53\times$ reduction to that of BDLP. This large reduction is mainly because Ladon does not use the NTT in the active-secure setting; therefore, the private-key coefficients can be represented using $\log(2\eta+1)$ bits rather than $\log q$ bits. Even if full $\log q$ bits are used, the private-key size becomes 5.76 KB at 128-bit security and 9.60 KB at 256-bit security. These sizes are still 4.3-$17.4\times$ smaller than the competitors.

TABLE 9: Deployment-authority cost of KMS Committee Provision operation in **Active-secure** settings.

Work$_{\lambda_c}$	KMS Committee Provision (ms)				$\mathsf{sk}_{\mathcal{KMS}}$ (KB)
	n=4	n=8	n=16	n=32	
LP$_{128}$	11.37	12.18	13.44	16.22	100
BDLP$_{128}$	0.73	1.13	1.77	2.32	25
BDLP-R$_{128}$	0.81	1.16	1.86	3.22	25
Ladon$_{128}$	14.91	15.12	16.18	16.87	**0.56**†
BDLP$_{256}$	1.58	2.36	3.78	6.75	50
BDLP-R$_{256}$	1.6	2.51	3.91	7.11	50
Ladon$_{256}$	42.43	42.60	43.2	44.3	**0.94**†

† In the active-secure setting, Ladon does not use NTT, so the private key can be represented in $\log(2\eta+1)$ bits per coefficient, saving storage. Using full $\log q$ bits yields 5760 bytes at 128-bit security and 9600 bytes at 256-bit security, which are relatively compact compared to the counterparts. Counterparts' computations are estimated.

Table 10 shows the passive-secure comparison. Pilvi requires an unshared secret key of up to 53.53 KB, which Ladon reduces to 5.63 KB, roughly $10\times$ smaller. Compared with the active variant, Ladon's passive variant can use the NTT to speed up KEM arithmetic. In this case, however, the private key is represented using $\log q$ bits per coefficient.

TABLE 10: Deployment-authority cost of KMS Committee Provision operation in **Passive-secure** settings.

Work$_{\lambda_c}$	KMS Committee Provision (ms)				$\mathsf{sk}_{\mathcal{KMS}}$ (KB)
	t=4	t=8	t=16	t=32	
Pilvi$_{128}$	38.26	65.99	196.62	601.70	34-54
MS$_{128}$	4.49	5.94	9.34	19.87	1.33
Ladon$_{128}$	**0.253**	**0.721**	**3.02**	**13.65**	**5.63**
Ladon$_{256}$	**0.512**	**1.27**	**5.27**	**22.6**	**9.4**

Counterparts' computations are estimated.

8. Related Work

In this section, we review prior work in three areas relevant to Ladon: (i) centralized key management systems, existing approaches for mitigating their compromise risks, and their limitations; (ii) PQ-secure threshold KEMs and public-key encryption (PKE) schemes, including PKE-based constructions used as non-CCA-secure KEMs [22]; and (iii) recent MPC-based techniques for thresholdizing encryption primitives, which help build more efficient solutions for the problem targeted in this work.

Google's Split-Trust Encryption Tool (STET) [53], [54] addresses centralized KMS trust in confidential computing by splitting key protection across two KMSs: one inside Google Cloud and one external. The asset owner secret-shares the key, obtains an independently wrapped share from each KMS, and uploads the wrapped shares to the cloud. When a TEE later needs the key, the KMSs unwrap their shares and deliver them to the TEE. Two operational characteristics shape STET's deployment. First, its implementation is tied to Google Cloud's API infrastructure, including `gcp-kms://` addresses for both internal and external KMSs. As a result, wrapping and TEE-side unwrapping both flow through Google's API stack, so distributed trust is realized only after wrapping and before final unwrapping. Second, the threshold access structure is fixed by the asset owner at the time of sharing rather than exposed as a configurable property of the KMS committee, leaving the committee's access structure transparent to the asset owner.

A cryptographic alternative that avoids both operational share splitting and the deployment limits of systems like STET is to distribute trust directly at the primitive level via threshold PKE or threshold KEMs, as discussed in Section 1. These primitives rely on threshold decryption: each committee member uses its secret-key share to produce a partial decryption, and the partial decryptions are combined to recover the plaintext. The key challenge is that a partial decryption can leak information about the member's secret-key share. The standard countermeasure is *noise flooding* [23], [55], in which each member masks its partial decryption with smudging noise large enough to statistically hide the share-dependent component. This technique has long been the workhorse for threshold decryption in PKE and fully homomorphic encryption (FHE) [21], [56], [57], [58], but it carries a steep parametric cost: the smudging noise must

be exponentially larger than the ciphertext noise in the statistical-security parameter, which forces a large modulus-to-noise ratio and inflated ciphertexts. Recent works reduce this overhead along different axes for the PKE. Pilvi [19] and Micciancio *et al*. [20] introduce alternative assumptions that permit smaller smudging noise and more compact ciphertexts; however, these assumptions remain conjectural [20], and the resulting schemes still carry large public keys and ciphertexts. Other approaches trade efficiency for weaker security notions [59] or obtain tighter parameters via improved Rényi-divergence analyses [60].

For threshold KEMs specifically, the literature is sparser. As discussed in Section 1, thresholdizing ML-KEM directly is fundamentally inefficient because its IND-CCA guarantee relies on the FO transform [27], whose re-encryption, equality check, and key-derivation steps are threshold-unfriendly and require heavy MPC machinery such as garbled circuits. A recent line of work [14], [15] sidesteps this obstacle by constructing IND-CCA threshold KEMs from threshold identity-based encryption combined with a signature scheme, following the BCHK transform [28]. The initial construction [14] reported public keys of 50 KB and ciphertexts of 450 KB; a follow-up [15] reduced these to 8 KB and 30 KB, respectively.

A parallel line of work in fully homomorphic encryption has moved away from noise flooding by using MPC-based protocols that isolate and cancel partial-decryption noise in shared form, recovering parameter sizes close to those of single-party schemes. Zyskind *et al*. [32], [61] replace noise flooding with a secure MPC rounding procedure that strips ciphertext noise while it remains shared, yielding UC-secure [62] threshold decryption with LWE parameters matching the standalone setting. Ajax [31] pursues a similar goal for FHEW-like cryptosystems [63] but avoids heavy MPC by adopting a mask-and-open technique built on double sharing: the noise term is masked, opened in one modulus, and removed in another, isolating it without ever revealing it in the clear. These works represent a recent trend toward efficient threshold encryption without large parameters and have been proposed as candidates in NIST's call for threshold cryptography.

