Auditable Continuous Group Key Agreement

Easwar Vivek Mangipudi Maddie Gorman Sasha Levinshteyn
Hoth Labs
{easwarmangipudi, maddie.b.gorman, sashaplevinshteyn}@gmail.com

Abstract

Continuous group key agreement (CGKA), the cryptographic core of Messaging Layer Security (MLS, RFC 9420), provides key management for large end-to-end encrypted group chats. It refreshes the group's keys as members join and leave, but offers no way for a designated auditor to recover past epoch keys, and no way to *check* that such recovery remains possible. Regulated deployments in finance, healthcare, and government therefore resort to plaintext server logging, abandoning end-to-end encryption entirely. Simply adding a key escrow admits a *silent escrow failure*: the group accepts an epoch whose escrow holds unrecoverable material, with no visible anomaly until a later audit.

Addressing this, we introduce *auditable CGKA* (Au-CGKA), an MLS-shaped protocol in which every admitted epoch carries a proof. The proof binds that epoch's key material to a well-defined secret recoverable by a threshold auditor committee. Every member checks that binding against the epoch secret it derives and refuses the commit if the two disagree, so *auditability* guarantees that the secret of every epoch an honest member accepts is threshold-recoverable. We give a post-quantum protocol, Π_A, realizing this property with STARK proofs. The committer escrows the epoch secret to an auditor committee, and the escrow ciphertext is a STARK-friendly encryption of Shamir shares. Its well-formedness is proven in-circuit at MLS commit time.

We prototype Au-CGKA in Rust with the proofs on a zero-knowledge, post-quantum custom multi-stage STARK. On an Apple M5 Pro, an auditability proof takes 1.38 s with proof-size 15.31 MB and verifies in 0.17 s, at every group size; the relation it proves is independent of the group size. The proof is checked at admission and then discarded, so it costs bandwidth on the commit and nothing in storage; the only persistent overhead is the fixed-size escrow. Adaptive post-quantum security holds in the secure-erasure model with straight-line reductions in the quantum random-oracle model, and carries to the implemented backend under a stated assumption; privacy and escrow soundness follow as game-based guarantees.

1 Introduction

End-to-end encrypted (E2EE) group messaging is now standardized. The Messaging Layer Security protocol (MLS, RFC 9420) [7] is designed for group chats of up to tens of thousands of members. Those members hold a shared secret key end-to-end, so that only current members can read the chat's traffic. The group advances in *epochs*: when the membership or keys change, one member *commits*, deriving fresh key material for the next epoch. This continual rekeying is carried out by TreeKEM (Tree-based Key Encapsulation Mechanism), the *continuous group key agreement* (CGKA) at the core of MLS [28]. It provides two properties that a single long-lived key cannot: *forward secrecy* (FS), so that compromising today's key does not expose past epochs, and *post-compromise security* (PCS), so that the group heals once a compromised member rekeys. §3 and Appendix A.1 recall the mechanics we rely on.

One class of deployments cannot use MLS as it stands. Record-keeping rules in finance (SEC Rule 17a-4 [79], FINRA Rule 4511 [38], MiFID II [36]), healthcare (HIPAA Security Rule audit controls [78]), and government (Federal Records Act [77], NARA electronic-records requirements [62]) require that an authorized party be able to reconstruct communications after the fact. This is precisely the recoverability that end-to-end encryption is designed to deny. Organizations under these mandates have two options today: disable E2EE and log plaintext at a server, exposing every message to a single breach; or augment the protocol with ad hoc, unverified key escrow, whose correct operation is neither observable nor verifiable.

The second option admits a *silent escrow failure*. Escrow's premise is that each commit message also seals the epoch's secret for the authorized party. MLS, however, gives no way to check that this sealed ciphertext holds the *same* secret the commit used to derive the group's keys. A malicious or faulty committer can therefore post a commit message that verifies for every member (the keys agree and MLS's confirmation tag passes) while the escrow holds unrecoverable material, with

no visible anomaly. The fault surfaces only when an auditor attempts decryption, often years later. By then, the epoch is unrecoverable.

This paper makes recoverability a *cryptographic invariant* of the key agreement: checked at commit-message admission and private against every party but a threshold of auditors, in the post-quantum setting. We give a construction and its post-quantum security analysis. We call the resulting primitive *auditable CGKA* (Au-CGKA).

The construction escrows the epoch's root secret es_e, from which the epoch's keys are derived, and publishes a commitment c_{root} to it. The proof establishes that the escrow shares a preimage of the published c_{root}, and every member checks that preimage against the epoch secret it derives when it admits the commit, so a mis-escrow is refused by the group. The tie between the escrow and the epoch is therefore established at admission, by the parties that already hold the secret, and needs no in-circuit key schedule.

Closing the silent-escrow gap at commit time has two costs. The commit message must prove that the escrow holds the *same* secret from which the group derived the epoch's keys. Encoded directly, that requirement must cover the entire TreeKEM update path and the SHA-512 [64]/HKDF [51] key schedule of MLS. That schedule comprises tens of hash compressions per commit, each at $\sim 10^5$ constraints. A monolithic per-commit proof over that schedule is therefore too slow for a per-commit client operation, and classical SNARKs (Groth16 [42], Bulletproofs [19]) would forfeit the post-quantum security a long-retention setting demands. Reserving recovery to a *threshold* of auditors rather than one is a second obstacle: the standardized Module-Lattice-based Key Encapsulation Mechanism (ML-KEM) [67] is not a threshold scheme, and lattice threshold public-key encryption (PKE) [15, 17, 25, 54] is not yet practical at these parameters. Composing existing primitives independently does not meet these requirements simultaneously. Meeting them requires a joint design: a commit structure whose recoverability is verified at modest circuit cost, threshold escrow whose well-formedness is STARK-verified, and an epoch-0 bootstrap, together with a security argument that the pieces compose.

This design yields Π_{A}, a protocol achieving *auditability*: the secret of every admitted epoch is recoverable by any threshold subset of the auditor committee, and that recoverability is checked by every member when it admits the epoch. The member check carries the guarantee, so the guarantee is conditioned on some member performing it: what Π_{A} establishes is that the secret of every epoch an *honest* member accepts is threshold-recoverable. A group all of whose members collude can accept an epoch whose escrow holds a different value, and no check made from the log alone detects this, no admission check being able to recompute a secret it does not hold. That case is out of scope; §D.3.3 states the boundary. We state the property formally as an ideal functionality (§4.2).

1.1 Setting and construction

The setting Au-CGKA targets a *large, long-lived group chat* over MLS whose membership *churns*. The group advances in *epochs*. In each epoch one member, the *committer*, rotates the keys and posts a commit message to a public, append-only log kept by the delivery service. A fixed *auditor committee* of N parties is designated: any (threshold) t or more of them can later reconstruct that epoch's escrowed secret and recover its epoch key k_e for a compliance audit. Recovering arbitrarily old epochs is the point, so the committee cannot forget: a coalition of t auditors assembled at any moment recovers every past epoch and no later rekeying repairs that (Rem. 1). Trading forward secrecy against the committee for recoverability is the premise, not an artifact. Every commit message carries a zero-knowledge proof that the epoch's secret is escrowed to this committee, so any member or third party can check the escrow without trusting the committer or the server.

The construction In Π_{A} (*auditability*), the committer escrows the epoch's root secret es_e, threshold secret-sharing it under the auditor committee's post-quantum PKE keys, and publishes a commitment $c_{\mathsf{root}} = H_{\mathsf{P2}}(\texttt{"crot"} \| G \| e \| es_e)$ (Fig. 2). The proof checks that the escrow opens to a preimage of c_{root}. Since members derive es_e themselves, they verify c_{root} against their own epoch secret when they process the commit message, so a mis-escrow is rejected *at admission*, before the epoch is accepted. The proof's only hash work is the commitment and the escrow, and the MLS key schedule stays outside it. Recovery is *per-epoch independent*: any t auditors reconstruct es_e for an epoch without touching any other, and an unrecoverable escrow ciphertext costs exactly that epoch. The auditor committee is *offline between audits* and comes online only when an audit is requested.

Π_{A} does not fix the update path at admission: the auditor recovers the epoch keys but obtains no independent account of how they were distributed. That scope is what keeps the proof small. The relation carries no term indexed by a path level, so its statement, its witness and its cost are the same whatever the group size: a Π_{A} proof is 15.31 MB and is produced in 1.38 s. That proof is checked at admission and has no use afterwards, so a deployment may drop it once the epoch is admitted. The 15.31 MB is a cost on the commit, not on the archive (§7).

Π_{A} therefore guarantees that every member accepting an epoch derives the same *recoverable* secret. The proof is post-quantum and zero-knowledge, and its verification is stateless: it needs the log and nothing else, which is what lets the delivery service refuse a malformed escrow before it is ever admitted. The remaining step — that the escrowed preimage is the group's own epoch secret — is checked by each member against the secret it derived, and the paper is careful throughout to keep the two apart (§D.3.3).

1.2 Contributions

1. **Au-CGKA interface.** We define the syntax of *auditable CGKA*. It extends the standard CGKA interface [5] with AuditSetup, GroupCreate and VerifyCreate (epoch-0 bootstrap), VerifyCommit (stateless, log-only), and Audit (threshold, offline) (§4).
2. **A security model for auditable CGKA.** We capture the recoverability guarantee as a UC ideal functionality $\mathcal{F}_{\mathsf{aud}}$. We extend the insider-security model of [5] with creation, audit, and verification interfaces in a single specification (Fig. 1). The functionality is realized against *adaptive* adversaries in the secure-erasure model (§6); game-based privacy and escrow soundness follow as corollaries (Appendix D.3.1; §D.3).
3. **A post-quantum construction.** Π_{A} escrows the epoch secret and has recoverability checked at admission against c_{root}. It realizes $\mathcal{F}_{\mathsf{aud}}$ from Module-LWE, CGKA path secrecy for the underlying TreeKEM layer [5], and a straight-line zero-knowledge argument of knowledge in a programmable global QROM, instantiated by post-quantum transparent STARK proofs [11]. We obtain it by jointly designing two standard techniques that keep the per-commit STARK tractable (§5). (i) *Committed escrow*: Π_{A} commits to the escrowed value, so admission needs only a commitment opening in circuit, and the MLS key schedule stays outside it. (ii) *Verifiable threshold escrow*: Shamir shares encrypted under a STARK-friendly Module-LWE PKE and proven well-formed in-circuit. This closes the silent-escrow gap.
4. **Implementation and evaluation.** We prototype Au-CGKA in Rust and prove the complete escrow well-formedness relation on a real witness end-to-end (§7–8) on a custom multi-stage STARK. For the evaluated configuration we arithmetize the relation directly: dedicated ML-KEM and field chips composed through shared lookup arguments. We also apply prover-side optimizations, chiefly stacking same-shape sub-computations, to reduce proof time and size. On an Apple M5 Pro an auditability commit proves in 1.38 s and verifies in 0.17 s, at every group size.

The full protocol pseudocode is given in Appendix B.2.

2 Related Work

Table 1 places Au-CGKA against the closest prior work on the three properties it claims.

CGKA and MLS security The CGKA abstraction was formalized by Alwen et al. [3] with a game-based analysis of TreeKEM. Subsequent work added active and insider security in the UC framework [4], efficient key management for overlapping groups [2], adaptive and active security via tainted trees [49], and machine-checked proofs of TreeSync [80]. Our starting point is the insider-security functionality of Alwen, Jost, and Mularczyk [5]; $\mathcal{F}_{\mathsf{aud}}$ extends it with creation, audit, and verification interfaces.

Table 1: Au-CGKA vs. closest prior work. Audit = epoch-key recoverability by a designated party; PQ = post-quantum; UC = UC-proven.

Work	Audit	PQ	UC
Message franking / AMF [44, 75]	—	—	—
DDF robustness NIZK [32]	—	—	—
TreeKEM / MLS [3, 7]	—	○	✓
Key transparency [52, 53, 58, 59]	—	—	—
Quarantined-TreeKEM [26]	○	○	✓
Verifiable encryption [20, 55]	○	○	—
UC auditable surveillance [37]	✓	—	✓
Au-CGKA (this work)	✓	✓	✓*

✓ = provides the property; ○ = partial, or provided for a different object; — = not provided. "Audit" for Au-CGKA is threshold recoverability of the secret of every epoch an honest member accepts, checked at admission (Theorem 13); no property is claimed against a group all of whose members collude. * in the $\mathcal{F}_{\mathsf{NIZK}}$-hybrid model; the implemented backend needs (A3′) (Appendix D.2.2).

TreeSync [80] authenticates *group management* (membership and tree consistency) with machine-checked proofs, complementary to Au-CGKA rather than a proof of it. Quarantined-TreeKEM (QTK) [26] uses Shamir sharing so inactive users' key material can be recovered under a *restricted-leakage* threat model, but is not a designated-auditor escrow whose per-commit well-formedness is proven and checked at admission. Au-CGKA is orthogonal: TreeSync concerns who and what is in the group, QTK concerns whether inactive users can resynchronize without unbounded leakage, and Au-CGKA concerns whether each epoch secret is threshold-recoverable by an auditor committee.

Zero knowledge for update paths Devigne, Duguey, and Fouque (DDF) [32] use classical NIZKs to prove well-formedness of MLS update messages, targeting *robustness* against malicious insiders. Our goal is different and our proof covers a different object: we target *auditability*, the recovery of epoch secrets by a designated party, with post-quantum security, and $\mathcal{R}_{\mathsf{A}}$ constrains the escrow rather than the update path. The two are complementary and compose in the obvious way, a deployment wanting both carrying both proofs.

Accountable escrow and auditable surveillance That exceptional access should be *verifiable* is not new. Verifiable (partial) key escrow [9] originates the requirement that an escrow provably contains the right secret; our *silent escrow failure* is that requirement for a per-epoch CGKA seed. Accountable metadata-hiding escrow [50] and AUDIT [39] instead make the *use* of access accountable after the fact, a question orthogonal to an escrow's well-formedness. Closest to our model, *universally composable auditable surveillance* [37] gives a UC functionality for threshold escrow with a judge and a ledger-based audit trail. Those works target generic

decryption. Au-CGKA builds the audit invariant into each CGKA commit message, making every MLS epoch secret's recoverability a per-commit, post-quantum property. To our knowledge, $\mathcal{F}_{\mathsf{aud}}$ is the first ideal functionality for an *auditable CGKA* (extending [5]).

Verifiable encryption and escrow Classical verifiable encryption [20] proves knowledge of a plaintext inside a ciphertext; post-quantum lattice analogues exist both in one-shot form [55] and in recent proof toolkits [56]. Proving the escrow ciphertexts well formed is the part of our statement such an argument discharges natively, in one proof for the whole committee; joining it to the rest would need a value committed in both systems. We prove the whole statement in one system instead, keeping a single extractor and a single assumption class. Committing to the escrowed value keeps the statement that small: clause (iii) ties the escrow to one published digest, so the whole relation has one secret input and the MLS key schedule stays outside the circuit. The share-then-encrypt composition behind our escrow (Shamir shares wrapped under each auditor's KEM) is the standard route to threshold access from non-threshold primitives, dating to fair public-key cryptosystems [60] and threshold cryptosystems [31]. Au-CGKA adds a *STARK-verified* proof that the sharing is well-formed, closing the silent-escrow gap. Proactive secret sharing [46] mitigates slow coalition assembly on long-lived auditor committee shares; the exceptional-access debate [1, 81] and anamorphic channels [69] apply to any escrow design.

Transparency and accountability in messaging Key transparency (KT) systems (CONIKS [59], WhatsApp KT [52], OPTIKS [53], Parakeet [58]) make *identity↔key bindings* auditable and detect equivocation of long-term keys. They do *not* recover MLS epoch secrets or prove escrow well-formedness. Au-CGKA instead audits *epoch-secret recoverability* at the key-agreement layer, and is complementary. Private hierarchical governance (MlsGov) [61] adds in-protocol moderation above MLS, orthogonal to Au-CGKA's invariant on the key schedule.

Abuse reporting and content moderation in E2EE A parallel line of work tackles accountability at the *message* layer: message franking [44], its metadata-private asymmetric variant [75], and Hecate's sealed-sender tokens [48] let a recipient report an abusive message unforgeably, and Scheffler and Mayer [71] systematize the design space. Au-CGKA operates at the key-agreement layer instead: the audited object is a per-epoch key seed, not a plaintext, and no Au-CGKA interface reports or attributes *message content*. The two are complementary.

Post-quantum building blocks ML-KEM [67] and SHAKE [65]/SHA-2 [64] are our on-wire primitives. The escrow ciphertext is a STARK-friendly instantiation of Kyber's CPA core [16] under the *same* Module-LWE assumption as ML-KEM (§A.3.1). Rather than a Fujisaki–Okamoto transform, the integrity of a *logged* escrow ciphertext comes from proving well-formedness in zero knowledge (*verifiable encryption* [20, 55]); this is a property of the logged object, not non-malleability of the scheme. IND-CPA suffices, because there is no escrow-ciphertext decryption oracle. STARKs provide post-quantum transparency [11]; Poseidon2 [40] is used for all in-circuit hashing.

3 Preliminaries

3.1 Notation

Let $\lambda = 384$ denote the security parameter. Key material on the MLS wire — path secrets and the epoch secret — has the RFC 9420 derivation width $N_h = 512$ bits at SHA-512, and the construction escrows it at that width. The escrow coin η the committer samples per commit is λ bits, and the escrow messages μ_j are 256 bits, the message capacity of one escrow-PKE ciphertext. It is η that fixes λ; see Appendix D.2.11 for details. We write $x \xleftarrow{\$} S$ for uniform sampling from a set S, and $x \xleftarrow{\$} \mathsf{Alg}$ for the output of a randomized algorithm Alg. We write $[N]$ for $\{1,\ldots,N\}$ and $\{0,1\}^*$ for the set of all bit strings. A function $\varepsilon : \mathbb{N} \to \mathbb{R}_{\geq 0}$ is *negligible*, written $\varepsilon(\lambda) = \mathsf{negl}(\lambda)$, if it vanishes faster than any inverse polynomial. **Table 5** (Appendix A) summarizes all critical symbols used throughout the paper.

3.2 MLS and TreeKEM

We recall the aspects of MLS (RFC 9420) [7] relevant to this work; see Appendix A.1 for full details. Throughout, we describe a single epoch transition: a commit advances the group from epoch $e-1$ to epoch e, and epoch-dependent quantities carry e as a subscript.

Ratchet tree and update paths MLS maintains a left-balanced binary tree of n_e leaves, each node holding a Hybrid Public Key Encryption (HPKE) [8] (ML-KEM-based) key pair. A node is *blank* if it holds no key pair, for instance after a removal. A *commit* by the member at leaf ℓ replaces the key pair of ℓ and of every node on ℓ's *filtered direct path* [7], the direct path from ℓ to the root with those nodes removed whose copath child has an empty resolution (Appendix A.1.1). We write $m_e = |\mathsf{FDP}(\ell, e)|$ for its length and index the updated nodes $i = 0,\ldots,m_e$, with $i = 0$ the leaf and $i = m_e$ the topmost node the path contains — the tree root except in degenerate trees. The tree *depth* $h_e = \lceil \log_2 n_e \rceil$ satisfies $m_e \leq h_e$, with equality when no node is blank; h_e is used only in cost statements about the unblanked case, and every path secret and ciphertext is indexed by m_e. The symbol ℓ denotes a leaf index

throughout; the published lane count of Appendix D.2.2 is ℓ_{pub}.

Throughout the paper, *commit message* denotes the object the committer posts, and *commits to* is reserved for commitment schemes.

Path secret derivation and node keys The committer samples a fresh per-epoch secret at the RFC 9420 derivation width ($N_h = 512$ bits at SHA-512) and derives a *path secret chain* along the filtered direct path by DeriveSecret, as defined in RFC 9420 [7]: $\mathsf{ps}_i = \mathsf{DeriveSecret}(\mathsf{ps}_{i-1})$. Each updated node takes its key pair from its own path secret,

$$\begin{aligned} \mathsf{ns}_i &= \mathsf{DeriveSecret}(\mathsf{ps}_i, \text{"node"}) \\ (sk_i, pk_i) &= \mathsf{KeyGen}(\mathsf{ns}_i) \end{aligned} \tag{1}$$

and the recipient of ps_i recomputes (1) and checks the resulting pk_i against the key the UpdatePath publishes. The *commit secret* is the topmost path secret, and the *epoch secret* es_e is derived from it and the previous epoch's *init_secret*. es_e is the object Π_{A} escrows, and this derivation is the RFC's, unchanged (§3.3).

Path encryption Each updated node $i \in [m_e]$ has a *copath resolution* of ρ_i HPKE public keys $pk_{i,1}, \ldots, pk_{i,\rho_i}$, the resolution of node i's copath child, i.e. the sibling of node $i-1$. The members holding those keys must receive ps_i, so the committer produces $\mathsf{ct}_{i,k} = \mathsf{KEnc}(pk_{i,k}, \mathsf{ps}_i)$ for $i \in [m_e]$, $k \in [\rho_i]$. The leaf secret ps_0 is never encrypted, so there is no level-0 ciphertext. A member decrypts its ps_{i_0}, recomputes $\mathsf{ps}_{i_0+1}, \ldots, \mathsf{ps}_{m_e}$ via $\mathsf{ps}_i = \mathsf{DeriveSecret}(\mathsf{ps}_{i-1})$, checks each node key by (1), and derives its keys.

Transcript and confirmation Every commit extends the *transcript hash* τ_e, a rolling hash over all public commit data; the *confirmation tag*, an HMAC over τ_e under the epoch's confirmation key, binds the epoch keys to the transcript, so a member who decrypts the wrong ps_i fails confirmation.

Key schedule and external PSKs The epoch secret is $\mathit{epoch\_secret} = \mathsf{HKDF}(\mathit{init\_secret} \| \mathit{commit\_secret})$, from which all application and handshake keys derive (Appendix A.1 gives the exact RFC 9420 schedule). MLS also supports *external pre-shared keys (ePSKs)* [7], which mix out-of-band entropy into the schedule before *epoch_secret*. Π_{A} escrows *epoch_secret* itself, downstream of that mixing, so an audit recovers whatever an ePSK contributed and the construction places no restriction on ePSK proposals.

3.3 Au-CGKA and the RFC 9420 Commit

Au-CGKA is *MLS-shaped*: it keeps the ratchet tree, the resolution rules, the filtered direct path, the wire format, the key schedule and the confirmation mechanism of RFC 9420, and it leaves the generation of a commit's secret material untouched. Π_{A} runs the RFC commit procedure verbatim and then escrows the epoch secret that procedure produces. What changes is the *payload*: a commit carries the commitment c_{root}, the escrow ciphertext $C_{\mathsf{esc}e}$ and the proof π, and group creation carries the same three objects in its GroupInfo (§B.1). Every party additionally holds the group-wide public parameter mpk. Appendix A.1.3 lists the complete set of design changes; because the commit procedure is the RFC's, that list is short, and none of its entries touches how path secrets, node keys or path ciphertexts are produced.

The cryptographic building blocks we use (Shamir secret sharing, ML-KEM, the PRF and hash functions) and the STARK background are recalled in Appendix A.

3.4 Secure-Erasure Model

All our security statements are in the *secure-erasure* model: an honest party deletes each *ephemeral* secret the instant it is no longer needed, retaining only the *persistent* state required for future operations. Concretely, once a committer has posted its commit message $\mathcal{M}_e$ it erases the sharing randomness η, every Shamir share σ_j and encapsulated key K_j, and the STARK witness w_e; the escrowed epoch secret es_e follows the ordinary MLS ratchet, erased once the epoch's keys are derived from it. The committer thereafter keeps only what standard MLS keeps: the decryption keys of the nodes it now owns and the epoch application/handshake secrets it is entitled to. Likewise, a group member erases a path secret, a node secret and a rotated HPKE secret as soon as it has derived the next one (the FS discipline of RFC 9420). The auditor committee is the one exception. Each A_j *must* retain its long-term decapsulation key $sk_j^{\mathbb{A}}$, since audit may occur arbitrarily far in the future. This key is therefore not erasable, and adaptive corruption of A_j unavoidably exposes it. This is exactly the model under which [5] states its adaptive insider security, and it matches Au-CGKA's erasure discipline. A corruption thus hands the adversary only the corrupted party's *current, non-erased* state.

3.5 CGKA Syntax

A CGKA [3] for a group of up to $N_{\max}$ members is a tuple of probabilistic polynomial-time (PPT) algorithms. Two objects recur below. The *group key state gks* is a member's private local state for the current epoch; the *roster* is the list of member identities and their key material, written ros_e once epochs are indexed (§4).

- $\mathsf{Init}(1^\lambda) \to \mathit{gik}$: generates a party's *group info key gik*, the long-term key material with which it creates or joins groups.
- $\mathsf{Create}(\mathit{gik}, \mathit{roster}) \to (\mathit{gks}, \mathcal{M}_0)$: creates a group, returning the creator's initial group key state *gks* and the epoch-0 (Welcome) message $\mathcal{M}_0$.

- $\mathsf{Propose}(gks, \mathsf{op}) \to (p, gks')$: generates a proposal p for operation $\mathsf{op} \in \{\mathsf{add}, \mathsf{remove}, \mathsf{update}\}$.
- $\mathsf{Commit}(gks, \{p_i\}) \to (gks', \mathcal{M}_e, commit\_secret_e)$: commits the pending proposals $\{p_i\}$, producing the next group key state gks', the commit message $\mathcal{M}_e$, and the epoch's commit secret $commit\_secret_e$ (in TreeKEM the root path secret ps_{m_e} of §3.2).
- $\mathsf{Process}(gks, \mathcal{M}_e) \to (gks', commit\_secret_e)$: processes a received commit message, returning the updated state gks' and the same commit secret $commit\_secret_e$.
- $\mathsf{Key}(gks) \to k_e$: extracts the current epoch key k_e from the state.

Security relative to the insider model of [5], and the inheritance of path confidentiality by Au-CGKA, are stated in §6.

4 Auditable CGKA: Syntax and Ideal Functionality

4.1 Extended Syntax

An Au-CGKA is parameterized by an *auditor committee* $\mathbb{A} = \{A_j\}_{j\in[N]}$ of N parties and a *threshold* $t \le N$: any t or more of them can audit an epoch; fewer cannot. We write $T \subseteq [N]$ for the subset of auditors that comes together for a given audit, pp for the public parameters shared by all algorithms, and $\mathcal{L}$ for the append-only public log of commit messages kept by the delivery service.

An Au-CGKA inherits the CGKA algorithms of §3, with Init, Propose, Process, and Key unchanged; it *extends* Create (as GroupCreate) and Commit so their messages carry an escrow and a *proof of correctness*. The commit message becomes $\mathcal{M}_e = (c_{\mathsf{root}}, C_{\mathsf{esc}e}, \pi_e)$: the epoch's *commitment* c_{root}, the *escrow ciphertext* $C_{\mathsf{esc}e}$ carrying the threshold-shared secret, and a proof π_e attesting that epoch e's secret is escrowed to the committee (checked by VerifyCommit). An Au-CGKA further *adds* the following five algorithms.

