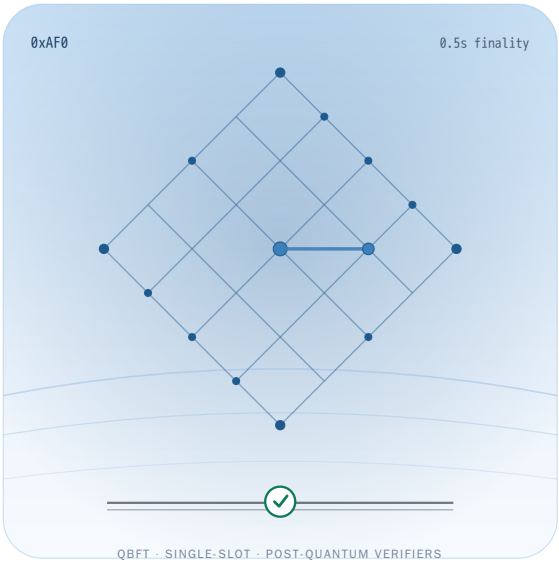


WHITEPAPER · CHAIN ID 2800 · 0XAF0

AERE Network

A non-custodial EVM Layer 1 built for settlement that stays safe when the cryptography changes.

Final as math, *safe when the math changes.*

CONSENSUS	EXECUTION	SUPPLY	CHAIN ID
0.5s single-slot BFT	EVM Pectra + Fusaka	2.8B fixed, no inflation	2800

PQC VERIFIERS
6 on-chain

VERSION 2.1.76 STATUS 2026-08-23 FOUNDATION · SEYCHELLES

Print or save as PDF

VERSION AND VERIFICATION

VERSION	CONTENT LAST CHANGED	FIGURES MEASURED AT BLOCK
2.1.76	2026-08-23	15,293,919

MEASURED AT

2026-08-23 20:17:27 UTC

CONTENT HASH (SHA-256)

`10977e2a47bb99823a4db68a6ef71b89191c14b6679bcad4bc5a7e6eb9dadc01`

DOCUMENT HASH (SHA-256)

`bb64a2dd17e47ef1377df674521695e3abe3ee7651f4b6df697974fde67b7a41`

EDITORIAL HASH (SHA-256)

`3b521f65addb001c01bf119ba1d0251a743681621ddda4df039317aee122f032`

Recompute the content hash yourself. This command fetches the published page, extracts the delimited content region, the measured figures, the table of contents and the nine chapters, and hashes exactly that. It is the hash of what this whitepaper claims, and edits to the surrounding page cannot move it:

```
curl -sL https://aere.network/whitepaper | sed -n '/<!--WP-CONTENT-START-->/,/<!--WP-CONTENT-EN
```

To verify the exact bytes of the whole page instead, remove this version block (so the hash cannot depend on itself) and hash the rest. This stricter hash moves on any edit anywhere in the page, including ones that do not touch the whitepaper's content:

```
curl -sL https://aere.network/whitepaper | sed '/<!--WP-VERSION-BLOCK-START-->/,/<!--WP-VERSION
```

How this version number works. The version changes only when the content changes. It is not a date stamp and must never be incremented on a schedule. A number that moves every day whether or not anything was written tells a reader nothing; a number that moves only on substance, and can be checked by hashing the document, tells them everything.

Three hashes, because they answer three different questions. The content hash proves the whitepaper's claims are the ones we published, and is the one to check first. The document hash proves you are reading the exact bytes we published, and it moves whenever anything moves, including a refreshed on-chain figure or a cosmetic page edit. The editorial hash ignores the generated figures entirely, so it moves only when the writing, structure or claims change. That last one is tied to the version number, which is why refreshing a measurement does not manufacture a new version.

Every figure marked as measured in this document is read from a public node by `scripts/generate-whitepaper-figures.mjs` and carries the block height it was read at. Values that cannot be read from a node, such as the fixed genesis supply or the throughput ceiling, are labelled as specification rather than presented as measurements. If the node cannot be read, the generator refuses to build rather than republish a stale number.

Full version history and hashes · Machine-readable figures (JSON)

MEASURED ON CHAIN

These figures are not typed by hand. They are read from a public AERE node and regenerated, all at one pinned block height so the set is internally consistent. Every value below was read at block 15,293,919, at 2026-08-23 20:17:27 UTC. If the node cannot be read, the build fails rather than republishing a stale number.

CHAIN ID 2800 MEASURED	BLOCK INTERVAL 565.9 ms MEASURED	BASE FEE 1 Gwei MEASURED
VALIDATORS 9 MEASURED	FAULT TOLERANCE f = 2 MEASURED	NAKAMOTO COEFFICIENT 1 SPECIFICATION
PQC PRECOMPILES LIVE 5 MEASURED	BURN RATE 37.5 % MEASURED	BURN CAP 50 % MEASURED
LIFETIME BURNED 0.137352 AERE MEASURED	TOTAL SUPPLY 2,800,000,000 SPECIFICATION	THROUGHPUT CEILING 273,000 TPS DESIGN CEILING

Consensus is classical. The 9 validators sign with secp256k1 ECDSA under QBFT. Post-quantum cryptography on this chain lives at the account and application layer, through the precompiles below, and not in consensus. All 9 validators are Foundation-operated, which is why the Nakamoto coefficient above is stated as 1 and marked as specification: a node can prove there are nine distinct addresses, it cannot prove they are independently run.