9. Conclusion

Current KMS deployments concentrate trust in a single component, so a breach exposes every asset key it protects. A PQ-secure threshold KEM removes this single point of trust by distributing decapsulation across a KMS committee. In this paper, we presented Ladon, a PQ-secure threshold KEM built on NIST's ML-KEM standard that partitions decapsulation along its linear/FO line: asset owners run public encapsulation to derive an asset key, and later, when the TEE requests the key, the committee evaluates only the linear decryption step using basic secret-sharing techniques, in particular double sharing, while the TEE locally completes the FO transform inside protected memory to derive the key. We instantiated Ladon at both 128-bit and 256-bit security levels. Compared with the state of the art, Ladon yields public keys up to $8.8\times$ smaller, ciphertexts up to $75\times$ smaller, and end-to-end key-release communication up to $27\times$ lower than prior PQ-secure threshold KEMs at matched security levels and for a committee of three members. Ladon also provides two variants that are secure against passive and active corruptions of the committee members, each with a full simulation-based security proof. An open-source Rust implementation of Ladon accompanies this work.

Acknowledgments

This work is supported by the National Science Foundation under grants CNS-2350213 and CNS-2350215.

References

[1] NVIDIA, "NVIDIA Confidential Containers Reference Architecture," NVIDIA Documentation, 2026, accessed: 2026-06-06. [Online]. Available: https://docs.nvidia.com/datacenter/cloud-native/confidential-containers/latest/overview.html

[2] Microsoft, "Azure Confidential Computing Products," Microsoft Learn, 2026, accessed: 2026-06-06. [Online]. Available: https://learn.microsoft.com/en-us/azure/confidential-computing/overview-azure-products

[3] Amazon Web Services, "AWS Nitro Enclaves," AWS Product Page, 2026, accessed: 2026-06-06. [Online]. Available: https://aws.amazon.com/ec2/nitro/nitro-enclaves/

[4] "AWS KMS concepts - AWS Key Management Service Developer Guide," AWS Documentation, 2025, accessed: 2026-06-09. [Online]. Available: https://docs.aws.amazon.com/kms/latest/developerguide/concepts.html

[5] Microsoft, "Azure confidential computing: Secure key release (SKR)," Microsoft Azure Documentation, 2024, accessed 2026-06-02. [Online]. Available: https://learn.microsoft.com/en-us/azure/confidential-computing/concept-skr-attestation

[6] Confidential Containers Project, "Trustee: Attestation and secret delivery components for confidential containers," GitHub repository, CNCF Confidential Containers, 2024, accessed 2026-06-09. [Online]. Available: https://github.com/confidential-containers/trustee

[7] Amazon Web Services, "Asymmetric keys in AWS KMS," https://docs.aws.amazon.com/kms/latest/developerguide/symmetric-asymmetric.html, 2026, accessed: 2026-06-11.

[8] S. C. P. Gingupalli, "Hardening hsm clusters: Resolving key sync vulnerabilities for robust cu isolation," in *European Symposium on Research in Computer Security*. Springer, 2025, pp. 144–163.

[9] V. Shoup, "A proposal for an iso standard for public key encryption," *Cryptology ePrint Archive*, 2001.

[10] C. Brzuska, M. Klooß, and I. K. Woo, "Threshold public-key encryption: definitions, relations, and cpa-to-cca transforms," in *IACR International Conference on Public-Key Cryptography*. Springer, 2026, pp. 162–197.

[11] L. Brandão and R. Peralta, "Nist first call for multi-party threshold schemes," *doi*, vol. 10, p. 6028, 2023.

[12] Cybersecurity and Infrastructure Security Agency, National Security Agency, and National Institute of Standards and Technology, "Quantum-readiness: Migration to post-quantum cryptography," *Quantum readiness: Migration to post quantum cryptography*, 2023.

[13] Amazon Web Services, "AWS KMS cryptographic details," https://docs.aws.amazon.com/pdfs/kms/latest/cryptographic-details/kms-crypto-details.pdf, 2026, accessed: 2026-06-11.

[14] O. Lapiha and T. Prest, "A lattice-based ind-cca threshold kem from the bchk+ transform," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2025, pp. 461–494.

[15] K. Boudgoust, R. del Pino, O. Lapiha, and T. Prest, "Ind-cca lattice threshold kem under 30 kib," in *IACR International Conference on Public-Key Cryptography*. Springer, 2026, pp. 38–69.

[16] J. Katz and Y. Lindell, *Introduction to modern cryptography: principles and protocols*. Chapman and hall/CRC, 2007.

[17] National Institute of Standards and Technology, "FIPS 203 federal information processing standards publication module-lattice-based key-encapsulation mechanism standard," *NIST*, vol. 47, 2024.

[18] Google Cloud, "Announcing quantum-safe key encapsulation mechanisms in Cloud KMS," https://cloud.google.com/blog/products/identity-security/announcing-quantum-safe-key-encapsulation-mechanisms-in-cloud-kms, 2025, accessed: 2026-06-11.

[19] V. Cini, R. W. Lai, and I. K. Woo, "Pilvi: Lattice threshold pke with small decryption shares and improved security," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2025, pp. 539–569.

[20] D. Micciancio and A. Suhl, "Simulation-secure threshold PKE from LWE with polynomial modulus," *IACR Communications in Cryptology*, vol. 1, no. 4, 2025.

[21] H. Okada and T. Takagi, "Low communication threshold fhe from standard (module-) lwe," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2025, pp. 165–198.

[22] B. Zhou, H. Jiang, and Y. Zhao, "Cpa-secure kems are also sufficient for post-quantum tls 1.3," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2024, pp. 433–464.

[23] R. Bendlin and I. Damgård, "Threshold decryption and zero-knowledge proofs for lattice-based cryptosystems," in *Theory of Cryptography Conference*. Springer, 2010, pp. 201–218.

[24] B. Schlüter, C. Wech, and S. Shinde, "Heracles: Chosen plaintext attack on amd sev-snp," in *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, 2025, pp. 3810–3824.

[25] Z. Gu, E. Valdez, S. Ahmed, J. J. Stephen, M. Le, H. Jamjoom, S. Zhao, and Z. Lin, "Nvidia gpu confidential computing demystified," *arXiv preprint arXiv:2507.02770*, 2025.