- $\mathsf{AuditSetup}(1^\lambda, N, t) \to (\mathsf{mpk}, \{\mathsf{ask}_j\}_{j\in[N]}, \mathsf{pp})$: generates the auditor committee's public configuration mpk, comprising the auditors' public keys $\{pk_j^{\mathbb{A}}\}_{j\in[N]}$ and the threshold t; the per-auditor secret keys ask_j, written $sk_j^{\mathbb{A}}$ once instantiated in §5; and the public parameters pp. The keys are for the escrow PKE (§A.3).
- $\mathsf{GroupCreate}(gks, \mathsf{pp}, \mathsf{mpk}, roster) \to \mathcal{M}_0$: extends MLS Create so the epoch-0 message carries an additional epoch-0 escrow proof π_0 attesting that the genesis epoch secret is threshold-escrowed to the auditor committee and bound to the epoch-0 GroupInfo hash (§B.1).
- $\mathsf{VerifyCreate}(\mathsf{pp}, \mathsf{mpk}, \mathcal{M}_0) \to \{0,1\}$: verifies π_0 on the same terms; the log functionality $\mathcal{F}_{\mathsf{Log}}$ that maintains $\mathcal{L}$ (§6) enforces this as an admission check on GroupInfo.
- $\mathsf{VerifyCommit}(\mathsf{pp}, \mathsf{mpk}, \mathcal{L}, \mathcal{M}_e) \to \{0,1\}$: stateless verification from the log, on a statement *recomputed from it* rather than supplied by the committer (§B.2.1). Accepts iff $\mathcal{M}_e$ carries a valid proof π binding the escrow to the commitment $c_{\mathsf{root}} = H_{\mathsf{P2}}(\texttt{"crot"} \| G \| e \| es_e)$. Whether that preimage is the group's own epoch secret is what each member checks against the secret it derives, so VerifyCommit and member processing together are what admit an epoch.
- $\mathsf{Audit}(\{\mathsf{ask}_j\}_{j\in T}, \mathcal{L}, e)$ for $|T| \ge t$: reconstructs the escrowed secret s_e and returns it together with the epoch key, (s_e, k_e). Recovery is per-epoch independent: no other epoch is read. The escrow target is part of the output, since it is what the committee reconstructs and what the ideal AUDIT returns (item (R1), §D.2.5).

4.2 The Ideal Functionality $\mathcal{F}_{\mathsf{aud}}$

We call the guarantee realized by $\mathcal{F}_{\mathsf{aud}}$ *auditability*, and specify it in Fig. 1. $\mathcal{F}_{\mathsf{aud}}$ maintains a *history graph* $\mathcal{H}$ of epoch nodes; each node v_e stores roster ros_e, tree shape $tree_e$, filtered-path length m_e, the escrow target s_e, the epoch key k_e, commitment c_{root}, escrow ciphertext $C_{\mathsf{esc}e}$, proof π_e, the accept set $\mathsf{Acc}_e \subseteq ros_e$, the transcript $\mathcal{M}_e$, and flag honest.

The functionality samples. It is *abstract*: for an honest committer it samples s_e and k_e independently and evaluates no cryptography, so k_e is never derived from s_e inside it, at any interface. The real protocol's relation $k_e = \mathsf{KeySchedule}(es_e)$ is established in the simulation by programming that oracle wherever a legitimate exposure reveals both sides; a functionality computing it would fix the same points, forbid that programming and be unrealizable (§D.2.5). Every hybrid of Appendix D refers to this one functionality.

Figure 1 is the formal definition; we highlight the design points salient to the construction and proofs.

Design points. $\mathcal{F}_{\mathsf{aud}}$ exposes a *single* creation interface and otherwise inherits the proposal/commit/process/key interfaces of [5]. Two things change at the commit. The transcript is the Au-CGKA message $\mathcal{M}_e = (c_{\mathsf{root}}, C_{\mathsf{esc}e}, \pi_e)$, stored opaquely; and for an honest committer the functionality samples s_e and k_e internally, leaking only the public metadata $L(e)$, whereas for a corrupt one $\mathcal{S}$ obtains them from the straight-line extractor run on the logged proof, so what is registered is what the relation proves of the logged data rather than an assumption about $\mathcal{S}$ (Lemma 6). Ideal VerifyCommit is then an equality check against the registered node, and registration is itself conditioned on acceptance: $\mathcal{S}$ registers a corrupt commit message only if the concrete VerifyCommit accepts, including the STARK relation ($\mathcal{R}_{\mathbb{A}}$, Eq. (2)). A mis-escrow therefore fails admission and is never registered. AUDIT returns the recorded pair (s_e, k_e) — the escrow target as well as the key, because real auditors reconstruct the target itself — and epochs are escrowed independently, so recovery is *per-epoch independent*. Since $\mathcal{F}_{\mathsf{Log}}$ admits at most one node per position, group splitting is refused at the log level. Privacy of epoch e holds exactly on executions meeting $\mathsf{safe}_{\mathsf{ESC}}(e)$, that is, fewer than t of its auditors *ever* corrupted.

Functionality $\mathcal{F}_{\mathsf{aud}}$. Parameters: auditor committee size N, threshold t. Interacts with parties $\mathcal{P}$, an auditor committee $\mathbb{A} = \{A_j\}_{j\in[N]}$, and the ideal adversary $\mathcal{S}$. Maintains a history graph $\mathcal{H}$ of epoch nodes v_e, each linked to its parent and storing the *abstract* state $(ros_e, tree_e, m_e, s_e, k_e)$, the accept set $\mathsf{Acc}_e \subseteq ros_e$, the transcript $\mathcal{M}_e$, and the flag honest, where m_e is the length of the committer's filtered direct path at e (§3.2). $\mathcal{C}$ is the corrupt set. Each party has a current node, set on creation or joining and advanced by PROCESS.

Leakage. $L(e) = (ros_e, tree_e, m_e, P_e, PK_e)$: roster, tree shape, filtered-path length, committer, and the ratchet-tree node public keys at e; all are public in the real execution. $\mathcal{S}$ supplies PK_e and the transcript $\mathcal{M}_e$ at every registration; the functionality stores $\mathcal{M}_e$ and only compares it for equality. $T \subseteq \mathbb{A}$ is the auditing subset.

On (GROUPCREATE, G, ros_0) **from** P: require that $(G,0)$ is unoccupied; set ros_0 and derive $tree_0, m_0$. If P is honest, sample $s_0, k_0 \xleftarrow{\$} \{0,1\}^{N_h}$ independently, put $\mathsf{Acc}_0 = ros_0$, and receive $(\mathcal{M}_0, PK_0)$ from $\mathcal{S}$; register v_0 with flag honest and record that s_0 is threshold-escrowed to $\mathbb{A}$. If P is corrupt, receive $(s_0, k_0), \mathsf{Acc}_0, \mathcal{M}_0, PK_0$ from $\mathcal{S}$ and register v_0. Set P's node to v_0; send $L(0)$ to $\mathcal{S}$.

On (PROPOSE, op) **from** P: require $P \in ros_e$ at P's node; record the proposal and send it to $\mathcal{S}$.

On (COMMIT, $\{p_i\}$) **from** P: require $P \in ros_{e-1}$ at P's node v_{e-1} and that (G,e) is unoccupied; validate $\{p_i\}$ and apply them to ros_{e-1}, giving $ros_e, tree_e, m_e$. If P is honest, sample $s_e, k_e \xleftarrow{\$} \{0,1\}^{N_h}$ independently, put $\mathsf{Acc}_e = ros_e$, receive $(\mathcal{M}_e, PK_e)$ from $\mathcal{S}$, register v_e as the child of v_{e-1} with flag honest, return k_e to P, and send $L(e)$ to $\mathcal{S}$. If P is corrupt, receive $(s_e, k_e), \mathsf{Acc}_e \subseteq ros_e, \mathcal{M}_e, PK_e$ from $\mathcal{S}$ and register v_e.

On (PROCESS, $\mathcal{M}_e$) **from** P: if $\mathcal{M}_e$ does not equal the registered transcript, or $P \notin ros_e$, return $\perp$. Else if $P \in \mathsf{Acc}_e$, set P's node to v_e and return k_e; else return $\perp$.

On (VERIFYCREATE, $\mathcal{M}_0$) **from any party** (public): return 1 iff $\mathsf{v}_0 \in \mathcal{H}$ and $\mathcal{M}_0$ equals the registered genesis transcript.

On (VERIFYCOMMIT, $\mathcal{M}_e$) **from any party** (public): return 1 iff $\mathsf{v}_e \in \mathcal{H}$ and $\mathcal{M}_e$ equals the registered transcript.

On (AUDIT, T, e) **from every** $A_j \in T$, $T \subseteq \mathbb{A}$: if $\mathsf{v}_e \notin \mathcal{H}$ or $|T| < t$, return $\perp$. Else return the recorded pair (s_e, k_e) to every $A_j \in T$ *and to* $\mathcal{S}$. No other node of $\mathcal{H}$ is read: recovery is per-epoch independent. No honesty condition is placed on T. An audited epoch is outside $\mathsf{safe}_{\mathsf{ESC}}$.

On (LEAK, e) **from** $\mathcal{S}$: if $\mathsf{v}_e \notin \mathcal{H}$ return $\perp$; if at least t of epoch e's auditors are in $\mathcal{C}$, return (s_e, k_e) to $\mathcal{S}$; else return $\perp$.

On (CORRUPT, P) **from** $\mathcal{S}$ *at any point* (adaptive): add P to $\mathcal{C}$ and hand $\mathcal{S}$ the party's current, non-erased abstract state, namely its roster membership, its current node and k_e at that node. For an auditor A_j this discloses $\mathsf{ask}_j = sk_j^{\mathbb{A}}$ permanently, as it is not erasable. While P stays corrupt, release the same abstract state at each later epoch and mark that epoch outside $\mathsf{safe}_{\mathsf{ESC}}$.

Figure 1: The ideal functionality $\mathcal{F}_{\mathsf{aud}}$. Justifications for the features that realizability forces are in §D.2.5.

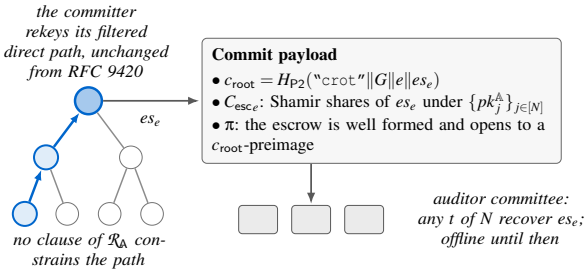

Figure 2: The escrow construction Π_{A} (§1.1). The committer runs the ordinary RFC 9420 commit, then escrows the epoch (root) secret es_e it produces and publishes the commitment c_{root} to it. VerifyCommit checks from the log that the escrow is a well-formed t-of-N sharing of a c_{root}-preimage; each member checks c_{root} against the epoch secret it derives before advancing; and any t auditors later recover the epoch, touching no other. Notation: Table 5.

Acceptance is recorded per party, in the accept set $\mathsf{Acc}_e \subseteq ros_e$. $\mathcal{R}_{\mathsf{A}}$ determines the escrow and nothing else, so the functionality asserts nothing about which roster members reach the epoch, and one fixing $\mathsf{Acc}_e = ros_e$ on every admitted epoch would not be realizable by Π_{A}. The guarantee $\mathcal{F}_{\mathsf{aud}}$ makes is about the epoch's secret, and it is unconditional on the accept set: every admitted epoch's s_e is threshold-recoverable. Appendix D.2.5 shows which of these features are forced by realizability rather than chosen, and §B.1 gives the epoch-0 anchor.

5 Construction

Figure 3 gives an overview of the construction and its audit flow. Π_{A} makes every epoch's secret recoverable by any t-of-N auditors and proves it at commit time under a verifiable Shamir-over-escrow-PKE escrow (§A.3.2, A.3.1). It escrows the epoch's root secret and commits to it, so a mis-escrow fails admission: the stateless VerifyCommit rejects an escrow that is not a well-formed sharing of a c_{root}-preimage, and each member rejects a c_{root} that disagrees with the epoch secret it derived. The committer *proves the sharing and every component of the escrow ciphertext well-formed in-circuit*.

5.1 Π_{A}: Auditability by Committed Escrow

Setup Each A_j generates an escrow-pke keypair $(pk_j^{\mathbb{A}}, sk_j^{\mathbb{A}})$; the escrow PKE is Kyber's CPA core (K-PKE) [16] under Module-LWE, not ML-KEM (§A.3.1). The system publishes $\mathsf{mpk} = (\{pk_j^{\mathbb{A}}\}_{j\in[N]}, t)$. The auditor committee is *entirely offline* until an audit is requested. The fields of each PRF and keyed-stream input (domain labels and indices) are of fixed width.

Commit at epoch e

1. **Root secret and commitment.**
   - Run the MLS commit as usual and let es_e be the resulting epoch (root) secret, from which the key schedule derives k_e.
   - Publish the escrow root $c_{\mathsf{root}} = H_{\mathsf{P2}}(\texttt{"crot"} \| G \| e \| es_e)$.

   The domain separator $\texttt{"crot"} \| G \| e$ is required: without it one fixed function would map secret to digest across every epoch of every group, and the published roots would form

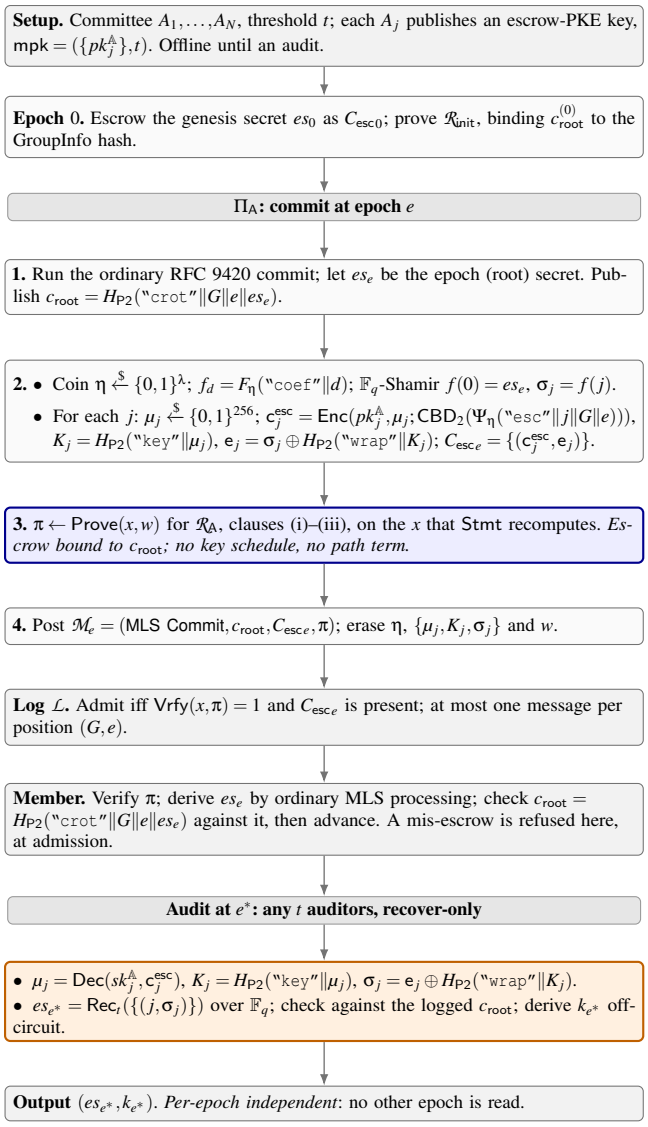

Figure 3: Protocol flow for Π_{A}, read top to bottom: setup, epoch-0 anchoring, the four commit steps, admission at the log and at each member, and audit. Blue is proved in-circuit; orange is audit. Notation in Table 5.

a multi-target preimage problem rather than one single-target problem each (Appendix D.2.2).

2. **Threshold escrow.**
   - Sample an escrow coin $\eta \xleftarrow{\$} \{0,1\}^\lambda$ and derive the sharing coefficients $f_d = F_\eta(\texttt{"coef"} \| d)$ for $d \in [t-1]$.
   - Shamir-share es_e over $\mathbb{F}_q$ (§A.3.2) into $(\sigma_j)_{j \in [N]}$, so $f(0) = es_e$ and $\sigma_j = f(j)$. Because the coefficients are derived from the single λ-bit coin η rather than drawn independently, privacy against $t-1$ auditors is computational in the real scheme, in the random-oracle model for F; the perfect t-privacy the analysis uses is available only once the coefficients are idealized (§A.3.2).
   - For each $j \in [N]$, sample $\mu_j \xleftarrow{\$} \{0,1\}^{256}$ and encrypt it under $pk_j^{\mathbb{A}}$ with CPA escrow-PKE noise $\mathsf{CBD}_2(\Psi_\eta(\texttt{"esc"} \| j \| G \| e))$, giving Module-LWE ciphertext $\mathsf{c}_j^{\mathsf{esc}} = (\mathbf{u}_j, v_j)$. The noise label carries (G, e) for the same single-target reason as c_{root}: the published $\mathbf{u}_j$ is a fixed function of the absorbed coin block under the auditor's long-lived key.
   - Set $K_j = H_{\mathsf{P2}}(\texttt{"key"} \| \mu_j)$ and wrap the share under a one-time pad, $\mathsf{e}_j = \sigma_j \oplus H_{\mathsf{P2}}(\texttt{"wrap"} \| K_j)$.
   - The escrow ciphertext is $C_{\mathsf{esc}e} = \{(\mathsf{c}_j^{\mathsf{esc}}, \mathsf{e}_j)\}_{j \in [N]}$.

   Both the wrap key and the pad masking the share are derived by the field-native H_{P2}, so clause (ii) proves the whole chain $\mu_j \to K_j \to \text{pad} \to \mathsf{e}_j$ in-circuit at $O(1)$ rows per auditor.
3. **Proof.**
   - Compute $\pi \leftarrow \mathsf{Prove}(x, w)$ for the relation $\mathcal{R}_{\mathsf{A}}$ below, on the statement x that Stmt recomputes from the log rather than one the committer supplies (§B.2.1).
   - Absorb x into the proof's Fiat–Shamir transcript, so the challenges are bound to the epoch's published inputs (§7).

$$\mathcal{R}_{\mathsf{A}}: \quad \begin{array}{l} \text{(i) } f(X) = es_e + \sum_{d=1}^{t-1} f_d X^d \text{ over } \mathbb{F}_q \text{ with} \\ f_d = F_\eta(\texttt{"coef"} \| d), \text{ and } \sigma_j = f(j) \text{ for all } j \in [N] \\ \text{(ii) } \forall j \in [N]: \\ \mathsf{c}_j^{\mathsf{esc}} = \mathsf{Enc}(pk_j^{\mathbb{A}}, \mu_j; \mathsf{CBD}_2(\Psi_\eta(\texttt{"esc"} \| j \| G \| e))) \\ \text{(every noise coefficient range-checked),} \\ K_j = H_{\mathsf{P2}}(\texttt{"key"} \| \mu_j), \text{ and} \\ \mathsf{e}_j = \sigma_j \oplus H_{\mathsf{P2}}(\texttt{"wrap"} \| K_j) \\ \text{(iii) } c_{\mathsf{root}} = H_{\mathsf{P2}}(\texttt{"crot"} \| G \| e \| es_e), \text{ on the same } es_e \\ \text{that is clause (i)'s constant term} \end{array} \tag{2}$$

Public inputs: $(G, e, \mathsf{ctx}_e, \mathsf{mpk}, C_{\mathsf{esc}e}, c_{\mathsf{root}})$, where $\mathsf{mpk} = (\{pk_j^{\mathbb{A}}\}_{j \in [N]}, t)$ carries the auditor keys and ctx_e binds the statement to the epoch's public context.

Witness: $(es_e, \eta, \{\mu_j, K_j, \sigma_j\}_{j \in [N]})$ with $\mu_j \in \{0,1\}^{256}$ and $\sigma_j \in \mathbb{F}_q$; κ is the escrow-PKE module rank and 256 the ring degree (§A.3.1).

The commit message is $\mathcal{M}_e = (\mathsf{MLS\ Commit}, c_{\mathsf{root}}, C_{\mathsf{esc}e}, \pi)$. $\mathcal{R}_{\mathsf{A}}$ asserts nothing beyond well-formed threshold escrow of a c_{root}-preimage: no in-circuit $\mathsf{KeySchedule}(es_e) = k_e$, no SHA-512 key schedule and no confirmation tag, its only hash work being the Poseidon2 commitment and the escrow clauses. The epoch key is bound to the escrow by the two-stage argument of §1: members hold es_e from the ordinary MLS commit and check c_{root} before advancing, and audit derives k_e from the opened escrow off-circuit under CGKA path secrecy (Theorem 1). A mis-escrow is refused at admission. The relation carries no term indexed by the path, so its statement, its witness and its cost are the same at every group size.

Erasure Once $\mathcal{M}_e$ is posted the committer erases the escrow ciphertext ephemerals $\eta, \{\mu_j, K_j, \sigma_j\}$ and the witness w per the discipline of §3.4; es_e follows the normal MLS

ratchet. Only the auditor committee's escrow ciphertext can then recover es_e. This is the discipline the adaptive proof relies on (§6).

Audit at epoch e^* (recover-only, per-epoch independent) Any $T \subseteq [N]$ with $|T| = t$:

- Each $A_j \in T$ decrypts $\mu_j \leftarrow \mathsf{Dec}(sk_j^{\mathbb{A}}, \mathsf{c}_j^{\mathsf{esc}})$.
- Each A_j sets $K_j = H_{\mathsf{P2}}(\text{"key"} \| \mu_j)$ and unwraps $\sigma_j = \mathsf{e}_j \oplus H_{\mathsf{P2}}(\text{"wrap"} \| K_j)$.
- The committee Lagrange-interpolates $es_{e^*} = \mathsf{Rec}_t(\{(j, \sigma_j)\}_{j \in T})$ over $\mathbb{F}_q$ and checks it against the logged c_{root}.
- It derives $k_{e^*} \leftarrow \mathsf{KeySchedule}(es_{e^*})$ *off-circuit* and returns the pair (es_{e^*}, k_{e^*}).

No other epoch is touched. An epoch whose escrow is unrecoverable therefore costs exactly that epoch, and an audit of epoch e^* costs $O(1)$ rather than $O(e^*)$.

What is proved where VerifyCommit rests on three mechanisms. *In circuit*: the escrow and its sharing (i)–(ii) and the commitment (iii). *Recomputed from the log by* Stmt: mpk, the epoch position and ctx_e. *Left to member processing*: the equality of c_{root} with the commitment to the epoch secret the member itself derived, and the MLS confirmation tag. No other component of a commit is claimed to be verified; in particular the update path is the RFC's and no clause of $\mathcal{R}_{\mathsf{A}}$ constrains it.

Group creation (epoch 0) is a base case the construction handles directly, escrowing the genesis epoch secret; we defer it to Appendix B.1.

6 Universal Composability

Π_{A} realizes auditability in the UC framework: it realizes $\mathcal{F}_{\mathsf{aud}}$ (Fig. 1, §4). A realization inherits privacy against sub-threshold coalitions and threshold recoverability. We summarize the model and result here. The complete treatment is the self-contained security analysis of Appendix D, covering the threat model, the simulator and hybrids, the game-based properties, and the adaptive post-quantum UC proof.

Threat model We prove security against an *adaptive*, quantum adversary in the secure-erasure model. It drives creation, proposals, and commits and, as a *corrupt committer*, may submit an adversarially crafted commit message. Adaptive corruption of any member or auditor reveals that party's current, non-erased state, including an auditor's decryption key. Path ciphertexts also have a chosen-ciphertext oracle (the MLS wire uses ML-KEM). The escrow ciphertext has no decryption oracle, because auditors decapsulate only at audit. IND-CPA security therefore suffices (§A.3.1). The committee-corruption budget is *global in time*: fewer than t members of an epoch's committee are *ever* corrupted. The fixed committee and the archived escrow ciphertext allow a coalition of t or more auditors to reconstruct the escrowed secret by design (the auditor exception). Rem. 1 explains why the auditor committee cannot be forward-secret, so a coalition of t auditors assembled at any time recovers every past epoch; Appendix B.3 records a deployment add-on that narrows the window in which such a coalition can be assembled, at fixed committee keys. Five post-quantum assumptions carry the theorem, and we refer to them by these numbers throughout:

(A1) *Module-LWE encryption security*: IND-CPA of the escrow PKE and IND-CCA2 of ML-KEM-768 for the path wire.

(A2) *STARK zero-knowledge* in the QROM, with error $\varepsilon_{\mathsf{ZK}}$.

(A3) *STARK straight-line knowledge soundness* in the QROM, with error $\varepsilon_{\mathsf{ks}}$, extraction being online from the compressed-oracle database. Plain knowledge soundness suffices because the hybrid sequence extracts *before* it simulates: at that point the honest proofs are still real, so the reduction still holds their witnesses and produces them with the real prover, and the extractor is never run alongside a simulation oracle. The stronger property, simulation-extractability, is therefore not assumed (Appendix D.2.2).

(A4) *Commitment binding*: collision resistance of H_{P2}, with error $\varepsilon_{\mathsf{bind}}$.

(A5) *CGKA path secrecy*: the adaptive insider-security hypothesis of [5] for a TreeKEM path layer whose node keys are freshly generated, with error $\varepsilon_{\mathsf{CGKA}}$. Π_{A} runs exactly that layer, unmodified: it adds a payload to the commit message and changes no path secret, node key or ciphertext, so the hypothesis is applied to its own object and needs no bridging lemma. We do not re-prove path-secret confidentiality.

The proof is carried out in the $(\mathcal{F}_{\mathsf{NIZK}}, \mathcal{F}_{\mathsf{Log}}, \mathcal{G}_{\mathsf{RO}}, \mathcal{F}_{\mathsf{PKI}})$-hybrid model, so the realization statement below assumes no property of the proof system: $\mathcal{F}_{\mathsf{NIZK}}$ is the ideal NIZK, and (A2)–(A4) enter only when it is instantiated (Cor. 2). The model conditions, the QROM techniques the proof uses, the lifting of (A2)–(A4) to the bundle of sub-proofs that VerifyCommit verifies, the adversary's two goals, and the instantiation assumption (A3′) are stated in Appendix D.

Safety predicate Following the standard CGKA convention [3,5], we state the realization *with respect to a safety predicate*. The predicate is the admissible corruption/interaction pattern under which an honest epoch's key must remain secret: for an honest challenge epoch e,

- $\mathsf{safe}_{\mathsf{ESC}}(e)$: the epoch is CGKA-fresh (committer honest, no exposed on-path secret, no decryption-oracle shortcut) *and* fewer than t of epoch e's auditors are *ever* corrupted.

$\mathsf{safe}_{\mathsf{ESC}}$ is the privacy game's freshness predicate Fresh (Appendix D.3.1).

Theorem 1 (Adaptive post-quantum UC security of Π_{A})**.** *In the secure-erasure model and the programmable global* quantum *random-oracle model (QROM) [14], with $N = O(1)$, Π_{A} UC-realizes $\mathcal{F}_{\mathsf{aud}}$ with respect to $\mathsf{safe}_{\mathsf{ESC}}$ in the $(\mathcal{F}_{\mathsf{NIZK}}, \mathcal{F}_{\mathsf{Log}}, \mathcal{G}_{\mathsf{RO}}, \mathcal{F}_{\mathsf{PKI}})$-hybrid model, against* quantum *polynomial-time (QPT) adversaries that adaptively corrupt any number of group members and, for each honest epoch, fewer than t of its auditors, under the QROM forms of (A1), (A4) and (A5).*

Corollary 2 (Instantiation)**.** *Replace $\mathcal{F}_{\mathsf{NIZK}}$ by the implemented multi-stage AIR backend of §7. Under the instantiation assumption (A3′) of Appendix D.2.2, the realization holds against the same adversary class, and the advantage of any QPT environment is bounded by Eq.* (6)*.*

The theorem follows one hybrid argument: $\mathcal{S}$ equivocates escrow openings and programs $\mathcal{G}_{\mathsf{RO}}$ on adaptive corruption/audit, reducing REAL to $\mathcal{F}_{\mathsf{aud}}$ on every $\mathsf{safe}_{\mathsf{ESC}}$-respecting execution (gap Eq. (6)). $\mathcal{R}_{\mathsf{A}}$ binds only the escrow and c_{root}, and the commit message is carried on an ordinary TreeKEM UpdatePath. In particular, $\mathsf{KeySchedule}(es_e) \leftrightarrow k_e$ is charged to CGKA path secrecy, not to a missing in-circuit clause of $\mathcal{R}_{\mathsf{A}}$: what makes the escrowed value the members' own epoch secret is their admission-time c_{root} check, so a divergence is an H_{P2} collision rather than an unproven step. CGKA path secrecy is the adaptive CGKA insider-security hypothesis of [5], taken as an explicit black box and applied to Π_{A}'s unchanged TreeKEM update. The simulator, QROM assumptions, hybrids, and full proofs are in Appendix D (§D.2.2–D.2.8).