Post-quantum verifiers, proved live rather than asserted. Mainnet-live at block 15,293,919: 0x0AE1, 0x0AE2, 0x0AE3, 0x0AE4, 0x0AE5, active since the AerePQC fork, whose `activationTime` of 1783820272 the network reports directly through `eth_config`; block 9,189,161 is the first block whose timestamp is at or after it. That block contains no transactions, because a fork activation is a client configuration change and not a transaction, so its significance is its timestamp rather than its contents, and it is not the place to look for evidence. Verified as not present on mainnet, and therefore testnet-only: 0x0AE6, 0x0AE7. Presence is measured by gas differential against a control address holding no code. An address that charges more than the control ran code; one that charges exactly the control did not. A returned value alone would prove nothing, because a live precompile given malformed input and

an address that does not exist both return empty. Measured at `latest` on both public endpoints, in gas above the codeless control: Falcon-512 40,000, Falcon-1024 75,000, ML-DSA-44 55,000, SLH-DSA-128s 350,000, SHAKE256 60 plus 12 per 32-byte word, and 0x0AE6 and 0x0AE7 exactly zero. The probe is calibrated first against the IDENTITY precompile at 0x04, whose cost the EVM specification fixes at 15 plus 3 per word, and it reproduces that formula exactly. Those five figures are the literal gas constants in the client source, so the measurement and the published code can be compared line for line.

The burn, at its true scale. The burn takes 37.5 percent of the validator coinbase reward, capped by contract at 50 percent. It is not a base-fee burn. Lifetime burned to date is 0.137352 AERE, which is 0.049 parts per billion of supply, or 0.0000000049 percent. Validator coinbase revenue is effectively zero, so there is almost nothing for the burn to take a share of. AERE is not deflationary today, and the burn should be read as mechanism that is live and correct, not as supply pressure that is currently meaningful. The splitter contracts' own cumulative counters agrees with the vault balance, which is the cross-check that keeps this number honest.

Block interval, target against measured. The configured QBFT target is 500 ms. The measured long-run mean is 565.9 ms, taken over a 20,000 block window ending at the measurement block, so the figure is stable rather than an artefact of whichever handful of blocks was sampled. The window deliberately includes degraded periods: a validator outage, for example, makes every rotation absorb a round-change timeout and pushes this mean well above the target until the validator returns, and the figure is published anyway, because a measurement that excluded the bad hours would be an advertisement. Where this document describes blocks or finality as sub-second, it is describing the configured target; whether the network currently delivers it is decided by this measured figure, not by the target.

Throughput. The 273,000 TPS figure is an architectural ceiling implied by the block parameters. It is not a measured mainnet rate and is never presented as one.

CONTENTS

01

OVERVIEW

Overview and Thesis

What AERE is, the thesis that shapes every subsystem, and the current state of the network stated plainly, so that nothing downstream reads as spin.

Chain 2800 · genesis-v2 2026-05-07 · companion deep-dives: `docs/ROADMAP-PHASES.md`, `research/pqc-onchain-verification.md`

AERE Network is an EVM-compatible Layer 1 blockchain designed for one job done to a high standard: settlement that is fast, final, and durable against the cryptographic transition the whole industry is walking toward.

PARAMETER	
Network	
Consensus	
Execution	
Supply	
Status date	
Source of truth for addresses	
Foundation (owner of most Ownable contracts; 61 of the 79 registry contracts that answer owner() return it, 18 re	

> On reading this document
>
> Every contract address in this document is copied verbatim from the canonical registry `sdk-js/src/addresses.ts`. Contracts that are deployed and demonstrated but not yet promoted into that registry are named by contract and artifact rather than by canonical address. Every live capability names the contract that backs it; every capability that is not yet live is labelled in development or roadmap with the honest dependency that unlocks it. The tone is deliberate: a young, single-operator chain earns trust by being precise about what is real and what is planned, and every claim here is meant to be checked on-chain rather than taken on faith.

Companion deep-dives cited throughout this document, each of which carries its own honesty preamble and its own verbatim address list: the phased roadmap (`docs/ROADMAP-PHASES.md`), the post-quantum research paper (`research/pqc-onchain-verification.md`), and the six subsystem specifications under `research/specs/` covering the zero-knowledge stack, the anti-MEV mempool, account abstraction, parallel execution, identity and compliance, and the flywheel economics. This overview summarizes and frames; the companion documents carry the full depth, the exact on-chain transaction hashes, and the per-contract trust boundaries.

Abstract

AERE Network is an EVM-compatible Layer 1 blockchain, chain ID 2800, built on Hyperledger Besu with QBFT consensus. It is designed for one job done to a high standard: settlement that is fast, final, and durable against the cryptographic transition the whole industry is walking toward. Blocks are produced every half second and reach single-slot Byzantine-fault-tolerant finality in well under a second, so a confirmed transaction does not later reorganize away. The execution environment tracks Ethereum's own ruleset through the Pectra and Fusaka hard forks, giving functional parity with Ethereum mainnet, so contracts, wallets, and tooling that work on Ethereum work here unchanged. Supply is fixed at 2,800,000,000 AERE, set at genesis and never inflated; the token economics run on a burn-and-yield flywheel funded by real fees, not on new issuance.

What distinguishes AERE from a conforming EVM chain is a suite of post-quantum signature verifiers that run on-chain today at the application and account layer. Full, spec-complete verifiers for the hash-based schemes WOTS+, XMSS, and SLH-DSA (FIPS 205), and the lattice schemes Falcon-512, Falcon-1024, and ML-DSA-44 (FIPS 204), each validated bit-for-bit against official NIST Known-Answer-Test and ACVP vectors, are

deployed as ordinary immutable contracts. We are not aware of any other public chain that verifies all of these on-chain. This capability is the concrete meaning of the AERE thesis, "Final as math, safe when the math changes."