[26] A. Langlois and D. Stehlé, "Worst-case to average-case reductions for module lattices," *Designs, Codes and Cryptography*, vol. 75, no. 3, pp. 565–599, 2015.

[27] E. Fujisaki and T. Okamoto, "Secure integration of asymmetric and symmetric encryption schemes," in *Annual international cryptology conference*. Springer, 1999, pp. 537–554.

[28] D. Boneh, R. Canetti, S. Halevi, and J. Katz, "Chosen-ciphertext security from identity-based encryption," *SIAM Journal on Computing*, vol. 36, no. 5, pp. 1301–1328, 2007.

[29] M. J. Dworkin *et al.*, "Sha-3 standard: Permutation-based hash and extendable-output functions," 2015.

[30] X. Wang, S. Ranellucci, and J. Katz, "Global-scale secure multiparty computation," in *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, 2017, pp. 39–56.

[31] Z. Hu, H. Liang, X. Wang, X. Xie, K. Yang, Y. Yu, and W. Zhang, "Ajax: Fast threshold fully homomorphic encryption without noise flooding," *Cryptology ePrint Archive*, 2025.

[32] G. Zyskind, D. Zarchy, M. Leibovich, and C. Peikert, "High-throughput universally composable threshold fhe decryption," in *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, 2025, pp. 2339–2353.

[33] V. Goyal, Y. Song, and C. Zhu, "Guaranteed output delivery comes free in honest majority mpc," in *Annual International Cryptology Conference*. Springer, 2020, pp. 618–646.

[34] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[35] D. Escudero, "An introduction to secret-sharing-based secure multiparty computation," *Cryptology ePrint Archive*, 2022.

[36] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *Annual cryptology conference*. Springer, 2012, pp. 643–662.

[37] R. Cramer, I. Damgård, D. Escudero, P. Scholl, and C. Xing, "Spdz2k: efficient mpc mod 2k for dishonest majority," in *Annual International Cryptology Conference. Springer*, 2018, pp. 769–798.

[38] A. Dalskov, D. Escudero, and A. Nof, "Fast fully secure multiparty computation over any ring with two-thirds honest majority," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 653–666.

[39] Microsoft, "Protect Intellectual Property on Azure VMs with Attestation-Gated Secure Key Release," Microsoft Learn, Sep. 2026, last updated: September 30, 2026. [Online]. Available: https://learn.microsoft.com/en-us/azure/virtual-machines/secure-key-release-pattern-trusted-launch

[40] M. J. Dworkin, "Sp 800-38d. recommendation for block cipher modes of operation: Galois/counter mode (gcm) and gmac," 2007.

[41] Amazon Web Services, "AWS KMS Concepts," https://docs.aws.amazon.com/kms/latest/developerguide/concepts-intro.html, 2026, aWS Key Management Service Developer Guide.

[42] Microsoft, "Secure Your Azure Key Vault," https://learn.microsoft.com/en-us/azure/key-vault/general/secure-key-vault, 2026, microsoft Learn.

[43] Google Cloud, "Cloud KMS Encryption and Key Management Deep Dive," https://docs.cloud.google.com/docs/security/key-management-deep-dive, 2026, google Cloud Documentation.

[44] A. Kyster, F. H. Nielsen, S. Oechsner, and P. Scholl, "Rushing at spdz: On the practical security of malicious mpc implementations," in *2025 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2025, pp. 2491–2508.

[45] Microsoft, "Rotate customer-managed keys for confidential VMs," https://learn.microsoft.com/en-us/azure/confidential-computing/key-rotation-offline, 2022, accessed: October 1, 2026.

[46] Google Cloud, "Enable and disable key versions," https://docs.cloud.google.com/kms/docs/enable-disable, accessed: October 1, 2026.

[47] J. B. Almeida, S. Arranz Olmos, M. Barbosa, G. Barthe, F. Dupressoir, B. Grégoire, V. Laporte, J.-C. Léchenet, C. Low, T. Oliveira *et al.*, "Formally verifying kyber: Episode v: Machine-checked ind-cca security and correctness of ml-kem in easycrypt," in *Annual International Cryptology Conference*. Springer, 2024, pp. 384–421.

[48] V. Lyubashevsky, "Basic lattice cryptography: The concepts behind kyber (ml-kem) and dilithium (ml-dsa)," *Cryptology ePrint Archive*, 2024.

[49] Amazon Web Services, "AWS Fault Isolation Boundaries: Availability Zones," AWS Whitepaper, 2024, accessed: 2026-06-09. [Online]. Available: https://docs.aws.amazon.com/whitepapers/latest/aws-fault-isolation-boundaries/availability-zones.html

[50] National Institute of Standards and Technology, "Submission requirements and evaluation criteria for the post-quantum cryptography standardization process," 2016.

[51] G. Borin, S. Celi, R. del Pino, T. Espitau, S. Katsumata, G. Niot, T. Prest, and K. Takemure, "Hermine: An efficient lattice-based frost-like threshold signature," *Cryptology ePrint Archive*, 2026.

[52] Y. Desmedt, G. Di Crescenzo, and M. Burmester, "Multiplicative non-abelian sharing schemes and their application to threshold cryptography," in *International Conference on the Theory and Application of Cryptology*. Springer, 1994, pp. 19–32.

[53] Google Cloud, "Split-trust encryption tool," https://docs.cloud.google.com/confidential-computing/docs/split-trust-encryption-tool, 2026, google Cloud Confidential Computing documentation. Accessed: 2026-05-25.

[54] Google, "Split-trust encryption tool (STET) source code," GitHub repository, 2023, accessed 2026-06-02. [Online]. Available: https://github.com/GoogleCloudPlatform/stet

[55] G. Asharov, A. Jain, A. López-Alt, E. Tromer, V. Vaikuntanathan, and D. Wichs, "Multiparty computation with low communication, computation and interaction via threshold fhe," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2012, pp. 483–501.

[56] A. Passelègue and D. Stehlé, "Low communication threshold fully homomorphic encryption," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2024, pp. 297–329.

[57] M. Dahl, D. Demmler, S. El Kazdadi, A. Meyre, J.-B. Orfila, D. Rotaru, N. P. Smart, S. Tap, and M. Walter, "Noah's ark: Efficient threshold-fhe using noise flooding," in *Proceedings of the 11th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, 2023, pp. 35–46.

