QuantLayer: privacy-preserving verification with post-quantum authorization

A technical description of the protocol: its constructions and assumptions, measured costs, security analysis, and what is implemented, experimental or planned.

Version	1.0
Date	17 November 2026

Abstract

QuantLayer is a verification layer for on-chain finance. It lets a holder prove a statement about an identity or credential (membership in a set, ownership of a credential, a balance above a threshold, eligibility under a policy) without disclosing the underlying data, and it lets a relying party require that an action be authorized by post-quantum and hybrid classical-plus-post-quantum signatures.

The system combines zero-knowledge circuits written in Noir and proved with UltraHonk over BN254; NIST post-quantum algorithms (ML-DSA, SLH-DSA, ML-KEM) implemented in Rust; a deterministic, fail-closed authorization engine; and seven Solidity contracts deployed on Arbitrum Sepolia. This paper specifies the constructions, states their assumptions, reports measured costs, and separates what is implemented from what is experimental.

We are explicit about one limit throughout: QuantLayer's signatures and key establishment are post-quantum, but its production proofs (pairing-based, over BN254) and the EVM accounts that use them are not. A hash-based STARK backend exists as an experimental prototype. No independent security audit has yet been completed.

1 Introduction

1.1 Problem statement

On-chain financial applications increasingly need to know facts about the people and institutions that interact with them: that an account belongs to a verified person, that a holder is not on a restricted list, that a balance exceeds a threshold. Today these facts are usually established by disclosure. The holder reveals the document, the verifier keeps a copy, and every check multiplies the number of places sensitive data is stored.

Two further problems compound this. First, the signatures that authorize actions (ECDSA over secp256k1, Ed25519) rest on the hardness of discrete logarithms, which Shor's algorithm solves in polynomial time on a large fault-tolerant quantum computer [4]. Signed data and recorded traffic captured today can be forged or decrypted later (“harvest now, decrypt later”). Second, verifying a statement on-chain must not let it be replayed on another chain, contract, application or action.

1.2 Objectives and contributions

1. **Prove facts, not data.** Six circuits prove statements about committed identities and credentials; only commitments, roots, nullifiers and proofs leave the holder's device.
2. **Post-quantum authorization.** ML-DSA, SLH-DSA and ML-KEM (FIPS 203–205 [1, 2, 3]) in Rust, with hybrid signatures that require both a classical and a post-quantum signature over one domain-separated input.
3. **Deterministic, fail-closed decisions.** An authorization engine whose checks run in a fixed order, whose output carries a full trace, and which never yields an allow when a dependency fails.
4. **Binding and replay protection on-chain.** Every proof and request is bound to chain, contract, application and action; nonces, proof contexts and nullifiers are spent exactly once.
5. **Honest assurance.** Each on-chain result reports whether it was verified trustlessly or attested by a quorum, and the system states its quantum limits plainly.

1.3 Scope and non-claims

What this paper does not claim

- That the protocol as a whole is quantum-resistant. Production proofs and EVM accounts are quantum-sensitive (Section 9.3).
- That QuantLayer makes transactions private. It proves facts about credentials.
- That any component has been independently audited. None has, yet.
- Any performance figure not measured in the repository. All figures cite their measurement.

2 System architecture

2.1 Trust layers

The architecture is drawn as three nested trust layers. The *private* layer is the holder's device: identity secrets, credential openings and Merkle paths. The *protocol* layer holds commitments, circuits and policies. The *public* layer is the chain and its verifiers. Exactly one artefact crosses from private to public: the proof, with its public inputs.

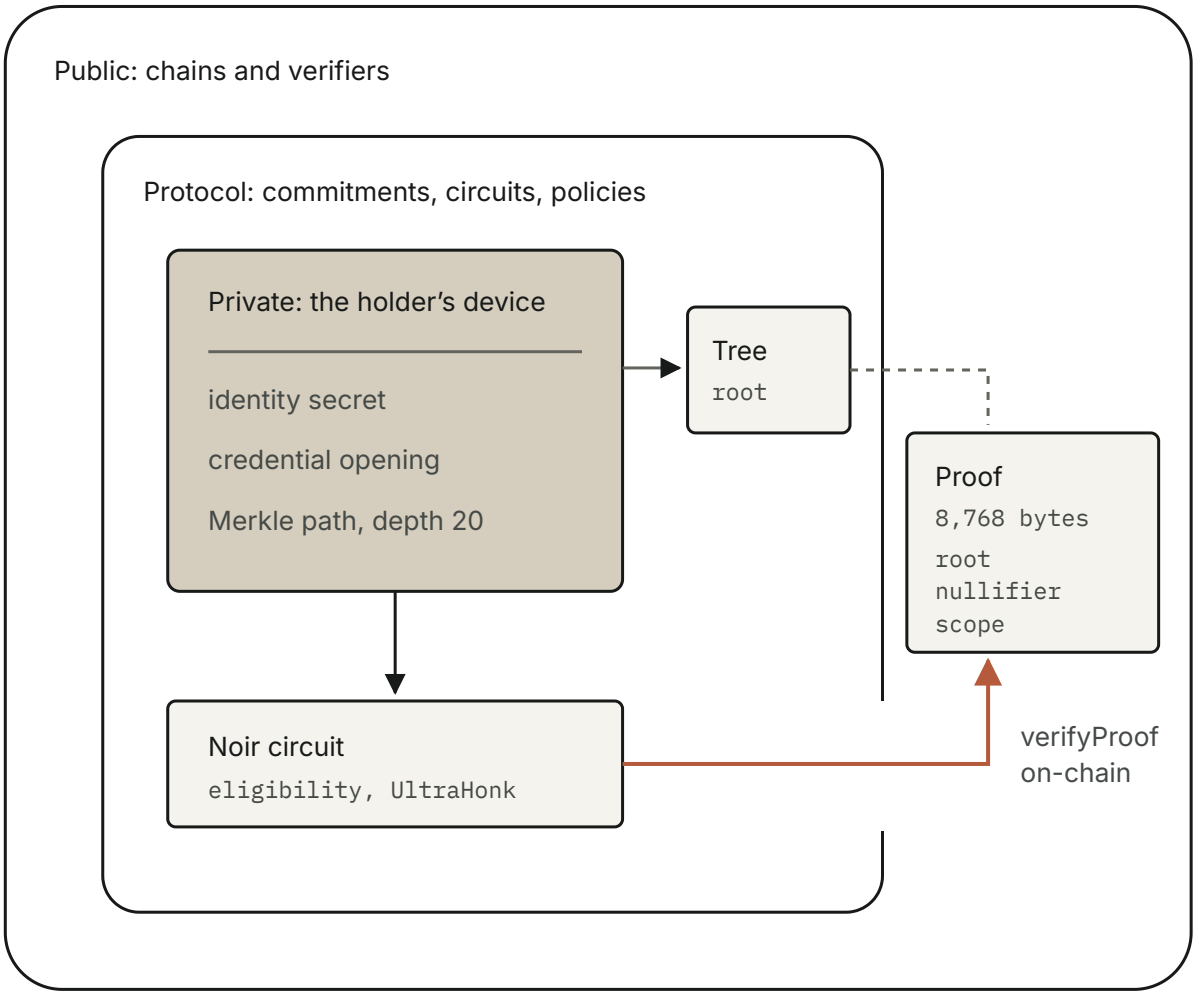


Figure 1. The QuantLayer boundary. The proof (8,768 bytes) is the only value that crosses into public view, where `QuantLayerProofVerifier` checks it on-chain.