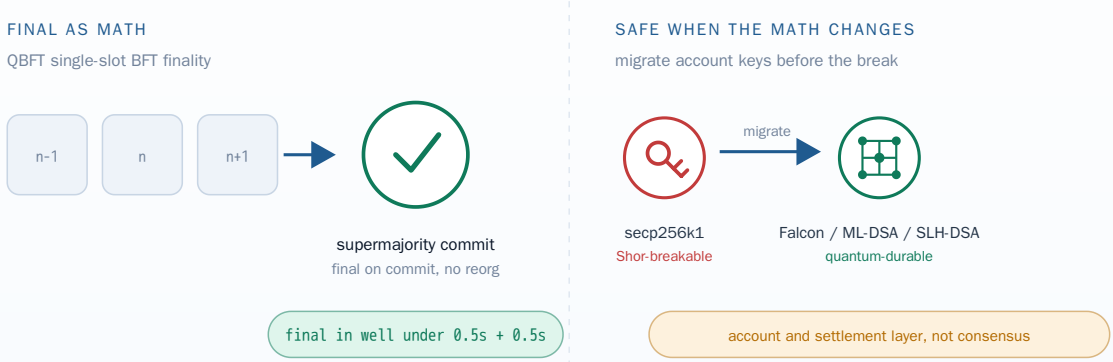
We are equally clear about what AERE is not yet. The network runs nine validators, all operated by the Foundation, on a single client, only two-fault Byzantine tolerance today (quorum six of nine), no external security audit, thin real usage, and no exchange listing. Consensus is hybrid post-quantum in a precise and limited sense, since 2026-08-14: classical secp256k1 ECDSA QBFT finalizes every block, and every 32nd block (an anchor block) must additionally carry a certificate of at least three valid Falcon-512 validator seals (f+1 of nine) (raised at block 14,961,456, August 21, 2026: the enforced minimum is now six of nine, a full 2f+1 quorum) or it is rejected. Correction published 2026-08-19: this document previously stated that from block 14,050,000 every block required a 2f+1 (six-of-nine) post-quantum quorum to finalize. The per-block Falcon quorum rule armed at that height is, in the shipped code, retired in favour of the anchor rules from block 13,014,000, and a re-reading of the code and of the chain (non-anchor blocks carry no Falcon seals) shows it changed no enforcement; that statement is withdrawn. What is enforced is the anchor certificate described here. Since block 13,014,000 the block hash itself covers a post-quantum certificate: every 32nd block carries, inside the hashed part of its header, a 32-byte digest that binds Falcon-512 seals from the validators. Since 2026-08-14 every node enforces a minimum of three seals per anchor block; three is f+1 for this set, the guarantee that at least one honest validator signed. At block 14,961,456 (August 21, 2026) the enforced minimum was raised to six of nine, a full 2f+1 quorum. Since 2026-08-16 the cap is nine and, measured on 2026-08-19 across the most recent anchors, certificates carry eight or nine seals, all nine signers appearing. The key manifests behind those seals are published at /pq/manifests/, pinned in an immutable on-chain contract, and a public tool verifies every seal cryptographically from a public RPC endpoint alone. That binds a post-quantum certificate into the chain's hash structure so it cannot be stripped or altered without changing the hash; it does not make block production post-quantum. These are stated plainly at the front so that nothing downstream reads as spin. The rest of this whitepaper describes a real, working foundation and an honest, gated path to change each of those facts in turn.

The thesis: final as math, safe when the math changes

Two words in that thesis carry the whole design.

FIG 1.1

The thesis, in two halves: deterministic finality now, and a migration path before the cryptography breaks



What it shows. The left half is deterministic finality: a QBFT block is final the moment a supermajority commits it, so there is no confirmation-depth wait and no deep-reorg probability. The right half is durability: because a public ledger permanently exposes keys and signatures, AERE lets accounts migrate to post-quantum keys before a cryptographically relevant quantum computer exists. The honest scope, drawn in amber, is that this durability is at the account and settlement layer; consensus signatures remain classical.

Final as math. AERE uses QBFT, a Byzantine-fault-tolerant consensus protocol, not longest-chain proof-of-work or probabilistic proof-of-stake. Under BFT finality, once a supermajority of validators commits a block, that block is final by the protocol's own rules: there is no confirmation-depth heuristic to wait through and no probability of a deep reorganization that quietly rewrites a settled transaction. Finality is a mathematical property of the commit, reached here in a single slot in well under a second. For a settlement layer this is the point. A payment or a trade that the chain calls final is final, and downstream systems can act on it immediately rather than waiting out a reorg window.

Safe when the math changes. The security of today's account signatures rests on the hardness of the elliptic-curve discrete logarithm, which Shor's algorithm solves in polynomial time on a sufficiently large quantum computer. A public ledger is uniquely exposed to this because it publishes, permanently and to everyone, the two artifacts an attacker most wants: account public keys and the signatures made under them. Signatures are public, so nothing needs to be harvested and decrypted here; the ledger-specific threat is retroactive forgery. The break can come years later, when a cryptographically relevant quantum computer exists, and it applies retroactively to any account whose key material is

already public: a recovered key can sign new statements about old history. (Harvest-now-decrypt-later, by contrast, threatens encrypted data, which a ledger also carries wherever payloads are confidential.) AERE's response is to make the chain able to verify post-quantum signatures now, so that users can migrate account and settlement authorization onto quantum-resistant keys well before the break arrives. Hash-based schemes (WOTS+, XMSS, SLH-DSA) rest only on the pre-image and collision resistance of their hash functions, which a quantum adversary attacks only with Grover's quadratic speedup, absorbed by parameter sizing. Lattice schemes (Falcon, ML-DSA) rest on structured-lattice problems for which no efficient quantum algorithm is known. AERE verifies both families on-chain today. The honest boundary, restated because it matters, is that this durability is at the account and settlement layer; consensus signatures remain classical, and making consensus itself post-quantum is a separate, explicitly approved decision, delivered through coordinated per-node activation heights rather than a re-genesis, and never a silent implication of the thesis.

Design philosophy

Five commitments shape every part of the system described in the chapters that follow.