[58] D. Boneh, R. Gennaro, S. Goldfeder, A. Jain, S. Kim, P. M. Rasmussen, and A. Sahai, "Threshold cryptosystems from threshold fully homomorphic encryption," in *Annual International Cryptology Conference*. Springer, 2018, pp. 565–596.

[59] B. Katharina and S. Peter, "Simple threshold (fully homomorphic) encryption from lwe with polynomial modulus," in *International Conference on the Theory and Application of Cryptology and Information Security*, 2023, pp. 371–404.

[60] K. Boudgoust and A. Costache, "Improved rényi arguments for lattice-based threshold encryption," *Cryptology ePrint Archive*, 2025.

[61] S. Chatel, B. Fisch, N. Genise, S. Halevi, L. Huguenin-Dumittan, G. Itzhaki, A. Kim, Y. Lee, Z. Liu, J. Mall *et al.*, "Pantheria: Threshold fhe for rlwe-based cryptosystems."

[62] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*. IEEE, 2001, pp. 136–145.

[63] L. Ducas and D. Micciancio, "Fhew: bootstrapping homomorphic encryption in less than a second," in *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 2015, pp. 617–640.

[64] I. Damgård, M. Fitzi, E. Kiltz, J. B. Nielsen, and T. Toft, "Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation," in *Theory of Cryptography Conference*. Springer, 2006, pp. 285–304.

[65] I. Damgård, D. Escudero, T. Frederiksen, M. Keller, P. Scholl, and N. Volgushev, "New primitives for actively-secure mpc over rings with applications to private machine learning," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 1102–1120.

Appendix A. Intractability Assumptions

We recall the Module Learning-With-Errors (MLWE) assumption [26], in its decisional form over the ring $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^N+1)$.

Definition 1 (Module LWE (MLWE))**.** Let $\mathsf{pp} = (N, k, \ell, q, \eta)$, where N, k, ℓ, q, η are positive integers. We say that MLWE holds with respect to pp if for every PPT algorithm $\mathcal{A}$, the advantage $\mathsf{Adv}^{\mathsf{MLWE}}_{\mathcal{A}}(\mathsf{pp})$ is negligible in λ, where

$$\mathsf{Adv}^{\mathsf{MLWE}}_{\mathcal{A}}(\mathsf{pp}) := \left| \begin{array}{l} \Pr\left[\begin{array}{l} \mathbf{A} \leftarrow_{\$} \mathcal{R}_q^{k\times\ell},\ \mathbf{s} \leftarrow_{\$} \beta_\eta^{k+\ell}; \\ \mathbf{t} := [\mathbf{I}_k \mid \mathbf{A}] \cdot \mathbf{s} \in \mathcal{R}_q^k; \\ b \leftarrow \mathcal{A}(\mathsf{pp}, \mathbf{A}, \mathbf{t}) :\ b = 1 \end{array}\right] \\ \quad - \Pr\left[\begin{array}{l} \mathbf{A} \leftarrow_{\$} \mathcal{R}_q^{k\times\ell},\ \mathbf{t} \leftarrow_{\$} \mathcal{R}_q^k; \\ b \leftarrow \mathcal{A}(\mathsf{pp}, \mathbf{A}, \mathbf{t}) :\ b = 1 \end{array}\right] \end{array} \right|.$$

Appendix B. Background on FIPS 203 (ML-KEM)

For completeness, we restate the three ML-KEM algorithms following the specification of FIPS 203 [17]. ML-KEM is built on top of an internal IND-CPA-secure public-key encryption scheme, which we refer to as *K-PKE* (with algorithms K-PKE.KeyGen, K-PKE.Enc, and K-PKE.Dec), whose security rests on the Module Learning with Errors (MLWE) assumption [26] (see Appendix A). The KEM algorithms invoke K-PKE as a subroutine; we inline K-PKE within each ML-KEM algorithm in a gray dashed box.

Notation. Following FIPS 203, let $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^N+1)$ denote the polynomial ring with modulus $q = 3329$, degree $N = 256$, and module rank k. We use lowercase letters for ring elements (*e.g.,* $a \in \mathcal{R}_q$), bold lowercase for vectors (*e.g.,* $\mathbf{v} \in \mathcal{R}_q^k$), and bold uppercase for matrices (*e.g.,* $\mathbf{A} \in \mathcal{R}_q^{k\times k}$). The centered binomial distribution β_η generates short secret and noise coefficients in $[-\eta, \eta]$. Let $\mathcal{B}^{32}$ denote 32-byte strings used for seeds and hash outputs. We use cryptographic hash functions $G : \mathcal{B}^* \to \mathcal{B}^{32} \times \mathcal{B}^{32}$ and $H : \mathcal{B}^* \to \mathcal{B}^{32}$, and a key-derivation function $\mathsf{KDF} : \mathcal{B}^* \to \mathcal{B}^{32}$. Finally, $\mathsf{Compress}_q(\cdot, d_b)$ and $\mathsf{Decompress}_q(\cdot, d_b)$, defined in Section 2.1, perform ciphertext rounding to reduce the bit-width of transmitted ring elements.

B.1. ML-KEM Core Algorithms

The three ML-KEM algorithms are presented below. Algorithm 4 (ML-KEM.KeyGen) generates a fresh public/secret key pair on input of the security parameter. Algorithm 5 (ML-KEM.Encaps) takes a public key and outputs a ciphertext together with a key. Algorithm 6 (ML-KEM.Decaps) takes a secret key and a ciphertext, and outputs the key recovered from the ciphertext.

B.2. Correctness of the ML-KEM's PKE

We write $\|\cdot\|_\infty$ for the coefficient-wise infinity norm: for $a = \sum_i a_i X^i \in \mathcal{R}_q$, $\|a\|_\infty := \max_i |a_i|$, where each a_i is taken as its representative in $(-q/2, q/2]$. The norm extends entry-wise to vectors $\mathbf{v} \in \mathcal{R}_q^k$.