2.2 Components and stack

Component	Technology	Responsibility
Protocol kernel	TypeScript packages	canonical encodings, identities, credentials, privacy primitives, authorization engine, network registry
Circuits	Noir 1.0.0, Barretenberg 6.0.0 (UltraHonk)	six statements over Poseidon commitments and depth-20 Merkle trees
Post-quantum module	Rust (libcrux, fips205), Node-API binding	ML-DSA, SLH-DSA, ML-KEM; hybrid signatures; key lifecycle and sealing
Contracts	Solidity 0.8.28, Foundry, OpenZeppelin	registries, proof verification, attestation, authorization
Backend	NestJS on Fastify, PostgreSQL (Prisma), Redis (BullMQ)	API, delegated proving worker, chain indexer, metering
SDK and web	TypeScript SDK, Next.js	developer integration; the application and this site
Experimental	Rust, Plonky3 0.8.0	hash-based STARK/FRI backend, off by default

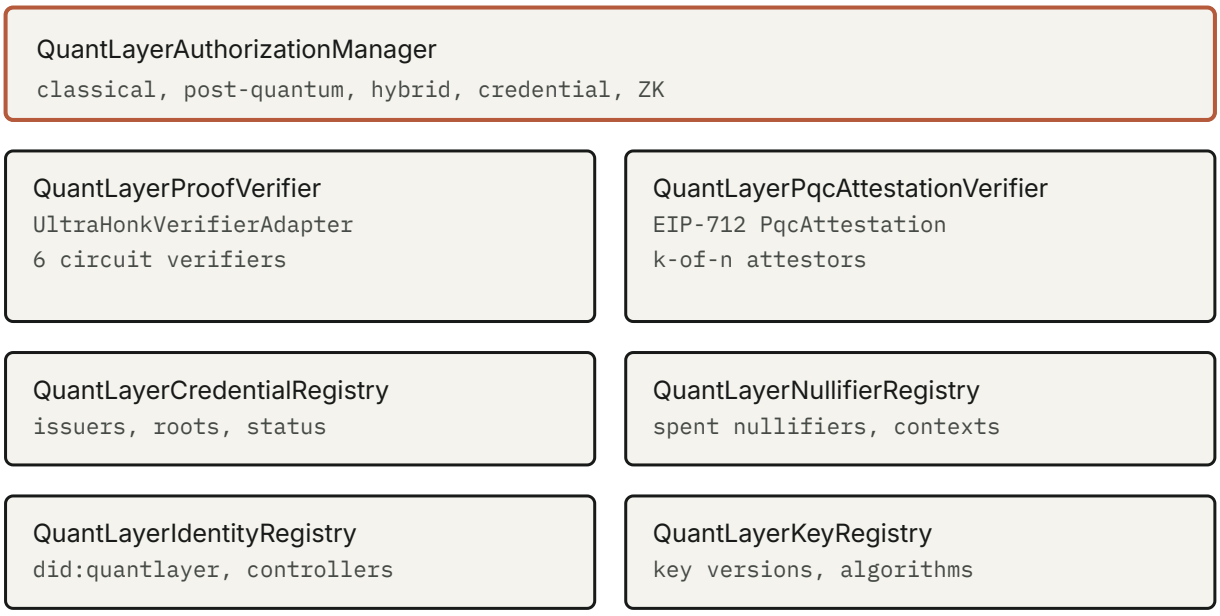


Figure 2. The on-chain contract stack and how its parts depend on each other.

2.3 Data flow

A proof can be computed on the holder's device or delegated to QuantLayer's proving worker. Figure 3 shows the delegated path and the form the witness takes at each hop. The witness crosses the network under TLS, is sealed with ML-KEM-768 and AES-256-GCM on receipt, reaches the queue only as ciphertext, and is never persisted. Client-side proving removes the network hop entirely and is preferred where the device can afford it.

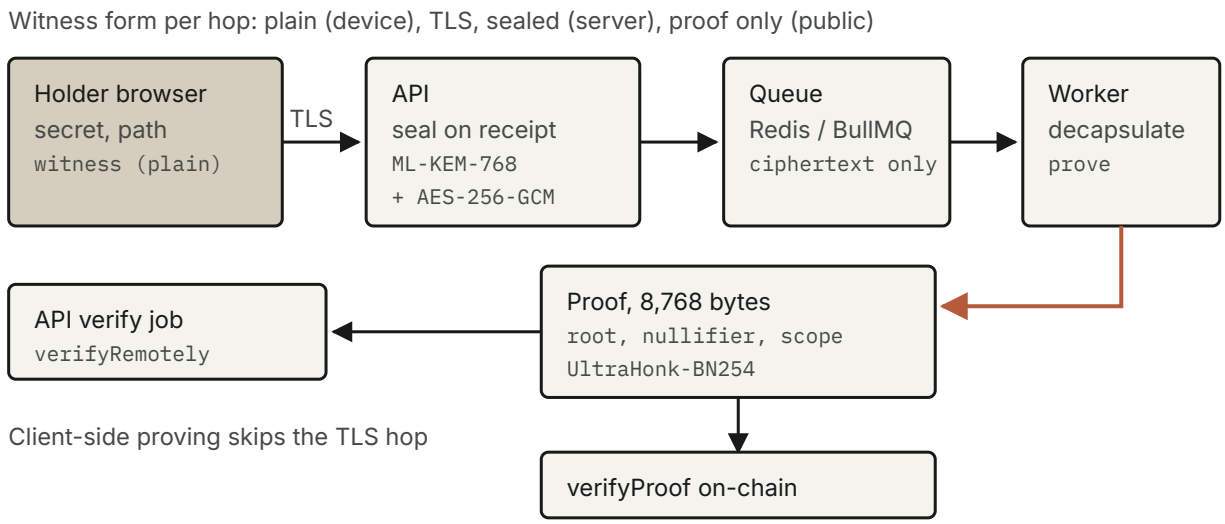


Figure 3. Delegated proving data flow. The TLS hop is the residual exposure to a future quantum adversary recording traffic (Section 9.3).

3 Cryptographic preliminaries

3.1 Notation and fields

All circuit arithmetic is over the scalar field of the BN254 curve, F_p, with

$$p = 0\text{x}30644\text{e}72\text{e}131\text{a}029\text{b}85045\text{b}68181585\text{d}2833\text{e}84879\text{b}9709143\text{e}1\text{f}593\text{f}000(\quad (1)$$

In decimal, `p = 21888242871839275222246405745257275088548364400416034343698204186575808495617`, the order of the BN254 pairing groups (about $2^{253.6}$).

We write $\|$ for byte concatenation, H for the Poseidon permutation-based hash over F_p [7] (the circomlib-compatible instance used by `noir-lang/poseidon` v0.4.0), and `keccak256` for the EVM hash.

3.2 Domain-separated hashing

Every hash over private data carries a domain tag as its first input. A tag is the big-endian integer of a versioned ASCII label of at most 31 bytes:

$$\tau(\ell) = \sum_{i=0}^{|\ell|-1} b_i \cdot 256^{|\ell|-1-i} < 2^{248} < p \quad (2)$$

$$H_D(x_1, \ \dots, x_k) = \text{Poseidon}(\tau(\ell_D), x_1, \ \dots, x_k) \quad (3)$$

ℓ the label, for example `quantlayer.identity.v1`; its bytes are b_i.

D the domain (commitment, identity, nullifier, merkle-node, credential), each with a fixed arity.

Because at most 31 bytes are encoded, every tag is below 2^{248} and therefore a canonical field element; the map from labels to tags is injective. With a fixed arity per domain, two different purposes can never present the same input to H, even when their payloads coincide. Labels carry a version suffix, so changing a construction means a new label rather than silently reusing an old one. The circuit constants are asserted equal to the encoded labels in the circuit library's tests.

3.3 Commitments and identities

$$C = H_{\text{commitment}}(v, r) \tag{4}$$

$$I = H_{\text{identity}}(s, t) \tag{5}$$

v, r a committed value and a uniformly random blinding factor in F_p.

s, t the identity's nullifier secret and trapdoor, two random field elements generated on the holder's device and never published.

I the identity commitment: the leaf a holder registers in a membership set.

Assumptions

Binding of (4) and (5) reduces to collision resistance of Poseidon; hiding holds computationally when r (respectively t) is uniform and Poseidon behaves as a random function on the domain. Both are standard assumptions for algebraic hashes [7]; Poseidon's cryptanalysis is younger than SHA-2's, which we note as a risk (Section 13).

3.4 Merkle membership

Identity commitments are leaves of a binary Merkle tree of depth $d = 20$, so a set holds up to $2^{20} = 1{,}048{,}576$ members. With path siblings σ_j and direction bits β_j:

$$h_0 = I, \quad h_{j+1} = \begin{cases} H_{\text{node}}(h_j, \sigma_j) & \beta_j = 0 \\ H_{\text{node}}(\sigma_j, h_j) & \beta_j = 1, \end{cases} \quad \text{root} = h_{20} \tag{6}$$

Verifying membership costs exactly d = 20 hash evaluations, independent of the set size. Empty subtrees are represented by precomputed zero hashes $z_{j+1} = H_{\text{node}}(z_j, z_j)$, so a sparse tree needs storage proportional to its occupied leaves.