The condition $N = O(1)$ is not cosmetic: Eq. (6) charges the escrow term over $2^N NE$ events, so 128 bits requires $N + \log_2 N + \log_2 E \le 36$, which is $N \le 12$ at $E \le 2^{20}$ epochs and $N \le 8$ at $E \le 2^{24}$. We measure at N=5; §D.2.11 derives the ceiling.

Game-based security properties UC realization implies the game-based guarantees as corollaries: privacy with the auditor exception, and escrow soundness. Their games, exact advantage bounds, and reductions are in Appendix D, together with the scope statement that keeps the two halves of admission apart: what VerifyCommit establishes from the log alone, and what each member establishes against the secret it derived.

7 Implementation

Software stack and what is prototyped We prototype the Au-CGKA *relation*, the escrow it constrains, and the MLS path updates it rides on. This is a modified MLS stack, the ciphersuite is custom (Appendix A.1.3). The measured harness runs one recipient per copath level, with no message signing. It proves the relation on real witnesses end to end, and we evaluate the costs. We implement Au-CGKA in Rust, targeting the MLS key schedule and hash format (RFC 9420): the SHA-512 DeriveSecret/HKDF chain, the SHAKE256 PRF, and the TreeKEM path-secret derivation. On top of this we build the evaluated configuration, an *optimized custom AIR* (algebraic intermediate representation) on a multi-stage Plonky3 [70] backend with dedicated ML-KEM chips, wired by an assembler into a single multi-stage proof and committed throughout under a FRI-based [10] *hiding* polynomial commitment scheme (PCS). We also realize the relation as a single proof in the general-purpose SP1 [73] zkVM, prototyped and measured in §8. ML-KEM-768 is provided by the formally verified, constant-time `libcrux` [30]. A standalone audit tool replays a log snapshot.

Circuit architecture The relation is discharged by two families of chips; a third, the SHA-512 chip the stack also implements, is *not* exercised by $\mathcal{R}_{\mathsf{A}}$, which evaluates no MLS derivation in circuit. *(1) An ML-KEM chip* proves each encryption in the escrow ciphertext $C_{\mathsf{esc}e}$ under the escrow PKE, which is Kyber's CPA core (K-PKE) [16] rather than ML-KEM itself (§A.3.1). The chip is built from a modular multiply, the NTT butterfly, inner products, and their chained composition (Table 7, Appendix C). Written in the time domain, encryption is $\mathbf{u} = \mathrm{INTT}(\hat{\mathbf{A}}^{\top} \circ \hat{\mathbf{r}}) + \mathbf{e}_1$ and the analogous accumulation for v, so per auditor it is one noise sampler, κ forward transforms, κ accumulations and the compression to the logged ciphertext. The matrix $\hat{\mathbf{A}}$ is a public input and costs nothing in circuit. Appendix C states what is assembled and what is not. *(2) A field chip* handles the Poseidon2 commitment and PRF derivations, which are field-native over BabyBear ($O(1)$ rows each), together with the Shamir escrow over $\mathbb{F}_q$. The prototype instantiates $\mathbb{F}_q$ at $q_{\mathsf{S}} = \mathrm{nextprime}(2^{256})$, whose 17 base-$2^{16}$ limbs and single signed carry chain per share cost about half of what a prime above 2^{512} would; because the STARK field is BabyBear, each share evaluation and Lagrange step is a small, constant number of range-checked multi-limb modular multiplications rather than one field operation. §A.3.2 carries an N_h-bit secret over that field and shows why doing so changes no relation clause and no security statement.

A multi-stage lookup prover LogUp [45] needs a running-sum column built from a challenge drawn *after* the looked-up values are committed, which Plonky3's [70] default `uni-stark` cannot host. We therefore build a multi-stage prover over its public primitives (two-adic FRI [10], duplex-sponge transcript, evaluation-domain helpers), with a range check over a fixed table and a tuple lookup instantiated as an XOR table. Lookups are *batched* into one running sum separated by powers of a further challenge, without which unbalanced lookups cancel and the argument accepts an unbalanced multiset; we test this with a crafted cancellation attack. And the ML-KEM chips bind computation to those lookups inside a single proof, so a chip's output feeds the

next and tampering with an intermediate fails the consuming constraint rather than only an opening.

Validation Every self-contained cryptographic operation the construction needs is implemented as a Plonky3 AIR, proven, verified, and checked to reject a tampered public input. The three hash primitives of §3 are validated against official test vectors; and the ML-KEM arithmetic, the CBD_2 sampler, the compression maps and the $\mathbb{F}_q$ Shamir evaluation are adversarially tested against plain-Rust references. Every clause is then assembled on the engine and measured end-to-end (§8), with a cross-table binding bus enforcing producer/consumer equality on the shared secret. Appendix C gives the component-by-component account.

Cross-clause binding A bundle needs one witness consistent across its sub-proofs, not a sum of independent ones. The escrowed secret es_e flows from the Shamir sharing (i) into the escrow wrap (ii) and the commitment (iii). Clause (i) shares that secret directly and publishes a commitment in the sharing's packing, so a single ℓ_{pub}-lane equality ties the escrowed secret to the commitment the members check. Without it a committer could Shamir-share one value while publishing another's commitment. That is a silent escrow failure that no per-clause check detects.

Statement binding Every chip's Fiat–Shamir transcript absorbs a domain-separated digest of the published statement — the committee configuration, the epoch context, the logged escrow ciphertext, c_{root}, the cross-proof ties and the parameter set — before any commitment, so a transcript is bound to its statement by construction rather than by a downstream check. The digest is absorbed rather than the pin columns, which are full-height and are a deterministic function of the statement that `Stmt` recomputes; the cost is one sponge per commit, measured at 0.28 ms, and the proof is unchanged because the verifier recomputes the digest from inputs it already holds. This is what puts the compiled argument inside the hypotheses of the QROM analyses of Appendix D.2.2; a regression proves under one statement and verifies under another on a chip that publishes no pin, so the property is decided by the transcript alone.

Two mechanisms enforce it. A cross-chip tie folds column equality into the owning chip's quotient polynomial without publishing an extra opening; a cross-table bus, a tuple-LogUp [45] multiset equality over committed limbs, ties produced and consumed copies, and the bundle verifies only if the multisets match. The harness checks both a consistent witness (accept) and a deliberately inconsistent one (reject). Every clause is proven under the hiding PCS, so the transcript carries only truncated Poseidon2 commitments, public ciphertexts and public context, with no es_e, η or plaintext share in it. Appendix C records the stacking and FRI configuration behind the measured bundle, its transcript surface, and the residual assumptions.

Table 2: Multi-stage lookup STARK under the hiding PCS (BabyBear, single core). Proof size in KB, prove/verify in ms.

Trace height $\mathcal{T}$	Proof size	Prove	Verify
$2^4 = 16$	235.7	49.6	1.7
$2^6 = 64$	306.1	63.2	2.0
$2^8 = 256$	386.5	83.3	2.1
$2^{10} = 1024$	476.8	119.7	2.3

FRI parameters and cost The challenge extension is degree 10 over BabyBear, set by the Grover bound on Fiat–Shamir: knowledge soundness is at most $\frac{1}{2}\log_2|\mathbb{F}_{\mathrm{ext}}|$, which a quartic extension would cap at 62 bits (Appendix D.2.2). A single multi-stage lookup STARK serializes to ∼236–477 KB across trace heights and verifies in 2–3 ms (Table 2); proof size grows slowly with the trace because the FRI query openings dominate and are near-constant. Stacking same-shape sub-computations then collapses the full-relation bundle to 2 proofs / 15.31 MB, proving in 1.38 s (§8 prove/verify and sizes). Table 3 gives the non-proof costs under ML-KEM-768: commit-message sizes excluding π, committer/member latency, and 3-of-5 audit.

The proof is not retained π is verified once by the delivery service at admission and once by each member before it advances, and is then of no further use: `VerifyCommit` is stateless, and `Audit` reads only $C_{\mathsf{esc}e}$ and c_{root}. A deployment may therefore drop π once the epoch is admitted, retaining the 5 637 B those two objects occupy (Table 3). The 15.31 MB is a bandwidth and latency cost on the commit, not a term in the archive a long-retention setting carries, and it grows no party's stored state. A deployment that wants the log independently checkable afterwards retains π and pays that per epoch; $\mathcal{F}_{\mathsf{aud}}$ reads neither, so §6 is unaffected either way.

8 Evaluation

Our evaluation answers three questions. **(Q1)** What does auditability cost *off* the proof, in commit-message bytes, per-commit committer work, and audit throughput? **(Q2)** What does it cost to prove the escrow relation $\mathcal{R}_{\mathsf{A}}$ in zero knowledge on our custom multi-stage STARK backend? **(Q3)** How does that dedicated backend compare with running the same relation as a single proof in a general-purpose zkVM? *Every proving figure on the custom backend is zero-knowledge*: under the hiding PCS of §7 the transcript reveals nothing beyond the public inputs. The guarantee is computational, at $\varepsilon_{\mathsf{ZK}} \le 2^{-299}$ against the hiding FRI commitment (Lemma 11); every cross-proof tie the bundle publishes carries a secret salt derived from η, so no weaker one-wayness term stands against that figure. Appendix D.2.9 gives the execution level.

Table 3: Non-STARK costs, excluding the proof π. Top: commit-message sizes in bytes. Bottom: per-commit latency and 3-of-5 audit cost. Apple M5 Pro, N=5, t=3, ML-KEM-768.

Constr.	h_e	MLS base	c_{root}	escrow	added	overhead %
Π_{A}	4	5 808	32	5 605	5 637	97
	10	12 816	32	5 605	5 637	44
	20	24 496	32	5 605	5 637	23

Constr.	h_e	commit (ms)	audit (μs)
Π_{A}	any	0.031	93

Parameters. The figures are measured at the deployed parameters, the prototype derives its FRI query count from the target itself ($nq \cdot lb \geq 2(128+24+4)$, so $nq = 156$ at $lb = 2$, which against Π_{A}'s own 2^{21} event class leaves 135 quantum bits after the knowledge-soundness charge), and every published commitment is width-32 truncated to $\ell_{\mathsf{pub}} = 16$ lanes. Every row of Table 8 therefore applies to the numbers below, the weakest being the 137-bit MMCS digest. The rows do not share a proof status, and the levels are conditional on (A3′) and on the ethSTARK conjecture behind the FRI row (Appendix D.2.2).

Measurement methodology All figures come from the reference implementation (§7) on an Apple M5 Pro (48 GB). ML-KEM-768 is the formally verified `libcrux` [30]; SHA-512/HKDF and Shamir field arithmetic are exact; the in-circuit commitment hash H_{P2} is Poseidon2 over Baby-Bear (§3); the PRF F and the path AEAD use SHAKE256, while the escrow ciphertext's wrap key and pad are derived by H_{P2} so that clause (ii) proves them in-circuit at field-native cost. Byte sizes are exact serialized lengths at true ML-KEM-768 sizes. Auditor committee N=5, threshold t=3 throughout. Each figure is the arithmetic mean of five runs at the operating point and of three per depth in the sweep. The harness runs the *unblanked* tree, $m_e = h_e$ and $\rho_i = 1$: blanking shortens the filtered path and widens the resolutions, which moves bytes between levels of the MLS base but changes no figure for Π_{A}, whose relation is indexed by neither (§B.2.3).

The reported figures use the STARK-friendly Module-LWE committee escrow, whose in-circuit derivation is Poseidon2 and so needs no Keccak on that path; the standardized ML-KEM escrow (§A.3.1) is implemented but not reported. The MLS update-path ciphertexts are ML-KEM-768 throughout.

Each end-to-end bundle prove time is the *arithmetic mean of five successful wall-clock runs*, measured around the whole prove (resp. verify) call on one coherent witness rather than summed from per-chip microbenchmarks; proof size is deterministic. Measurements are taken on a verified-idle machine, under the thread and scheduling settings of Appendix C. Absolute seconds are single-machine research-prototype figures and drift by a few percent with sustained load.

Q1: Non-proof cost Au-CGKA adds a fixed escrow ciphertext of 5 605 B to each commit message, plus the epoch's commitment (Table 3). That figure is N compressed escrow-PKE ciphertexts plus one wrap each (Appendix E.1); the harness realizes the escrow-PKE and H_{P2} paths as reference stand-ins (SHAKE) at those wire sizes, and the path ML-KEM and SHA-512 operations dominate them, so the figure is a lower bound on the committee half of a commit. At group size 1024 (h_e=10) the MLS update path is 12 816 B, and Au-CGKA adds 5 637 B: the escrow ciphertext is the dominant part and does not grow with the tree, the commitment c_{root} contributing 32 B once. That 32 B is the stand-in digest width; at the deployed ℓ_{pub}=16 lanes the commitment is 64 B, which adds 32 B to the added column at every depth. Because the addition is fixed and the base grows, the relative overhead falls as the group grows, from 97% at h_e=4 to 23% at h_e=20 (a group of $2^{20} \approx 10^6$): at small group sizes the addition is comparable to the update path itself, and only at $h_e \geq 20$ does it become a modest fraction of it. The baseline here is a pure ML-KEM-768 MLS update path (1 168 B per level); standardized RFC 9420 ciphersuites carry far smaller path ciphertexts, against which the relative overhead is correspondingly larger. Committer cryptography outside the proof system is sub-millisecond and independent of the group: a commit costs 0.031 ms at every depth (Table 3, bottom), dominated by the N committee encryptions. The member-side addition is a single hash of the epoch secret it already holds, which is why no separate verify figure is meaningful. The figures exclude the MLS signature, which this prototype does not perform. A 3-of-5 audit costs 93 μs and touches one epoch, so auditing epoch e^* costs 93 μs irrespective of e^*. These costs are independent of the proof system, which dominates the total.

Q2: Zero-knowledge backend cost We assemble every clause of the relation (Eq. (2)) on the custom backend. At group size 1024, N=5, on one coherent real-witness instance, Π_{A} proves in **1.38 s** and verifies in **0.17 s** for a **15.31 MB** bundle of **2** sub-proofs. Prove and verify times are the mean of five runs at the operating point and the proof size is deterministic. The cost is dominated by the N in-circuit lattice encryptions and the hiding-PCS commitments; the bundle discharges no Keccak permutation, no DeriveSecret link and no path NTT, because no clause asks for one. Beyond the stacking of §7, two mechanisms hold these costs down: row-independent chips (compression, CBD, parse, matrix–vector) are concatenated into one proof each, and the sub-computations fixed by public inputs (the key hash and the ML-KEM matrix A) are recomputed by the verifier rather than proven.

Q3: the zkVM realization As a general-purpose single-proof alternative we run the relation checker for $\mathcal{R}_{\mathsf{A}}$ as a guest program in the SP1 STARK zkVM. The guest escrows under the standardized ML-KEM instantiation, with the `libcrux` ML-KEM-768 encapsulations proven in-circuit, whereas the

custom-backend figures use the STARK-friendly Module-LWE escrow (§A.3.1), so the ratio below bounds the backend difference rather than isolating it.

At the operating point of Q2, with SP1 proving shard-parallel across all 18 cores, the guest runs 1.97 M cycles and Π_{A} proves in **30.0 s**, ∼22× the dedicated-AIR backend's figure; the ratio compares recursively compressed proof to recursively compressed proof, and both backends use the same 18 cores. In exchange the zkVM emits a single recursively compressed proof of constant size **1.21 MiB** that verifies in ∼**27 ms**: a ∼12× size reduction for that ∼22× prove-time increase. Times are means of five runs. The two therefore trade prove time against proof size, and Appendix E.2 states the relation under which the custom bundle can be aggregated to the same constant size.

The cycle count matches that accounting: the 1.97 M cycles cover the N-committee escrow and one commitment, with no key-schedule and no path work in the guest. These are conservative CPU figures, since SP1 lacks ML-KEM precompiles and the guest therefore runs as plain RISC-V. The depth sweep confirms what the relation already implies: every measured depth returns the same cycle count and the same 30.0 s, because the guest program's work does not depend on the tree.

Scaling with group size Table 4 sweeps the update-path depth h_e from 4 to 20 (group size $n = 2^{h_e}$, i.e. 16 to ∼1M users), reporting the commit-message bytes alongside the proof cost on both backends.

$\mathcal{R}_{\mathsf{A}}$ contains no update-path term at all, so Π_{A} is independent of the group *by construction* — its cost is $O(N)$ in the committee size and $O(1)$ in the group size, N being a protocol constant. At every depth it proves the same 2 sub-proofs over the same witness shape, for a byte-identical 15.31 MB proof; across the full $h_e = 3..20$ sweep the measured prove time spans 1.37–1.49 s, and the residual spread is run-to-run and thermal drift rather than depth. We quote the range rather than the point for that reason. The zkVM path is flat for the same reason, at 30.0 s and a constant 1.21 MiB. Only the wire cost varies, as Q1 describes.

Table 4: Cost versus group size for Π_{A}. The proof columns do not move: $\mathcal{R}_{\mathsf{A}}$ has no clause indexed by a path level, so the prover does the same work at every depth — the same 2 sub-proofs and a byte-identical proof — and the zkVM guest is flat for the same reason. What moves is the wire: the escrow is a fixed addition to a base that grows with the tree, so the relative overhead falls by a factor of four across the range. Apple M5 Pro, N=5, t=3, ML-KEM-768.

		commit message (B)			proof	
h_e	group	MLS base	Au-CGKA	ovh. %	prove (s)	size
custom multi-stage STARK						
4	16	5 808	11 445	97	1.38	15.31 MB
10	1 024	12 816	18 453	44	1.38	15.31 MB
20	∼1M	24 496	30 133	23	1.38	15.31 MB
SP1 zkVM, recursively compressed						
4	16	—	—	—	30.0	1.21 MiB
10	1 024	—	—	—	30.0	1.21 MiB
20	∼1M	—	—	—	30.0	1.21 MiB

Prove time is the mean of five runs at the operating point; across the full $h_e = 3..20$ sweep it spans 1.37–1.49 s, a spread that is run-to-run and thermal drift rather than depth. Proof size is deterministic. Verification is 0.17 s on the custom backend and ∼27 ms on the zkVM, at every depth. The commit-message columns are the same measurement as Table 3 and are a property of the wire, not of the backend.

9 Conclusion

We show that MLS-shaped group messaging can make epoch-key recoverability a property each commit message carries and every honest member checks. We formalize this as a UC ideal functionality $\mathcal{F}_{\mathsf{aud}}$ that extends the MLS insider-security model with group-creation, audit, and verification interfaces, and we give a post-quantum construction that realizes it. Auditability here is a verifiable property of the key-agreement protocol itself: Π_{A} fixes recoverability of the epoch secret at admission, before any member advances, and an audit then recovers one epoch without reading another. The scope is what keeps the cost low. Because the relation commits to the escrowed value rather than re-deriving it, the MLS key schedule and the update path both stay outside the circuit, and the proof's cost is independent of the group. On our prototype the dominant cost is proof generation and proof size, not verification: under ML-KEM-768, an auditability commit message proves in 1.38 s and verifies in 0.17 s at 15.31 MB, at every group size.

References

[1] Harold Abelson, Ross Anderson, Steven M. Bellovin, Josh Benaloh, Matt Blaze, Whitfield Diffie, John Gilmore, Matthew Green, Susan Landau, Peter G. Neumann, Ronald L. Rivest, Jeffrey I. Schiller, Bruce Schneier, Michael A. Specter, and Daniel J. Weitzner. Keys under doormats: Mandating insecurity by requiring government access to all data and communications. *Journal of Cybersecurity*, 1(1):69–79, 2015.

[2] Joël Alwen, Benedikt Auerbach, Mirza Ahad Baig, Miguel Cueto Noval, Karen Klein, Guillermo Pascual-Perez, Krzysztof Pietrzak, and Michael Walter. Grafting key trees: Efficient key management for overlapping groups. In *TCC*, pages 222–253, 2021.

[3] Joël Alwen, Sandro Coretti, Yevgeniy Dodis, and Yiannis Tselekounis. Security analysis and improvements for the IETF MLS standard for group messaging. In *CRYPTO*, pages 248–277, 2020.

[4] Joël Alwen, Sandro Coretti, Daniel Jost, and Marta Mularczyk. Continuous group key agreement with active security. In *TCC*, pages 261–290, 2020.

[5] Joël Alwen, Daniel Jost, and Marta Mularczyk. On the insider security of MLS. In *CRYPTO*, pages 34–68, 2022.

[6] Andris Ambainis, Mike Hamburg, and Dominique Unruh. Quantum security proofs using semi-classical oracles. In *CRYPTO*, pages 269–295, 2019.

[7] Richard Barnes, Benjamin Beurdouche, Raphael Robert, Jon Millican, Emad Omara, and Katriel Cohn-Gordon. The Messaging Layer Security

(MLS) Protocol. Request for Comments 9420, Internet Engineering Task Force (IETF), 2023. URL: https://www.rfc-editor.org/info/rfc9420.

[8] Richard Barnes, Karthikeyan Bhargavan, Benjamin Lipp, and Christopher A. Wood. Hybrid Public Key Encryption. Request for Comments 9180, Internet Engineering Task Force (IETF), 2022. URL: https://www.rfc-editor.org/info/rfc9180.

[9] Mihir Bellare and Shafi Goldwasser. Verifiable partial key escrow. In *ACM CCS*, pages 78–91, 1997.

[10] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast Reed–Solomon interactive oracle proofs of proximity. In *ICALP*, pages 14:1–14:17, 2018.

[11] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. IACR Cryptology ePrint Archive, Report 2018/046, 2018. URL: https://eprint.iacr.org/2018/046.

[12] Eli Ben-Sasson, Dan Carmon, Yuval Ishai, Swastik Kopparty, and Shubhangi Saraf. Proximity gaps for Reed–Solomon codes. In *FOCS*, pages 900–909, 2020.

[13] Benjamin Beurdouche, Eric Rescorla, Emad Omara, Srinivas Inguva, and Alan Duric. The Messaging Layer Security (MLS) Architecture. Request for Comments 9750, Internet Engineering Task Force (IETF), 2025. URL: https://www.rfc-editor.org/info/rfc9750.

[14] Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. Random oracles in a quantum world. In *ASIACRYPT*, pages 41–69, 2011.

[15] Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter M. R. Rasmussen, and Amit Sahai. Threshold cryptosystems from threshold fully homomorphic encryption. In *CRYPTO*, pages 565–596, 2018.

[16] Joppe W. Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Kyber: A CCA-secure module-lattice-based KEM. In *IEEE EuroS&P*, pages 353–367, 2018.

[17] Luís T. A. N. Brandão and René Peralta. NIST First Call for Multi-Party Threshold Schemes. NIST Internal Report (IR) 8214C ipd (Initial Public Draft), National Institute of Standards and Technology, 2023. URL: https://doi.org/10.6028/NIST.IR.8214C.ipd.

[18] Gilles Brassard, Peter Høyer, and Alain Tapp. Quantum cryptanalysis of hash and claw-free functions. In *LATIN*, pages 163–169, 1998.

[19] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *IEEE S&P*, pages 315–334, 2018.

[20] Jan Camenisch and Victor Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *CRYPTO*, pages 126–144, 2003.

[21] Ran Canetti. Universally composable security. *Journal of the ACM*, 67(5):28:1–28:94, 2020.

[22] Ran Canetti, Uriel Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In *STOC*, pages 639–648, 1996.

[23] Ran Canetti, Abhishek Jain, and Alessandra Scafuro. Practical UC security with a global random oracle. In *ACM CCS*, pages 597–608, 2014.

[24] André Chailloux, María Naya-Plasencia, and André Schrottenloher. An efficient quantum collision search algorithm and implications on symmetric cryptography. In *ASIACRYPT*, pages 211–240, 2017.

[25] Marina Checri, Renaud Sirdey, Aymen Boudguiga, and Jean-Paul Bultel. On the practical CPA^D security of "exact" and threshold FHE schemes and libraries. In *CRYPTO*, pages 3–33, 2024.

[26] Céline Chevalier, Guirec Lebrun, Ange Martinelli, and Abdul Rahman Taleb. Quarantined-TreeKEM: A continuous group key agreement for MLS, secure in presence of inactive users. In *ACM CCS*, pages 2400–2414, 2024. IACR ePrint 2023/1903.

[27] Alessandro Chiesa, Peter Manohar, and Nicholas Spooner. Succinct arguments in the quantum random oracle model. In *TCC*, pages 1–29, 2019.

[28] Katriel Cohn-Gordon, Cas Cremers, Luke Garratt, Jon Millican, and Kevin Milner. On ends-to-ends encryption: Asynchronous group messaging with strong security guarantees. In *ACM CCS*, pages 1802–1819, 2018.

[29] Cryspen. Post-Quantum OpenMLS. Cryspen engineering blog, 2024. Experimental X-Wing hybrid cipher suite; code point unregistered. URL: https://cryspen.com/post/pq-openmls/.

[30] Cryspen. libcrux: The formally verified crypto library for Rust. Software repository, 2025. URL: https://github.com/cryspen/libcrux.

[31] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In *CRYPTO*, pages 307–315, 1989.

[32] Julien Devigne, Céline Duguey, and Pierre-Alain Fouque. MLS group messaging: How zero-knowledge can secure updates. In *ESORICS*, pages 587–607, 2021.

[33] Jelle Don, Serge Fehr, Christian Majenz, and Christian Schaffner. Security of the Fiat–Shamir transformation in the quantum random-oracle model. In *CRYPTO*, pages 356–383, 2019.

[34] Jelle Don, Serge Fehr, Christian Majenz, and Christian Schaffner. Efficient NIZKs and signatures from commit-and-open protocols in the QROM. In *CRYPTO*, pages 729–757, 2022. IACR ePrint 2022/270.

[35] Jelle Don, Serge Fehr, Christian Majenz, and Christian Schaffner. Online-extractability in the quantum random-oracle model. In *EUROCRYPT*, pages 677–706, 2022. IACR ePrint 2021/280.

[36] European Commission. Recording of Telephone Conversations or Electronic Communications. Art. 76, Commission Delegated Regulation (EU) 2017/565 (MiFID II), 2017. URL: https://eur-lex.europa.eu/eli/reg_del/2017/565/oj/eng.

[37] Valerie Fetzer, Michael Klooß, Jörn Müller-Quade, Markus Raiber, and Andy Rupp. Universally composable auditable surveillance. In *ASIACRYPT*, pages 453–487, 2023.

[38] Financial Industry Regulatory Authority. General Requirements. FINRA Rule 4511 (Books and Records Requirements), 2023. URL: https://www.finra.org/rules-guidance/rulebooks/finra-rules/4511.

[39] Jonathan Frankle, Sunoo Park, Daniel Shaar, Shafi Goldwasser, and Daniel J. Weitzner. Practical accountability of secret processes. In *USENIX Security*, pages 657–674, 2018. The "AUDIT" system; IACR ePrint 2018/697.

[40] Lorenzo Grassi, Dmitry Khovratovich, and Markus Schofnegger. Poseidon2: A faster version of the Poseidon hash function. In *AFRICACRYPT*, pages 177–203, 2023.

[41] Alex B. Grilo, Kathrin Hövelmanns, Andreas Hülsing, and Christian Majenz. Tight adaptive reprogramming in the QROM. In *ASIACRYPT*, pages 637–667, 2021.

[42] Jens Groth. On the size of pairing-based non-interactive arguments. In *EUROCRYPT*, pages 305–326, 2016.

[43] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *STOC*, pages 212–219, 1996.