Additive, not invasive. Post-quantum security can be introduced at two very different layers. Changing consensus so that validators sign blocks with a post-quantum scheme touches the client, the gossip and finality protocols, and every operator. Changing the account and settlement layer is additive: a contract that verifies a Falcon or SLH-DSA signature deploys like any other contract, and a smart account can bind its authorization to a post-quantum key with no protocol change at all. AERE takes the additive path deliberately, which is what lets it offer a usable quantum-resistant primitive today while leaving consensus untouched. The post-quantum research paper (`research/pqc-onchain-verification.md`) develops this argument and its limits in full.

Parity with Ethereum, not a private dialect. AERE tracks Ethereum's ruleset rather than forking away from it. Pectra and Fusaka are both active, so the chain has EIP-7702 account delegation, the EIP-2537 BLS precompiles, the RIP-7951 secp256r1 precompile for native passkey verification, transient storage, and the EIP-7825 per-transaction gas cap of 16,777,216 (2^{24}) gas. AERE keeps that cap deliberately rather than leaning on an unbounded local gas limit, so the chain stays a functional peer of Ethereum mainnet and its constraints are honest ones that mainnet contracts also face. One measurement note, because this number is used as a reference point elsewhere in the document: the cap is a

client-level transaction validation rule, and it is not readable from chain state. It has not been measured here, since the only conclusive test is broadcasting a transaction above the cap. What is measured is the block gas limit in the live header, 9,007,199,254,740,991, so the 2^{24} figure is a client rule rather than something derivable from the block limit. The account-abstraction spec (`research/specs/spec-account-abstraction.md`) details how the wallet layer is built on these features.

Immutable, ownerless where it matters. The post-quantum verifiers, the burn-and-yield router, and the burn endpoint are deployed with no owner, no admin, and no upgrade path. They are pure logic: given the inputs, they return the same decision every time, and no key can change that. Ownership concentration is a real weakness of the current network, discussed below, but it is bounded: the contracts that carry the core guarantees cannot be altered by the Foundation or anyone else. Where a contract does have an owner, this whitepaper names that owner as the Foundation account and does not pretend otherwise.

Fixed supply, real-fee flywheel, no adaptive issuance. Total supply is 2,800,000,000 AERE, capped at genesis-v2 and never increased. No mechanism in the system mints new tokens. The supply-reduction and staking-yield mechanism is an immutable three-bucket revenue router, AereSink `0x69581B86A48161b067Ff4E01544780625B231676`, which splits fee inflow on a fixed 15 / 40 / 45 basis across burning AERE, buying back and burning AERE, and lifting the exchange rate of the liquid-staking receipt sAERE `0xA2125bE9C6fd4196D9F94757Df18B3a2A5e650b0`. AERE is not deflationary today, and this whitepaper does not describe it as such. The router is live, immutable and ownerless, and it burns strictly in proportion to validator coinbase reward. That reward is currently zero, for two independent reasons given in full in Chapter 6: the genesis QBFT configuration declares no `blockreward`, and the 1 Gwei base fee is destroyed by EIP-1559 rather than paid to the proposer. Lifetime burned to date is 0.13735259 AERE against a fixed supply of 2,800,000,000 AERE, readable at any time as the balance of `AereFeeBurnVault` `0x696afDF4f814e6Fd6aa45CE14C498ed9375fB2c6`. We publish that figure rather than imply a larger one. Staking rewards are a finite, reserve-funded subsidy today, with the intended end-state that real protocol-fee volume, not the reserve, pays stakers. The flywheel spec (`research/specs/spec-flywheel-economics.md`) states the exact formulas and reserve mechanics. AERE is the network's only token; there is no stablecoin issuance and no second asset.

Honesty as a credibility asset. A young, small, single-operator chain earns trust by being precise about what is real and what is planned, not by rounding up. Every live capability in

this whitepaper names the contract that backs it, with the address copied verbatim from the canonical registry. Every not-yet-live capability is labelled in development or roadmap and states the honest dependency that unlocks it: code, money, people, or a founder signature. Contracts that are deployed but not yet promoted into the canonical registry are referred to by name and artifact rather than having their addresses printed as canonical. This discipline is not decoration; it is the reason a reader can check the claims rather than take them on faith.

Current state, stated plainly

This is what runs on mainnet 2800 today. It is a complete and real foundation. It is not a decentralized or audited one yet, and both halves of that sentence are stated here so nothing downstream has to walk anything back.

A COMPLETE, REAL FOUNDATION

- **0.5-second blocks**, single-slot BFT finality (halved mid-chain at block 2,137,652)
- **EVM parity:** Pectra at block 2,075,341, Fusaka at block 2,106,597
- **Fixed supply** of 2,800,000,000 AERE across six genesis wallets
- **Six NIST KAT-validated** post-quantum verifiers live on-chain
- **Deployed stack:** WAERE, staking, sAERE and AereSink, a proven lending engine, a multi-prover zk surface, a rollup validity anchor

NOT DECENTRALIZED OR AUDITED YET

- **Nine validators, one operator** (Foundation), two-fault Byzantine tolerance at n=9 (quorum 6-of-9)
- **One producing client** (Besu) on mainnet; the second client validates and follows there, and reaches consensus with Besu only on an isolated testnet
- **No external audit** of contracts or client changes yet
- **Thin usage**, no listing, no fiat on-ramp; PQC scope is not consensus

What is live and working. Besu QBFT with 0.5-second blocks, halved from one second mid-chain at block 2,137,652, and single-slot BFT finality. The EVM ruleset is Pectra, activated at block 2,075,341, plus Fusaka, activated at block 2,106,597, for functional parity with Ethereum mainnet. Supply is fixed at 2,800,000,000 AERE across six genesis wallets (100M Strategic Investor, 180M Foundation, 1.4B Reserve (scheduled), 560M Ecosystem Reserve, 420M Team Reserve, 140M Airdrop Reserve). The deployed and Foundation-

controlled stack includes the wrapped token WAERE, the staking pool, the sAERE and AereSink flywheel plumbing, a lending engine proven end to end on a proof market, a multi-prover zero-knowledge verification surface behind a route-by-selector gateway, and a rollup validity anchor, AereRollupValidity `0x38772063572DF94E90351e44ccbBEefD5F497fbd`, whose first epoch was recorded from a real SP1 Groth16 proof. The block parameters admit a design ceiling on the order of 273,000 transactions per second; this is a theoretical maximum implied by the gas and block configuration, not a measured or sustained figure, and realized throughput on the live network is a small fraction of it.