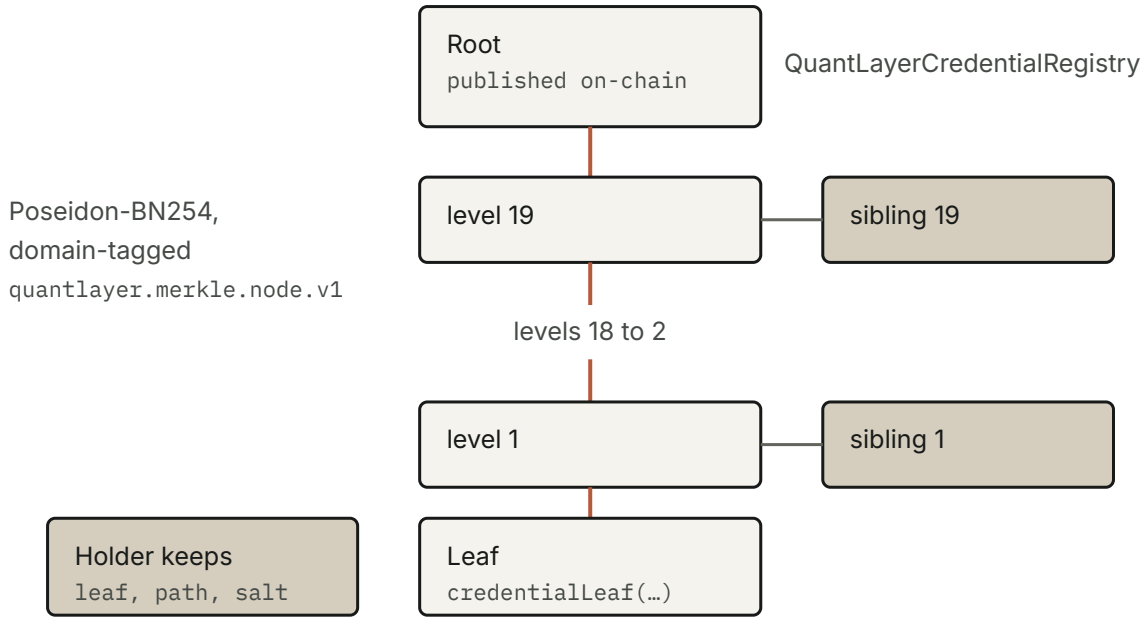


Figure 4. A membership path from a leaf to the published root.

3.5 Nullifiers and scopes

$$\nu = H_{\text{nullifier}}(s, \text{scope}) \tag{7}$$

$$\text{scope} = \text{keccak256}(D_{\text{scope}}, \text{chainId}, \text{verifier}, \text{appId}, \text{action}) \gg 8 \tag{8}$$

D_{scope} `keccak256("QUANTLAYER-ZK-SCOPE-V1")`; the right shift by 8 bits maps the 256-bit digest below p.

The nullifier depends on the nullifier secret and the scope only, not on the trapdoor: the same identity yields the same ν for one (chain, verifier, application, action) and unrelated values for different scopes. Spending ν once per scope stops double claims; unlinkability across scopes rests on the pseudo-randomness of Poseidon keyed by s.

4 Zero-knowledge layer

4.1 Circuit statements

Each circuit proves knowledge of a private witness (an identity, and where relevant a credential and its Merkle path) such that a public statement holds. Every circuit takes the context (Section 4.3) and a root as its first public inputs. The three circuits that enforce one use per scope also take the scope and output the nullifier (7).

Circuit	Public inputs (→ output)	Proves, without revealing the witness
`membership`	`context, root, scope ⟶ nullifier`	the prover's identity commitment (5) is a leaf under `root`; anonymous, once per scope
`credential-ownership`	`context, root, issuer, credential_type`	the prover holds a credential of `credential_type` from `issuer`, committed under `root` and bound to their identity
`credential-validity`	`…, now`	ownership, and the validity window `issued_at ≤ now < expires_at`
`balance-threshold`	`…, now, threshold`	a current balance credential whose hidden amount is at least `threshold`
`nullifier-uniqueness`	`…, scope ⟶ nullifier`	a credential holder's one-time tag for `scope`; the verifier contract rejects reuse
`eligibility`	`…, now, threshold, scope ⟶ nullifier`	a current credential of the required type with attribute ≥ `threshold`, usable once per scope (for example, one claim per sale)

Because the prover chooses public inputs such as `threshold` and `credential_type`, the on-chain policy pins them: a proof that is valid for a lower threshold is rejected by a policy that requires a higher one.

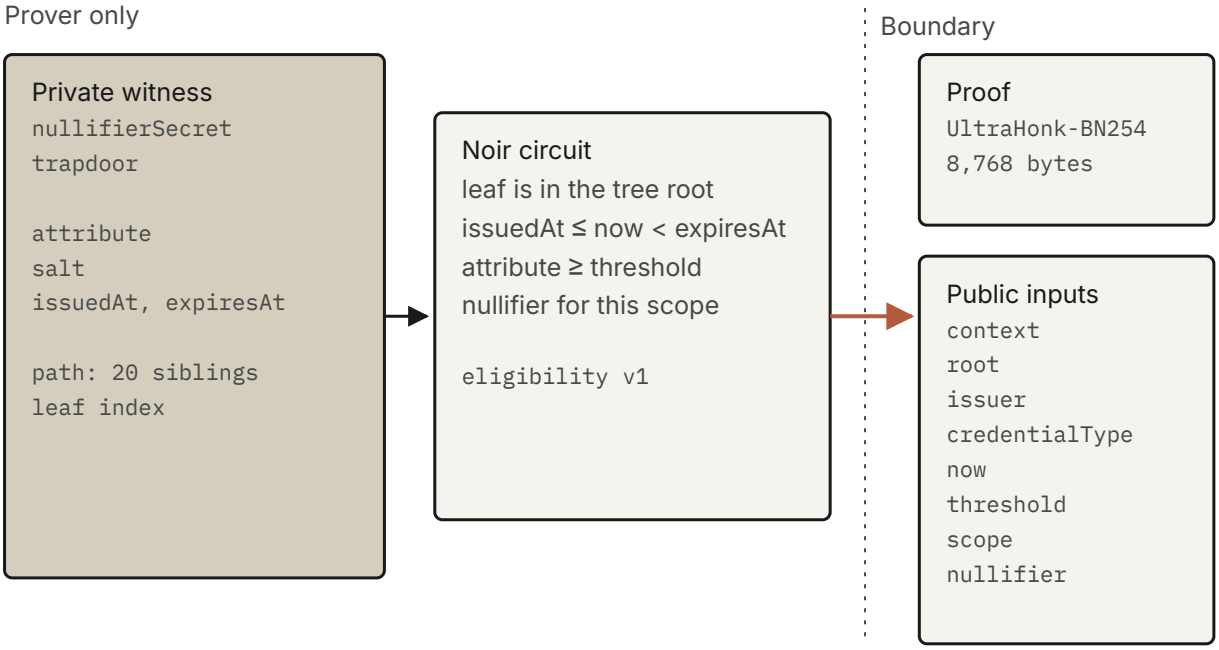


Figure 5. Witness and public inputs separated by the circuit: the witness stays private, only public inputs and the proof are output.

4.2 Proof system

Circuits are written in Noir 1.0.0 and proved with Barretenberg 6.0.0's UltraHonk, a PLONK-family [9] argument with KZG polynomial commitments [8] over BN254. It uses a universal structured reference string (no per-circuit trusted setup), produces constant-size proofs, and verifies in Solidity through BN254 pairing precompiles. Verifier contracts are generated from each circuit's verification key; a reproducible build recompiles all six circuits and checks the artifacts and Solidity verifiers byte for byte.

Soundness and zero knowledge

Knowledge soundness of UltraHonk rests on the algebraic group model and the q-type assumptions behind KZG over BN254, in the random-oracle model for the Fiat–Shamir transform (Keccak transcript). Zero knowledge holds against classical adversaries. Neither property is claimed against quantum adversaries (Section 9.3).

4.3 Context binding

$$\mathrm{ctx} = \mathrm{keccak256}(D_{\mathrm{ctx}}, \mathrm{chainId}, \mathrm{verifier}, \mathrm{consumer}, \mathrm{appId}, \mathrm{circuitId}, \mathrm{nonce}) \gg 8 \quad (9)$$

The context is a public input. `QuantLayerProofVerifier` recomputes (9) from the calling contract (`consumer = msg.sender`) and the current chain, so a proof produced for one chain, deployment, consumer, application or circuit fails everywhere else. Each context is consumed once, and each nullifier once per scope; all writes happen after every check, so a rejected call consumes nothing.