[44] Paul Grubbs, Jiahui Lu, and Thomas Ristenpart. Message franking via committing authenticated encryption. In *CRYPTO*, pages 66–97, 2017.

[45] Ulrich Haböck. Multivariate lookups based on logarithmic derivatives. IACR Cryptology ePrint Archive, Report 2022/1530, 2022. URL: https://eprint.iacr.org/2022/1530.

[46] Amir Herzberg, Stanisław Jarecki, Hugo Krawczyk, and Moti Yung. Proactive secret sharing or: How to cope with perpetual leakage. In *CRYPTO*, pages 339–352, 1995.

[47] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the Fujisaki–Okamoto transformation. In *TCC*, pages 341–371, 2017.

[48] Rawane Issa, Nicolas Alhaddad, and Mayank Varia. Hecate: Abuse reporting in secure messengers with sealed sender. In *USENIX Security*, pages 2335–2352, 2022.

[49] Karen Klein, Guillermo Pascual-Perez, Michael Walter, Chethan Kamath, Margarita Capretto, Miguel Cueto Noval, Ilia Markov, Michelle Yeo, Joël Alwen, and Krzysztof Pietrzak. Keep the dirt: Tainted TreeKEM, adaptively and actively secure continuous group key agreement. In *IEEE S&P*, pages 268–284, 2021.

[50] Markulf Kohlweiss and Ian Miers. Accountable metadata-hiding escrow: A group signature case study. *PoPETs*, 2015(2):206–221, 2015.

[51] Hugo Krawczyk and Pasi Eronen. HMAC-based Extract-and-Expand Key Derivation Function (HKDF). Request for Comments 5869, Internet Engineering Task Force (IETF), 2010. URL: https://www.rfc-editor.org/info/rfc5869.

[52] Sean Lawlor and Kevin Lewi. Deploying Key Transparency at WhatsApp. Engineering at Meta, 2023. URL: https://engineering.fb.com/2023/04/13/security/whatsapp-key-transparency/.

[53] Julia Len, Melissa Chase, Esha Ghosh, Kim Laine, and Radames Cruz Moreno. OPTIKS: An optimized key transparency system. In *USENIX Security*, 2024.

[54] Baiyu Li and Daniele Micciancio. On the security of homomorphic encryption on approximate numbers. In *EUROCRYPT*, pages 648–677, 2021.

[55] Vadim Lyubashevsky and Gregory Neven. One-shot verifiable encryption from lattices. In *EUROCRYPT*, pages 293–323, 2017.

[56] Vadim Lyubashevsky, Gregor Seiler, and Patrick Steuer. The LaZer library: Lattice-based zero knowledge and succinct proofs for quantum-safe privacy. In *ACM CCS*, pages 3125–3137, 2024.

[57] Rohan Mahy and Richard Barnes. ML-KEM and Hybrid Cipher Suites for Messaging Layer Security. Internet-Draft draft-mahy-mls-pq-00, Internet Engineering Task Force (IETF), 2025. Work in progress; cipher suite code points provisional. URL: https://datatracker.ietf.org/doc/draft-mahy-mls-pq/00/.

[58] Harjasleen Malvai, Lefteris Kokoris-Kogias, Alberto Sonnino, Esha Ghosh, Ercan Ozturk, Kevin Lewi, and Sean Lawlor. Parakeet: Practical key transparency for end-to-end encrypted messaging. In *NDSS*, 2023.

[59] Marcela S. Melara, Aaron Blankstein, Joseph Bonneau, Edward W. Felten, and Michael J. Freedman. CONIKS: Bringing key transparency to end users. In *USENIX Security*, pages 383–398, 2015.

[60] Silvio Micali. Fair public-key cryptosystems. In *CRYPTO*, pages 113–138, 1992.

[61] Armin Namavari, Barry Wang, Sanketh Menda, Ben Nassi, Nirvan Tyagi, James Grimmelmann, Amy X. Zhang, and Thomas Ristenpart. Private hierarchical governance for encrypted messaging. In *IEEE S&P*, pages 2610–2629, 2024.

[62] National Archives and Records Administration. Electronic Records Management. 36 C.F.R. Part 1236, 2023. URL: https://www.law.cornell.edu/cfr/text/36/part-1236.

[63] National Institute of Standards and Technology. The Keyed-Hash Message Authentication Code (HMAC). FIPS PUB 198-1, 2008. URL: https://doi.org/10.6028/NIST.FIPS.198-1.

[64] National Institute of Standards and Technology. Secure Hash Standard (SHS). FIPS PUB 180-4, 2015. URL: https://doi.org/10.6028/NIST.FIPS.180-4.

[65] National Institute of Standards and Technology. SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. FIPS PUB 202, 2015. URL: https://doi.org/10.6028/NIST.FIPS.202.

[66] National Institute of Standards and Technology. Module-Lattice-Based Digital Signature Standard. FIPS PUB 204, 2024. URL: https://doi.org/10.6028/NIST.FIPS.204.

[67] National Institute of Standards and Technology. Module-Lattice-Based Key-Encapsulation Mechanism Standard. FIPS PUB 203, 2024. URL: https://doi.org/10.6028/NIST.FIPS.203.

[68] Jesper Buus Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In *CRYPTO*, pages 111–126, 2002.

[69] Giuseppe Persiano, Duong Hieu Phan, and Moti Yung. Anamorphic encryption: Private communication against a dictator. In *EUROCRYPT*, pages 34–63, 2022.

[70] Polygon Zero Team. Plonky3: A toolkit for polynomial IOPs. Software repository, 2025. URL: https://github.com/Plonky3/Plonky3.

[71] Sarah Scheffler and Jonathan R. Mayer. SoK: Content moderation for end-to-end encryption. *PoPETs*, 2023(2):403–429, 2023.

[72] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[73] Succinct Labs. SP1: A performant, open-source zero-knowledge virtual machine for RISC-V. Software repository, 2025. URL: https://github.com/succinctlabs/sp1.

[74] StarkWare Team. ethSTARK documentation. IACR Cryptology ePrint Archive, Report 2021/582, 2021. Conjectured FRI soundness in the list-decoding regime, i.e. proximity gaps up to capacity $1-\rho$ rather than the proven Johnson radius $1-\sqrt{\rho}$. URL: https://eprint.iacr.org/2021/582.

[75] Nirvan Tyagi, Paul Grubbs, Julia Len, Ian Miers, and Thomas Ristenpart. Asymmetric message franking: Content moderation for metadata-private end-to-end encryption. In *CRYPTO*, pages 222–250, 2019.

[76] Dominique Unruh. Universally composable quantum multi-party computation. In *EUROCRYPT*, pages 486–505, 2010.

[77] U.S. Congress. Records Management by Agency Heads: General Duties. 44 U.S.C. § 3101 (Federal Records Act), 2014. URL: https://uscode.house.gov/view.xhtml?req=granuleid:USC-prelim-title44-section3101.

[78] U.S. Department of Health and Human Services. Technical Safeguards: Audit Controls. 45 C.F.R. § 164.312(b) (HIPAA Security Rule), 2013. URL: https://www.law.cornell.edu/cfr/text/45/164.312.

[79] U.S. Securities and Exchange Commission. Records to Be Preserved by Certain Exchange Members, Brokers and Dealers. 17 C.F.R. § 240.17a-4 (Securities Exchange Act Rule 17a-4), 2023. URL: https://www.law.cornell.edu/cfr/text/17/240.17a-4.

[80] Théophile Wallez, Jonathan Protzenko, Benjamin Beurdouche, and Karthikeyan Bhargavan. TreeSync: Authenticated group management for messaging layer security. In *USENIX Security*, pages 1217–1233, 2023.

[81] Charles V. Wright and Mayank Varia. Crypto crumple zones: Enabling limited access without mass surveillance. In *IEEE EuroS&P*, pages 288–306, 2018.

[82] Mark Zhandry. How to record quantum queries, and applications to quantum indifferentiability. In *CRYPTO*, pages 239–268, 2019.

A Background

A.1 MLS and TreeKEM Background

This appendix records the MLS mechanisms the paper depends on beyond the summary of §3.2. All details follow RFC 9420 [7] and RFC 9750 [13].

Table 5: Symbolic notation used in the paper.

Symbol	Meaning
Parameters & identifiers	
λ	Security parameter (384 bits)
N_h	RFC 9420 derivation width (512 bits at SHA-512)
G	Group identifier
e	Epoch number
n_e	Group size at epoch e
$h_e = \lceil \log_2 n_e \rceil$	Tree depth (cost statements only)
$m_e = \lvert \mathsf{FDP}(\ell, e) \rvert$	Filtered-direct-path length; indexes every path object, $m_e \le h_e$
τ_e	Confirmed transcript hash
Commit context, path secrets & commitments	
ctx_e	Commit context: tree $\|$ props $\|$ τ_{e-1}
$\mathsf{ps}_i : \{0,1\}^{N_h}$, $0 \le i \le m_e$	Path secret at node i (MLS background only; no clause of $\mathcal{R}_{\mathsf{A}}$ mentions it)
c_{root}	Poseidon2 commitment $H_{\mathsf{P2}}(\text{"crot"} \| G \| e \| es_e)$, published by Π_{A}
ns_i	Node secret $\mathsf{DeriveSecret}(\mathsf{ps}_i, \text{"node"})$
ρ_i	Size of the copath resolution of level i (recipients of ps_i)
$\mathsf{ct}_{i,k}$, $i \in [m_e]$, $k \in [\rho_i]$	Path ciphertext at level i, copath-resolution index k
k_e	Epoch key at epoch e
Escrowed secrets, coins & escrow ciphertext	
$es_e : \{0,1\}^{N_h}$	Epoch secret (key-schedule root); the escrowed secret
$\eta : \{0,1\}^{\lambda}$	Coin for escrow ciphertext randomness
f_d, $d \in [t-1]$	Shamir coefficient $F_\eta(\text{"coef"} \| d)$; $f(0)$ is the secret
$\sigma_j = f(j)$, $j \in [N]$	Shamir share of the escrowed secret for A_j
$C_{\mathsf{esc}e}$	Escrow ciphertext for epoch e

Symbol	Meaning
Primitives & algorithms	
$H_{\mathsf{SHA}} : \{0,1\}^* \to \{0,1\}^{512}$	SHA-512 wire hash
$H_{\mathsf{P2}} : \mathbb{F}_q^* \to \mathbb{F}_q$	Poseidon2 (in-circuit)
$F_k(\cdot) : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$	PRF $\mathsf{SHAKE256}(k \| \cdot)$; keyed at λ or N_h
ℓ_{pub}	Published lanes of a truncated Poseidon2 digest
ℓ	Leaf index in the ratchet tree
$\Psi_k(x)$	Poseidon2 keyed stream (escrow noise; not F)
$\mathsf{CBD}_2(\cdot)$	Centered binomial sampler of parameter 2, on a $\Psi()$ stream
$\mathsf{DeriveSecret} : \{0,1\}^* \to \{0,1\}^{512}$	DeriveSecret (HKDF-Exp-Label)
$\mathsf{Encap}, \mathsf{Decap}$	ML-KEM-768 encap/decap
KeyGen	Deterministic ML-KEM-768 key generation from a node secret
$\mathsf{KEnc}, \mathsf{KDec}$	ML-KEM-768 + AEAD enc/dec
$\mathsf{Share}_{t,N}, \mathsf{Rec}_t$	Shamir share/reconstruct over $\mathbb{F}_q$
$(\mathsf{Prove}, \mathsf{Vrfy}, \mathsf{Ext})$	STARK prover, verifier, extractor
Auditor committee	
$\mathbb{A} = \{A_j\}_{j=1}^N, t$	Auditor committee and threshold
$\mathsf{mpk} = (\{pk_j^{\mathbb{A}}\}, t)$	Auditor committee public configuration
ask_j	Secret key share of auditor A_j
Constructions, relations, log & UC model	
Π_{A}	The auditability construction
$\mathcal{F}_{\mathsf{aud}}$	The functionality it realizes
$\mathcal{R}_{\mathsf{A}}$	STARK relation proved at commit
$\mathsf{safe}_{\mathsf{ESC}}$	Safety predicate of the realization statement
$\mathcal{L}$	Append-only public transcript log
$\mathcal{S}, \mathcal{A}, \mathcal{Z}$	UC simulator, adversary, environment
$Q_{\mathcal{G}_{\mathsf{RO}}}, \Pi_{\mathsf{p}}$	Oracle queries; punctured key families
$n_\pi, n_{\mathsf{tie}}, n_{\mathsf{tie}}^{\mathsf{u}}$	Bundle sub-proofs; published ties; unsalted ties
$\mathcal{T}, \mathcal{K}, d_{\mathsf{ext}}$	AIR trace height, constraint count, extension degree

A.1.1 Copath Resolution

A node is *blank* if it holds no key pair, for instance after a removal or when the tree is extended for a join; blank nodes are skipped during path derivation, which makes the number of recipients per level vary. The *resolution* $\mathsf{Res}(v)$ of a node v is the smallest set of non-blank nodes covering all leaves in v's subtree, defined recursively: $\mathsf{Res}(v) = \{v\}$ if v is non-blank; $\mathsf{Res}(v) = \emptyset$ if v is a blank leaf; and $\mathsf{Res}(v) = \mathsf{Res}(\mathsf{left}(v)) \cup \mathsf{Res}(\mathsf{right}(v))$ if v is blank and internal. The *filtered direct path* $\mathsf{FDP}(\ell, e)$ of leaf ℓ is its direct path with every node removed whose copath child has an empty resolution, those nodes having no recipients and being left blank by the update. It is the filtered path, of length $m_e \le h_e$, that a commit rekeys, and every index i in this paper runs over it. For $i \in [m_e]$ the committer encrypts ps_i to every key in the copath resolution of level i, that is to $\{pk_v : v \in \mathsf{Res}(\mathsf{sibling}(\mathsf{dp}_{i-1}))\}$, where dp_i is the i-th node on the filtered path and $\mathsf{dp}_0 = \ell$; the recipients are therefore the members under node i's copath child. Writing ρ_i for the size of that resolution, the number of path ciphertexts is $\sum_{i=1}^{m_e} \rho_i$, with $\rho_i = 1$ per level and $m_e = h_e$ when no nodes are blank. The leaf secret ps_0 is encrypted to nobody — the topmost node has no relevant sibling and the leaf has no copath child — so the levels carrying ciphertexts are exactly $1, \ldots, m_e$.

A.1.2 Path Secret Derivation and Key Schedule

Each node on the filtered direct path derives its key pair from the chain by $\mathsf{ns}_i = \mathsf{DeriveSecret}(\mathsf{ps}_i, \text{"node"})$ and $(pk_i, sk_i) = \mathsf{KeyGen}(\mathsf{ns}_i)$, and a recipient recomputes that key pair from the path secret it recovered and rejects the commit if the derived pk_i differs from the one the UpdatePath publishes. AuCGKA changes none of this: the committer's leaf key pair is generated independently of the chain and the chain begins at the first filtered parent, exactly as RFC 9420 specifies. The topmost path secret ps_{m_e} becomes the *commit_secret*, which

enters the key schedule:

$$\begin{aligned} \mathsf{js} &= \mathsf{HKDF.Ext}(\mathit{init\_secret}_{e-1}, \mathit{commit\_secret}), \\ \mathsf{es} &= \mathsf{HKDF.Ext}(\mathsf{js}, \mathsf{GC}_e), \\ \mathsf{app\_secret} &= \mathsf{DeriveSecret}(\mathsf{es}, \text{“}\texttt{app}\text{”}), \\ \mathsf{conf\_key} &= \mathsf{DeriveSecret}(\mathsf{es}, \text{“}\texttt{confirm}\text{”}), \\ \mathit{init\_secret}_e &= \mathsf{DeriveSecret}(\mathsf{es}, \text{“}\texttt{init}\text{”}). \end{aligned}$$

Here GC_e is the GroupContext: group id G, epoch e, tree hash, confirmed transcript hash τ_e and ciphersuite identifier, hashed into the schedule at every epoch so that the keys are bound to the public group state. The *confirmation tag* is $\mathsf{CT}_e = \mathsf{HMAC}(\mathsf{conf\_key}, \tau_e)$ [63], which every member verifies before accepting epoch e. The epoch secret es above is the object Π_{A} escrows: it is downstream of the whole update path and of any external PSK mixed in, which is why recovering it recovers the epoch without reference to any other. A *Welcome* message carries the new member's path secret under its init key and the current *init_secret* under a pre-distributed KeyPackage; in Au-CGKA it additionally carries the epoch's commitment c_{root}, so a joining member can check it against the epoch secret its Welcome yields.

HPKE ciphersuite MLS uses HPKE (RFC 9180) for all asymmetric encryption. The post-quantum-safe ciphersuite used here substitutes the Diffie–Hellman KEM (DHKEM) with ML-KEM-768 (FIPS 203), giving $\mathsf{HPKE} = (\text{ML-KEM-768}, \mathsf{HKDF\text{-}SHA512}, \mathsf{SHAKE256\text{-}DEM})$: the KEM step yields a shared key K and encapsulation ct, and K seeds the data-encapsulation mechanism (DEM), in our prototype a SHAKE256 stream-and-tag construction, so the ciphersuite rests only on lattice and hash assumptions. The same KEM+DEM protects Au-CGKA's on-wire path ciphertexts, which are the RFC's own and carry the RFC's own coins; the auditor-committee escrow ciphertexts instead use the STARK-friendly escrow PKE of §A.3.1, with a one-time PRF pad in place of the DEM.

A.1.3 Design Delta from MLS

Au-CGKA keeps the ratchet tree, the resolution rules, the key-schedule stages, the transcript and the confirmation tag, and it runs the RFC 9420 commit procedure unchanged. It is nonetheless *not* an RFC 9420-conformant extension: the per-commit payload is a field appended to `Commit`, and that is a change to the wire. We claim no code point and no interoperability with current implementations. Four of the five items below do sit on extension points the RFC already defines; the fifth is the addition, and we state why no defined extension point can carry it.

(D1) *Auditor committee: a GroupContext extension.* Every party holds $\mathsf{mpk} = (\{pk_j^{\mathbb{A}}\}, t)$, a public input to $\mathcal{R}_{\mathsf{A}}$. It is group-wide, public, and fixed at creation, which is the shape of a GroupContext extension (RFC §7.2). Carried there it is covered by the GroupInfo hash, which is the pinning $\mathcal{R}_{\mathsf{init}}$ relies on (§B.1), and a committee change is a GroupContextExtensions proposal rather than an out-of-band step. The pinning matters: a committer free to substitute mpk would escrow to a committee it controls.

(D2) *Genesis payload: a GroupInfo extension.* Group creation carries $(C_{\mathsf{esc}0}, \pi_0)$ in its GroupInfo (RFC §12.4.3.1), admitted by VerifyCreate, so epoch 0 is recoverable on the same terms as every later epoch (§B.1). How the genesis secret is *generated* is unchanged.

(D3) *Per-commit payload: one field on* `Commit`. Commits carry $(c_{\mathsf{root}}, C_{\mathsf{esc}e}, \pi)$ and Welcomes the epoch's c_{root}. This is the one addition to the wire. `Commit` has no extensions field, and neither of the two defined routes can carry the payload. A new proposal type (RFC §12.1) cannot: a proposal is an *input* to the commit, covered by the transcript hash before the commit secret exists, whereas the payload is a function of es_e, which the commit produces. A GroupInfo extension cannot either: GroupInfo is built after es_e exists, but it reaches only Welcome recipients and external joiners, not the members processing an ordinary commit, so no member could check the escrow at admission — which is the guarantee Π_{A} makes. The `authenticated_data` field of `FramedContent` fails for the proposal's reason rather than the GroupInfo's: it is covered by the confirmed transcript hash, so it too is fixed before the commit secret from which es_e derives exists.

(D4) *Erasure is mandatory.* What RFC 9420 treats as hygiene the adaptive proof requires: on posting, erase η, every σ_j and K_j, and the witness. Auditor decapsulation keys are the sole exception. This is a requirement on implementations, not a change to the wire.

(D5) *Ciphersuite.* The prototype runs HPKE as $(\text{ML-KEM-768}, \mathsf{HKDF\text{-}SHA512}, \mathsf{SHAKE256\text{-}DEM})$, for which no RFC §17.1 suite is a match [29, 57]. This is a post-quantum choice of our prototype, not a requirement of the escrow: $\mathcal{R}_{\mathsf{A}}$ constrains no path ciphertext, so a deployment may run any registered ciphersuite and keep every security statement of this paper.

Unchanged: the ratchet tree and its algebra, blanking and resolution, the filtered direct path, path-secret derivation and node-key generation, the coins of every path ciphertext, the UpdatePath and Welcome wire structure, every proposal type including external PSKs, the key-schedule stages and labels, the GroupContext fields the RFC defines, the transcript hash and the confirmation tag, and the requirement that every member verify that tag before accepting an epoch.

A.2 Cryptographic and STARK Background

A.3 Cryptographic Building Blocks

Hash functions, PRF, and keyed stream We model H_{SHA} (SHA-512) and H_{P2} (Poseidon2 [40]) as *independent* random oracles, so the security of one does not rest on the other. H_{SHA} is the on-wire MLS derivation hash (the path-secret ladder $\mathsf{ps}_i = \mathsf{DeriveSecret}(\mathsf{ps}_{i-1})$ and the key schedule), and H_{P2} does all *field-native* in-circuit work: the commitment, escrow-ciphertext wrap-key derivation and FRI-internal hashing. $\mathcal{R}_{\mathsf{A}}$ evaluates no H_{SHA} in circuit at all, which is the point of committing to the escrowed value (§5.1). Both are post-quantum, Grover reducing preimage security on an n-bit input to $n/2$ bits, which is 192 bits at $\lambda = 384$ and 256 at the $N_h = 512$ derivation width. The PRF is $F_k(x) = \mathsf{SHAKE256}(k\|x)$, modeled as a random oracle keyed by the first λ bits and used under the standard PRF definition. Separately, $\Psi_k(x)$ is the keyed Poseidon2 sponge stream used only for field-native escrow-PKE noise and is not F; its output lanes are rejection-sampled by discarding the unique non-canonical BabyBear value $p-1$, and their low 24 bits are consumed in order.

Commitments The epoch secret es_e is committed as:

$$c_{\mathsf{root}} \;=\; H_{\mathsf{P2}}(\text{"crot"}\|G\|e\|es_e). \tag{3}$$

c_{root} is *computationally binding* (a collision breaks collision resistance of H_{P2}) and computationally but *not* statistically hiding, since an unbounded adversary can enumerate the committed value. The proofs therefore rely on *high-entropy hiding*, whose relevant quantity is min-entropy: the epoch secret carries the full derivation width $N_h = 512$, so a QPT adversary recovers it from c_{root} with probability at most $Q_{\mathcal{G}_{\mathsf{RO}}} 2^{-512}$ classically and 2^{-256} under Grover. Hiding also requires the published digest to be truncated, a full-width digest of a bijective permutation being invertible whatever the entropy of its input (Appendix D.2.2). The domain separator $\text{"crot"}\|G\|e$ makes each published root a single-target problem rather than one function shared across every epoch of every group. Hiding against the committer is irrelevant, since the extractor recovers a malicious committer's witness regardless.

Path encryption: ML-KEM-768 (FIPS 203) Path secrets travel on the MLS wire under the standard KEM-DEM (HPKE, RFC 9180) composition *over* ML-KEM [67], written $\mathsf{KEnc}/\mathsf{KDec}$: $\mathsf{Encap}(pk) \to (c_{\mathsf{kem}}, K)$ agrees on a fresh key K and an AEAD under K carries the payload, yielding $\mathsf{ct} = (c_{\mathsf{kem}}, c_{\mathsf{dem}})$. $\mathsf{KEnc}/\mathsf{KDec}$ are *not* FIPS-203 primitives but this composition. The path wire is untouched by Π_{A}: no clause of $\mathcal{R}_{\mathsf{A}}$ mentions it, and its confidentiality is charged to CGKA path secrecy (§D.2.2); the escrow ciphertext is a separate scheme with its own decryption-failure bound (Appendix E.1).

A.3.1 Escrow PKE: a STARK-friendly Module-LWE Scheme

The escrow ciphertext wraps each Shamir share under an auditor's key, so the in-circuit cost of proving it well-formed is a primary design constraint. Rather than run ML-KEM's SHAKE-heavy sampler inside the STARK, we escrow under a *STARK-friendly* CPA public-key encryption scheme instantiating Kyber's K-PKE core [16] under the *same* Module-LWE assumption; it is not ML-KEM and makes no FIPS 203 claim. Over $R_{q_R} = \mathbb{Z}_{q_R}[X]/(X^{256}+1)$ with module rank κ, a key is $sk = \mathbf{s}$ (short) and $pk^{\mathbb{A}} = (\mathbf{A}, \mathbf{b}{=}\mathbf{A}\mathbf{s}{+}\mathbf{e})$ with $\mathbf{A} \in R_{q_R}^{\kappa\times\kappa}$; encryption of μ with coins $(\mathbf{r}, \mathbf{e}_1, e_2)$ is $\mathbf{u} = \mathbf{A}^\top\mathbf{r} + \mathbf{e}_1$, $v = \mathbf{b}^\top\mathbf{r} + e_2 + \mathsf{Encode}(\mu)$, and $\mathsf{Dec}(\mathbf{s}, (\mathbf{u}, v)) = \mathsf{Decode}(v - \mathbf{s}^\top\mathbf{u})$. Three changes make it inexpensive to prove while keeping it post-quantum. $\mathbf{A}$ is a *public input*, recomputed by the verifier, so no matrix expansion (SHAKE/XOF) appears in-circuit. The noise $(\mathbf{r}, \mathbf{e}_1, e_2)$ is $\mathsf{CBD}_2(\Psi_\eta(\text{"esc"}\|j\|G\|e))$, consumed in order and *range-checked* in-circuit, the stream's rejection step making these coins exactly the reference CBD_2 distribution. And we drop the Fujisaki–Okamoto transform: the escrow ciphertext needs no decryption oracle, auditors decapsulating only at audit, so IND-CPA suffices, and integrity of the *logged* escrow ciphertext follows from the STARK proof of well-formedness (verifiable encryption) rather than from FO (Lemma 15; the scheme itself is not claimed non-malleable). The absent decryption oracle is an obligation on the deployment, not a property of the scheme: an auditor must decapsulate only ciphertexts admitted to $\mathcal{F}_{\mathsf{Log}}$, since one that decrypted an attacker-supplied ciphertext would supply the oracle the argument assumes away. Audit reads from the log for this reason (§B.2.1).

We use the scheme in KEM mode: $\mathsf{Encap}(pk_j^{\mathbb{A}})$ encrypts a random $\mu_j \xleftarrow{\$} \{0,1\}^{256}$ and sets $K_j = H_{\mathsf{P2}}(\text{"key"}\|\mu_j)$, and the share is wrapped by a one-time pad $\mathsf{e}_j = \sigma_j \oplus H_{\mathsf{P2}}(\text{"wrap"}\|K_j)$, which the simulator equivocates through the random oracle (§D.2.7). The width of μ_j is 256 bits, both the message capacity of one such ciphertext and the min-entropy the pad point needs: that point is *reprogrammed* rather than punctured, so it is governed by the $\sqrt{Q_{\mathcal{G}_{\mathsf{RO}}}}$-bounded term of Appendix D.2.2 and not by the $Q_{\mathcal{G}_{\mathsf{RO}}}$-linear one that fixes λ. The parameter set, ciphertext compression, the decryption-failure bound, the IND-CPA reduction and the verifiable-encryption statement are in Appendix E.