Decryption of K-PKE relies on the algebraic cancellation between the public-key relation $\mathbf{t} = \mathbf{A}\mathbf{s} + \mathbf{e}$ and the ciphertext structure. Let $e_v := v' - v \pmod q$ and $\mathbf{e}_u := \mathbf{u}' - \mathbf{u} \pmod q$ denote the rounding errors introduced by ciphertext compression and subsequent decompression of the two ciphertext components. Substituting the definitions

Algorithm 4 ML-KEM.KeyGen

Input: Security parameter λ
Output: Public key pk and secret key sk
$\rho \leftarrow \mathcal{B}^{32}$ ▷ seed for public matrix
$z \leftarrow \mathcal{B}^{32}$ ▷ for implicit rejection
$(\mathsf{pk}_{\mathsf{PKE}}, \mathsf{sk}_{\mathsf{PKE}}) := \mathsf{K\text{-}PKE.KeyGen}(\rho)$ *(expanded below)*

K-PKE.KeyGen(ρ)

1. $\mathbf{A} \leftarrow \mathcal{R}_q^{k \times k}$ *expanded from seed ρ*
2. $(\mathbf{s}, \mathbf{e}) \leftarrow \beta_{\eta_1}^k \times \beta_{\eta_1}^k$ *short secret, error*
3. $\mathbf{t} := \mathbf{A}\mathbf{s} + \mathbf{e}$
4. **return** $(\mathsf{pk}_{\mathsf{PKE}}, \mathsf{sk}_{\mathsf{PKE}}) := \big((\rho, \mathbf{t}),\ \mathbf{s}\big)$

$\mathsf{pk} := \mathsf{pk}_{\mathsf{PKE}}$
$\mathsf{sk} := (\mathsf{sk}_{\mathsf{PKE}},\ \mathsf{pk},\ H(\mathsf{pk}),\ z)$
return $(\mathsf{pk}, \mathsf{sk})$

Algorithm 5 ML-KEM.Encaps

Input: Public key $\mathsf{pk} = (\rho, \mathbf{t})$
Output: Ciphertext c and session key K
$m \leftarrow \mathcal{B}^{32}$ ▷ random 32-byte seed
$(\bar{K}, r) := G(m \,\|\, H(\mathsf{pk}))$
$c := \mathsf{K\text{-}PKE.Enc}(\mathsf{pk}, m;\ r)$ *(expanded below)*

K-PKE.Enc($\mathsf{pk}, m;\ r$)

1. Parse $(\rho, \mathbf{t}) := \mathsf{pk}$, and recover $\mathbf{A} \leftarrow \mathcal{R}_q^{k \times k}$ *expanded from ρ*
2. $(\mathbf{r}, \mathbf{e}_1, e_2) \leftarrow \beta_{\eta_1}^k \times \beta_{\eta_2}^k \times \beta_{\eta_2}$ *seeded from r*
3. $\mathbf{u} := \mathsf{Compress}_q\big(\mathbf{A}^T\mathbf{r} + \mathbf{e}_1,\ d_u\big)$
4. $v := \mathsf{Compress}_q\big(\mathbf{t}^T\mathbf{r} + e_2 + \lfloor q/2 \rceil \cdot m,\ d_v\big)$
5. **return** $c := (\mathbf{u}, v)$

$K := \mathsf{KDF}(\bar{K} \,\|\, c)$
return (c, K)

Algorithm 6 ML-KEM.Decaps

Input: Secret key $\mathsf{sk} = (\mathsf{sk}_{\mathsf{PKE}}, \mathsf{pk}, h, z)$, ciphertext c
Output: Session key K
Parse $(\mathsf{sk}_{\mathsf{PKE}}, \mathsf{pk}, h, z) := \mathsf{sk}$
$m' := \mathsf{K\text{-}PKE.Dec}(\mathsf{sk}_{\mathsf{PKE}}, c)$ *(expanded below)*

K-PKE.Dec($\mathsf{sk}_{\mathsf{PKE}}, c$)

1. Parse $\mathbf{s} := \mathsf{sk}_{\mathsf{PKE}}$ and $(\mathbf{u}, v) := c$
2. $\mathbf{u}' := \mathsf{Decompress}_q(\mathbf{u}, d_u)$
3. $v' := \mathsf{Decompress}_q(v, d_v)$
4. **return** $m' := \mathsf{Compress}_q(v' - \mathbf{s}^T\mathbf{u}',\ 1)$

$(\bar{K}', r') := G(m' \,\|\, h)$
$c' := \mathsf{K\text{-}PKE.Enc}(\mathsf{pk}, m';\ r')$
if $c = c'$ **then**
  return $K := \mathsf{KDF}(\bar{K}' \,\|\, c)$
else
  return $K := \mathsf{KDF}(z \,\|\, c)$
end if

of v and $\mathbf{u}$ from K-PKE.Enc (Algorithm 5) into the decryption expression and grouping terms yields

$$\begin{aligned} v' - \mathbf{u}'^T\mathbf{s} &= \big(\mathbf{r}^T\mathbf{t} + e_2 + \tilde{m} + e_v\big) - \big(\mathbf{r}^T\mathbf{A} + \mathbf{e}_1^T + \mathbf{e}_u^T\big)\,\mathbf{s} \\ &= \tilde{m} + \underbrace{\mathbf{r}^T\mathbf{e} + e_2 + e_v - \mathbf{e}_1^T\mathbf{s} - \mathbf{e}_u^T\mathbf{s}}_{e_{\mathsf{total}}}, \end{aligned} \tag{1}$$

where $\tilde{m} := \mathsf{Decompress}_q(m, 1)$ lifts each bit of m to either 0 or $\lceil q/2 \rceil$ in $\mathcal{R}_q$. The dominant term $\mathbf{r}^T\mathbf{A}\mathbf{s}$ cancels exactly, leaving $\tilde{m}$ plus a residual noise e_{total} that aggregates three contributions: the key-generation noise ($\mathbf{r}^T\mathbf{e}$), the encryption noise ($e_2 - \mathbf{e}_1^T\mathbf{s}$), and the compression error ($e_v - \mathbf{e}_u^T\mathbf{s}$). The final rounding step $\mathsf{Compress}_q(\cdot, 1)$ rounds each coefficient of (1) as closer to 0 or to $\lceil q/2 \rceil$, recovering the original bit (and hence m) correctly whenever $\|e_{\mathsf{total}}\|_\infty < \lceil q/4 \rceil$. The parameters q, η, d_u, d_v are chosen so that $\|e_{\mathsf{total}}\|_\infty$ exceeds $\lceil q/4 \rceil$ only with negligible probability. We refer to Table 11 for the complete list of parameters.

TABLE 11: ML-KEM parameter sets. Sizes are in bytes. The decryption-failure probability δ is the FIPS 203 upper bound on the probability that decryption of the underlying PKE fails.