4.4 Measured cost

All six circuits compile to 2^{16} gates, dominated by 22–23 Poseidon hashes. Measured on an Apple M2 (Node 26, 4 prover threads):

Measure	Value
Witness generation	112–117 ms
Prove, native	706–825 ms
Prove, WebAssembly (browser path)	858–931 ms
Verify, native	2.2–2.4 ms
Proof size	8,768 bytes
Peak memory, native prover	672–725 MiB
On-chain `verifyProof` (Cancun, incl. 21,000 base and calldata)	2,641,497–2,673,331 gas
Verifier runtime size	18,032–18,096 bytes

Source: docs/zk/BENCHMARKS.md (median latencies after warm-up).

5 Post-quantum layer

5.1 Algorithms and parameters

QuantLayer Quantum implements the three NIST post-quantum standards: ML-KEM (FIPS 203) [1], ML-DSA (FIPS 204) [2] and SLH-DSA (FIPS 205) [3]. ML-DSA and ML-KEM come from Cryspen's libcrux, whose portable code is produced from hax/F*-verified sources [21]; SLH-DSA comes from the `fips205` crate. ML-DSA-65 is the default signature (NIST category 3).

ML-DSA is a Fiat–Shamir-with-aborts signature over the module lattice $R_q = Z_q\left[X\right] / (X^{256} + 1)$. A public key is (ρ, t_1) with $t = As_1 + s_2$, and verification accepts $(\tilde{c}, z, h)$ when

$$\|z\|_\infty < \gamma_1 - \beta \quad \wedge \quad \tilde{c} = H(\mu \| \mathrm{UseHint}(h, Az - c \cdot t_1 \cdot 2^d)) \tag{10}$$

q 8,380,417; the ring modulus.

A a k × ℓ matrix over R_q expanded from the seed ρ; for ML-DSA-65, (k, ℓ) = (6, 5).

γ_1, β, d for ML-DSA-65: $\gamma_1 = 2^{19}$, $\beta = \tau \cdot \eta = 49 \cdot 4 = 196$, d = 13 dropped bits of t.

μ the message representative, a hash of the public key and the message (with its context string).

Security reduces to Module-LWE and Module-SIS, for which no efficient quantum algorithm is known. ML-KEM-768 is the IND-CCA2 key encapsulation obtained by the Fujisaki–Okamoto transform of a Module-LWE public-key scheme with $q = 3329, k = 3$. SLH-DSA's security rests only on its hash function (SHA-2 here).

Algorithm	NIST cat.	Public key	Signature / ciphertext	Sign or encaps (Rust)	Verify or decaps
ML-DSA-44	2	1,312 B	2,420 B	111.9 µs	30.5 µs
ML-DSA-65	3	1,952 B	3,309 B	155.9 µs	45.6 µs
ML-DSA-87	5	2,592 B	4,627 B	192.2 µs	70.4 µs
SLH-DSA-SHA2-128s	1	32 B	7,856 B	877 ms	741.6 µs
SLH-DSA-SHA2-128f	1	32 B	17,088 B	39 ms	2.19 ms
SLH-DSA-SHA2-256s	5	64 B	29,792 B	1,323 ms	1.63 ms
ML-KEM-512	1	800 B	768 B	6.4 µs	6.6 µs
ML-KEM-768	3	1,184 B	1,088 B	11.0 µs	11.5 µs
ML-KEM-1024	5	1,568 B	1,568 B	15.3 µs	16.3 µs

Source: docs/pqc/BENCHMARKS.md; Apple M2, single thread, medians after warm-up.

5.2 Hybrid signatures

During the transition, a request can require both a classical and a post-quantum signature. Both sign one domain-separated input, so neither component can be stripped or reused under another scheme [13]:

$$m^* = \text{"QUANTLAYER-HYBRID-V1"} \| 0\text{x}00 \| |\text{id}| \| |\text{id}| \| |\text{ctx}| \| |\text{ctx}| \| m \tag{11}$$

$$\text{accept} \iff V_{\text{c}}(\text{pk}_{\text{c}}, m^*, \sigma_{\text{c}}) = 1 \quad \wedge \quad V_{\text{pq}}(\text{pk}_{\text{pq}}, m^*, \sigma_{\text{pq}}) = 1 \tag{12}$$

`id` the scheme identifier, `Ed25519+ML-DSA-65` or `ES256K+ML-DSA-65`.

`ctx` the application context, `quantlayer.authorization/1` for authorizations (at most 255 bytes).

Security argument

Under (12), a forgery for the hybrid scheme yields a forgery for *each* component on the same message m^*. The construction is therefore EUF-CMA secure if either component is, which is the property that matters when one family may fail: a quantum break of the classical half alone, or an unforeseen weakness in the lattice half alone, does not authorize anything. ECDSA halves must be low-S (malleable signatures are rejected).

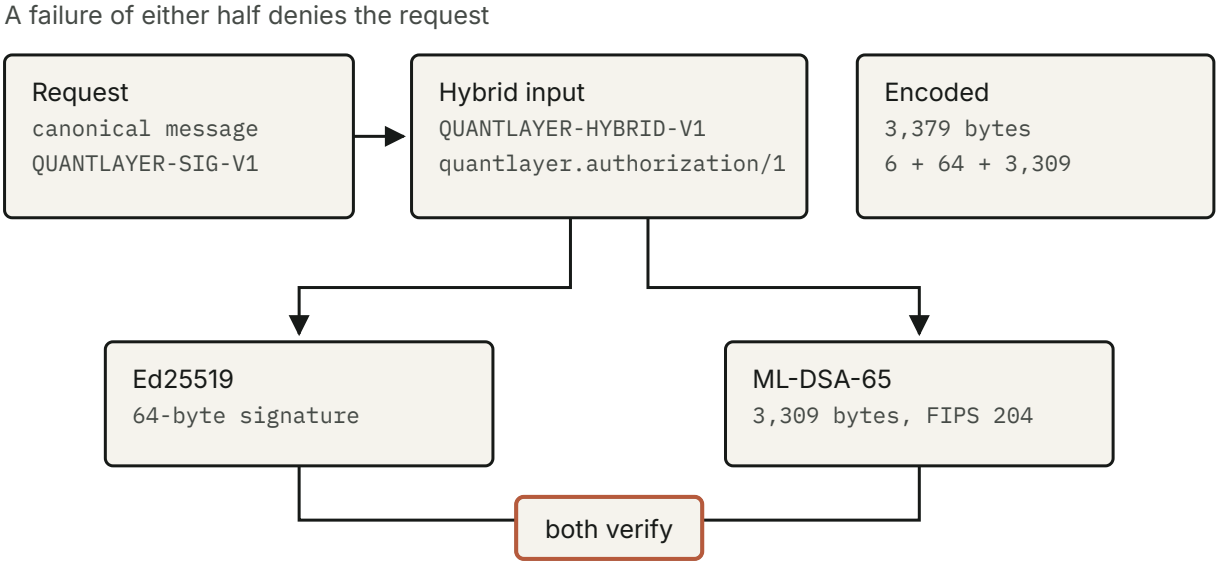


Figure 6. Hybrid authorization: both halves sign the same encoded input; either failure denies.

5.3 Key lifecycle and custody

Custody keys follow the state machine of Figure 7. Rotation retires version n and activates $n+1$ with a dual-signed transition statement; migration does the same across algorithms

(for example ML-DSA-65 → ML-DSA-87). Revocation stops verification and wipes the sealed secret.

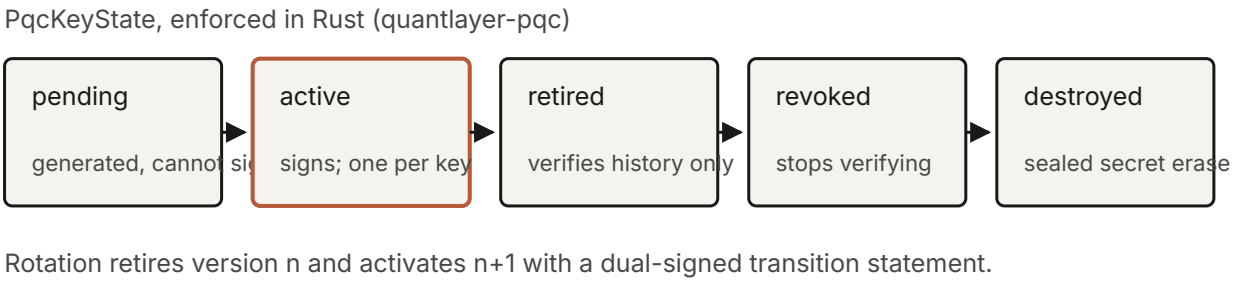


Figure 7. Post-quantum key states, enforced in Rust.