A.3.2 Shamir Secret Sharing over $\mathbb{F}_q$

We use Shamir's threshold scheme [72] over a prime field $\mathbb{F}_q$ with $q > 2^{N_h}$, so an escrowed secret embeds injectively as an integer in $\{0,\dots,2^{N_h}-1\} \subset \mathbb{F}_q$. $\mathsf{Share}_{t,N}$ samples one degree-$(t-1)$ polynomial $f(X) = s + \sum_{d=1}^{t-1} f_d X^d$ over $\mathbb{F}_q$ with $f_d = F_\eta(\text{"coef"}\|d)$ and outputs $\sigma_j = f(j)$ for $j \in [N]$; Rec_t Lagrange-interpolates $f(0)$ from any t shares. *One* sharing carries *one* secret: this is the object clause (i) constrains and

what the security analysis assumes throughout.

Arithmetizing one sharing over a 256-bit field The prototype instantiates $\mathbb{F}_q$ at $q_{\mathsf{S}} = \text{nextprime}(2^{256})$ (§7), in which an N_h-bit secret does not fit, and therefore carries the sharing as a *pair* of $\mathbb{F}_q$ elements: it splits the secret into two 32-byte halves, shares each under its own coefficient family $f_{b,d} = F_{\eta}(\text{"coef"}\|b\|d)$ and its own pad $H_{\mathsf{P2}}(\text{"wrap"}\|b\|K_j)$ for $b \in \{0,1\}$, and reconstructs by interpolating both halves at zero and concatenating. The half index must be inside both derivations: without it in the coefficients the two polynomials coincide and one share reveals $s_0 - s_1$, and without it in the pad the two wraps satisfy $\mathsf{e}_{0,j} \oplus \mathsf{e}_{1,j} = \sigma_{0,j} \oplus \sigma_{1,j}$, a linear relation between shares available to any holder of the escrow ciphertext. Instantiating $\mathbb{F}_q$ above 2^{512} drops the split. Because the relation, the functionality and the proofs are stated over the *single* sharing above, the substitution needs an argument rather than a remark:

Lemma 3 (The split-field sharing realizes the specified one)**.** *Fix $t \le N$ and let $s \in \{0,1\}^{N_h}$ with halves $s_0, s_1 \in \{0,1\}^{N_h/2}$. Define $\mathsf{Share}'_{t,N}(s;\eta) = \big((\sigma_{0,j},\sigma_{1,j})\big)_{j\in[N]}$ with $\sigma_{b,j} = f_b(j)$ for the two degree-$(t-1)$ polynomials f_b over $\mathbb{F}_q$ with $f_b(0) = s_b$ and coefficients $f_{b,d}$, and Rec'_t the pairwise interpolation at zero followed by concatenation. Then $(\mathsf{Share}', \mathsf{Rec}')$ is a (t,N)-threshold scheme for s:* (a) *Rec'_t returns s from any t index-distinct pairs;* (b) *for coefficients uniform and independent in $\mathbb{F}_q$, any $t-1$ pairs are distributed independently of s; and* (c) *its share vector is a deterministic relabelling of one $\mathsf{Share}_{t,N}$ share vector for the same t, N and secret, so every statement of this paper that quantifies over "the sharing" holds verbatim for it.*

Proof. Each half is an ordinary Shamir sharing over $\mathbb{F}_q$, so t index-distinct pairs give t evaluations of each f_b and concatenation inverts the split; perfect t-privacy of each half holds for uniform coefficients, and the two families are domain-separated by b, hence jointly uniform; and both objects are the evaluation vector at $[N]$ of an $\mathbb{F}_q$-linear map of $(s, \text{coefficients})$ with the same threshold structure, the relabelling being $j \mapsto (j,b)$. □

The hypothesis of (b) requires care: the coefficients come from one λ-bit coin, so their joint entropy is at most λ where $2(t-1)$ uniform ones carry $2(t-1)\log_2 q$ bits. Real-world privacy is therefore computational, in the random-oracle model for F; perfect t-privacy is available only after the coefficients are idealized at $H_3 \to H_4$, the only place we use it (Lemma 8). Embedding an N_h-bit secret in the prime costs the negligible statistical distance $(q - 2^{N_h})/q$. The committer proves the sharing well-formed in-circuit, so a verifier is assured the escrow ciphertext escrows the committed secret. The STARK field is BabyBear ($p = 2^{31} - 2^{27} + 1$), which cannot hold an $\mathbb{F}_q$ element natively, so clause (i) represents each element in base-2^{16} limbs and proves one signed carry chain per share; the committee index j being a small public scalar, each $f_d \cdot j^d$ is a per-limb scalar multiply rather than a big-by-big one.

A.3.3 STARKs and the (Prove, Vrfy, Ext) Triple

We use a STARK (Scalable Transparent ARgument of Knowledge) [11] instantiated as a Plonky3-style AIR (Algebraic Intermediate Representation) over a small field. It is a non-interactive argument in the random oracle model with five properties: completeness; *knowledge soundness* via a straight-line extractor Ext that, from any accepted (x,π^*), outputs w with $(x,w) \in \mathcal{R}$ except with negligible probability; zero knowledge, which we model by the $\mathcal{F}_{\mathsf{NIZK}}$ hybrid (§D.1); post-quantum security, its underlying IOP being information-theoretically sound and its Fiat–Shamir compilation knowledge-sound with a straight-line extractor in the QROM [27, 33]; and succinctness, at proof size $O(\lambda \cdot \text{polylog}(\mathcal{T}))$ in the trace height $\mathcal{T}$ and independent of the witness size. The proof assumes all five hold *simultaneously* of the system we instantiate, which is assumption (A3′) of §6 and is not established here; §7 shows that the relation *can* be arithmetized efficiently (in-circuit ML-KEM and Poseidon2), which is a statement about cost rather than about (A3′).

Global random oracle model Following Canetti, Jain, and Scafuro [23], we model H_{SHA}, H_{P2} *and the keyed function* F as a single joint global random oracle $\mathcal{G}_{\mathsf{RO}}$ accessible by all parties, the simulator and the environment; a keyed evaluation $F_k(x)$ is the query $\mathcal{G}_{\mathsf{RO}}(k\|x)$ under domain separation. Placing F inside $\mathcal{G}_{\mathsf{RO}}$ lets the proof *program* the wrap, sharing and commitment derivations, and means no step invokes a PRF assumption. We do *not* characterize a programmable point as one the environment "has not queried": against superposition access that event is undefined. The two admissible mechanisms, reprogramming at a point of min-entropy ζ and puncturing with semi-classical O2H, are stated with their costs in §D.2.4, and every programmed and punctured point is tracked in the hybrid sequence.

B Constructions and Protocol Specifications

This appendix gives the epoch-0 bootstrap and the per-algorithm steps not stated in §5.

B.1 Epoch-0 Bootstrap

Epoch 0 is created rather than committed, so its escrow is fixed at group creation. Auditability is a per-epoch property and epoch 0 is an epoch, so it needs the same escrow, commitment and admission check as every later epoch; without them the first epoch of every group would be the one epoch no audit can reach. We therefore state group creation as an

algorithm of its own, Π_{init}. It keeps the RFC 9420 creation procedure unchanged, including how the initial key-schedule input is sampled and the convention that an epoch with no predecessor runs the key schedule with $\mathit{commit\_secret}_0 = \mathbf{0}$. Nothing about how the genesis secret is *generated* changes: Π_{A} escrows the epoch secret the RFC procedure already produces, so no derivation is redirected through an escrowed value. What creation gains is the payload and its proof.

Genesis (Π_{init}) The creator runs RFC 9420 creation, obtaining the genesis epoch secret es_0. It samples $\eta_0 \xleftarrow{\$} \{0,1\}^\lambda$, forms $C_{\mathsf{esc}0} = \mathsf{Escrow}(es_0, \eta_0, \mathsf{mpk})$ exactly as in Π_{A} step 2, publishes $c^{(0)}_{\mathsf{root}} = H_{\mathsf{P2}}(\texttt{"crot"} \| G \| 0 \| es_0)$, and proves $\mathcal{R}_{\mathsf{init}}$:

$$\mathcal{R}_{\mathsf{init}}: \boxed{\begin{array}{l}\text{(i) escrow ciphertext } C_{\mathsf{esc}0} \text{ Shamir-escrows } es_0 \text{ over} \\ \mathbb{F}_q \text{ under } \mathsf{mpk} \text{ (as } \mathcal{R}_{\mathsf{A}}\text{(i–ii))} \\ \text{(ii) } c^{(0)}_{\mathsf{root}} = H_{\mathsf{P2}}(\texttt{"crot"} \| G \| 0 \| es_0), \text{ on the same } es_0 \\ \text{that is clause (i)'s constant term, bound to the epoch-0} \\ \text{GroupInfo hash } \mathit{GI\text{-}hash}_0\end{array}} \tag{4}$$

Public inputs $(\mathsf{mpk}, C_{\mathsf{esc}0}, c^{(0)}_{\mathsf{root}}, \mathit{GI\text{-}hash}_0)$, witness $(es_0, \eta_0, \{\mu_{0,j}, K_{0,j}, \sigma_{0,j}\})$; the GroupInfo extension carries $(C_{\mathsf{esc}0}, \pi_0)$. Epoch 0 has no update path — the group is created with its full roster and no node is rekeyed by a commit — so $m_e = 0$ at the genesis node of $\mathcal{H}$. A joining member obtains the epoch-0 secret from its Welcome message and admits the genesis only if $c^{(0)}_{\mathsf{root}}$ matches the commitment to that secret, so VerifyCreate refuses a mis-escrowed genesis on the same member-enforced terms as a later commit (§5.1). Audit is Audit (§B.2.2) on $C_{\mathsf{esc}0}$: any t auditors recover es_0 and derive the epoch-0 key, touching no other epoch.

Why the anchor is a payload rather than a derivation Binding $c^{(0)}_{\mathsf{root}}$ to $\mathit{GI\text{-}hash}_0$ is what makes the committee the group intended the committee the genesis escrows to: mpk is a public input of $\mathcal{R}_{\mathsf{init}}$, so a creator that substituted a committee of its own keys would produce a proof against a mpk the GroupInfo does not carry. No party retains epoch-0 material afterwards: the genesis secret is Shamir-escrowed under the committee's keys, and the creator, which holds it transiently while building the Welcome messages, erases it on the same discipline as a later committer (§3.4). The simulator gains one group-creation case, handled as an honest commit (§D.2.7), which does not affect Theorem 1.

B.2 Full Protocol Specifications

We give the per-algorithm pseudocode of Π_{A} not already stated in §5, starting with the subroutines it shares with the epoch-0 bootstrap.

B.2.1 Common Subroutines

$\mathsf{Escrow}(s, \eta, \mathsf{mpk})$, $\mathsf{Recover}(T, C_{\mathsf{esc}}, \{sk^{\mathbb{A}}_j\})$. Escrow is step 2 of §5.1 on an arbitrary secret s in place of es_e, returning $C_{\mathsf{esc}} = \{(\mathsf{c}^{\mathsf{esc}}_j, \mathsf{e}_j)\}_{j\in[N]}$; Recover, for $|T| = t$, is the corresponding audit step, decrypting $\mu_j \leftarrow \mathsf{Dec}(sk^{\mathbb{A}}_j, \mathsf{c}^{\mathsf{esc}}_j)$, unwrapping $\sigma_j \leftarrow \mathsf{e}_j \oplus H_{\mathsf{P2}}(\texttt{"wrap"} \| H_{\mathsf{P2}}(\texttt{"key"} \| \mu_j))$ and returning $\mathsf{Rec}_t(\{(j, \sigma_j)\}_{j\in T})$ over $\mathbb{F}_q$. Neither routine reads any epoch but its own.

$\mathsf{Stmt}(\mathsf{pp}, \mathsf{mpk}, \mathcal{L}, \mathcal{M}_e)$. Recompute the STARK statement from the wire, taking *no* field from the committer: read G, e and ctx_e from the logged GroupContext; take $\mathsf{mpk} = (\{pk^{\mathbb{A}}_j\}_{j\in[N]}, t)$ from the group's public parameters in $\mathcal{L}$ rather than from $\mathcal{M}_e$; and take $C_{\mathsf{esc}e}$ and c_{root} from $\mathcal{M}_e$, which occupies the unique log position (G, e). Return the resulting x. The statement has no path-derived field, so Stmt reads neither the ratchet tree nor the UpdatePath.

$\mathsf{ValidateCommit}(\mathsf{pp}, \mathsf{mpk}, \mathcal{L}, \mathcal{M}_e)$. Set $x \leftarrow \mathsf{Stmt}(\mathsf{pp}, \mathsf{mpk}, \mathcal{L}, \mathcal{M}_e)$ and check: (a) $\mathsf{Vrfy}(x, \pi) = 1$; (b) escrow ciphertext $C_{\mathsf{esc}e}$ is present and covered by π; (c) e is the next expected epoch for G in $\mathcal{L}$. Reject on any failure.

Why c_{root} is outside τ_e c_{root} is *not* hashed into the transcript hash, and cannot be: it is a function of $es_e = \mathsf{HKDF.Ext}(\mathsf{js}, \mathsf{GC}_e)$ and GC_e contains τ_e, so a committer could compute neither value first. No binding is lost: c_{root} is a public input of x and hence absorbed into the Fiat–Shamir transcript, it is recomputed from the wire by Stmt, and $\mathcal{M}_e$ occupies the unique log position (G, e). Step (a) likewise verifies against the *recomputed* x, which costs nothing, every field being public in $\mathcal{L}$; Appendix D.3.1 states why a committer-supplied statement would make the check vacuous.

B.2.2 Per-Algorithm Steps Not Stated in §5

Commit, Escrow and Audit are given in full in §5.1, and AuditSetup in its setup paragraph. The remaining interfaces are these.

VerifyCommit. Run ValidateCommit on public inputs mpk, $C_{\mathsf{esc}e}$ and c_{root}. It is stateless: it reads the log and the message, holds no group state, and establishes that the logged escrow is a well-formed t-of-N sharing of a c_{root}-preimage. It does not establish that the preimage is the group's epoch secret; that is the member-side check below, and §D.3.3 states the boundary precisely.

$\mathsf{Process}(\mathit{gks}, \mathcal{M}_e)$. Run ValidateCommit; run the ordinary RFC 9420 commit processing — decrypt the path secret the UpdatePath carries for this member, recompute the chain above it, check each node key and the confirmation tag — to derive es_e and k_e; then check $c_{\mathsf{root}} = H_{\mathsf{P2}}(\texttt{"crot"} \| G \| e \| es_e)$ and abort if not. Only the last step is new.

Audit, key-schedule step. After Recover returns es_e, check it against the logged c_{root}, run the key schedule forward on it, and output (es_e, k_e). The step needs no other epoch's material:

es_e is the key schedule's own root at epoch e, already downstream of the previous epoch's *init_secret* and of any external PSK mixed in. The c_{root} check is the only failure condition.

B.2.3 Public and Private Operations, Variable Depth

VerifyCommit is stateless and needs only the log; §D.3.3 states what it establishes and what it leaves to the members. Audit is neither stateless nor open: reconstructing the escrowed secret requires a threshold of auditors, leaves no record on the log, and returns its output only to the requesting committee.

Group size and blanking both change each epoch, so the filtered-path length m_e varies. This costs Π_{A} nothing. $\mathcal{R}_{\mathbb{A}}$ has no clause indexed by a path level: the statement is the escrow, its sharing and one commitment, so the trace, the witness, the proof size and the prove time are the same whether the tree holds eight members or a million, and whether or not nodes are blank. The measured configuration of §8 reports the unblanked case $m_e = h_e$, $\rho_i = 1$ for the non-proof wire figures, where blanking does move bytes between levels; the proof figures are independent of both.

B.3 Proactive Refresh of the Committee Keys

The safety predicate of §6 counts auditor corruptions over the *whole* execution (Rem. 1), because audit must recover arbitrarily old epochs and the committee therefore cannot forget; an adversary that corrupts one auditor per year defeats a 3-of-5 committee in three years. We record here an add-on that narrows that window. It is a deployment measure rather than a part of Π_{A}: the base theorem is the fixed-sharing case, and no clause of $\mathcal{R}_{\mathbb{A}}$, no object on $\mathcal{F}_{\mathsf{Log}}$ and no logged proof depends on it.

Re-randomizing the *per-epoch* sharing does not narrow the window: the shares of epoch e are wrapped inside $C_{\mathsf{esc}e}$ on $\mathcal{F}_{\mathsf{Log}}$ under the auditors' long-term keys, so refreshing it would mean decrypting and re-encrypting every archived epoch — $O(E)$ work per period, the committee online over the whole archive, and a window in which plaintext shares exist — and would invalidate the logged proofs, $C_{\mathsf{esc}e}$ being a public input of x. We refresh one level down instead. Each auditor's escrow decapsulation key $sk_j^{\mathbb{A}}$ is itself held as a k_{c}-of-n_{c} sharing among n_{c} custodians of auditor j, and *that* sharing is proactively re-randomized [46] every period: at period θ the custodians hold $\mathsf{sh}_{j,1}^{(\theta)}, \ldots, \mathsf{sh}_{j,n_{\mathsf{c}}}^{(\theta)}$ with $\mathsf{Rec}_{k_{\mathsf{c}}}$ of any k_{c} equal to $sk_j^{\mathbb{A}}$, the period-θ shares being information-theoretically independent of the period-θ' shares for $\theta \neq \theta'$ given fewer than k_{c} of each. Nothing on $\mathcal{F}_{\mathsf{Log}}$ changes: $pk_j^{\mathbb{A}}$ is fixed forever, so every archived $C_{\mathsf{esc}e}$ stays decryptable and every logged proof stays valid. An audit becomes a two-level reconstruction, k_{c} custodians of each of t auditors reassembling $sk_j^{\mathbb{A}}$ before the committee interpolates as before. Because $pk_j^{\mathbb{A}}$ cannot rotate, the refresh bounds the *probability* that an adversary acquires $sk_j^{\mathbb{A}}$ and not the *consequence*: an $sk_j^{\mathbb{A}}$ obtained in any one period opens that auditor's slot in every archived epoch. That is the residual cost of Rem. 1, and no scheduling of the refresh removes it.

Proposition 4 (Refined safety predicate)**.** *Under periodic refresh, epoch e is private if for fewer than t indices $j \in [N]$ the adversary ever obtains $sk_j^{\mathbb{A}}$, and it obtains $sk_j^{\mathbb{A}}$ only by corrupting $\geq k_{\mathsf{c}}$ custodians of auditor j* within a single refresh period. *Writing $\mathsf{safe}'_{\mathsf{ESC}}$ for that predicate, Theorem 1 holds with respect to $\mathsf{safe}'_{\mathsf{ESC}}$ in place of $\mathsf{safe}_{\mathsf{ESC}}$, with one additional term $EN\varepsilon_{\mathsf{prs}}$ for the refresh protocol's own secrecy error.*

Proof sketch. The simulator is unchanged, still revealing $sk_j^{\mathbb{A}}$ on a corruption of A_j; only which corruption events yield $sk_j^{\mathbb{A}}$ differs. On a custodian corruption below the threshold $\mathcal{S}$ hands the adversary uniform shares of the period, distributed exactly as real by k_{c}-privacy of the sub-sharing, and programs nothing; the wrap-point equivocation of Lemma 7 is reached only once $sk_j^{\mathbb{A}}$ itself is exposed. Cross-period share consistency is the refresh protocol's own guarantee, charged $\varepsilon_{\mathsf{prs}}$, which we do not instantiate here. □

The refresh runs among auditor j's custodians only, needing no interaction with the group, the delivery service or the other auditors, so it can run while the committee is otherwise offline. It replaces "fewer than t auditors ever corrupted" with "fewer than k_{c} custodians of any auditor corrupted within one period, for all but $t-1$ auditors", which is the weaker condition only if the period is short relative to the time to corrupt k_{c} custodians. The period is a deployment parameter.

Cost The refresh is a standalone crate re-sharing $sk_j^{\mathbb{A}}$ under a post-quantum Feldman VSS with Ajtai/SIS-style linear commitments over $\mathbb{F}_q$. It is a statement-layer add-on: custodian state and committee membership change never enter $\mathcal{R}_{\mathbb{A}}$, so $pk_j^{\mathbb{A}}$ and the logged $C_{\mathsf{esc}e}$ are fixed for an epoch and no logged proof is affected. Table 6 reports wall-clock means at N=5, t=3, n_{c}=5, k_{c}=3; these are add-on operations, not STARK proving. A full Herzberg refresh round (`refresh_panel`) costs 5.3 ms and a leave+join archiving the leaver key among the new committee (`replace_rotate_leave_join`) costs 2.7 ms; every other operation — VSS dealing and verification, delta verification, two-level audit — is under one millisecond. All of it runs off the group's critical path.

C Implementation Details

Compute chips and cross-clause binding Every chip of §7 is cross-checked against a plain-Rust reference and rejects a tampered input. The dominant cost is the escrow ciphertext's lattice arithmetic, 62,080 modular operations at N=5, proven as one fused multi-stage STARK with the field terms additive. The binder of the cross-clause tie (§7) is the Shamir

Table 6: Auditor re-sharing microbenchmarks (statement-layer add-on, not STARK proving). PQ Feldman VSS with Ajtai/SIS linear commitments over $\mathbb{F}_q$; wall-clock means, release build. Apple M5 Pro; committee N=5, t=3; custodians n_c=5, k_c=3. The full refresh panel and leave+join are the heaviest operations.

Operation	Mean
PQ Feldman VSS	
`vss::deal`	287 µs
`vss::verify_share`	57 µs
Custodian sharing and proactive refresh	
`share_seed`	1.0 ms
`verify_custodian_state`	98 µs
`propose_refresh`	489 µs
`verify_delta`	100 µs
`apply_refresh`	501 µs
`refresh_panel` (full round)	5.3 ms
`reconstruct_seed`	165 µs
`redistribute`	863 µs
Committee membership and archive	
`setup_committee`	4.8 ms
`replace_rotate_leave_join`	2.7 ms
`verify_archive_share`	95 µs
`reconstruct_from_archive`	152 µs
Two-level audit	
`open_historical_askc`	463 µs
`audit_with_askc`	79 µs

band's own packing of es_e, exported as one ℓ_{pub}-lane digest and consumed by the commitment and wrap chips, with no public value exposing es_e, η or the plaintext shares. A commitment reading a different secret and a Shamir/commitment leaf mismatch are each rejected by test. The measured harness instantiates one recipient per copath level (ρ_i=1) for the wire figures; a resolution of size ρ multiplies the MLS base's ciphertext and AEAD work by ρ and leaves every proof figure and every security figure unchanged, the relation having no path-indexed clause. The stack also implements a SHA-512 chip, validated against the official test vectors and used by the LogUp engine's regression suite — LogUp cuts one compression from ∼100k plain constraints to ∼8k lookup checks — but $\mathcal{R}_{\mathsf{A}}$ evaluates no SHA-512 in circuit, so it contributes nothing to the measured bundle. Table 7 reports each ML-KEM chip proven on the engine at full trace utilization: the $[0, q_R)$ modulus table fixes the height at $\mathcal{T} = 4096$, so the chips share a trace size and differ in width, in how many columns are range-checked and in the number of computation constraints, with proof size tracking the column count.

Where the proof size is Tables 2 and 7 report *single* arguments, and the measured bundle is not their sum: stacking concatenates the row-independent chips into 2 wide traces, so each sub-proof commits to many times the columns of any single row of Table 7. Column count is what sets the size. At a fixed $nq = 156$ each query opens every committed column at its index and carries a Merkle path per commitment, so

Table 7: ML-KEM chip costs under the hiding PCS. columns = main-trace columns, lookups = batched range lookups, constraints = computation constraints; proof size in KB, prove/verify in ms.

Chip	cols	lookups	constraints	Proof	Prove	Verify
mod-mul ab mod q	8	2	1	587.9	278.9	2.9
NTT butterfly	21	6	3	628.3	353.8	3.4
inner product $\sum a_i s_i$	8	2	2	587.9	242.5	2.8
chained FMA	15	4	2	608.3	292.0	3.1

serialized bytes scale with columns × queries and only logarithmically with height — the same effect that makes Table 2 near-flat in the trace. The N escrow encryptions supply those columns, their 62,080 modular operations being the widest band, and the hiding PCS adds the leaf salt and the masking columns on top of each. Size is therefore $O(N)$ in the committee and $O(1)$ in the group, and halving nq — which is what the ethSTARK conjecture buys — roughly halves it.

Configuration behind the measured bundle The measured bundle uses stacked aggregation with parallel chip scheduling and FRI log-blowup 2; every chip of $\mathcal{R}_{\mathsf{A}}$ is degree ≤3, so no part of the bundle needs the higher blowup the SHA-512 chain would. The non-hiding assembler, which exposes the shares publicly for offline auditing, is not the measured configuration. Timings are taken under the tip Rayon pool at the core count (18 threads), FRI-merge off, and shared-DFT prewarm defaults; bundle verification checks sub-proofs on scoped OS threads and runs the FRI query phase in parallel once the indices are drawn serially, so neither the transcript nor the proof bytes depend on the thread count.

End-to-end zero knowledge Under the hiding PCS the commit message and π expose only the intended publics: c_{root}, the context, the committee public keys and the wrapped escrow ciphertexts. No witness secret appears — not es_e, η, the escrow messages, the wrap keys, the pads or the shares. Values crossing between sub-proofs do so as a Poseidon2 digest truncated to ℓ_{pub} lanes, under a secret salt of twelve further input lanes; the salt is derived from η, so sub-proofs sharing a value publish an identical digest, and η is carried at the full λ bits because it determines the coefficients, the noise and thence every wrap. *Every* tie in the bundle is salted this way — both chips that share a value hold η — so $n^{\mathsf{u}}_{\mathsf{tie}} = 0$ in Lemma 11 and the transcript carries no unsalted commitment to a secret. The wrap key is bound to its escrow message in-circuit: the chip that hashes μ_j derives K_j from that digest under a range-checked canonical bit decomposition and publishes only the truncated commitment, so no published value determines K_j. A harness test scans the reconstructed published statement for the secrets and for any window of lanes from which the wrap key could be re-derived, and confirms that the wrap opens with the witness key but not from the

transcript.

The assumptions these mechanisms rest on — collision resistance of the truncation under idealized Brassard–Høyer–Tapp, and the ethSTARK conjecture [74] behind the FRI figure — are stated with their levels in Appendix D.2.2 and charged over an execution in Appendix D.2.11. Pad expansion is not among them: the field-native pad lets clause (ii) prove $\mu_j \to K_j \to \mathrm{pad} \to \mathsf{e}_j$, so an auditor recovering K_j can always unwrap σ_j.

D Security Analysis

This appendix is the complete security treatment of AuCGKA, post-quantum throughout: the threat model and UC framework (§D.1); the assumptions and the adaptive post-quantum UC theorem, with its simulator and hybrid sequence (§D.2); and the game-based properties derived as corollaries, with their games and reductions (§D.3).

Adversary powers and goals $\mathcal{A}$ has the powers granted by the oracle set of Appendix D.3.1: it drives the group through $O_{\mathsf{create}}, O_{\mathsf{prop}}, O_{\mathsf{commit}}$ and may, as a corrupt committer, submit an arbitrary commit message; O_{corrupt} reveals a party's current non-erased state, including an auditor's $\mathsf{ask}_j = sk_j^{\mathbb{A}}$; and O_{dec} on path ciphertexts makes the model chosen-ciphertext, excluded by freshness on the challenge epoch's own logged ciphertexts rather than on positions. The network is the append-only log $\mathcal{F}_{\mathsf{Log}}$, which admits at most one node per position (G,e), so equivocation is refused at the log layer, and $\mathcal{F}_{\mathsf{PKI}}$ supplies authentication. Against this model the adversary pursues one of two goals, formalized as the games of Appendix D.3.1: **(a)** learn a fresh epoch's key without a threshold of auditors ($\mathsf{Game}_{\mathsf{priv}}$), or **(b)** produce a commit message that verifies and is accepted by an honest member yet whose escrow is unrecoverable ($\mathsf{Game}_{\mathsf{es}}$, the silent-escrow attack). Both goals are about the escrow, which is what $\mathcal{R}_{\mathsf{A}}$ constrains.