The post-quantum verifier suite. This is the genuinely differentiated part of the current state, and it is worth stating precisely: we are not aware of any other public EVM Layer-1 with native post-quantum signature verification precompiles live on mainnet, callable by anyone, from any contract or any `eth_call`, at a published public RPC endpoint. Not a testnet, not a demonstration contract, not an announcement of intent. Five precompiles went live at block 9,189,161 and have been serving traffic since: Falcon-512 at `0x0AE1`, Falcon-1024 at `0x0AE2`, ML-DSA-44 at `0x0AE3`, SLH-DSA-128s at `0x0AE4` and SHAKE256 at `0x0AE5`. Five algorithms rather than one is the substantive part of the design: a contract can select a lattice scheme or a hash-based scheme at runtime, so an application is not hostage to a single family surviving cryptanalysis. Two further precompiles, ML-KEM-768 and a Falcon hash-to-point helper, exist on testnet only and are deliberately not counted here. Six full, spec-complete, official NIST KAT-validated verifiers run on-chain: the hash-based WOTS+ (AerePQCVerifier), XMSS-SHA2_10_256 (AereXmssVerifier), and SLH-DSA-SHA2-128s (AereSphincsVerifier), all three of which record on-chain, alongside the lattice schemes Falcon-512 (AereFalcon512Verifier), which also records on-chain, and Falcon-1024 (AereFalcon1024Verifier) and ML-DSA-44 / Dilithium2 (AereMLDSA44Verifier). Chapter 3 carries the full table with each contract's verbatim address, cryptographic family, and on-chain status. We are not aware of any other public chain that verifies all of these on-chain. Two of the six used to be the honest catch. Falcon-1024 (a full record transaction costs roughly 21.7M gas) and ML-DSA-44 (roughly 52.9M gas) both exceed the 2^{24} per-transaction cap in pure Solidity, so their pure-Solidity verifiers stay read-only `eth_call` references. Their cryptography is identical to the recorded schemes and demonstrated against the same official vectors; the only obstacle was the execution budget. That obstacle is removed by a native-precompile path that was built and KAT-validated on an isolated Besu 26.4.0 scratch fork (chain ID 28099) and then activated on mainnet 2800 by the AerePQC fork at block 9,189,161 (2026-07-12). On mainnet every scheme now records within the cap, with measured verify-and-record gas of Falcon-512

86,336, Falcon-1024 145,496, ML-DSA-44 351,050, SLH-DSA-SHA2-128s 558,276, and SHAKE256 21,470, all far under 16,777,216, and all NIST KAT vectors including negatives pass (41 of 41 accept and reject cases). Activation was a coordinated client-only hard fork with no re-genesis that also carried the extended EIP-2935 block-hash lookback (an 8191-block window); it added the precompiles and EIP-2935 only and did not change consensus, and Block-STM parallel execution was not part of it and remains a testnet demonstration. Since 2026-08-14 AERE mainnet consensus is hybrid in a precise and limited sense: every 32nd block (an anchor block) does not finalize without a certificate of at least three valid Falcon-512 validator seals (f+1 of nine) (raised at block 14,961,456, August 21, 2026: the enforced minimum is now six of nine, a full 2f+1 quorum) under its block hash, with classical ECDSA (secp256k1) QBFT finalizing every block; post-quantum verification is also available as on-chain precompiles. The verifier suite is already usable, not merely readable: a post-quantum ERC-4337 smart account whose sole owner is a Falcon-512 key, and a hybrid authorizer that requires both a secp256k1 and a Falcon-512 signature over the same hash, both record on-chain. The research paper carries the full method, the committed fixtures, and the per-scheme gas analysis.

The weaknesses, without softening. These are facts about the network as it stands, not risks in the abstract.

> Nine validators, one operator. All nine validators are held and run by the Foundation. With n=9 the tolerated Byzantine faults is two (quorum 6-of-9), so the chain now survives two simultaneous validator outages, but all nine are one operator. This is not a decentralized network today.

> One producing client. AERE produces every mainnet block with a single execution client, Besu. A second, independent client (a patched Nethermind on a different codebase) has validated and followed live chain 2800 and would cross-check a consensus-relevant divergence at runtime; as of 2026-08-10 it is stalled behind the head, and the gap is published rather than hidden, and it is now complete enough to reach QBFT consensus with Besu as an equal validator, but that was proven on an isolated testnet only. It produces no mainnet blocks, so client diversity on the live network does not exist today and is not claimed.

> No external audit yet. The contracts and the client changes have not undergone a third-party security audit. Internal audit discipline has found and fixed numbered findings tracked in the registry, and the verifiers are validated against official NIST vectors and independent reimplementations, but that is not a substitute for an external audit and is not presented as one.

> Thin usage. These are recently deployed primitives with limited real traffic. The demonstrations cited throughout are genuine on-chain transactions and `eth_call` results, not production load.

> No listing and no fiat on-ramp. There is no centralized-exchange listing and no fiat on-ramp. The native order-book market map is empty, so price discovery for AERE does not exist on a public venue yet.