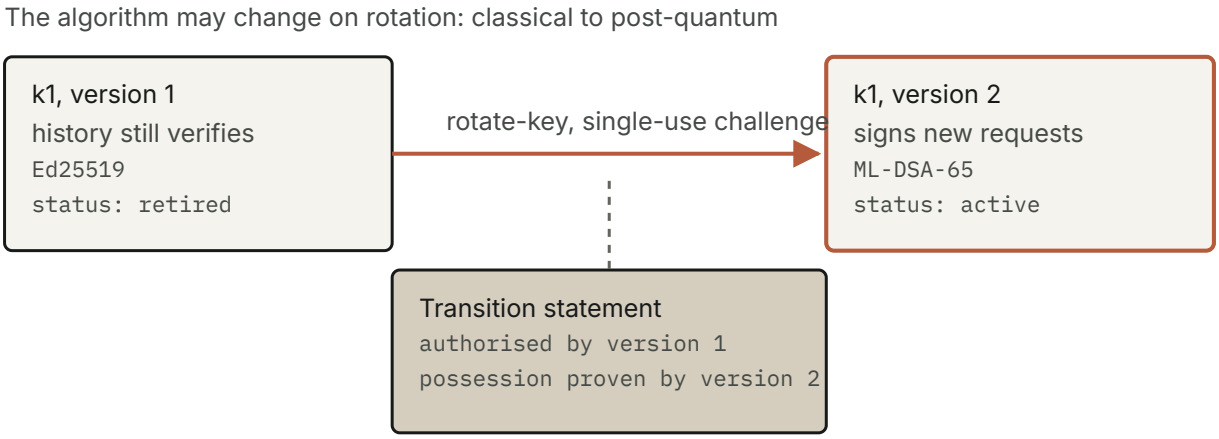


Figure 8. Rotation and migration with a transition statement signed by both versions.

At rest, a key is stored as its FIPS key-generation seed, sealed with XChaCha20-Poly1305 under a 256-bit key-encryption key. The associated data is a header naming the algorithm, key id, version and public-key fingerprint, so a sealed key cannot be relabelled or moved between identities; any tampering fails closed without detail.

5.4 Conformance testing

Test	Coverage
NIST ACVP vectors (hash-checked subset of ACVP-Server)	ML-DSA keygen 15, siggen 36, sigver 45; ML-KEM keygen 15, encaps/decaps 105; SLH-DSA keygen 9, siggen 24, sigver 42; all nine parameter sets
Differential testing	libcrux against the independent fips203 / fips204 crates, byte for byte
Interoperability vectors	Rust against the TypeScript bindings and @noble/post-quantum
Sealed-key fixture	keys sealed by an earlier build must keep opening after dependency upgrades

6 Authorization protocol

6.1 Requests and modes

An authorization request (`quantlayer.authorization-request` v1) names a subject, an action and resource, canonical parameters, and a domain (application, chain id, contract). It carries a single-use nonce, an issue and expiry time, the exact policy id and version, a mode, and the key id, version and algorithm of every signer. Parsing is strict: unknown fields, non-canonical values and inconsistent modes are rejected before anything else runs.

Mode	Evidence	On-chain assurance
classical	EIP-712 ECDSA by the registered key	Trustless
post-quantum	ML-DSA or SLH-DSA signature	Attested
hybrid	classical and post-quantum over one input (11)	Attested (ECDSA half verified on-chain)
credential	a selective-disclosure presentation; status checked on-chain	Attested
zk-eligibility	an UltraHonk proof bound to the request digest	Trustless

A classical authorization through the SDK

Copy

```
import { createQuantLayerClient, generateEd25519Signer } from '@quantlayer-p

const ql = createQuantLayerClient({
  api: { baseUrl: process.env.QUANTLAYER_API_URL, apiKey: process.env.QUANTL
  network: { network: 'arbitrum-sepolia', rpcUrl: process.env.ARBITRUM_SEPOL
});
const { signer } = generateEd25519Signer('k1');
const identity = await ql.identity.create({ signer });
// `request`: an AuthorizationRequest (Section 6.1) naming this identity's k
const evidence = await ql.authorization.sign(request, [signer]);
const decision = await ql.authorization.decide(request, evidence); // allow
```

6.2 Decision procedure

The engine evaluates nine steps in a fixed order and stops at the first failure:

$$\text{allow} \iff \bigwedge_{i=1}^{9} \text{check}_i(\text{request}, \text{evidence}, \text{state}) \tag{13}$$

The steps are, in order: request (schema, digest), evidence (shape, size), policy (exists, not revoked, version ≥ minimum, mode), binding (chain, contract, application, action), time (lifetime, skew, expiry), revocation (request id), subject (identity active), verification (key registry, then signature, credential or proof) and replay (consume the nonce, then the nullifier).

Properties

- **Determinism.** Identical inputs over identical state yield identical decisions; the clock is an injected dependency.
- **Fail closed.** A failing dependency becomes `VERIFIER_UNAVAILABLE`; an unconfigured capability becomes `CAPABILITY_NOT_CONFIGURED`. Neither can produce an allow.
- **Replay.** The nonce is consumed only after every other check passes, so a denied request does not burn it; the atomic consume makes concurrent submissions resolve to exactly one allow.
- **No secrets in decisions.** A decision lists key references, algorithms and claim *names*; it never contains signatures, keys, claim values or witnesses.

6.3 Execution sequence

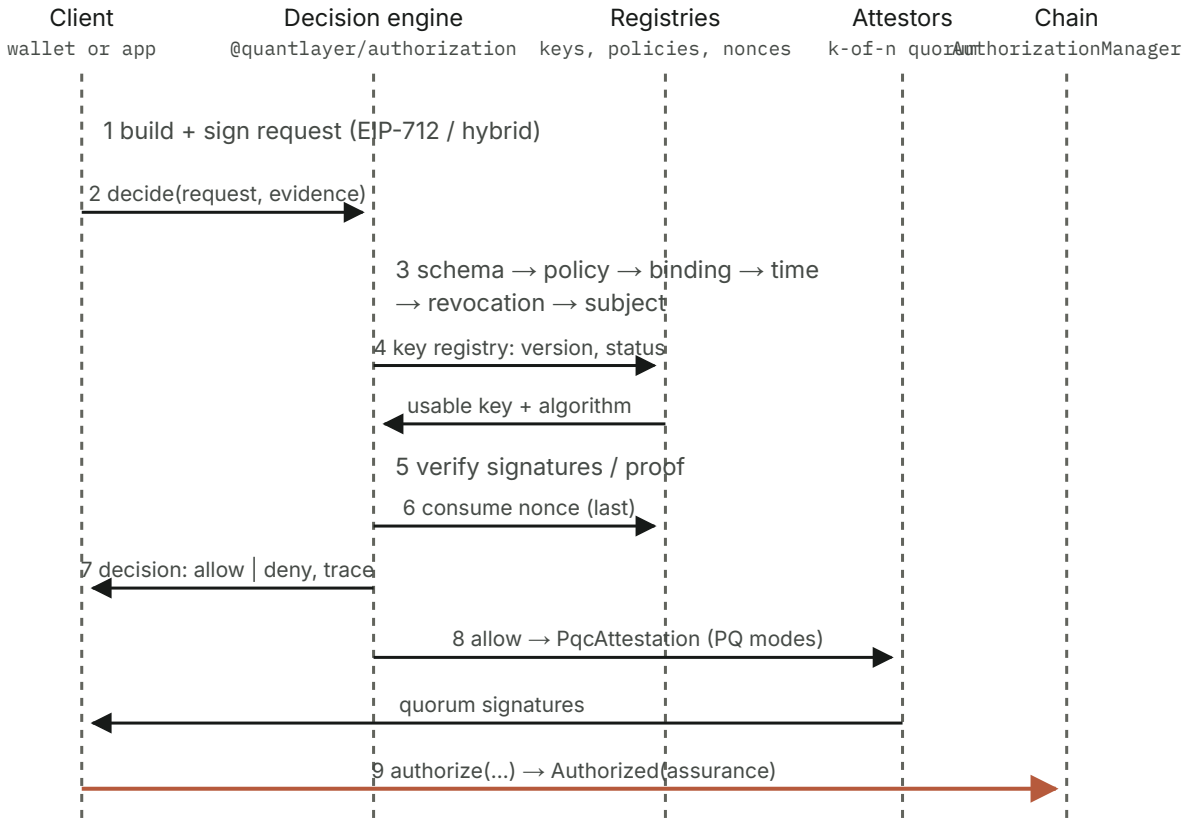


Figure 9. One authorization, from request to on-chain result. The attestation step occurs only for allow decisions in the post-quantum, hybrid and credential modes.