Security	k	η_1	η_2	d_u	d_v	δ	$\lvert\mathsf{pk}\rvert$	$\lvert\mathsf{sk}\rvert$	$\lvert c\rvert$
128-bit	2	3	2	10	4	2^{-139}	800	1,632	768
192-bit	3	2	2	10	4	2^{-164}	1,184	2,400	1,088
256-bit	4	2	2	11	5	2^{-174}	1,568	3,168	1,568

Appendix C. Correctness Analysis

Our goal in this section is to establish the correctness of the partial-decryption protocol in Figure 2. We walk through each step and show two things: the plaintext value that the step produces as a share, and the noise it accumulates. The cumulative noise term governs the per-instance success probability via the bound used in Section 5.1.

Our analysis follows the noise-tracking approach of Ajax [31], originally developed for the LWE regime, adapted here to the MLWE setting underlying Ladon's KEM. Throughout this section, $\|\cdot\|_\infty$ denotes the coefficient-wise infinity norm introduced in Appendix B.2. For ease of reference, we briefly recap the protocol structure from Section 4.3. After the offline preprocessing of ℓ double-shared random polynomials $d_1, \ldots, d_\ell$, the online phase proceeds in five steps (Figure 2): a linear decryption applied to $\mathsf{ct}_{\mathsf{ak}}$ (Step 1), a modulus switch from q to q' (Step 2), a reduction modulo μ' to isolate the noise (Step 3), a masked-opening round (Step 4), and a final local computation producing the share of $\mu' \cdot m$ over $\mathbb{Z}_{q'}$ (Step 5). We analyze each in turn.

Step 1- Linear decryption. Each committee member $\mathcal{KMS}_i$ uses its share of $\mathbf{s}$ to locally compute $[\![w]\!]_i^q := v - [\![\mathbf{s}^T]\!]_i\mathbf{u} \pmod q$, applied to the encapsulating ciphertext $\mathsf{ct}_{\mathsf{ak}} = (\mathbf{u}, v)$. Because subtraction and multiplication by

a public vector are linear, summing the $[\![w]\!]_i^q$ across the committee yields the K-PKE linear-decryption expression:

$$w = v - \mathbf{s}^T\mathbf{u} = \tilde{m} + e_{\mathsf{total}} \pmod q.$$

Step 2- Modulus switch. Each committee member next rescales its share from modulus q down to modulus q', where q' is even and $\mu' := q'/2$. Concretely, each committee member computes $[\![w']\!]_i^{q'} := \left\lfloor \frac{q'}{q} \cdot [\![w]\!]_i^q \right\rceil \pmod{q'}$, which equals $\frac{q'}{q} \cdot [\![w]\!]_i^q + e_{\mathsf{modsw},i}$, where $e_{\mathsf{modsw},i}$ is the local rounding noise introduced by the rescaling at that committee member. To track the noise cleanly, we note that $\tilde{m} = \lfloor q/2 \rceil \cdot m = (q/2 + e_{\mathsf{round}}) \cdot m$, where e_{round} captures the rounding offset from $\lfloor\cdot\rceil$. The shared value w therefore takes the form

$$w = \tfrac{q}{2} \cdot m + e_{\mathsf{round}} \cdot m + e_{\mathsf{total}} \pmod q,$$

and summing the shares gives $\sum_i [\![w]\!]_i^q = w + \kappa \cdot q$ for some $\kappa \in \mathbb{Z}$ arising from the additive-sharing wrap-around at modulus q. Combining with the rescaling above,

$$\begin{aligned}\sum_{i=1}^{t} [\![w']\!]_i^{q'} &= \tfrac{q'}{q} \cdot \sum_{i=1}^{t} [\![w]\!]_i^q + \sum_{i=1}^{t} e_{\mathsf{modsw},i} \\ &= \tfrac{q'}{q} \cdot (w + \kappa \cdot q) + e_{\mathsf{modsw}} \\ &= \mu' \cdot m + \tfrac{q'}{q}(e_{\mathsf{round}} m + e_{\mathsf{total}}) + \kappa \cdot q' + e_{\mathsf{modsw}},\end{aligned}$$

where $e_{\mathsf{modsw}} := \sum_{i=1}^{t} e_{\mathsf{modsw},i}$ is the aggregate modulus-switch rounding noise across the t participating authorities. Reducing modulo q' yields

$$\begin{aligned} w' &= \mu' \cdot m + e' \pmod{q'}, \\ e' &:= \tfrac{q'}{q}\big(e_{\mathsf{round}} m + e_{\mathsf{total}}\big) + e_{\mathsf{modsw}}. \end{aligned}$$

Bound on $\|e'\|_\infty$. Given that the rounding operations produce error in $[-0.5, 0.5)$, with the aggregate satisfying $\|e_{\mathsf{modsw}}\|_\infty \le t/2$, that the message coefficients m are bit-valued (so $\|e_{\mathsf{round}} \cdot m\|_\infty \le 1/2$), and that e_{total} is the ML-KEM linear-decryption noise bounded with overwhelming probability (Appendix B.2), combining these yields

$$\|e'\|_\infty \le \tfrac{q'}{q} \cdot \|e_{\mathsf{total}}\|_\infty + \tfrac{q'}{2q} + \tfrac{t}{2}.$$

Per the discussion in Section 5.1, the per-instance success probability is bounded by

$$p_{\mathsf{dec\text{-}succ}} \ge 1 - N \cdot \|e'\|_\infty/\mu' = 1 - 2N \cdot \|e'\|_\infty/q'.$$

The strategy for parameter selection is to choose the KEM's parameters such that the dominant term is small: a large q together with small error coefficients keeps $\|e_{\mathsf{total}}\|_\infty/q$ small, driving the success probability close to 1.

Step 3- Isolating the noise. At this point, the committee jointly holds a share of $w' = \mu' \cdot m + e'$ over $\mathbb{Z}_{q'}$, where $\mu' = q'/2$ encodes the message bits and e' is the accumulated noise. To proceed, each committee member extracts the noise alone by reducing its share modulo μ'. Because $\mu' \mid q'$, the message component $\mu' \cdot m$ vanishes upon reduction, leaving a share of e' alone over $\mathbb{Z}_{\mu'}$. With a share of the noise, the committee can mask and open it (Step 4), and then subtract the mask (Step 5) to obtain a share of m.