D.1 UC Framework and Hybrid Model

We use the UC framework of Canetti [21]: protocol Π *UC-realizes* ideal functionality $\mathcal{F}$ if for every QPT adversary $\mathcal{A}$ there is a QPT simulator $\mathcal{S}$ such that no QPT environment $\mathcal{Z}$ distinguishes $\mathsf{REAL}_{\Pi,\mathcal{A}}$ from $\mathsf{IDEAL}_{\mathcal{F},\mathcal{S}}$ with more than negligible advantage. We work in the $(\mathcal{F}_{\mathsf{NIZK}}, \mathcal{F}_{\mathsf{Log}}, \mathcal{G}_{\mathsf{RO}}, \mathcal{F}_{\mathsf{PKI}})$-hybrid model. $\mathcal{F}_{\mathsf{NIZK}}$ is the ideal NIZK, providing prove/verify with *simulation* from public inputs and *straight-line extraction* from adversarial proofs, instantiated by the STARK in the QROM; the extractor being online, it stays sound against a quantum prover. $\mathcal{F}_{\mathsf{Log}}$ is an append-only bulletin board admitting at most one entry per position (G,e) with publicly readable history, modeling the delivery service's ordering guarantee. $\mathcal{G}_{\mathsf{RO}}$ is a programmable global quantum random oracle [14,23] queried in superposition by all parties including $\mathcal{S}$, run by the simulator as a compressed oracle [82] and reprogrammed at the derivation points the proof tracks explicitly (Appendix D.2.6). The MLS key schedule is modeled inside $\mathcal{G}_{\mathsf{RO}}$, which is what lets $\mathcal{S}$ *program* the relation between the escrowed secret and the epoch key at the points where an exposure reveals both. The functionality itself samples the two independently and derives neither (Fig. 1); every hybrid of Appendix D.2 refers to that one functionality (Appendix D.2.5). $\mathcal{F}_{\mathsf{PKI}}$ is an ideal PKI mapping identities to public keys, whose realization — credential issuance and revocation, signature security — is orthogonal to auditable key agreement.

Corruption model. Corruptions are *adaptive*, in the secure-erasure model (§3.4): $\mathcal{A}$ may corrupt any party at any point and learns only that party's current, non-erased state, and any subset of group members may be corrupt, exactly as in [5]. Auditor corruption is adaptive too, but the threshold budget is counted over the *whole* execution, so adaptivity changes only when and which auditors $\mathcal{A}$ selects, not how many (Remark 1).

Remark 1 (Auditor-committee forward secrecy is precluded by audit). The global-in-time committee budget is a structural consequence of auditability. Audit must recover the secret of an *arbitrarily old* epoch on demand, so the committee must retain, for every past epoch, the means to reconstruct it: no committee-side forward secrecy is achievable, and a $\geq t$ breach is necessarily *retroactive* across all epochs whose escrow ciphertexts remain on $\mathcal{F}_{\mathsf{Log}}$. Post-compromise security is void against such a coalition for the same reason, and neither property is recovered by later rekeying, the threshold structure limiting how much of the committee an adversary must reach and not what it obtains once it has. Accordingly Fresh (Appendix D.3.1) counts auditor corruptions over the whole execution rather than in a per-epoch window, which also covers an adversary accumulating them gradually without ever holding t shares at once. Appendix B.3 gives a mechanism that narrows the window and the refined predicate it delivers; the base theorem assumes a fixed sharing.

D.2 Adaptive Post-Quantum UC Security

We prove that Π_{A} is *adaptively* UC-secure against a QPT adversary in the secure-erasure model (§3.4) and the programmable global quantum random-oracle model (QROM) [14]: it realizes $\mathcal{F}_{\mathsf{aud}}$.

D.2.1 Model

We keep the UC framework, the $(\mathcal{F}_{\mathsf{NIZK}}, \mathcal{F}_{\mathsf{Log}}, \mathcal{G}_{\mathsf{RO}}, \mathcal{F}_{\mathsf{PKI}})$-hybrid model and the adaptive secure-erasure corruption model of §D.1. All machines $(\mathcal{Z}, \mathcal{A}, \mathcal{S})$ are QPT; $\mathcal{G}_{\mathsf{RO}}$ is quantum-accessible, $Q_{\mathcal{G}_{\mathsf{RO}}}$ denotes the total number of superposition queries, and $\mathcal{S}$ may lazily sample $\mathcal{G}_{\mathsf{RO}}$ on fresh inputs and adaptively reprogram it at the points of §D.2.7. In the QROM, $\mathcal{F}_{\mathsf{NIZK}}$ simulates proofs from public inputs and admits a straight-line (online) extractor [27, 34].

D.2.2 Building blocks in the QROM

Each block is post-quantum and QROM-secure; ε_X is the QPT advantage against the corresponding game. Table 8 collects the levels and §D.2.11 charges them over a whole execution.

Escrow PKE, $\varepsilon_{\mathsf{esc}}$. IND-CPA of the scheme of §A.3.1 reduces to quantum Module-LWE at the parameters of Appendix E.1 (Lemma 14), and suffices because there is no escrow-ciphertext decryption oracle and integrity of a *logged* ciphertext comes from the proof of well-formedness (Lemma 15, proof-bound rather than scheme-level).

Oracles. $F = \mathsf{SHAKE256}$, $H_{\mathsf{P2}} = \mathsf{Poseidon2}$, the keyed stream Ψ and the RFC 9420 key schedule are all modeled inside $\mathcal{G}_{\mathsf{RO}}$, hence programmable; Ψ is included because $H_2 \to H_3$ punctures the "esc" label family. Modeling KeySchedule inside $\mathcal{G}_{\mathsf{RO}}$ is a *model condition*: the functionality evaluates it nowhere, and $\mathcal{S}$ programs $\mathsf{KeySchedule}(es_e, \mathsf{ctx}_e) := k_e$ at exposure points, which §D.2.5 shows is forced. The programmed points are counted in R. The path-secret ladder DeriveSecret needs no such treatment here: no clause of $\mathcal{R}_{\mathsf{A}}$ reaches it, and the path layer is handled whole by the TreeKEM simulator of [5].

Commitment binding and hiding, $\varepsilon_{\mathsf{bind}}$. Publishing ℓ_{pub} BabyBear lanes gives $n \approx 30.9\,\ell_{\mathsf{pub}}$ bits. Collision costs are $2^{n/2}$ without large memory, $\tilde{O}(2^{2n/5})$ under QRACM [24] and $2^{n/3}$ with BHT quantum memory [18]; we quote BHT conservatively, so no result rests on a memory restriction. Hiding requires truncation, the permutation being invertible given all W lanes: an attacker guesses the $W - \ell_{\mathsf{pub}}$ withheld lanes, inverts and reads the packed secret, at Grover cost $p^{(W-\ell_{\mathsf{pub}})/2} = 15.45(W - \ell_{\mathsf{pub}})$ bits. The right guess is recognizable (es_e recomputes the confirmation tag), and the salt, recovered by the same inversion, strengthens only direct preimage search. Over the event class $T = E(1+n_{\mathsf{tie}})$, $\log_2 T = 21$ at Π_{A}'s bundle, hiding delivers $15.45(W - \ell_{\mathsf{pub}}) - 10.5$ bits and binding $(30.9\ell_{\mathsf{pub}} - 21)/3$: at the prototype's $W = 32, \ell_{\mathsf{pub}} = 16$ that is 236.7 and 157.8, matching the permutation width. Every published commitment is width-32 truncated to 16 lanes, a checked invariant (§C).

Clause (iii) commits the whole secret in one region because direct Grover search at $n_v = N_h$ delivers 245.5 bits over the execution against 117.5 for either 256-bit half. Hiding claims also require uniform committed secrets, which is no technicality: hiding is *computational*, so an epoch secret drawn below the full derivation width would fall to dictionary search on the published c_{root} whatever the figures say. Multi-target search applies only where a published value is a direct function of the searched secret; this holds for c_{root} and escrow noise, whose (G, e) separators leave 241 bits even at $M = 2^{30}$.

STARK $\mathcal{F}_{\mathsf{NIZK}}$: $\varepsilon_{\mathsf{ZK}}, \varepsilon_{\mathsf{ks}}$. The FRI IOP is information-theoretically sound and its Fiat–Shamir compilation is knowledge-sound with a straight-line extractor and zero-knowledge against quantum verifiers in the QROM [27, 33, 82].

Three effects bound $\varepsilon_{\mathsf{ks}}$. The additive DEEP, quotient and proximity-gap terms are at least $O(\mathcal{T} \cdot \mathcal{K})/|\mathbb{F}_{\mathrm{ext}}|$ for a degree-d_{ext} challenge extension. A bad Fiat–Shamir challenge is found by Grover search over the transcript nonce, so every challenge-derived term is halved in the exponent and knowledge soundness is at most $\frac{1}{2}\log_2 |\mathbb{F}_{\mathrm{ext}}|$ whatever the height, query count or blowup. And FRI is multi-round, with $n_{\mathsf{fold}} = \Theta(\log \mathcal{T})$ folding rounds, whose per-round QROM compilation loss [27] we absorb into the *conjectured* ethSTARK query budget [74] rather than charge separately. At $d_{\mathrm{ext}} = 10$, $|\mathbb{F}_{\mathrm{ext}}| \approx 2^{309}$, so the Grover cap is ≈ 155 bits and the additive terms $\approx 2^{-279}$ at $\mathcal{T} \cdot \mathcal{K} \approx 2^{30}$: the binding constraint is the query budget, whose $q^2/2^{nq \cdot lb}$ term is charged in §D.2.11. We deploy $nq = 156$, $lb = 2$, which credits each query with lb bits — the soundness of FRI at list-decoding capacity $1 - \rho$ for rate $\rho = 2^{-lb}$, i.e. the ethSTARK conjecture [74]. Proximity gaps are proven only to the Johnson radius $1 - \sqrt{\rho}$ [12], crediting $lb/2$ bits per query: at the same target that is roughly twice the queries and, the openings dominating, roughly twice the proof size — about 30 MB for the 15.31 MB of §8, at the same prove time to within the query-phase share. The figures of §8 are therefore conjectural in size as well as in level, and this is the one row of Table 8 that no engineering removes, only pays for. Grinding is zero, being Grover-halved like the query bits. $d_{\mathrm{ext}} = 10$ is a floor, not a preference: a quartic extension caps knowledge soundness at 62 bits at every trace height and degree 8 at ≈ 124, and the degree costs roughly $3\times$ prove time and $1.5\times$ proof size over a quartic one.

$\varepsilon_{\mathsf{ZK}}$ is the advantage against the hiding FRI commitment, not the one-wayness of the published bindings. The prover blinds each committed polynomial with a uniform mask of degree exceeding the opened points, so the residual advantage is that the mask degree is exhausted, at most $n_{\mathsf{fold}}\, n_{\mathsf{open}}/|\mathbb{F}_{\mathrm{ext}}|$ plus the QROM simulation loss; at $d_{\mathrm{ext}} = 10$, $\varepsilon_{\mathsf{ZK}} \le 2^{-299}$. It is not Grover-halved, the mask being the prover's own randomness. The bound has a precondition the interface does not enforce: the commitment interleaves a height-h trace with h uniform rows, so each column has degree below $2h$ with h blinding coefficients and the openings determine the trace once more than h distinct evaluations are revealed. Hiding therefore requires $nq \le h$ *per committed matrix* rather than per batch, a short matrix being opened at the query index shifted right by the difference of log-heights. Every committed matrix is padded to at least the query count, a checked invariant of the prover.

CGKA path secrecy, $\varepsilon_{\mathsf{CGKA}}$. The main UC-realization theorem of [5]: Insider-Secure TreeKEM realizes $\mathcal{F}_{\mathsf{CGKA}}$ against

adaptive corruptions in the $(\mathcal{F}_{\mathsf{AS}}, \mathcal{F}_{\mathsf{KS}})$-hybrid ROM. Instantiated with ML-KEM HPKE (IND-CCA2 under Module-LWE via the FO transform [47]) and the SHA/HKDF schedule its components are post-quantum, its generalized-selective-decryption analysis is straight-line with polynomial loss, and the path wire's chosen-ciphertext security is subsumed. We rely on it as a hypothesis and require $\varepsilon_{\mathsf{CGKA}} \leq 2^{-136}$ so that it is not the binding row of §D.2.11; its analysis is in the ROM rather than the QROM, and is the only assumption we lift rather than instantiate natively. Π_{A} applies it to its own object: the update path it runs is an unmodified TreeKEM update, with independently generated node keys and independent encryption coins, so the hypothesis transfers with no intermediate step.

Authentication. $\mathcal{F}_{\mathsf{PKI}}$ credentials use ML-DSA-65 (FIPS 204 [66]), EUF-CMA in the QROM.

Model conditions rather than assumptions. The programmable global QROM $\mathcal{G}_{\mathsf{RO}}$, a committee of constant size $N = O(1)$, and the RFC 9420 key schedule modeled as an oracle shared by the functionality, the parties and the simulator are conditions of the model, not computational assumptions. F is modeled as a random oracle and therefore carries no separate PRF assumption.

The instantiated backend, (A3′). Assumptions (A2)–(A4) are properties of a single argument, whereas VerifyCommit verifies a bundle of sub-proofs tied by published digests. Lemmas 10 and 11 lift these three assumptions from a single argument to the bundle (§D.2.9), and Eq. (6) carries the lifted forms. Corollary 2 uses them in one instantiated form:

(A3′) the *instantiated* system — our multi-stage AIR bundle over a FRI-based hiding commitment with Poseidon2 hashing, batched LogUp lookups and cross-table ties, Fiat–Shamir compiled — is a zero-knowledge straight-line argument of knowledge in the QROM for $\mathcal{R}_{\mathsf{A}}$, at the errors of (A2)–(A4).

(A3′) is an assumption rather than a theorem of this paper, and it is not one assumption: it bundles five claims of very different standing, and separating them is what makes the residual small enough to state. Four are properties of *our* compiler, established in the artifact; the fifth is the conjecture the FRI row of Table 8 already carries.

(a) The compilation matches the cited theorems. The QROM analyses of [27, 33] compile an IOP whose challenges are $H(x\|\text{prefix})$, and the online extractor recovers a witness *for the statement the transcript commits to*. Every chip therefore absorbs a domain-separated digest of the published statement — committee configuration, epoch context, logged escrow ciphertexts, c_{root}, the cross-proof ties and the parameter set — before any commitment: the digest rather than the pin columns, those being full-height and a deterministic function of what Stmt recomputes. Enforcing the pins only downstream, by the constraint check at ζ, gives soundness for a fixed statement but leaves the challenges unbound, so the cited theorems would not apply as stated and the Grover charge of §D.2.11 would assume what the compiler does not enforce, that a prover fixes its statement before grinding. A regression proves under one statement and verifies under another on a chip publishing no pin, so the property is decided by the transcript alone.

(b) The lookup argument. The per-argument error Lemma 10 lifts is that of a *batched* LogUp: ℓ_{pub} lookups in one running sum separated by powers of a challenge δ, tuple and lookup challenges drawn after the trace commitment. Its statement and its two hypotheses — post-commitment challenges, δ-separation — are recorded with the engine, and the crafted cancellation attack that δ-separation defeats is mounted in the test suite rather than described.

(c) The cross-table bus. The multiset equality turning n_π per-sub-proof witnesses into one witness for the conjunction is stated as its own lemma with its own error, so what Lemma 10 assumes of a tie is written where the tie is implemented.

(d) The arithmetization decides the relation. No cited analysis says whether a constraint system decides its intended relation, and an under-constrained column is invisible because every honest proof still verifies. A mutation harness therefore perturbs every cell of the trace under test and requires rejection, reporting survivors by position, replacing a chosen set of negative tests by an exhaustive one over the cells it covers. This item remains partial: the harness covers the lookup engine every chip is built on, not yet every chip.

(e) The masking preconditions. The hiding bound requires at least nq rows per committed matrix, a hard assertion on every prover entry point, and live blinding randomness, which a fixed seed would satisfy silently; a regression requires two proofs of one witness to differ.

What (A3′) still assumes is the FRI soundness of §D.2.2 in the list-decoding regime — the ethSTARK conjecture [74], already the C row of Table 8 — with the per-round QROM compilation loss absorbed into it, and the completion of (d) to every chip. Only Corollary 2 rests on any of it: Theorem 1 is stated in the $\mathcal{F}_{\mathsf{NIZK}}$-hybrid model and holds independently of how $\mathcal{F}_{\mathsf{NIZK}}$ is realized.

D.2.3 Straight-line reductions

A reduction is valid against a QPT adversary precisely when it is *straight-line* — it runs the adversary once, forwarding messages and superposition queries, never rewinding, cloning or measuring it — and black-box.

Proposition 5 (Straight-line simulation)**.** *The simulator of §D.2.7 and every hybrid-step reduction of §D.2.8 are straight-line and black-box in $\mathcal{A}$. Measurements are confined to the oracle register of the compressed oracle and to the single*

measure-and-reprogram query of (T3), both licensed against a quantum adversary [33, 82].

Proof. By inspection of §D.2.8: every step is online extraction (T1–T2), one measure-and-reprogram query (T3), a call to $\mathcal{F}_{\mathsf{NIZK}}.\mathsf{Sim}$, a single-run left-or-right reduction, puncturing with O2H (T4), an invocation of the TreeKEM simulator of [5] (straight-line by hypothesis), or oracle reprogramming. $\mathcal{Z}$ drives a single instance, so no quantum rewinding is needed for composition [76]. □

D.2.4 QROM toolkit

Four standard techniques carry the proof; none rewinds, clones or measures $\mathcal{A}$. **(T1)** $\mathcal{S}$ runs $\mathcal{G}_{\mathsf{RO}}$ as Zhandry's *compressed oracle* [82] — a purifying register whose basis states are databases D of at most $Q_{\mathcal{G}_{\mathsf{RO}}}$ point-value pairs, perfectly indistinguishable from a random oracle — because answering from a classical transcript would disturb a superposition query; "$\mathcal{S}$ reads D" means a measurement. **(T2)** The extractor of $\mathcal{F}_{\mathsf{NIZK}}$ is *online* [34, 35]: it recovers a witness from a verifying Fiat–Shamir proof by inspecting D, and for a commit-and-open argument that witness is the set of committed values D already fixes. **(T3)** Where the value committed at a challenge point is needed we use *measure-and-reprogram* [33], whose $O(Q^2_{\mathcal{G}_{\mathsf{RO}}})$ loss is already inside $\varepsilon_{\mathsf{ks}}$ [27]. **(T4)** Steps that would otherwise condition on "$\mathcal{A}$ has not queried $\mathcal{G}_{\mathsf{RO}}$ at a secret point" — not well defined against superposition access — use semi-classical one-way-to-hiding [6]: for a set S of secret points and $\mathcal{G}_{\mathsf{RO}}\backslash S$ the punctured oracle,

$$\left|\Pr[\mathcal{A}^{\mathcal{G}_{\mathsf{RO}}}=1]-\Pr[\mathcal{A}^{\mathcal{G}_{\mathsf{RO}}\backslash S}=1]\right| \;\le\; 2\sqrt{(Q_{\mathcal{G}_{\mathsf{RO}}}+1)\Pr[\mathsf{Find}]},$$
$$\Pr[\mathsf{Find}] \;\le\; 4Q_{\mathcal{G}_{\mathsf{RO}}}\,p_{\max},$$

with $p_{\max}$ bounding the probability that any single point of S takes a given value in $\mathcal{A}$'s view. Every S we puncture on is determined by a uniform λ-bit secret, so $p_{\max}\le 2^{-\lambda}$ and each step costs

$$\varepsilon_{\mathsf{O2H}}(Q_{\mathcal{G}_{\mathsf{RO}}}) \;=\; 4\sqrt{Q_{\mathcal{G}_{\mathsf{RO}}}(Q_{\mathcal{G}_{\mathsf{RO}}}+1)}\cdot 2^{-\lambda/2}. \qquad (5)$$

D.2.5 Realizability of the functionality

Five features of Fig. 1 are forced by realizability rather than chosen, each against an alternative an environment separates. (R1) AUDIT returns (s_e,k_e) and reaches $\mathcal{S}$ on every audit, because $\mathcal{S}$ must explain the logged escrow ciphertext as a sharing of the reconstructed value and cannot invent s_e, and because the environment checks the returned s_e against the public c_{root}, which $\mathcal{S}$ can only have programmed if it learned the value; releasing s_e leaks nothing the real audit does not. It returns *only* epoch e's pair, which makes the real interface realizable in the reverse direction too: a functionality releasing ancestors would leak more than Audit computes, Audit reading one escrow ciphertext and deriving k_e from its own recovered es_e. (R2) AUDIT is honesty-oblivious, real recovery interpolating over any t shares and a corrupt A_j contributing a valid one, so a functionality returning $\perp$ would refuse an audit the protocol completes; the residual leakage is stated directly instead, through the output reaching $\mathcal{S}$ when $T\cap C\neq\emptyset$ and through LEAK. (R3) The functionality evaluates no cryptography and samples s_e,k_e independently. The alternative is not dependent sampling but a functionality that *computes* KeySchedule, which is unrealizable: it would fix the oracle points Lemma 9 must reprogram at an exposure, $\mathcal{G}_{\mathsf{RO}}$ being global. Independent sampling is consistent with what the environment can check because $\mathcal{S}$ programs the relation wherever an exposure reveals both sides, at the cost of Lemma 7. (R4) Acceptance is per party, because $\mathcal{R}_{\mathsf{A}}$ determines the escrow and nothing else: an admitted commit message fixes the escrowed secret and fixes nothing about what each roster member obtains from Process. A functionality with $\mathsf{Acc}_e = ros_e$ on every admitted epoch would be unrealizable by Π_{A}, and $\mathcal{S}$ instead computes Acc_e by replay from public data (Fig. 4); the guarantee is unaffected, AUDIT reading s_e and not Acc_e. (R5) $L(e)$ names the committer and tree keys, genesis carries its own verification interface and a sampled k_0, and $\mathcal{S}$ supplies $\mathcal{M}_e$ and PK_e at every registration: $\mathcal{R}_{\mathsf{A}}$'s statement is not determined by the roster, so no UC statement would otherwise cover π_0. All are public in the real execution.

Lemma 6 (Corrupt-committer values are extracted, not chosen)**.** *For every commit message with a corrupt committer that* VERIFYCOMMIT *admits, the tuple $\mathcal{S}$ registers is the one the relation proves of the logged data: the escrow target is the value the logged escrow ciphertext shares and the logged c_{root} commits to, and the transcript is the logged $\mathcal{M}_e$. This fails only if extraction fails on a verifying proof, with the probability of Lemma 10 per epoch.*

Proof. $\mathcal{S}$ obtains the values from the online extractor (T2). Clauses (i)–(iii) give es^* whose commitment is the logged c_{root}, shared by the logged escrow. The epoch key is sampled by $\mathcal{S}$ and tied to the extracted value by programming KeySchedule. The lemma is a statement about a simulator built as in Fig. 4, not a check the functionality performs; the corresponding statement about the protocol is escrow soundness (Theorem 13). □

D.2.6 Equivocation in the programmable QROM

Every value a corruption can force $\mathcal{S}$ to open is $\mathcal{G}_{\mathsf{RO}}$-derived — the wraps e_j and the commitment c_{root} (Eq. (3)) — hence equivocable by lazy reprogramming. This replaces *non-committing* encryption [22, 68], which has no standardized post-quantum instantiation, at a constant 2^N committee-guessing cost in $H_2\to H_3$. The path ciphertexts need no equivocation of their own: they are the TreeKEM simulator's, and

no interface of $\mathcal{F}_{\mathsf{aud}}$ publishes a value that would let $\mathcal{Z}$ recompute them. The substitution is not free, moving the obligation to the deployment, which must honor the erasure discipline of §3.4 with hardware-enforced erasure wherever software erasure is untrusted. The one secret deliberately outside that discipline is each auditor's long-term decapsulation key: an audit may be requested arbitrarily far in the future, so adaptive corruption of A_j exposes it unavoidably, and Appendix B.3 bounds the window over which such corruptions accumulate.

Lemma 7 (Equivocation, QROM)**.** *Let $\mathcal{S}$ emit, at commit time, independent uniform strings for every wrap e_j and for the commitment c_{root} of an honest epoch, leaving the corresponding $\mathcal{G}_{\mathsf{RO}}$-points unprogrammed, and reprogram $\mathcal{G}_{\mathsf{RO}}$ at such a point only when an adaptive* CORRUPT *or an* Audit *legitimately exposes the keying material. Let ζ lower-bound the min-entropy of the reprogrammed point in $\mathcal{A}$'s view until that exposure. Then any QPT distinguisher making $Q_{\mathcal{G}_{\mathsf{RO}}}$ superposition queries detects a schedule of R such reprogrammings with advantage*

$$\mathbf{Adv}^{\mathsf{equiv}}_{\mathcal{S}}(\lambda) \;\le\; \tfrac{3}{2} R \sqrt{Q_{\mathcal{G}_{\mathsf{RO}}}\, 2^{-\zeta}}, \qquad R = O\big(E\,(N+2)\big),$$

with E the number of epochs and N the committee size; the 2 counts the commitment point and the key-schedule point of each epoch. The paper applies the lemma at two values of ζ: $\zeta = 256$ at the $O(EN)$ wrap points, whose entropy is the width of $\mu_{e,j}$, and $\zeta = N_h$ at the commitment, key-schedule and audit-leaf points, whose entropy is the width of es_e.

Proof. Each emitted string is the $\mathcal{G}_{\mathsf{RO}}$-image of a secret uniform input: a wrap pad at $(\texttt{"wrap"} \| K_{e,j})$ with $K_{e,j} = H_{\mathsf{P2}}(\texttt{"key"} \| \mu_{e,j})$, carrying the 256 bits of $\mu_{e,j}$, or a commitment, key-schedule image or audit leaf derived from the epoch secret, carrying N_h. Emitting a uniform string and later reprogramming $\mathcal{G}_{\mathsf{RO}}$ at that point to the required image is one instance of the adaptive resampling game of [41], whose theorem bounds a $Q_{\mathcal{G}_{\mathsf{RO}}}$-query quantum distinguisher by $\frac{3}{2}\sqrt{Q_{\mathcal{G}_{\mathsf{RO}}} 2^{-\zeta}}$ at a position of min-entropy ζ; the R points are disjoint and domain-separated (the commitment carries the $\texttt{"crot"}$ label the key schedule does not), so a hybrid over them accumulates additively, and per epoch there are $\le N$ wraps, one commitment, one key-schedule point and $\le 2+N$ audit points.

The hypothesis must be *information-theoretic* min-entropy, and at the wrap point the secrecy of $\mu_{e,j}$ is only computational in the real protocol, so we do not apply the lemma there directly: $H_2 \to H_3$ first replaces $\mu_{e,j}$ by an independent message under IND-CPA, after which the position is uniform and independent of the logged ciphertext. Substituting a computational bound for ζ inside [41] would not be a valid instantiation, which is why that step precedes this one. □

D.2.7 Simulator

Figure 4 gives $\mathcal{S}$ case by case. Emitting each honest transcript before the corruption schedule is known is possible for two reasons proper to the secure-erasure model: erasure of a committer's ephemerals means corrupting a sender exposes none of the randomness behind an emitted ciphertext or proof, and every value a receiver corruption exposes is $\mathcal{G}_{\mathsf{RO}}$-derived, hence equivocable (Lemma 7). The two lemmas below are the content of the reconciling handlers.