7 On-chain protocol

7.1 Contracts and roles

Seven contracts form the on-chain layer: identity, key, credential and nullifier registries; the proof verifier with one UltraHonk adapter per circuit; the post-quantum attestation verifier; and the authorization manager. Administration uses role-based access control. On a mainnet, the deployment script must hand every role to a contract admin (a multisig, ideally behind a timelock) and renounce the deployer's; a confirmation of the chain id prevents a mistyped RPC URL from deploying to a mainnet.

7.2 Replay and domain separation

Threat	Defence
The same authorization twice	each (subject, nonce) consumed once; the nonce key binds the chain and the manager
The same proof twice	each proof context (9) consumed once
One holder, fresh proofs	the nullifier (7) is spent once per scope (8)
Another chain or deployment	EIP-712 domain `{name, version, chainId, verifyingContract}` [16, 17]; the proof context binds chain, verifier and consumer
Stale requests, proofs, attestations	expiry and maximum lifetime; proof age ≤ 1 hour and a current root (1-day grace); attestation validity ≤ 1 hour
Front-running a signed request	the policy can pin the integrating contract as the only accepted caller

7.3 Post-quantum attestation

Native ML-DSA verification on the EVM was measured from its kernels (Keccak-f[1600], NTT, pointwise products) at a lower bound of about 22.5 million gas for ML-DSA-65, roughly 7,500 times `ecrecover` and 70% of Arbitrum's 32 million per-transaction limit, before decoding and norm checks. QuantLayer therefore verifies post-quantum evidence off-chain and brings the result on-chain through an explicit attestation:

$$\text{accept} \iff \text{usable}(\text{key}) \wedge \text{matches}(\text{att}, \text{key}, \text{digest}) \wedge \text{fresh}(\text{att}) \wedge |\text{signers}| \geq i \quad (14)$$

The attestation commits to the request digest, the subject, the key id and version, the algorithm, `keccak256(publicKey)` and `keccak256(signature)`, a validity window of at most one hour and the attestor-set version. With three or more attestors, $k \geq 2$. With no attestor set, these modes fail closed.

> **Trust model**
>
> An *Attested* result trusts that at least k registered attestors verified honestly and that their secp256k1 keys are uncompromised. The attestation layer itself is not post-quantum secure. Every result reports its assurance in the `Authorized` event, so a relying party can tell *Trustless* from *Attested*.

7.4 Deployment

The protocol is deployed on Arbitrum Sepolia (chain id 421614) from block 330,081,729. The deployment was verified on-chain (code, wiring, the six circuits and their verification keys, roles) and every contract's source matches on Sourcify. Robinhood Chain and the mainnets are not yet deployed.

Contract	Address
QuantLayerIdentityRegistry	`0x8fa74bc3c10A86C15a0406099071F36f4DbF62f3`
QuantLayerKeyRegistry	`0x36345d5a55ED0D07cB27dA1ece36c2404ccAe72B`
QuantLayerCredentialRegistry	`0xC899D084E3501c215906d29E79A129661579c7E7`
QuantLayerNullifierRegistry	`0x2c23A381529b4874F058E6E9BDB851919A1aAb4b`
QuantLayerProofVerifier	`0x365d679583d05f6E53316F1Cf33A77a8f91BD46D`
QuantLayerPqcAttestationVerifier	`0xE32766173f150F663c375d498682F9b035097f14`
QuantLayerAuthorizationManager	`0xEd7Ea8A06E847EEB0e42982694A18512D0DC9F5f`

8 Credentials and networks

8.1 Selective-disclosure credentials

QuantLayer's own credential format (`quantlayer.credential/2`) commits to each claim with a salted digest; the issuer signs the list of digests, possibly with a post-quantum key. A holder presents only the claims a verifier needs; the rest stay hidden behind their digests. A credential can also be committed as a Merkle leaf, so the circuits of Section 4 can prove facts about it without any disclosure.

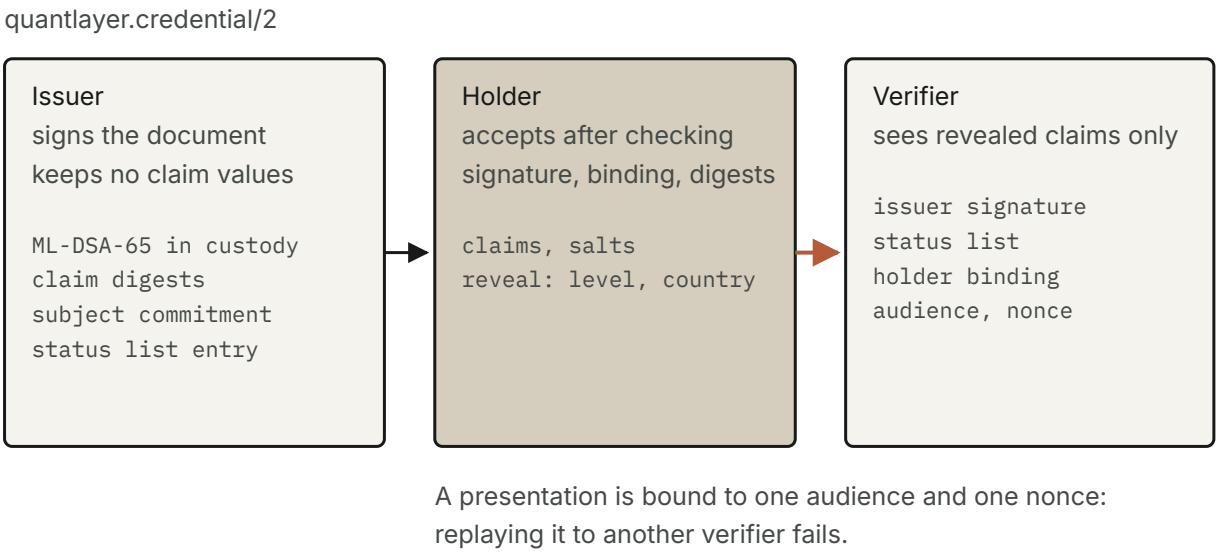


Figure 10. Issue, accept, present and verify a selective-disclosure credential.

8.2 External credentials

QuantLayer verifies credentials issued elsewhere: W3C Verifiable Credentials 2.0 [14] secured as JOSE or Data Integrity proofs, SD-JWT [15], and Bitstring Status Lists [23]. Trust is explicit: the verifier supplies the issuers it trusts and their keys; keys are never taken from the credential, only EdDSA, ES256 and ES256K are accepted, and the algorithm must equal the trusted key's. Status lists are supplied inline and verified under the same trust; no URL is ever fetched, so verification cannot be turned into a server-side request forgery. Status checks fail closed.

8.3 Cross-chain messages

Cross-chain messages carry a domain tag bound to the source network, contract and purpose:

$$\text{domain} = \text{keccak256}(\text{"QUANTLAYER-DOMAIN-V1|"} \| \text{caip2} \| \text{"|"} \| \text{contract} \| \text{"|"} \quad (15)$$

A message or proof whose context names one network is refused on another (for example, a request for Arbitrum One on a Robinhood-only policy). Cross-network attestation of issuer state trusts a quorum of independent RPC endpoints for block headers; it is a trust-based, not trustless, bridge.