> Post-quantum scope is not consensus. Restated once more because it is the easiest claim to overstate: the verifiers, the hybrid authorizer, and the post-quantum account all sit above consensus. Consensus is hybrid post-quantum in the limited, anchor-certificate sense described in chapter 3, since 2026-08-14, by a separate, explicitly approved activation; nothing in the application-layer chapters of this whitepaper is what did that. Since block 13,014,000 the block hash itself covers a post-quantum certificate: every 32nd block carries, inside the hashed part of its header, a 32-byte digest that binds Falcon-512 seals from the validators. Since 2026-08-14 every node enforces a minimum of three seals per anchor block; three is f+1 for this set, the guarantee that at least one honest validator signed. At block 14,961,456 (August 21, 2026) the enforced minimum was raised to six of nine, a full 2f+1 quorum. Since 2026-08-16 the cap is nine and, measured on 2026-08-19 across the most recent anchors, certificates carry eight or nine seals, all nine signers appearing. The key manifests behind those seals are published at /pq/manifests/, pinned in an immutable on-chain contract, and a public tool verifies every seal cryptographically from a public RPC endpoint alone. That binds a post-quantum certificate into the chain's hash structure so it cannot be stripped or altered without changing the hash; it does not make block production post-quantum.

The roadmap (`docs/ROADMAP-PHASES.md`), consolidated in Chapter 9, is the plan to change each of these facts in order, with the gate for every change named and none of them reached by announcement.

How to read this whitepaper

The chapters that follow move from this overview into the subsystems, each of which has a dedicated companion specification that carries the engineering depth this overview deliberately omits. The post-quantum verification layer is developed in `research/pqc-onchain-verification.md`. The wallet and onboarding layer, including passkey accounts, the paymasters, EIP-7702 delegation, and the post-quantum smart account, is in `spec-account-abstraction.md`. The proof-verification surface, covering SP1, RISC Zero, the storage-proof coprocessor, and the attestation registry, is in `spec-zk-stack.md`. The fair-ordering and settlement work is in `spec-antimev-mempool.md`, the throughput work in `spec-parallel-execution.md`, the regulatory and identity layer in `spec-identity-compliance.md`, and

the token mechanics in `spec-flywheel-economics.md`. The phased path from today's single-operator network to an independent validator set, binding on-chain governance, and native post-quantum precompiles is set out in `docs/ROADMAP-PHASES.md` and summarized in Chapter 9.

Read all of it against the tone set here. Every claim in this whitepaper is meant to be checkable on-chain, and it will read as accurate only for as long as it stays that way.

02

FOUNDATIONS

Architecture and Consensus

Deterministic QBFT finality, honest Ethereum parity, and a real, self-verifying parallel-execution engine that runs in the rollup layer while the base layer stays deliberately sequential.

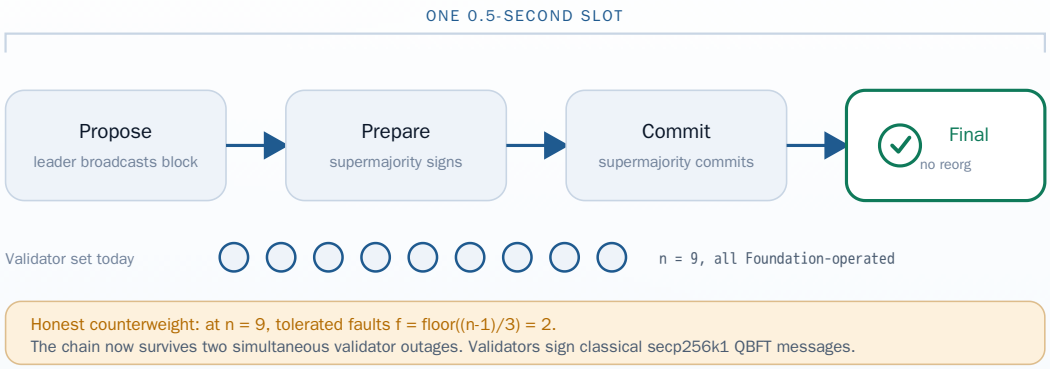
Chain 2800 · Besu QBFT · companion deep-dive: `spec-parallel-execution`

AERE is an EVM Layer-1 with chain ID 2800. Its architecture is deliberately conservative where correctness matters and deliberately forward-leaning where the execution budget allows. The chain runs a single, well-understood consensus engine; it tracks Ethereum's EVM ruleset feature-for-feature so that any tool, contract, or auditor familiar with mainnet is immediately at home; and it invests its novelty in two places that do not compromise base-layer safety: an application-layer post-quantum verifier suite (covered in Chapter 3) and a real, benchmarked parallel-execution engine that runs today in the rollup layer while the base layer stays deliberately sequential. This chapter describes the consensus, the finality model, the EVM upgrade posture, and the parallel-execution work, and it is honest about what is centralized, what is unproven at scale, and what is roadmap. The design thesis that runs through all of it is simple: final as math, safe when the math changes.

Consensus: Hyperledger Besu with QBFT

AERE's base layer is Hyperledger Besu running QBFT (Quorum Byzantine Fault Tolerant) consensus. QBFT is a proof-of-authority, leader-based BFT protocol: a known validator set takes turns proposing blocks, and a block is committed once a supermajority of validators has signed off on it in the prepare and commit rounds. There is no mining, no probabilistic longest-chain rule, and no reorg risk under honest-majority assumptions. When a block commits, it is final.

FIG 2.1 QBFT single-slot finality: propose, prepare, commit, final, all inside one 0.5-second slot



What it shows. A QBFT block is final the moment the validator quorum commits it, not after N confirmations, so there is no settlement lag and no reorg window. Halving the block period to 0.5 seconds halves time to finality directly. The amber band states the honest counterweight: with nine Foundation-operated validators of a single operator, fault tolerance is two and consensus signatures are classical.