Active variant: no modulus switch. In the active variant of Ladon, the construction operates directly over a power-of-two ring $\mathbb{Z}_{2^k}$ rather than the prime-modulus ring $\mathbb{Z}_q$, so no modulus switch is required (Section 4.5). The committee's partial decryption produces a share of $\tilde{m} + e_{\mathsf{total}}$ directly, where e_{total} is the ML-KEM linear-decryption noise; no additional rounding noise is introduced. The accumulated noise term therefore satisfies $e' = e_{\mathsf{total}}$, and the per-instance success probability simplifies to $p_{\mathsf{dec\text{-}succ}} \ge 1 - N \cdot \|e_{\mathsf{total}}\|_\infty/\mu$. Following our requirement that $\mu' \mid q'$ in the passive variant, we note that in the active variant we have $q = 2^k$ and $\mu = 2^{k-1}$. Although each value is shared over $\mathbb{Z}_{2^{k+s}}$, the divisibility condition $\mu \mid q$ still holds since $2^{k-1+s} \mid 2^{k+s}$. Hence, the protocol executes correctly.

Appendix D. Security Proof of Ladon

We prove the security of KMS Distributed Decryption using the simulation-based approach of Ajax [31], showing that it securely realizes the ideal functionality $\mathcal{F}_{\mathsf{LadonDec}}$, which models threshold decryption by a trusted party, against a semi-honest adversary corrupting up to t of the $t+1$ $\mathcal{KMS}$ members. Our proof applies to both Shamir and additive secret sharing. We define the ideal functionality, construct a simulator, establish indistinguishability of the adversary's view, and bound the statistical distance between the value reconstructed by the $\mathcal{TEE}$ and the output of Ladon's underlying decryption algorithm on $\mathsf{ct}_{\mathsf{ak}}$ using the unshared $\mathsf{sk}_{\mathcal{KMS}}$, accounting for the decryption-error probability analyzed in Section 5.1. Finally, we extend the proof to Ladon with active security in the dishonest-majority setting.

Ideal Functionality $\mathcal{F}_{\mathsf{LadonDec}}$. The functionality interacts with the $t+1$ participating $\mathcal{KMS}$ members, the $\mathcal{TEE}$ as the designated receiver, and the adversary. It receives the members' shares $\{[\![\mathsf{sk}_{\mathcal{KMS}}]\!]_i^q\}$ together with the ciphertext $\mathsf{ct}_{\mathsf{ak}} = (\mathbf{u}, v)$, reconstructs $\mathsf{sk}_{\mathcal{KMS}}$ internally, applies the linear decryption and the modulus switch from q to q', and outputs the noise-free value $\mu' \cdot m \in \mathcal{R}_{q'}$ to the $\mathcal{TEE}$. The adversary learns the ciphertext and nothing else. The FO-finalization steps performed by TEE Asset Key Derivation run locally inside the $\mathcal{TEE}$ after this output and are outside the scope of KMS Distributed Decryption.

Theorem 1. *Assuming the double-shared masks $\{d_j\}_{j\in[\ell]}$ used in KMS Distributed Decryption are produced by a secure preprocessing protocol [64], KMS Distributed Decryption securely realizes $\mathcal{F}_{\mathsf{LadonDec}}$ against any semi-honest adversary corrupting a subset* Corr *of the $t+1$ members with $|\mathsf{Corr}| \le t$.*

Simulator. Let $\mathsf{Hon} = [t+1] \setminus \mathsf{Corr}$ denote the honest members. The simulator $\mathcal{S}$ takes as input what the adversary itself sees: the corrupt parties' shares of the secret key $[\![\mathsf{sk}_{\mathcal{KMS}}]\!]_{\mathsf{Corr}}^q$, the ciphertext $\mathsf{ct}_{\mathsf{ak}}$, and the corrupt parties' shares of each preprocessed mask, $[\![d_j]\!]_{\mathsf{Corr}}^{\mu'}$ and $[\![d_j]\!]_{\mathsf{Corr}}^{q'}$,

for $j \in [\ell]$. $\mathcal{S}$ then walks through the five steps of KMS Distributed Decryption as follows:

In the Linear Decryption, Modulus Switch, and Isolate the Error steps, the operations are local on each authority's share and produce no inter-party messages. $\mathcal{S}$ performs them on the corrupt parties' shares directly to obtain $[\![w]\!]_i^q$, $[\![w']\!]_i^{q'}$, and $[\![e']\!]_i^{\mu'}$ for each $i \in \mathsf{Corr}$.

The Mask the Error and Open step is the only step where messages cross between honest and corrupt members: for each $j \in [\ell]$, the members jointly reveal $\tilde{e}_j = e' + d_j \pmod{\mu'}$ by broadcasting their shares. $\mathcal{S}$ simulates this step by first sampling $\tilde{e}_j$ uniformly at random from $\mathcal{R}_{\mu'}$. It then computes the corrupt parties' shares of $\tilde{e}_j$ as $[\![\tilde{e}_j]\!]_i^{\mu'} = [\![e']\!]_i^{\mu'} + [\![d_j]\!]_i^{\mu'}$ for each $i \in \mathsf{Corr}$, with both summands known to $\mathcal{S}$. Finally, $\mathcal{S}$ samples the honest parties' broadcast shares uniformly at random, subject to summing with the corrupt shares to $\tilde{e}_j$, and sends them to the adversary.

In the Obtain Share of the Seed step, each authority locally computes its share of $\mu' \cdot m_j$ and sends it to the $\mathcal{TEE}$. Since the $\mathcal{TEE}$ is honest in Ladon's threat model, these messages are not delivered to the adversary. Therefore, $\mathcal{S}$ only submits the corrupted parties' inputs to $\mathcal{F}_{\mathsf{LadonDec}}$ to complete the ideal execution.

Indistinguishability of the Adversary's View. The adversary's view in the real execution has three parts: its own input (the corrupt shares of $\mathsf{sk}_{\mathcal{KMS}}$ and of the d_j's), the local values it computes during the Linear Decryption, Modulus Switch, Isolate the Error, and Obtain Share of the Seed steps, and the broadcasts it receives during the Mask the Error and Open step. The first two parts are reproduced exactly by $\mathcal{S}$, since $\mathcal{S}$ holds the same input and runs the same local computations. The third part is uniform in $\mathcal{R}_{\mu'}$ from the adversary's view because the mask d_j is sampled uniformly from $\mathcal{R}_{\mu'}$ and shared additively across all $t+1$ members, so with only t shares the mask stays uniform from the adversary's view, and adding e' to a uniform d_j modulo μ' produces a uniform value. This matches how $\mathcal{S}$ samples $\tilde{e}_j$ in the simulation. The honest parties' broadcast shares are uniform in both worlds, conditioned to sum to $\tilde{e}_j$ with the corrupt shares, so they, too, are identically distributed. The real and simulated views coincide, and KMS Distributed Decryption securely realizes $\mathcal{F}_{\mathsf{LadonDec}}$.