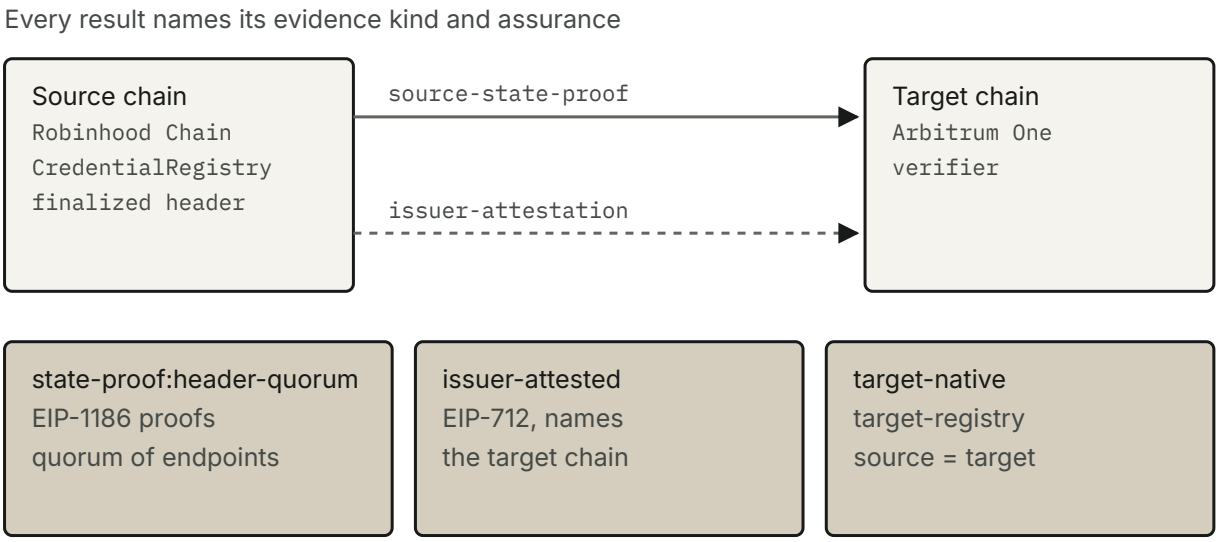


Figure 11. Cross-chain verification and the bindings that stop replay between networks.

9 Security analysis

9.1 Threat model

We consider network attackers, front-runners and relayers, malicious verifiers and applications, a compromised operator database, a colluding attestor quorum, and a future adversary with a cryptographically relevant quantum computer who records data today. The holder's device is trusted by its holder; the operator is trusted for availability but never holds private keys in plaintext; attestors are trusted only at their threshold.

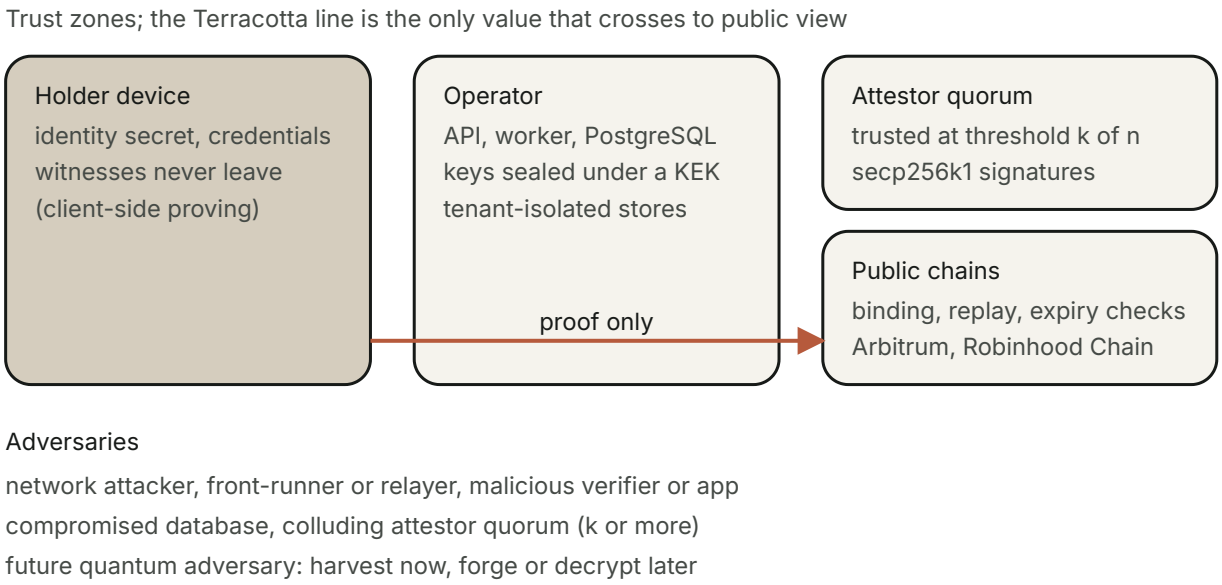


Figure 12. Trust zones, adversaries, and the single value that crosses into public view.

9.2 Attack vectors and mitigations

Attack	Mitigation	Residual risk
Replaying a proof or request elsewhere	context (9), scope (8), EIP-712 domains, single-use nonces and contexts	none known
Double claim by one identity	nullifier (7) spent once per scope	none known
Algorithm confusion or downgrade	algorithm identifiers pinned in the key registry and request; hybrid requires both halves	classical-only identity controllers remain valid
Key substitution	possession proofs; rotation authorized by a current key; transition statements	none known
Credential forgery or replay	issuer signature, holder binding with audience and nonce, status checks that fail closed	holder binding inherits the holder key’s algorithm
Cross-tenant access in the API	every store keyed by project; tenant-isolation tests	none known
Server compromise	no plaintext private keys; custody seeds sealed under a KEK; witnesses never persisted	a compromised running worker sees witnesses it is proving
Attestor collusion	k-of-n quorum, versioned sets, one-hour validity, assurance reported	k colluding attestors can attest falsely
Harvest now, forge or decrypt later	post-quantum signatures and key encapsulation	proofs, EVM accounts and TLS remain quantum-sensitive (9.3)

9.3 Quantum analysis

Shor's algorithm computes discrete logarithms in polynomial time [4]; Grover's search finds a preimage of an n-bit function in $O(2^{n/2})$ evaluations [5], and the BHT algorithm finds a collision in $O(2^{n/3})$ [6]. Applied to QuantLayer's components:

Component	Rests on	Against a quantum adversary
UltraHonk proofs, KZG, BN254 verifiers	discrete logarithms in pairing groups	**broken: proofs can be forged, nullifiers minted**
EVM accounts, contract admins, attestors	secp256k1 ECDSA	**broken**
ML-DSA, SLH-DSA, ML-KEM	Module-LWE/SIS; hash functions	believed secure at their NIST categories
Hybrid signatures (off-chain)	AND of ECDSA/EdDSA and ML-DSA-65	secure while ML-DSA is
Poseidon commitments and nullifiers	hash collision / preimage resistance	collision ≈ 2^{85} (BHT on a 254-bit field); unlinkability holds
Selective-disclosure salts	128-bit salts	≈ 2^{64} Grover work per guessed low-entropy claim
BN254 (classical)	exTNFS [12, 22]	≈ 100–110 bits classically, below 128
TLS to the proving API	classical key exchange	recorded witnesses could be decrypted later

Consequence

QuantLayer's post-quantum signatures and key establishment protect authorizations and stored keys against a future quantum adversary, but a cryptographically relevant quantum computer could still forge production proofs and control EVM accounts. The protocol as a whole is therefore **not** quantum-resistant today. The experimental STARK backend (Section 12.2) is the research path toward post-quantum proofs.

10 Performance and scalability

10.1 Complexity

Operation	Cost	Measured
Membership path	O(d) hashes, d = 20, independent of set size	20 Poseidon evaluations
Set capacity per tree	2^d leaves	1,048,576
Proof size	constant	8,768 bytes
Proving	quasi-linear in gates (2^16)	706–825 ms native, 858–931 ms WASM
Verification off-chain	constant	2.2–2.4 ms native
Verification on-chain	constant (pairings)	≈ 2.63–2.66 M gas
ML-DSA-65 sign / verify	O(k·ℓ·n log n) via NTT	155.9 µs / 45.6 µs
Authorization decision	9 fixed steps; one nonce write	not separately benchmarked

10.2 Bottlenecks

- **Prover memory.** A native proof peaks at 672–725 MiB. This is the main constraint on proving directly on low-memory devices, and the reason delegated proving exists.
- **On-chain verification.** One `verifyProof` uses about 2.66 M gas, roughly 8% of Arbitrum's 32 M per-transaction limit; throughput is bounded by block gas, not by QuantLayer.
- **Delegated proving throughput** scales with worker processes (configurable concurrency, Redis-backed queue); each proof is independent.
- **Post-quantum on-chain verification** is not feasible natively (≈ 22.5 M gas lower bound), hence attestation (Section 7.3).
- **Hash-based proofs are large.** The experimental STARK prototype produces 209–217 KB proofs (13.4 ms prove, 1.86 ms verify, Rust), about 24 times an UltraHonk proof; calldata cost is the obstacle to verifying them on-chain.