Two properties follow directly from this choice, and both are stated plainly rather than dressed up.

First, finality is deterministic and immediate. A QBFT block is final the moment it is committed by the validator quorum, not "final after N confirmations" the way a proof-of-work or Nakamoto-style chain is. There is no settlement lag to wait out and no economic-finality heuristic to reason about. This is the "final as math" half of the thesis: settlement on AERE is a property of the consensus algorithm, not a probability that grows over time.

Second, and this is the honest counterweight, AERE today runs nine validators, and all nine keys are held and operated by the Foundation. For a QBFT set of size n, the tolerated Byzantine or crash faults are f = floor((n - 1) / 3), which at n = 9 equals two, with a quorum of six of nine. In plain terms: the chain now survives two simultaneous validator outages, but with one operator there is no operator diversity yet. This is a permissioned, single-operator network at the consensus layer. It is a complete and real foundation, but it is not a decentralized one, and nothing in this chapter should be read as claiming otherwise. The expansion to nine is already done; the remaining path from nine validators to twenty-one, under an independent-operator Validator Charter, is the subject of the phased roadmap; it is roadmap, not a live property.

Consensus signatures are classical

Consensus signatures themselves are classical. Validators sign QBFT consensus messages with secp256k1 keys. AERE's post-quantum work lives strictly above consensus at the application and account layer, and does not change what validators sign. Any statement that AERE has post-quantum consensus would be false, and the architecture is explicit about that boundary.

Machine-checked safety and liveness

AERE's QBFT safety and liveness are formally modeled and checked. A z3 SMT model (44 checks, kept in `formal-consensus/`) proves QBFT agreement, that no two conflicting blocks can finalize against a Byzantine adversary, with quorum intersection proven for every set size N, and proves liveness under partial synchrony. The model also covers the post-quantum Falcon quorum extension: the pre-activation certificate is machine-checked to be a strict no-op on classical consensus, and the fleet-size guard for the per-block blocking post-quantum mode, nine keyed validators, is verified (that per-block mode is modelled and guarded, but not the mode enforced on mainnet, where the anchor certificate is; see the correction in this chapter).

The honest boundary. This is a bounded model check of the consensus *design*, not of the Besu client bytecode, and it assumes partial synchrony. Since 2026-08-14 the Falcon certificate in anchor headers is enforced at a minimum of three seals (f+1 of nine), raised to six of nine, a full 2f+1 quorum, at block 14,961,456 (2026-08-21): mainnet QBFT does not finalize an anchor block without it, with classical ECDSA (secp256k1) seals finalizing every block. (Correction 2026-08-19: the earlier claim that a Falcon quorum blocks every block from 14,050,000 is withdrawn; that rule is retired in the shipped code and changed no enforcement.) A formal model of the design is also not an external security audit of the contracts or the chain, which is a separate, still-pending step.

Sub-second deterministic finality

AERE runs a 0.5-second block time. The chain launched with one-second blocks and was moved to 0.5-second blocks by a mid-chain QBFT parameter change at block 2,137,652, applied without a re-genesis. Measured on 2026-08-01 across the last 100,000 blocks at the head, the mean interval is 0.557 seconds, and the validator set read from the chain at the same head is seven. Because QBFT finalizes on commit rather than by accumulation of work, halving the block period halves the time to finality directly: a transaction is final in well under a second from inclusion, with no confirmation count to wait through.

It is worth being precise about what this figure is and is not. The 0.5-second block period and single-slot commit are the source of AERE's sub-second finality claim, and that claim is about latency to irreversibility, not about throughput. Any headline peak-throughput number for the chain (the 273,000 TPS design ceiling that appears in AERE's broader materials) is a design ceiling derived from block-space arithmetic under ideal conditions. It has never been a measured result, and it is not asserted as one here. Throughput and finality are separate axes, and this chapter only makes the finality claim, which follows mechanically from QBFT plus the block period.

EVM upgrade posture: Pectra and Fusaka parity

AERE deliberately keeps its EVM ruleset at functional parity with Ethereum mainnet rather than forking off into a bespoke dialect. Two hard forks are activated on chain 2800:

- Pectra (the Prague and Cancun feature set) activated at Unix time 1780189051, block 2,075,341, on 2026-05-31.
- Fusaka (Osaka) activated at Unix time 1780220351, block 2,106,597, later the same day.

Correction of 2026-08-02, because these two numbers were published wrong for months. Earlier editions of this document gave the heights as 2,075,363 and 2,106,606. Both forks are gated by timestamp rather than by block number, so the height has to be read off the chain, and when it was, both published figures were too high, by 22 and 9 blocks. The corrected heights above were re-derived independently at head: 2,075,341 is the first block whose header carries a `requestsHash` field and also the first whose timestamp reaches `pragueTime`, and 2,106,597 is the first whose timestamp reaches `osakaTime`. The activation timestamps themselves were always right; only the block numbers derived from them were not.

PECTRA ACTIVATION	FUSAKA ACTIVATION	0.5S BLOCK CHANGE	PER-TX GAS CAP
2,075,341	2,106,597	2,137,652	16,777,216
block	block	block	2^{24}, EIP-7825

Parity is a strategic choice, not an incidental one. It means AERE is a functional peer of mainnet: the same compilers, the same wallet libraries, the same audit assumptions, and the same precompile addresses all carry over unchanged. The trade AERE accepts for that parity is that it inherits mainnet's constraints too, most notably a per-transaction gas cap,

and it keeps that cap on purpose rather than papering over it with an unbounded local gas limit.