Lemma 8 (Adaptive opening, QROM)**.** *On* CORRUPT(P) *at any time, $\mathcal{S}$ produces a state perfectly indistinguishable from a real corruption, conditioned on the event of Lemma 7 not occurring.*

Proof. If P is a member, $\mathcal{F}_{\mathsf{aud}}$ returns its current MLS state, which $\mathcal{S}$ renders through the adaptive TreeKEM simulator of [5]; by erasure no past ephemeral survives, so P's earlier commits stay unopened on the sender side, and if P holds the epoch secret of a released epoch $\mathcal{S}$ reprograms $\mathcal{G}_{\mathsf{RO}}$ so the logged c_{root} opens to it. If $P = A_j$, $\mathcal{S}$ reveals $sk_j^{\mathbb{A}}$ and for each emitted epoch draws $\sigma_{e,j} \xleftarrow{\$} \mathbb{F}_q$ and reprograms $H_{\mathsf{P2}}(\texttt{"wrap"} \| K_{e,j}) := \mathsf{e}_{e,j} \oplus \sigma_{e,j}$. While fewer than t auditors are corrupt these shares are jointly independent of the escrowed secret by Shamir's perfect t-privacy, so the uniform choice is distributed exactly as real — available here because the lemma is applied at H_4 and later, where the coefficients are already idealized as uniform; before that step the sharing is only computationally private, its coefficients being a λ-bit-keyed oracle image (Lemma 3). The point is fresh because $\mathcal{A}$ cannot have recovered $\mu_{e,j}$ before holding $sk_j^{\mathbb{A}}$. Whole-ciphertext consistency is deferred to audit, and each reprogramming is one instance of Lemma 7. □

Lemma 9 (Audit consistency, QROM)**.** *On* Audit(T,e) *with $|T| \ge t$ for an honest epoch, and on* (LEAK$,e$) *once $\ge t$ of epoch e's auditors are corrupt, $\mathcal{S}$ explains the logged escrow ciphertext as a Shamir sharing of the epoch secret es^* returned by $\mathcal{F}_{\mathsf{aud}}$, programming the wrap and commitment openings so that reconstruction matches es^* and* $\mathsf{KeySchedule}(es^*, \mathsf{ctx}_e) = k_e$*, except with the probability of Lemma 7. The argument does not depend on the honesty of T.*

Proof. $\mathcal{S}$ uses the returned es^* as the escrow opening and does not resample, since $\mathcal{Z}$ must see the values the functionality returns; it reprograms the unprogrammed c_{root} point onto es^*, the key-schedule point $\mathsf{KeySchedule}(es^*, \mathsf{ctx}_e) := k_e$, and the residual wrap points onto a degree-$(t-1)$ polynomial over $\mathbb{F}_q$ through es^* and the already-opened shares.

Such a polynomial exists by a dimension count: $r < t$ opened shares and $f(0) = es^*$ impose $r+1 \le t$ independent conditions on t coefficients, leaving an affine space of dimension $t-r-1 \ge 0$ and a single polynomial exactly at $r = t-1$, so no later opening contradicts an earlier one; the split field runs the same count per half. Residual points are programmed

Simulator $\mathcal{S}$ (for Π_{A}). $\mathcal{S}$ runs a copy of $\mathcal{A}$, relays between $\mathcal{Z}$ and $\mathcal{A}$, plays the honest parties toward $\mathcal{A}$, controls $(\mathcal{F}_{\mathsf{NIZK}}, \mathcal{F}_{\mathsf{Log}}, \mathcal{G}_{\mathsf{RO}}, \mathcal{F}_{\mathsf{PKI}})$, and interacts with $\mathcal{F}_{\mathsf{aud}}$ on its adversary interface. It keeps a local copy of $\mathcal{H}$ and a table Prog of programmed points. Corruption is *adaptive* in the secure-erasure model; $\mathcal{C}$ grows over time and per-epoch privacy holds while $|\mathcal{C} \cap \mathbb{A}(e)| < t$. Every honest output is emitted before $\mathcal{S}$ knows the corruption schedule, so all secret openings are deferred to the CORRUPT, AUDIT and LEAK cases (Lemmas 7, 8).

Oracles and setup. $\mathcal{S}$ runs $\mathcal{G}_{\mathsf{RO}}$, and the KeySchedule points modeled inside it, as Zhandry's compressed oracle [82] with a purifying database register D: programming a point updates D (T1) and extraction measures it (T2). $\mathcal{S}$ never records $\mathcal{A}$'s queries. It generates *all* committee keys $\{(pk_j^{\mathbb{A}}, sk_j^{\mathbb{A}})\}_{j\in[N]}$ via $\mathcal{F}_{\mathsf{PKI}}$ and publishes mpk; no secret key reaches $\mathcal{A}$ except on a CORRUPT.

On (GROUPCREATE) leakage $L(0)$, honest creator: for every $j \in [N]$ encrypt a random $\mu_{0,j}$ under $pk_j^{\mathbb{A}}$ and emit a uniform wrap $\mathsf{e}_{0,j} \xleftarrow{\$} \mathbb{F}_q$ (no share fixed); emit $c_{\mathsf{root}}^{(0)}$ as a uniform ℓ_{pub}-lane digest, point left unprogrammed; $\tilde{\pi}_0 \leftarrow \mathcal{F}_{\mathsf{NIZK}}.\mathsf{Sim}(x_0)$ on the public $\mathcal{R}_{\mathsf{unit}}$ statement; assemble $\mathcal{M}_0$, choose PK_0, supply both to $\mathcal{F}_{\mathsf{aud}}$, post $\mathcal{M}_0$ to $\mathcal{F}_{\mathsf{Log}}$, erase the ephemerals.

On $\mathcal{A}$ posting a GroupInfo, corrupt creator: if VerifyCreate rejects, ignore. Else extract a witness for $\mathcal{R}_{\mathsf{init}}$, giving s_0; sample $k_0 \xleftarrow{\$} \{0,1\}^{N_h}$, program $\mathsf{KeySchedule}(s_0, \mathsf{ctx}_0) := k_0$, set $\mathsf{Acc}_0 = ros_0$, and send $(s_0, k_0), \mathsf{Acc}_0, \mathcal{M}_0, PK_0$ to $\mathcal{F}_{\mathsf{aud}}$.

On (COMMIT) leakage $L(e) = (ros_e, tree_e, m_e, P_e, PK_e)$, honest committer:

1. commitment c_{root} as a uniform ℓ_{pub}-lane digest, point left unprogrammed;
2. escrow: encrypt a random $\mu_{e,j}$ under each $pk_j^{\mathbb{A}}$ and emit a uniform wrap $\mathsf{e}_{e,j}$, shares deferred (Lem. 7);
3. for $j \in \mathcal{C} \cap \mathbb{A}(e)$ the wrap point is queryable at once, so fix those slots now: draw $\sigma_{e,j} \xleftarrow{\$} \mathbb{F}_q$ if $|\mathcal{C} \cap \mathbb{A}(e)| < t$, else call $(\textsc{Leak}, e)$ and program from s_e;
4. the whole update path — node keys, path secrets and path ciphertexts — from the TreeKEM simulator of [5], using P_e and PK_e to fix the filtered path and copath resolutions. $\mathcal{S}$ owes no openings of its own here: no clause of $\mathcal{R}_{\mathbb{A}}$ constrains the path, and no interface of $\mathcal{F}_{\mathsf{aud}}$ publishes a value from which a path ciphertext is recomputable;
5. $\pi \leftarrow \mathcal{F}_{\mathsf{NIZK}}.\mathsf{Sim}(x_e)$ with x_e assembled from ctx_e, mpk and the emitted values; assemble $\mathcal{M}_e$, supply $(\mathcal{M}_e, PK_e)$ to $\mathcal{F}_{\mathsf{aud}}$, post to $\mathcal{F}_{\mathsf{Log}}$. Erasure means a later corruption of the committer owes no witness.

On $(\textsc{Corrupt}, P)$ at any time; $\mathcal{F}_{\mathsf{aud}}$ returns P's abstract state:

1. *member/committer:* render P's MLS state via the [5] TreeKEM simulator; if the released state determines the epoch secret of an epoch e, program $\mathcal{G}_{\mathsf{RO}}$ so the logged c_{root} opens to it and so $\mathsf{KeySchedule}(es_e, \mathsf{ctx}_e) := k_e$. Repeat at each later epoch while P stays corrupt;
2. *auditor A_j:* reveal $sk_j^{\mathbb{A}}$, add j to $\mathcal{C}$, and for each emitted epoch draw $\sigma_{e,j} \xleftarrow{\$} \mathbb{F}_q$ and program $H_{\mathsf{P2}}(\text{"wrap"} \| K_{e,j}) := \mathsf{e}_{e,j} \oplus \sigma_{e,j}$ with $K_{e,j} = H_{\mathsf{P2}}(\text{"key"} \| \mu_{e,j})$ (Shamir-independent while $|\mathcal{C} \cap \mathbb{A}(e)| < t$). When this corruption reaches t, switch to the LEAK handler.

On $\mathcal{A}$ posting $\mathcal{M}_e$ to $\mathcal{F}_{\mathsf{Log}}$, corrupt committer: if $\mathsf{Vrfy}(x_e, \pi) \neq 1$ for $x_e = \mathsf{Stmt}(\mathsf{pp}, \mathsf{mpk}, \mathcal{L}, \mathcal{M}_e)$, ignore. Else run Ext on the compressed-oracle database (T1–T2) to obtain a witness for $\mathcal{R}_{\mathbb{A}}$; on failure **abort** (Lem. 10). Set $s_e = es^*$, sample $k_e \xleftarrow{\$} \{0,1\}^{N_h}$, program $\mathsf{KeySchedule}(es^*, \mathsf{ctx}_e) := k_e$, and send $(s_e, k_e), \mathsf{Acc}_e, \mathcal{M}_e, PK_e$ to $\mathcal{F}_{\mathsf{aud}}$. All recorded values are extracted, not chosen (Lem. 6); the witness is not forwarded. $\mathcal{S}$ sets Acc_e by replaying each honest member's processing of the logged UpdatePath, as the [5] corrupt-committer handling does; the replay can return a proper subset of ros_e, which is item (R4).

On $(\textsc{Audit}, T, e)$, $|T| \geq t$; $\mathcal{F}_{\mathsf{aud}}$ returns (s_e, k_e) to $\mathcal{S}$ as well as to T: use $s_e = es^*$ as the escrow opening and do *not* resample; program the unprogrammed c_{root} point onto es^* and $\mathsf{KeySchedule}(es^*, \mathsf{ctx}_e) := k_e$; then fix the degree-$(t-1)$ polynomial f' over $\mathbb{F}_q$ through es^* and the already-opened shares and program every remaining wrap onto it, for *every* $j \in T$ whether corrupt or honest (Lem. 9). No path ciphertext is touched: an audit publishes no value from which one can be recomputed.

On $\geq t$ of epoch e's auditors being corrupt: call $(\textsc{Leak}, e)$ and program exactly as in the AUDIT case, so the shares $\mathcal{A}$ holds interpolate to es^*. No abort arises: the epoch is outside $\mathsf{safe}_{\mathsf{ESC}}$.

Aborts. Only (a) extraction failure on a verifying adversarial proof or (b) the semi-classical Find event on the points $\mathcal{S}$ must program, bounded by Lemmas 7 and 9. $\mathcal{S}$ makes no committee-subset guess and never aborts on an auditor corruption.

Figure 4: Simulator $\mathcal{S}$ for Theorem 1, case by case over $\mathcal{F}_{\mathsf{aud}}$. $\mathcal{S}$ matches the real transcript from public leakage and reconciles corruptions, audits and leakage by programming $\mathcal{G}_{\mathsf{RO}}$.

for every $j \in T$ regardless of honesty, real reconstruction being honesty-oblivious. The LEAK case is identical with the sharing fixed when the t-th auditor is corrupted, the polynomial being unique.

The logged path ciphertexts need no step of their own, which is where Π_{A}'s scope pays: an audit publishes es_e, from which no path ciphertext is recomputable, the coins being the RFC's independent randomness and the epoch secret downstream of the whole update. No party can compare a reproduced ciphertext against $\mathcal{F}_{\mathsf{Log}}$, so the TreeKEM simulator's ciphertexts survive the audit unchanged. The points are counted in R: per exposed epoch, one commitment, one key-schedule point and up to N residual wraps. □

D.2.8 Hybrid sequence

We transform the real execution H_0 into the ideal execution H_6. Each step names the change, a reduction $\mathcal{A}_i$ built from $\mathcal{Z}$, and the assumption it reduces to; by Proposition 5 every $\mathcal{A}_i$ is straight-line, and corruption is adaptive throughout, reconciled by Lemma 8. Of the features of §D.2.5, (R1)–(R2) make the ideal world leak more and are answered by Lemma 9, (R3) is discharged by the programming those lemmas perform, and (R4)–(R5) are computed by $\mathcal{S}$ from public data. *Audit execution* does get its own step, $H_2 \to H_{2.5}$: the interface an honest auditor runs changes there, and it must change *before* any ciphertext does.

$H_0 \to H_1$ **(extract from adversarial proofs).** For each corrupt-committer epoch with $\mathsf{Vrfy}(x_e, \pi_e) = 1$, run the straight-line QROM extractor on (x_e, π_e) and the compressed-oracle database (T1) to obtain a witness for $\mathcal{R}_{\mathbb{A}}$; abort with $\perp_{\mathsf{ks}}$ if extraction fails. $\mathcal{A}_1$ forwards each verifying pair to the knowledge-soundness game, so a $\perp$ on a verifying proof is exactly a break. Extraction *precedes* simulation: the honest proofs are still real here, so $\mathcal{A}_1$ still holds their witnesses and produces them with the real prover, and needs only plain knowledge soundness, whereas simulating first would require simulation-extractability. Bound: Lemma 10 per epoch, with no separate term for the registration (Lemma 6).

$H_1 \to H_2$ **(simulate honest proofs).** Replace each honest $\pi_e \leftarrow \mathsf{Prove}(x_e, w_e)$ by $\mathcal{F}_{\mathsf{NIZK}}.\mathsf{Sim}(x_e)$. $\mathcal{A}_2^{\mathsf{ZK}}$ runs $\mathcal{Z}$, calls its

zero-knowledge challenge oracle on x_e at each honest commit, splices the reply in, and forwards $\mathcal{Z}$'s bit. Bound: $\leq En_\pi \varepsilon_{\mathsf{ZK}}$ (Lemma 11).

$H_2 \to H_{2.5}$ **(route audit through the simulator).** Replace the *execution* of Audit at every honest auditor by the interface $\mathcal{S}$ runs in the ideal world: the auditor receives the reconstruction $\mathcal{S}$ produces by the programming of Lemma 9 rather than computing Dec, the wrap key and the interpolation; ciphertexts, wraps and keys are unchanged. The step is perfect, since at H_2 the logged $\mathsf{c}_j^{\mathsf{esc}}$ still decapsulates to the $\mu_{e,j}$ that keys the pad, so each returned value coincides with the programmed one and $\Pr[H_{2.5}] = \Pr[H_2]$. The order is forced: routing audit through $\mathcal{S}$ after the swap would require the mismatch to be argued unobservable.

$H_{2.5} \to H_3$ **(equivocate the escrow ciphertext).** Replace the escrow message under every honest slot by an independent $\mu'_{e,j}$, and emit every wrap as a fresh uniform string, leaving $(\texttt{"wrap"} \| K_{e,j})$ unprogrammed. Because $N = O(1)$, $\mathcal{A}_3^{\mathsf{esc}}$ guesses the subset $\mathbb{H} \subseteq [N]$ of slots that stay honest for the whole execution (a factor $\leq 2^N$), embeds an IND-CPA challenge key at each $j \in \mathbb{H}$ and holds real keys elsewhere, so it never owes a key it must later surrender; for $j \in \mathbb{H}$ it obtains $\mathsf{c}_j^{\mathsf{esc}}$ from its left-or-right challenge on $(\mu_{e,j}, \mu'_{e,j})$.

The step is *decisional* for a quantitative reason. Keeping the real $\mu_{e,j}$ and puncturing on $S_j = \{\texttt{"key"} \| \mu_{e,j}, \texttt{"wrap"} \| K_{e,j}\}$ instead would bound $\Pr[\mathsf{Find}]$ by the escrow PKE's one-wayness at $2\sqrt{(Q_{\mathcal{G}_{\mathsf{RO}}}+1)4Q_{\mathcal{G}_{\mathsf{RO}}}\,\varepsilon_{\mathsf{esc}}}$ per honest slot, which at $\varepsilon_{\mathsf{esc}} \approx 2^{-164}$ exceeds 2^{-53} already at $Q_{\mathcal{G}_{\mathsf{RO}}} = 1$ and reaches 1 for $Q_{\mathcal{G}_{\mathsf{RO}}} \geq 2^{53}$, so the bound is uninformative. After the decisional swap $\mu_{e,j}$ appears nowhere in the transcript, so $(\texttt{"wrap"} \| K_{e,j})$ carries its 256 bits as *information-theoretic* min-entropy and Lemma 7 applies at $\zeta = 256$, with no puncturing and no square root. O2H must be applied at a position that is already information-theoretically uniform, never as a substitute for making it so, and the same holds at $H_4 \to H_6$.

One consequence of the swap is accounted for rather than argued away. Under the swapped ciphertext the logged $\mathsf{c}_j^{\mathsf{esc}}$ no longer decapsulates to the $\mu_{e,j}$ that keys the pad, so the real Audit would unwrap a wrong share at every honest slot — but no party runs it: $H_{2.5}$ has already moved every honest auditor onto the programmed interface, and no corruption exposes a challenge slot's key, $\mathbb{H}$ being by definition the slots that stay honest throughout. Consistency of the escrow with the reconstructed secret is restored where the functionality hands that secret out, not before.

The reduction's challenge ciphertext uses the challenger's coins, whereas the protocol's are $\mathsf{CBD}_2(\Psi_\eta(\texttt{"esc"} \| j \| G \| e))$, so the $\texttt{"esc"}$ label family is punctured here; η is uniform and erased, and canonical-lane rejection makes the idealized coins exactly CBD_2-distributed. Bound: $\leq 2^N NE\,\varepsilon_{\mathsf{esc}} + E\,\varepsilon_{\mathsf{O2H}}(Q_{\mathcal{G}_{\mathsf{RO}}})$, plus the equivocation term.

$H_3 \to H_4$ **(idealize the sharing coefficients).** Treat the sharing coefficients $f_d = F_\eta(\texttt{"coef"} \| d)$ as independent uniform values of $\mathbb{F}_q$, which is what makes Shamir's t-privacy perfect rather than computational and is the hypothesis Lemma 8 needs. F is modeled inside $\mathcal{G}_{\mathsf{RO}}$, so $F_\eta(\cdot)$ is a uniform random function. We may not argue that $\mathcal{A}$ never queries $\eta \| \cdot$; instead we run the step against $\mathcal{G}_{\mathsf{RO}} \setminus S_\eta$ for $S_\eta = \{\eta \| z\}$, where the images are information-theoretically absent so the replacement is perfect, and transfer back by (T4). Since η is uniform, erased and never exposed, $p_{\max} \leq 2^{-\lambda}$ and the transfer costs $\leq E\,\varepsilon_{\mathsf{O2H}}(Q_{\mathcal{G}_{\mathsf{RO}}})$. The step reaches the escrow only: the update path carries the RFC's own independent coins, so no lemma is required to treat the path layer as an ordinary TreeKEM commit.

$H_4 \to H_5 \to H_6$ **(idealize the honest path layer and the commitment, jointly).** Replace each honest epoch's path-secret plaintexts inside its path ciphertexts by uniform values (H_5) and each honest c_{root} by a fresh uniform digest whose $\mathcal{G}_{\mathsf{RO}}$-point is opened only on legitimate exposure (H_6). Neither ordering works alone: idealizing the commitment first needs O2H with an information-theoretic $p_{\max}$, which the real es_e — a deterministic function of the commit secret inside the logged ciphertexts — does not provide, while idealizing the path layer first needs the TreeKEM simulator of [5], which cannot be invoked while the transcript still contains a c_{root} depending on the path secrets. We therefore do both in one reduction to CGKA path secrecy: $\mathcal{S}$ runs $\mathcal{S}_{\mathsf{TK}}$ for the path layer, adopting its openings on each member CORRUPT, and emits c_{root} uniformly. The CGKA hypothesis then makes the commit secret, hence es_e, uniform and independent of everything $\mathcal{A}$ sees, and only then is the O2H application on the commitment point legitimate. Bound: $\leq \varepsilon_{\mathsf{CGKA}} + E\,\varepsilon_{\mathsf{O2H}}(Q_{\mathcal{G}_{\mathsf{RO}}})$.

After H_6 the transcript is a function of public metadata, the extracted adversarial injections, the corruption schedule (Lemma 8) and the audit events (Lemma 9), which is exactly the output of $\mathcal{F}_{\mathsf{aud}}$ driven by $\mathcal{S}$.

D.2.9 Bundle soundness

Assumption (A3) concerns one argument, whereas VerifyCommit verifies a *bundle*: Π_{A} publishes 2 sub-proofs, tied by the mechanisms of §7. Extraction must yield one witness consistent across sub-proofs, so the bundle's error is not $\varepsilon_{\mathsf{ks}}$.

Lemma 10 (Bundle knowledge soundness)**.** *Let a bundle consist of n_π sub-proofs whose shared values are tied by n_{tie} published ℓ_{pub}-lane digests. If each sub-proof is straight-line knowledge-sound with error $\varepsilon_{\mathsf{ks}}$ and the truncated digest is collision-resistant with error $\varepsilon_{\mathsf{bind}}$, then from any accepted bundle the extractor outputs a single witness satisfying every clause, except with probability $n_\pi \varepsilon_{\mathsf{ks}} + n_{\mathsf{tie}} \varepsilon_{\mathsf{bind}}$.*

Proof. Run the extractor of (T2) on each sub-proof; a failure on any one is a knowledge-soundness break, giving $n_\pi \varepsilon_{\mathsf{ks}}$. The n_π witnesses satisfy their own clauses but need not agree

on a shared value; for each tie the producing and consuming witnesses both open the same published digest, so a disagreement is a collision, charged $\varepsilon_{\mathsf{bind}}$. Absent a collision their union is a single witness for the conjunction. □

Both n_π and n_{tie} are constants for Π_{A}: the relation has no clause indexed by a path level, so the bundle does not grow with the group. The ties are published at the same ℓ_{pub} as every other commitment and enter the union charge of §D.2.11 exactly as c_{root} does.

Lemma 11 (Bundle zero knowledge)**.** *A bundle of n_π sub-proofs under the hiding PCS, publishing n_{tie} tie digests of which $n^{\mathsf{u}}_{\mathsf{tie}}$ are unsalted commitments to values of width n, is simulatable from its public inputs except with probability $n_\pi \varepsilon_{\mathsf{ZK}} + \frac{3}{2} n^{\mathsf{u}}_{\mathsf{tie}} \sqrt{Q_{\mathcal{G}_{\mathsf{RO}}} 2^{-n}}$.*

Proof. The simulator obtains each sub-proof from $\mathcal{F}_{\mathsf{NIZK}}.\mathsf{Sim}$, at $n_\pi \varepsilon_{\mathsf{ZK}}$. A salted tie digest is simulated by a uniform string, since its preimage contains twelve lanes of fresh randomness from η. An unsalted tie digest is a deterministic public function of the committed value and cannot be simulated uniformly; it is emitted as a uniform digest and reprogrammed onto the true value on exposure, which is one instance of Lemma 7 at $\zeta = n$ — the same mechanism the wraps and the commitment use, and charged the same way rather than asserted directly. □

For Π_{A} the second term is *zero*: every tie the bundle publishes crosses two chips that both hold η, so all of them carry the secret salt and $n^{\mathsf{u}}_{\mathsf{tie}} = 0$ (§C). The lemma is stated in general form because Appendix E.2 applies it to an aggregated bundle.

Zero knowledge of the commit transcript. The bundle lemmas cover the proof object; the claim is about the whole commit message $\mathcal{M}_e$. Its parts are simulated at distinct steps: π_e by Lemma 11, c_{root} and the path ciphertexts at $H_4 \to H_6$, and $C_{\mathsf{esc}\,e}$ at $H_2 \to H_3$; their sum is the zero-knowledge part of Eq. (6), and the transcript is then a function of $L(e)$ alone. The governing term is zero knowledge of the proof system, the advantage against the hiding FRI commitment at $\varepsilon_{\mathsf{ZK}} \le 2^{-299}$ (278 bits over the execution); there is no weaker one-wayness term to set against it, Π_{A}'s ties all carrying a secret salt.

D.2.10 Concrete bound and main theorem

Summing the hybrid gaps, the equivocation term and Lemmas 10 and 11 gives, for any QPT $\mathcal{Z}$ making $Q_{\mathcal{G}_{\mathsf{RO}}}$ superposition queries,

$$\begin{aligned}\big|\Pr[\mathcal{Z}{=}1 \text{ in } \mathsf{REAL}] - \Pr[\mathcal{Z}{=}1 \text{ in } \mathsf{IDEAL}]\big| \\ \le\ & En_\pi\varepsilon_{\mathsf{ZK}} + En_\pi\varepsilon_{\mathsf{ks}} + E(1{+}n_{\mathsf{tie}})\varepsilon_{\mathsf{bind}} \\ & + 2^N NE\,\varepsilon_{\mathsf{esc}} + \varepsilon_{\mathsf{CGKA}} \\ & + \tfrac{3}{2}R_{\mathsf{w}}\sqrt{Q_{\mathcal{G}_{\mathsf{RO}}}}\,2^{-128} + \tfrac{3}{2}R_{\mathsf{p}}\sqrt{Q_{\mathcal{G}_{\mathsf{RO}}}}\,2^{-N_h/2} \\ & + \Pi_{\mathsf{p}}\,\varepsilon_{\mathsf{O2H}}(Q_{\mathcal{G}_{\mathsf{RO}}}). \qquad (6)\end{aligned}$$

Here n_π is the sub-proof count and n_{tie} the tie count of §D.2.9; the reprogrammed points split by min-entropy into $R_{\mathsf{w}} = O(EN)$ wrap points at $\zeta = 256$ and $R_{\mathsf{p}} = O(E)$ commitment, key-schedule and audit-leaf points at $\zeta = N_h$, with $R = R_{\mathsf{w}} + R_{\mathsf{p}}$ and $\Pi_{\mathsf{p}} = 3E$. No $n^{\mathsf{u}}_{\mathsf{tie}}$ term appears, by Lemma 11. The $E\,\varepsilon_{\mathsf{bind}}$ summand outside n_{tie} arises because $\mathcal{S}$ registers the extracted secret for a corrupt-committer epoch while honest members run PROCESS on the secret they derive, and by clause (iii) a difference is an H_{P2} collision on the logged commitment. Both oracle terms are $O(\mathrm{poly}(\lambda)\,Q_{\mathcal{G}_{\mathsf{RO}}}\,2^{-\lambda/2})$, the puncturing term dominating since it grows as $Q_{\mathcal{G}_{\mathsf{RO}}}$ against $\sqrt{Q_{\mathcal{G}_{\mathsf{RO}}}}$.

Theorem 1 (Adaptive post-quantum UC security of Π_{A}, restated)**.** *In the secure-erasure model and the programmable global QROM, and under the assumptions of §D.2.2, Π_{A} UC-realizes $\mathcal{F}_{\mathsf{aud}}$ with respect to $\mathsf{safe}_{\mathsf{ESC}}$ in the $(\mathcal{F}_{\mathsf{NIZK}}, \mathcal{F}_{\mathsf{Log}}, \mathcal{G}_{\mathsf{RO}}, \mathcal{F}_{\mathsf{PKI}})$-hybrid model, against QPT adversaries that adaptively corrupt any group members and, for each honest epoch, fewer than t of its auditors. Instantiating $\mathcal{F}_{\mathsf{NIZK}}$ under (A3′), the advantage of any QPT environment is bounded by Eq. (6), which is negligible in λ.*

Proof. By Proposition 5 the simulator and all reductions are straight-line and black-box; by §D.2.2 each assumption holds in the QROM; the simulator's interfaces are correct by Lemmas 8 and 9, whose only loss is the event of Lemma 7; the bundle steps are Lemmas 10 and 11; and §D.2.8 bounds each remaining gap. Summation yields Eq. (6). □

D.2.11 The query budget and the security of a whole execution

Eq. (6) is negligible for every polynomial $Q_{\mathcal{G}_{\mathsf{RO}}}$, but a concrete statement must charge each event class. Our bar is *attack cost* $> 2^{128}$: for $T \cdot \Theta(Q^a_{\mathcal{G}_{\mathsf{RO}}}/2^n)$, the query count at advantage 1 has

$$\log_2 Q_{\mathcal{G}_{\mathsf{RO}}} = (n - \log_2 T)/a.$$

Thus execution charging subtracts $\log_2 T/a$ from the primitive level n/a: one half of the union log for Grover ($a = 2$) [43], one third for BHT ($a = 3$) [18], twice it for reprogramming ($a = \frac{1}{2}$) [41], and all of it for puncturing ($a = 1$) [6].