11 Comparison with existing approaches

QuantLayer combines patterns that exist separately elsewhere. The comparison below is qualitative except where a figure is QuantLayer's own measurement.

Approach	What the verifier learns	Post-quantum signatures	Notes
Document disclosure (conventional KYC)	the full document	not applicable	every verifier keeps a copy
W3C VC 2.0 / SD-JWT [14, 15]	disclosed claims only	depends on the issuer’s algorithm	selective disclosure, but no predicates over hidden values
Semaphore [18]	membership and a scoped nullifier	no	the identity-commitment and nullifier pattern QuantLayer’s membership circuit follows; Groth16 proofs
QuantLayer	the proven predicate (membership, type, validity, threshold) and a scoped nullifier	yes: ML-DSA, SLH-DSA, hybrid	also verifies external W3C and SD-JWT credentials; reports trustless versus attested assurance

Property	Groth16	UltraHonk (QuantLayer production)	STARK/FRI (QuantLayer experim
Setup	per-circuit trusted setup	universal SRS	transparent (no setup)
Proof size	constant, three group elements	constant; 8,768 bytes measured	209,662–217,290 bytes measure
Prover (QuantLayer circuits)	not measured here	706–825 ms native	13.4 ms for the eligibility counter (Rust)
EVM verification	pairings	≈ 2.66 M gas measured	no verifier contract
Assumptions [8, 9, 10, 11]	pairings, knowledge assumptions	pairings (KZG), random oracle	collision-resistant hashing, rand oracle
Against a quantum adversary	soundness broken	soundness broken	plausibly secure (hash-based); conjectured 111 bits at current parameters

The choice of UltraHonk for production follows from its universal setup, mature tooling in Noir and Barretenberg [19], and an on-chain cost that fits within a transaction. Its quantum weakness is the reason a hash-based backend is being researched in parallel rather than adopted: at current sizes its proofs cannot be verified on-chain economically.

12 Implementation status and roadmap

12.1 Implemented

In the product path, with tests run in this repository:

- Six Noir circuits with reproducible artifacts and Solidity verifiers; UltraHonk proving off-chain (native and WASM) and on-chain verification.
- ML-DSA, SLH-DSA and ML-KEM in Rust, with NIST ACVP vectors for all nine parameter sets, through the Node binding, API, SDK and web application.
- Hybrid signatures (Ed25519+ML-DSA-65, ES256K+ML-DSA-65); key registration, rotation, migration and revocation.
- The authorization engine with five modes; replay, nonce and downgrade protection.
- Seven Solidity contracts, deployed and verified on Arbitrum Sepolia.
- Selective-disclosure credentials; verification of external W3C VC 2.0 and SD-JWT credentials.
- An API (PostgreSQL, Redis), delegated proving with ML-KEM-sealed witnesses, a chain indexer, a TypeScript SDK and a web application.

12.2 Experimental

Feature	State	Why not production
STARK/FRI backend (Plonky3 0.8.0, KoalaBear, Keccak)	implemented and tested; off by default behind a flag; not reachable from the API	111 conjectured / 102 proven bits, below the 128-bit target; no on-chain verifier; large proofs
Composite authorization policies	implemented and tested; requires explicit opt-in; the API does not enable it	awaiting review; a ZK nullifier is consumed even if another factor fails
Cross-chain messages	library only	no API or web integration yet

12.3 Roadmap

1. An independent external security audit of the contracts, circuits, post-quantum module and API (required by the release gate before any production deployment).
2. Deployment to Robinhood Chain testnet; a registered attestor set on testnet.
3. Mainnet deployments with a multisig admin behind a timelock, after the audit.

4. A post-quantum proof system in production: a STARK backend reaching 128 conjectured bits (a larger challenge field) with an economical verification path.
5. Hybrid key exchange (`X25519MLKEM768`) for witness sealing, and post-quantum TLS for delegated proving.
6. Post-quantum identity controllers, and retiring classical controllers and holder keys.

13 Limitations

- **Proofs are not post-quantum.** UltraHonk over BN254 can be forged by a cryptographically relevant quantum computer; so can every EVM account (Section 9.3).
- **On-chain post-quantum results are attested,** not verified on-chain: they trust a k-of-n quorum whose own signatures are classical.
- **ZK mode is anonymous:** a proof does not authenticate which identity produced it; the subject must be an active identity, but it is not bound to the prover.
- **QuantLayer proves facts about credentials;** it does not hide transaction amounts, senders or recipients.
- **BN254's classical security** is about 100–110 bits, below a 128-bit target [12, 22].
- **Poseidon's cryptanalysis** is younger than that of SHA-2 and SHA-3 [7].
- **The post-quantum libraries are pre-1.0** (libcrux 0.0.11, fips205 0.5.0); side-channel resistance was not measured.
- **No independent security audit** has been completed. All security statements in this paper are the authors' analysis.

References

1. National Institute of Standards and Technology. FIPS 203: Module-Lattice-Based Key-Encapsulation Mechanism Standard. 2024. doi:10.6028/NIST.FIPS.203.
2. National Institute of Standards and Technology. FIPS 204: Module-Lattice-Based Digital Signature Standard. 2024. doi:10.6028/NIST.FIPS.204.
3. National Institute of Standards and Technology. FIPS 205: Stateless Hash-Based Digital Signature Standard. 2024. doi:10.6028/NIST.FIPS.205.
4. P. W. Shor. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. SIAM Journal on Computing 26(5), 1997.
5. L. K. Grover. A Fast Quantum Mechanical Algorithm for Database Search. Proceedings of the 28th ACM Symposium on Theory of Computing (STOC), 1996.
6. G. Brassard, P. Høyer, A. Tapp. Quantum Cryptanalysis of Hash and Claw-Free Functions. LATIN 1998, LNCS 1380.
7. L. Grassi, D. Khovratovich, C. Rechberger, A. Roy, M. Schofnegger. Poseidon: A New Hash Function for Zero-Knowledge Proof Systems. USENIX Security Symposium, 2021.
8. A. Kate, G. M. Zaverucha, I. Goldberg. Constant-Size Commitments to Polynomials and Their Applications. ASIACRYPT 2010, LNCS 6477.
9. A. Gabizon, Z. J. Williamson, O. Ciobotaru. PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive Arguments of Knowledge. IACR Cryptology ePrint Archive 2019/953.
10. E. Ben-Sasson, I. Bentov, Y. Horesh, M. Riabzev. Fast Reed–Solomon Interactive Oracle Proofs of Proximity. ICALP 2018.
11. E. Ben-Sasson, I. Bentov, Y. Horesh, M. Riabzev. Scalable, Transparent, and Post-Quantum Secure Computational Integrity. IACR Cryptology ePrint Archive 2018/046.
12. T. Kim, R. Barbulescu. Extended Tower Number Field Sieve: A New Complexity for the Medium Prime Case. CRYPTO 2016, LNCS 9814.
13. N. Bindel, U. Herath, M. McKague, D. Stebila. Transitioning to a Quantum-Resistant Public Key Infrastructure. PQCrypto 2017, LNCS 10346.
14. W3C. Verifiable Credentials Data Model v2.0. W3C Recommendation, 2025.
15. IETF OAuth Working Group. Selective Disclosure for JSON Web Tokens (SD-JWT).
16. Ethereum Improvement Proposals. EIP-712: Typed Structured Data Hashing and Signing.
17. Ethereum Improvement Proposals. EIP-155: Simple Replay Attack Protection.
18. Privacy & Scaling Explorations. Semaphore: a zero-knowledge protocol for anonymous signalling. docs.semaphore.pse.dev.
19. Aztec. Noir language and Barretenberg (UltraHonk) documentation. noir-lang.org.
20. Polygon. Plonky3: a toolkit for polynomial IOPs. github.com/Plonky3/Plonky3.
21. Cryspen. libcrux: a formally verified cryptographic library (hax / F*). github.com/cryspen/libcrux.
22. R. Barbulescu, S. Duquesne. Updating Key Size Estimations for Pairings. Journal of Cryptology 32, 2019.

23. W3C. Bitstring Status List v1.0.