The argument above holds whether or not $e' + d_j$ wraps around modulo μ' in some coefficient. When wraparound does occur for a coefficient, the opened $\tilde{e}_j$ equals $(e' + d_j) \pm \mu'$ for that coefficient; but since d_j is uniform in $\mathcal{R}_{\mu'}$ coefficient-wise, $(e' + d_j) \pm \mu'$ is also uniform from the adversary's view. The adversary, therefore, cannot tell whether wraparound has occurred. Wraparound does affect correctness, since the $\mathcal{TEE}$ reconstructs $\mu' \cdot m_j \pm \mu'$ rather than $\mu' \cdot m_j$ for the affected coefficient, but this discrepancy stays inside the $\mathcal{TEE}$'s protected memory and reveals nothing to the adversary.

Statistical Distance to Direct Decryption. The gap between the $\mathcal{TEE}$'s recovered value at the end of KMS Distributed Decryption and the output of Ladon's underlying decryption algorithm applied to $\mathsf{ct}_{\mathsf{ak}}$ with the unshared $\mathsf{sk}_{\mathcal{KMS}}$ reduces to the wraparound event analyzed above. The two outputs agree whenever wraparound is absent in the majority of the ℓ parallel copies, so the gap is bounded by $p_{\mathsf{dec\text{-}fail}} = 1 - \sum_{i=\lceil \ell/2 \rceil}^{\ell} \binom{\ell}{i} p_{\mathsf{dec\text{-}succ}}^i (1 - p_{\mathsf{dec\text{-}succ}})^{\ell - i}$, where $p_{\mathsf{dec\text{-}succ}}$ is the per-instance success probability defined in Section 5.1. The bound decays exponentially in ℓ whenever $p_{\mathsf{dec\text{-}succ}} > 1/2$. With proper parameters, we can ensure the underlying decryption failure is almost negligible compared to the threshold decryption failure.

Extension to Active Security. The simulator and indistinguishability argument above carry over to the active variant of Ladon with minimal change. First, as with passive security, the double-sharings in this setting can be generated using [65] during the offline phase. The shares are lifted from $\mathbb{Z}_{2^k}$ to $\mathbb{Z}_{2^{k+\lambda_s}}$ and each share carries an information-theoretic MAC under SPDZ$_{2^k}$ [37], so the simulator additionally emulates the MAC tags on the corrupt parties' shares using SPDZ$_{2^k}$'s standard simulator. The Linear Decryption, Isolate the Error, Mask the Error and Open, and Obtain Share of the Seed steps are all linear and therefore automatically preserve MACs, so no changes are required to the simulation of any step. The Modulus Switch step is dropped entirely in the active variant, since $q = 2^k$ is fixed from KMS Committee Provision onward. The uniformity argument for $\tilde{e}_j$ runs unchanged, since it depends only on the additive sharing of d_j and the uniformity of the mask, both of which are preserved by SPDZ$_{2^k}$. The one additional step is the MAC checking. An active deviation by a corrupted authority is detected with probability at least $(1 - 2^{-\lambda_s + \log(\lambda_s + 1)})$ [37], causing the protocol to abort. The overall statistical security of the active variant is therefore dominated by whichever is weaker: the MAC bound or the decryption-failure probability derived for the active variant in Appendix C.

Appendix E. Preprocessing Phase in Ladon

Ladon requires preprocessing material consisting of masked double-shared polynomials. Each key release consumes ℓ such polynomials, totaling ℓN coefficients double-shared across two moduli. These are generated using standard MPC techniques for passive [64] or malicious security [65]. As discussed in Section 4.5, Ladon also uses agreed-upon random values (AGVs) to reduce MAC checks across protocol openings, which the KMS committee can likewise generate via MPC [37]. All preprocessing material is generated offline during setup.

Our local benchmarks show that, in the active variant, generating the double sharings for one random polynomial mask takes 57, 108, 202, and 426 microseconds for $n = 4$, 8, 16, and 32 parties, respectively. In the passive variant, the corresponding times are 31, 59, 113, and 221 microseconds. The cost of generating additional double-shared polynomial masks grows linearly with ℓ, since each key release consumes ℓ such masks.

Ethics Considerations

None. The development and evaluation of Ladon do not involve human participants, personal information, or sensitive datasets. All experiments use locally generated cryptographic inputs, publicly available specifications, and open-source software. We explicitly state our assumptions and security model, provide security proofs and evaluation methodology, and release our implementation and analysis scripts to facilitate independent verification. These materials enable key stakeholders, particularly cloud providers offering confidential computing services, to conduct further evaluations, cryptographic reviews, software hardening, and integration testing before deployment.

Appendix F. Meta-Review

The following meta-review was prepared by the program committee for the 2027 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

F.1. Summary

This paper presents Ladon, a post-quantum threshold KEM that replaces the centralized Key Management Service in confidential computing deployments. Rather than distributing the entire FO decapsulation across the KMS committee, which is expensive because the re-encryption and comparison steps are hash-heavy and MPC-unfriendly, Ladon distributes only the linear decryption step using a mask-and-open double-sharing technique, leaving the FO steps to the TEE already trusted in the key-release path. Two variants are provided, one resistant to passive and one to active committee corruption, and the evaluation reports smaller public keys and ciphertexts and lower communication cost than prior work.

F.2. Scientific Contributions

- Independent Confirmation of Important Results with Limited Prior Research
- Provides a Valuable Step Forward in an Established Field

F.3. Reasons for Acceptance

1) The paper provides a valuable step forward in an established field. The centralized KMS remains a critical single point of failure, and recognizing that the TEE already trusted in the key-release path can absorb the FO transform steps is a nice way to sidestep garbled-circuit-style MPC.
2) The design is clear, with both passive and active variants implemented end to end, and the reported improvements in key size, ciphertext size, and communication overhead over prior work are demonstrated. The open-source release will support reproducibility and follow-up research.

F.4. Noteworthy Concerns

1) The evaluation does not capture end-to-end system performance, making it difficult to assess real deployment impact. The system is never integrated with any of the confidential computing stacks it motivates with, so enclave memory encryption, attestation latency, and TEE-side working set costs do not appear.