For puncturing, $\Pi_{\mathsf{p}} = 3E$ covers the two label families keyed by η — the `"esc"` noise labels and the `"coef"` coefficients — and the commitment point. Its $4\Pi_{\mathsf{p}} Q_{\mathcal{G}_{\mathsf{RO}}}\,2^{-\lambda/2}$ term delivers 168.4 bits at $E \le 2^{20}$ and $\lambda = 384$. Reprogramming reaches advantage 1 at $Q_{\mathcal{G}_{\mathsf{RO}}} = 2^\zeta/(2.25R^2)$ [41]; the wrap and epoch-secret classes at $\zeta = 256, N_h$ deliver 210 and 466 bits, respectively. This permits the escrow's 256-bit message capacity while puncturing fixes λ. It is η that is punctured, and this term is what sets the security parameter: it needs $\lambda \ge 2(128 + 2 + \log_2 \Pi_{\mathsf{p}})$, about 303 bits at $E \le 2^{20}$ epochs, and 384 is the smallest standard length above it. At $\lambda = 256$ the same term would deliver 104 bits, below the target rather than at it.

The committee ceiling. The escrow-PKE row is the one entry whose event class grows faster than linearly in a parameter a deployment chooses. Its $2^N NE$ class charges $N + \log_2 N + \log_2 E$ bits against the 164-bit primitive level, so clearing the 128-bit bar requires $N + \log_2 N + \log_2 E \le 36$. At $E \le 2^{20}$ that is $N \le 12$ (35.6 bits at N=12, 36.7 at N=13); the measured N=5 charges 27.3 and leaves the 137 bits of Table 8. The two trade in the direction a deployment may not expect: the committee ceiling *rises* as the epoch budget falls, to $N \le 16$ at $E \le 2^{16}$, and falls as it grows, to $N \le 8$ at $E \le 2^{24}$. A larger committee needs a stronger escrow parameter set rather than a scheduling change. The 2^N is the subset guess of §D.2.8 ($H_{2.5} \to H_3$): a per-slot hybrid would charge N, but the honest set is not determined in advance under adaptive auditor corruption, and we do not know a reduction that avoids the guess. This is why Theorem 1 carries $N = O(1)$.

The remaining event logs are $\log_2(En_\pi) = 21$ and $\log_2(E(1+n_{\mathsf{tie}})) = 21$: both are constants times E, since Π_{A}'s bundle does not grow with the group. The prototype fixes the FRI query count from a conservative form of that charge, $nq \cdot lb \ge 2(128+24+4)$, i.e. $nq = 156$ at $lb = 2$; against Π_{A}'s own 21-bit event class this leaves 135 quantum bits after the knowledge-soundness charge (145.5 under the $q^2/2^{nq\cdot lb}$ accounting of Table 8); grinding is zero, being Grover-halved like the query bits. Poseidon2 binding delivers 157.8 bits at $\ell_{\mathsf{pub}} = 16$ and guess-and-invert hiding 236.7 at $(W, \ell_{\mathsf{pub}}) = (32, 16)$. The 14-lane MMCS digest delivers 137.2 BHT bits after the union and is the weakest row; a Grover search over the 11-lane hiding-MMCS leaf salt delivers 159.5. Fixed-advantage terms retain 137 bits for escrow PKE and 278 for FRI hiding, and CGKA path secrecy is required at 136 bits. Every row therefore clears the 128-bit bar at the configuration the measurements of §8 were taken at, which is the configuration Table 8 describes.

Table 8: Post-quantum levels at $\lambda = 384$, $N_h = 512$, $W = 32$ truncated to $\ell_{\mathsf{pub}} = 16$ lanes, $nq = 156$ FRI queries at log-blowup 2, 11-lane salt. "Exec." charges the event class over $E \le 2^{20}$ epochs at Π_{A}'s bundle ($n_\pi = 2$); [§] fixed advantage, [¶] attack cost. "St." is proof status: R proven reduction, H hardness estimate, Q QROM search/reprogramming, C ethSTARK-conjectural [74].

Primitive / role	Event	Prim.	Exec.	St.
Escrow PKE, Module-LWE CPA	$2^N NE$	164	137[§]	H
ML-KEM-768 wire, ML-DSA-65 auth.	E	164	144[§]	H
SHAKE256, PRF F / $\mathcal{G}_{\mathsf{RO}}$	Π_{p}	192	168.4[¶]	Q
HKDF/HMAC-SHA-512, schedule	Π_{p}	256	232.4[¶]	Q
Poseidon2 ℓ_{pub}-lane digests	$E(1+n_{\mathsf{tie}})$	165	158[¶]	Q
MMCS `DIGEST_ELEMS`= 14	En_π	144	**137**[¶]	Q
Duplex challenger, transcript	En_π	165	158[¶]	Q
Hiding FRI commitment, $\varepsilon_{\mathsf{ZK}}$	En_π	299	278[§]	R
Hiding-MMCS leaf salt, ZK	En_π	170	159[¶]	Q
FRI-STARK, degree-10, $\mathcal{F}_{\mathsf{NIZK}}$	En_π	156	145[¶]	C

The digest row is collision resistance under conservative BHT; the same digests give 237 bits against guess-and-invert and 246 against direct search. Union bounds are proven (§D.2.11); "St." grades the primitive-level entry. Two entries dominate the residual risk: the C row, and the CGKA hypothesis, required at 136 bits and absent from the table.

D.3 Game-Based Security Properties

UC realization is the primary result, but the auditability guarantee is best also stated as standalone games, since the integrity properties among them rest on assumptions a UC statement leaves implicit. Privacy follows directly from Theorem 1, read off $\mathcal{F}_{\mathsf{aud}}$ and transferred by UC indistinguishability. Escrow soundness is an argument-of-knowledge statement, proved by direct reduction (Appendix D.3.3) to $\varepsilon_{\mathsf{ks}}$ and $\varepsilon_{\mathsf{bind}}$ through Lemma 10. It is an admission-time statement: a commit message whose escrow is malformed fails VerifyCommit, and one whose escrow opens to something other than the group's epoch secret is refused by the members' own c_{root} check.

D.3.1 Shared game setup

We give the game-based formulations of the two security properties: privacy and escrow soundness. Both are parameterized by λ, the group parameters $(N_{\max}, t)$ and the number of epochs E, in Bellare–Rogaway notation with $\mathcal{A}$ a stateful QPT algorithm holding quantum access to $\mathcal{G}_{\mathsf{RO}}$ and classical access to the oracles below, against a stateful challenger Ch.

Shared oracle set Every game gives $\mathcal{A}$ the standard CGKA oracles O_{create}, O_{prop}, O_{commit} and O_{corrupt} of [5]. O_{create} returns only the public $\mathcal{M}_0$, since returning the creator's state would hand $\mathcal{A}$ the genesis secret; O_{corrupt} reveals a party's complete state — group key state, all decryption keys, all coins used since initialization — and for an auditor reveals $\mathsf{ask}_j = sk_j^{\mathbb{A}}$. Two oracles are new. $O_{\mathsf{audit}}(G, e, T)$ runs $\mathsf{Audit}(T, \mathcal{L}, e)$ for $|T| \ge t$ and returns its output with no honesty condition on T, since real recovery has none; this costs confidentiality of epoch e, which is why (F5) excludes it on the challenge. $O_{\mathsf{dec}}(G, e, i, k, \mathsf{ct})$ returns the plaintext of the *submitted* ct under the key of the member at position k of copath node i's resolution at epoch e, accepting any ct including ones $\mathcal{A}$ crafts, which makes the model chosen-ciphertext; clause (F3) excludes the submitted ciphertext, not the position.

Freshness predicate $\mathsf{Fresh}(e^*, \mathcal{A})$ Corruption is *adaptive* — $\mathcal{A}$ may call O_{corrupt} at any point, on group members or auditors — and the predicate is evaluated over the whole transcript. A challenge epoch e^* is *fresh* if all of the following hold:

(F1) The committer of e^* is honest at the time of the commit.

(F2) No call to O_{corrupt}, *at any time*, has revealed the group key state of a member from which k_{e^*} is derivable; equivalently, $\mathcal{A}$ never adaptively corrupts a member while it holds an unrotated secret on the path to k_{e^*}. This is the forward-secrecy/PCS window of the underlying CGKA, not a static-corruption restriction.

(F3) O_{dec} has not been called on any ciphertext ct appearing in the logged $\mathcal{M}_{e^*}$, nor on any ct whose plaintext determines $\mathsf{ps}_{m_{e^*}}$ (the commit secret). The exclusion is by ciphertext, so $\mathcal{A}$ retains full chosen-ciphertext access to every other ciphertext of epoch e^* and to all epochs.

(F4) Fewer than t of epoch e^*'s auditors are corrupted *at any point in the game*, before *or* after the challenge.

(F5) O_{audit} was never called on e^* itself: an audit of e^* hands k_{e^*} and the escrowed secret to the calling committee by design, so no confidentiality claim about e^* survives it. No *other* epoch is excluded, an audit reading one epoch's escrow ciphertext and nothing else. This clause is not implied by (F4): a subset T containing one corrupt auditor still has fewer than t corrupt members, yet the audit output reaches the adversary.

Fresh is evaluated on the *completed* transcript, after $\mathcal{A}$ halts; checking it at declaration time would let $\mathcal{A}$ declare a fresh e^*, take the challenge and only then corrupt t of its auditors. Clauses (F1)–(F5) are the safety predicate $\mathsf{safe}_{\mathsf{ESC}}$ under which the theorem of §6 is stated.

D.3.2 The two games

Each game begins with Ch running $\mathsf{AuditSetup}(1^\lambda, N, t)$ and handing $(\mathsf{mpk}, \mathsf{pp})$ to $\mathcal{A}$, which then queries the oracles above adaptively and may act as the committer in any epoch.

$\mathsf{Game}_{\mathsf{priv}}$ **(privacy with the auditor exception).** $\mathcal{A}$ declares a challenge epoch e^*; Ch samples $b \xleftarrow{\$} \{0,1\}$ and returns the real k_{e^*} if $b = 1$ and $k^* \xleftarrow{\$} \{0,1\}^{N_h}$ otherwise; $\mathcal{A}$ continues to query every oracle without restriction and outputs b'. It wins if $b' = b$ and $\mathsf{Fresh}(e^*, \mathcal{A}) = 1$ on the completed transcript. $\mathbf{Adv}^{\mathsf{priv}}_{\Pi}(\lambda) = |\Pr[\mathcal{A}\text{ wins}] - 1/2|$, bounded in Corollary 12.

$\mathsf{Game}_{\mathsf{es}}$ **(escrow soundness).** $\mathcal{A}$ outputs a target epoch e^* and a commit message $\mathcal{M}^*_{e^*}$ such that (a) $\mathsf{VerifyCommit}(\mathsf{pp}, \mathsf{mpk}, \mathcal{L}, \mathcal{M}^*_{e^*}) = 1$ and (b) some honest member P^* accepted e^* with $\mathsf{Key}(gks_{P^*}) = k_{P^*}$, the *member-derived* key. Ch selects any uncorrupted $T \subseteq [N]$ with $|T| \geq t$ and runs $\mathsf{Audit}(T, \mathcal{L}, e^*)$, and $\mathcal{A}$ loses if no such T exists; it wins if the *auditor-reconstructed* key k' in that output satisfies $k' \neq k_{P^*}$. Three keys occur in this game and are kept apart throughout: k_{P^*} (member-derived), k' (auditor-reconstructed) and, in the proof, $k^\dagger$ (the key implied by the extracted witness). $\mathbf{Adv}^{\mathsf{es}}_{\Pi}(\lambda) = \Pr[\mathcal{A}\text{ wins}]$, bounded in Theorem 13. Condition (b) is load-bearing rather than cosmetic: what ties the escrowed value to the group is the member's own c_{root} check, so the game quantifies over epochs a member accepted.

In both games the challenger runs VerifyCommit on the statement $x^* \leftarrow \mathsf{Stmt}(\mathsf{pp}, \mathsf{mpk}, \mathcal{L}, \mathcal{M}^*)$ that it *recomputes* from the wire, by the same deterministic procedure ValidateCommit uses (§B.2.1), never on a statement $\mathcal{A}$ supplies. Against a supplied statement the check would be vacuous: $\mathcal{A}$ could post an honestly generated $\mathcal{M}^*$ alongside an unrelated x^*, or claim a committee of its own keys and a threshold $t = 1$. This is why ValidateCommit recomputes x from the log and why escrow soundness is a statement about the group rather than about the committer's claims.

D.3.3 Corollaries and their reductions

Privacy follows from the UC theorem; escrow soundness is an argument of knowledge, proved by *direct reduction* to the named assumptions. Throughout, an "accepted commit message" is a $\mathcal{M}^*$ with $\mathsf{VerifyCommit} = 1$, hence with $\mathsf{Vrfy}(x^*, \pi^*) = 1$, and Ext is the straight-line extractor of §A.3.3.

Corollary 12 ($\mathsf{Game}_{\mathsf{priv}}$)**.** *For the game* $\mathsf{Game}_{\mathsf{priv}}$ *of Appendix D.3.2 (adversary corrupting any group members and at most $t-1$ auditor committee members, challenge epoch satisfying* Fresh*),* $\mathbf{Adv}^{\mathsf{priv}}_{\Pi}(\lambda) \leq \mathbf{Adv}^{\mathsf{UC}}_{\Pi}(\lambda) = \mathsf{negl}(\lambda)$*. FS and PCS hold as in [5].*

Proof. In $\mathcal{F}_{\mathsf{aud}}$ an honest epoch's escrow target s_e and epoch key k_e are sampled independently of the transcript, so k_e is uniform in $\mathcal{A}$'s view unless the functionality releases it, and there are exactly three release routes. PROCESS returns k_e to a member of Acc_e, closed by (F2), which forbids corrupting a member holding an unrotated secret on the path to k_{e^*}. $\mathsf{Audit}(T, e)$ with $|T| \geq t$ returns (s_e, k_e) to every $A_j \in T$ and to $\mathcal{S}$, closed by (F5), which excludes an audit of e^*; the clause excludes no other epoch, because AUDIT reads only ν_e. (F4) alone does *not* close this route, since a T containing one corrupt auditor has fewer than t corrupt members and is (F4)-fresh while the output still reaches $\mathcal{A}$. (LEAK, e) requires $\geq t$ of epoch e's auditors to be corrupt, closed by (F4). A distinguisher whose challenge epoch satisfies Fresh on the completed transcript is therefore an environment separating REAL from IDEAL on a $\mathsf{safe}_{\mathsf{ESC}}$-respecting execution, which Eq. (6) bounds. □

Theorem 13 (Escrow soundness, $\mathsf{Game}_{\mathsf{es}}$)**.** *For the game* $\mathsf{Game}_{\mathsf{es}}$ *of Appendix D.3.2,* $\mathbf{Adv}^{\mathsf{es}}_{\Pi}(\lambda) \leq n_\pi \varepsilon_{\mathsf{ks}} + (1+n_{\mathsf{tie}})\varepsilon_{\mathsf{bind}} + t\,\delta_{\mathsf{esc}} = \mathsf{negl}(\lambda)$*, in the bundle parameters of Lemma 10 (n_π sub-proofs, n_{tie} published ties; the additional $\varepsilon_{\mathsf{bind}}$ is the member-versus-extraction collision, the same summand that appears outside n_{tie} in Eq. (6)): no QPT adversary produces an accepted commit message for which t auditors running* Audit *recover a key different from the one honest members hold.*

The term $t\,\delta_{\mathsf{esc}}$ is the escrow-PKE decryption-failure probability over the opened slots, which clause (ii) does not remove: proving a ciphertext well formed is not proving that decryption returns the encrypted message, and a failure on any opened slot yields a wrong share and hence a wrong key, exactly the winning condition. It is therefore not a proof-system error and does not vanish with better proof parameters. One audit opens t slots in one epoch, so no factor in the number of epochs appears, recovery being per-epoch independent. The range-checked CBD_2 noise of §A.3.1 bounds $\delta_{\mathsf{esc}} \le 2^{-164}$ per ciphertext, hence $t\,\delta_{\mathsf{esc}} \le 2^{-162}$ per audit, at the parameters of Appendix E.1. The guarantee is established at *admission*: a commit message whose escrow is malformed fails VerifyCommit, and one whose escrow opens to a value other than the group's epoch secret fails the members' c_{root} check.

Proof. Let $\mathcal{A}$ win: it produces an accepted $\mathcal{M}^*_{e^*}$, an honest member holds key k_{P^*}, yet $\mathsf{Audit}(T,\mathcal{L},e^*)$ with $|T|=t$ uncorrupted returns $k' \neq k_{P^*}$. Run Ext on π^* (Lemma 10 for the bundle); except with probability $n_\pi\,\varepsilon_{\mathsf{ks}} + n_{\mathsf{tie}}\,\varepsilon_{\mathsf{bind}}$ it yields a witness $w^* \in \mathcal{R}_{\mathsf{A}}$, so by Eq. (2) the $\{\sigma^*_j\}$ are a degree-$(t-1)$ Shamir sharing over $\mathbb{F}_q$ of a value es^* (clause i) wrapped in the logged escrow ciphertext (clause ii) with $c_{\mathsf{root}} = H_{\mathsf{P2}}(\texttt{"crot"}\|G\|e\|es^*)$ (clause iii).

An honest member accepts e^* only if c_{root} matches the commitment to the epoch secret it derived; by binding, es^* is that epoch secret except with probability $\varepsilon_{\mathsf{bind}}$. Hence the escrow ciphertext Shamir-shares the members' actual epoch secret. By Shamir reconstruction on the proven-consistent shares, Audit recovers es^*, checks it against the logged c_{root} and derives $k^\dagger = \mathsf{KeySchedule}(es^*)$, which is the members' own key schedule on the members' own root secret, so $k' = k^\dagger = k_{P^*}$, contradicting $k' \neq k_{P^*}$. The audit emits no fraud verdict, a mis-escrow having been refused at admission. The remaining route to a win is a decryption failure on one of the t opened slots, which yields a wrong share; summing the failure modes gives the bound. □

The scope of VerifyCommit. Escrow soundness is a statement about an epoch an honest member accepted, and the reason is worth stating on its own, because it fixes what VerifyCommit alone is good for. VerifyCommit is complete on honestly generated commit messages, and on any message it accepts, Lemma 10 extracts a witness for $\mathcal{R}_{\mathsf{A}}$ against the statement Stmt recomputed from the log, except with probability $n_\pi\,\varepsilon_{\mathsf{ks}} + n_{\mathsf{tie}}\,\varepsilon_{\mathsf{bind}}$. That witness says the logged $C_{\mathsf{esc}\,e}$ is a well-formed t-of-N threshold sharing, under the committee mpk the log fixes, of *some* preimage of the logged c_{root}, with every component of the ciphertext well formed down to its noise ranges.

It does *not* say the preimage is the epoch secret the group derived. No clause of $\mathcal{R}_{\mathsf{A}}$ evaluates the MLS key schedule, and c_{root} is a commitment the committer chose, so that step is established by the members, each against the secret it derived for itself. $\mathcal{F}_{\mathsf{aud}}$ is accordingly realized by a protocol whose admission check is one the group performs, not one a bystander performs, and the paper claims no property of the form "a third party establishes auditability from the log alone". The boundary has a concrete consequence, stated in §1 and repeated here as the scope of condition (b): if *every* member of the group is corrupt, all of them may accept an epoch whose escrow opens to a value other than the one they use, and neither VerifyCommit nor $\mathcal{F}_{\mathsf{Log}}$ detects it, since no check made from the log alone can recompute a secret it does not hold. Auditability is a guarantee for groups retaining an honest member, and no design that keeps the key schedule outside the circuit can give more. VerifyCommit is a stateless filter — it is what lets $\mathcal{F}_{\mathsf{Log}}$ refuse a malformed escrow without holding group state — and its guarantee is the one above.

E Escrow PKE and Aggregation

E.1 The escrow PKE

Parameters The scheme of §A.3.1 runs at the ML-KEM-768 parameter set: ring degree 256, modulus $q_R = 3329$, module rank $\kappa = 3$, and CBD_2 noise on every coefficient of $(\mathbf{r},\mathbf{e}_1,e_2)$. Messages are the 256-bit escrow seeds μ_j, encoded one bit per coefficient of the message polynomial in the top half of $\mathbb{Z}_{q_R}$, 256 being both the capacity of one such ciphertext and, by §D.2.2, what the reprogramming bound needs. Ciphertext compression at $(d_u,d_v) = (10,4)$ is *enabled* in the measured configuration, which is what makes a per-auditor ciphertext 1 088 B; disabling it costs 448 B per auditor and changes no security statement.

Decryption failure Decryption recovers $v - \mathbf{s}^\top\mathbf{u} = \mathsf{Encode}(\mu) + e_2 + \mathbf{e}^\top\mathbf{r} - \mathbf{s}^\top\mathbf{e}_1 + \Delta_{\mathsf{cmp}}$, and a coefficient decodes wrongly exactly when that error exceeds $q_R/4$; at these parameters the quantities are the ML-KEM-768 ones, giving $\delta_{\mathsf{esc}} \le 2^{-164}$ per ciphertext and the $t\,\delta_{\mathsf{esc}} \le 2^{-162}$ term of Theorem 13. The range checks of clause (ii) bound every noise coefficient, which is what lets an honest-encryption failure bound apply to a logged ciphertext an adversary produced.

IND-CPA

Lemma 14 (IND-CPA of the escrow PKE)**.** *Any QPT adversary distinguishing* $\mathsf{Enc}(pk^{\mathbb{A}},\mu^0)$ *from* $\mathsf{Enc}(pk^{\mathbb{A}},\mu^1)$ *for chosen* μ^0,μ^1 *has advantage at most* $2\,\varepsilon_{\mathsf{MLWE}}(\kappa,q_R,\mathsf{CBD}_2)$.

Proof. Two single-run hops. Replace $pk^{\mathbb{A}} = (\mathbf{A},\mathbf{A}\mathbf{s}+\mathbf{e})$ by $(\mathbf{A},\mathbf{b})$ with $\mathbf{b}$ uniform, against which a distinguisher is a Module-LWE distinguisher in κ samples; then replace $(\mathbf{u}, v - \mathsf{Encode}(\mu))$ by a uniform pair, against which a distinguisher is one in $\kappa+1$ samples under the transposed matrix.

In the final game the ciphertext is independent of μ. Neither hop rewinds, clones or measures the adversary, so the reduction is straight-line, and the QROM plays no role since no hash is applied to the message. □

Verifiable encryption IND-CPA alone does not stop an adversary from mauling a logged escrow ciphertext; the proof supplies that. The property below is about the *logged object*, not about the encryption scheme: it does not make a CPA-secure PKE non-malleable in the standard sense, and we do not claim that. We call it *proof-bound ciphertext integrity*.

Lemma 15 (Proof-bound integrity of a logged escrow ciphertext)**.** *Let $\mathcal{M}_e$ be admitted, so* Vrfy *accepts its proof. Then for every j there is a witness, extractable by* Ext*, containing μ_j, σ_j and noise within the range-checked bounds such that $\mathsf{c}_j^{\mathsf{esc}} = \mathsf{Enc}(pk_j^{\mathbb{A}}, \mu_j; \cdot)$ and $\mathsf{e}_j = \sigma_j \oplus H_{\mathsf{P2}}(\text{"}wrap\text{"} \| K_j)$, except with the probability of Lemma 10. Consequently an adversary that modifies any component of a logged $C_{\mathsf{esc}e}$ must produce a fresh accepting proof for the modified statement, since $C_{\mathsf{esc}e}$ is a public input of x and x is recomputed from the wire by* Stmt*.*

Proof. Immediate from clause (ii) and straight-line knowledge soundness, together with the fact that Stmt takes $C_{\mathsf{esc}e}$ from $\mathcal{L}$ rather than from the committer (§B.2.1). □

This is the sense in which the construction uses verifiable encryption [20, 55]: the guarantee an FO transform would give for the primitive is given here for the logged object, which is the only one the protocol relies on.

E.2 Recursive aggregation of the bundle

A Π_{A} bundle is 2 sub-proofs and 15.31 MB. Aggregation collapses it to a single proof whose size is independent of the relation, and the security statement is a corollary of Lemma 10.

The aggregation relation Let Π_{in} be the bundle of §D.2.9: n_π sub-proofs $\{\pi_1, \ldots, \pi_{n_\pi}\}$ over statements $\{x_1, \ldots, x_{n_\pi}\}$ derived from one public x, tied by n_{tie} published digests $\{\delta_1, \ldots, \delta_{n_{\mathsf{tie}}}\}$. Define

$$\mathcal{R}_{\mathsf{agg}}: \boxed{\begin{array}{l}\text{(i) } \mathsf{Vrfy}(x_i, \pi_i) = 1 \text{ for every } i \in [n_\pi], \text{ where each } x_i \text{ is derived from the public } x \text{ by the fixed,} \\ \text{deterministic splitter of } \mathsf{Stmt} \\ \text{(ii) the } n_{\mathsf{tie}} \text{ tie digests carried by the } \{x_i\} \text{ agree pairwise as producer/consumer}\end{array}} \tag{7}$$

with public input x and witness $(\{\pi_i\}, \{\delta_l\})$. The aggregated proof is $\pi_{\mathsf{agg}} \leftarrow \mathsf{Prove}(x, (\{\pi_i\}, \{\delta_l\}))$ for $\mathcal{R}_{\mathsf{agg}}$, and VerifyCommit verifies π_{agg} alone.

Corollary 16 (Aggregation preserves both properties)**.** *If the outer argument is straight-line knowledge-sound with error $\varepsilon_{\mathsf{ks}}^{\mathsf{out}}$ and zero-knowledge with error $\varepsilon_{\mathsf{ZK}}^{\mathsf{out}}$, then from an accepting π_{agg} a witness for $\mathcal{R}_{\mathbb{A}}$ is extractable except with probability $\varepsilon_{\mathsf{ks}}^{\mathsf{out}} + n_\pi \varepsilon_{\mathsf{ks}} + n_{\mathsf{tie}} \varepsilon_{\mathsf{bind}}$, and the aggregated transcript is simulatable except with probability $\varepsilon_{\mathsf{ZK}}^{\mathsf{out}}$.*

Proof. Extraction is two-stage: the outer extractor yields $(\{\pi_i\}, \{\delta_l\})$ satisfying $\mathcal{R}_{\mathsf{agg}}$, at $\varepsilon_{\mathsf{ks}}^{\mathsf{out}}$; applying Lemma 10 to that bundle yields the single witness, at $n_\pi \varepsilon_{\mathsf{ks}} + n_{\mathsf{tie}} \varepsilon_{\mathsf{bind}}$. Both stages are straight-line, so the composition is. For zero knowledge, the inner proofs and the tie digests are now *witness* rather than transcript: the aggregated transcript is (x, π_{agg}) and nothing else, so the $n_{\mathsf{tie}}^{\mathsf{u}}$ term of Lemma 11 cannot arise at all and outer zero knowledge is the whole statement. For Π_{A} that term is already zero (§D.2.9), so what aggregation buys here is size rather than a strengthened hiding claim. □