Pectra brings, among other changes, EIP-7702 (EOA delegation, the native account-abstraction primitive), the EIP-2537 BLS12-381 precompiles at addresses 0x0b through 0x11, EIP-1153 transient storage (TSTORE and TLOAD), EIP-5656 (MCOPY), EIP-6780 (the revised SELFDESTRUCT semantics), and EIP-2935 (the historical block-hash contract). Two Pectra items are neutralized honestly because AERE is a QBFT chain with no separate consensus layer: EIP-4788 (parent beacon block root) returns 0x0, and EIP-7516 (BLOBBASEFEE) returns 0 because there are no blobs.

Fusaka adds the two features that matter most to AERE's actual product surface:

- RIP-7951, a secp256r1 (P-256) verification precompile at address 0x100. P-256 is the curve behind platform passkeys: Apple Face ID and Touch ID and the Secure Enclave, Windows Hello, Android biometric keystores, YubiKeys, and the EU Digital Identity Wallet. Verifying a P-256 signature in hand-written Solidity costs on the order of 250,000 gas; the native precompile does it for roughly 3,450 gas, about two orders of magnitude cheaper. This precompile is what makes native passkey-authenticated smart accounts practical on AERE rather than merely possible, and it is the direct precedent AERE cites for its own proposed native post-quantum precompiles.
- EIP-7825, a per-transaction gas cap of 2^{24} = 16,777,216 gas, independent of the block gas limit. AERE keeps this cap deliberately. It is the single most load-bearing constant in the chain's design because it draws a hard line between what a state-changing transaction can do and what only a read-only `eth_call` can do. That distinction is exactly what governs which of AERE's on-chain post-quantum verifiers record a result in a mined pure-Solidity transaction and which record only through the native precompiles that the AerePQC fork made live (Chapter 3), and it is also the ceiling that AERE's parallel-execution and precompile roadmap is measured against.

The SHA-256 precompile at 0x02 is also present and is used heavily by the hash-based verifiers. The important architectural point is that AERE gets these primitives the same way mainnet does, at the same addresses, so nothing about AERE's account, passkey, or post-quantum surface depends on a nonstandard EVM.

Execution model and parallel execution

The base layer executes sequentially on mainnet today. Besu processes the transactions in a block one after another, in order, exactly as Ethereum mainnet does. Making the L1 base layer execute in parallel is real, proven work, but stated precisely it is three distinct proof-of-approach artifacts, not a shipped mainnet feature. None is part of the AerePQC mainnet fork (which activated only the post-quantum precompiles and the extended EIP-2935 block-hash lookback, described in Chapter 3) and none is mainnet-active.

(1) A parallel state-commit rewrite. The Besu Bonsai world-state commit phase (the state-root hashing that follows execution) was rewritten and proven bit-identical to the stock client four independent ways, including re-importing 253 real exported chain-2800 blocks with zero root mismatch. This is the strongest real-data evidence in this area and the nearest-term candidate for a rolling validator upgrade after audit, since its output is byte-for-byte the same. (Its multi-threaded variant was measured to anti-scale, so the shipped candidate is the single-threaded rewrite.)

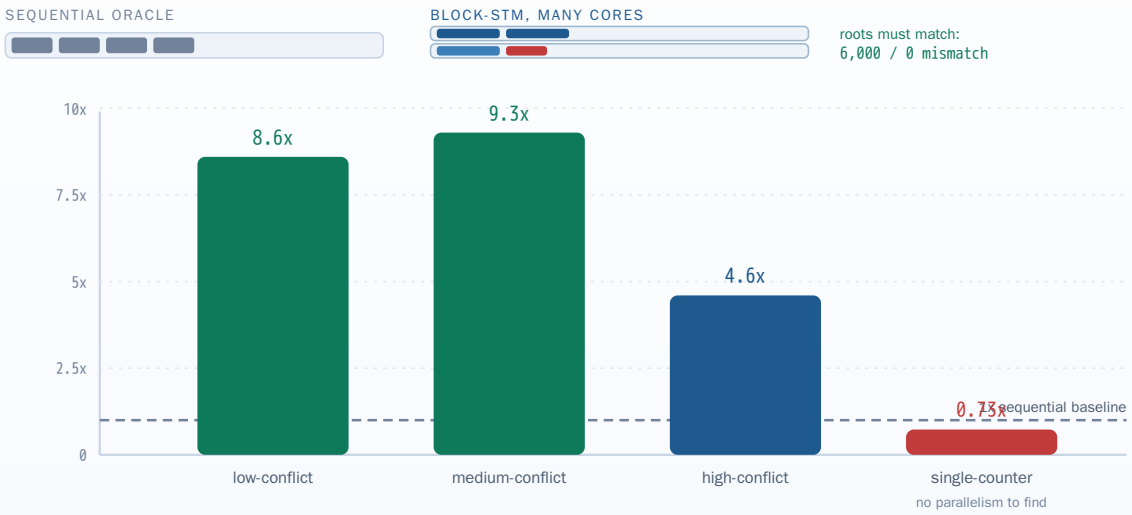
(2) A real parallel EVM. `aere-parallel-evm` runs real EVM bytecode through `revm` under a Block-STM multi-version-memory scheduler, proven bit-identical to sequential `revm` on real AERE contract bytecode: the WAERE ERC-20 and a constant-product AMM, with concurrent conflicting writes to a shared reserve slot, in-block `CREATE`, and reverts, at 1 to 32 worker threads. Beyond synthetic genesis, it now also executes REAL AERE mainnet blocks against live fork state: pinning the chain state at block N minus 1 over the public RPC, it replays block N in parallel and proves the result bit-identical to the canonical chain by matching every transaction's on-chain receipt, including gas used, the logs (down to a value embedded from the block timestamp), and the success or revert outcome. That makes it a validating parallel execution client on real chain-2800 state, not merely a synthetic benchmark. It now also computes a real Ethereum post-state root over its execution output, cross-checked byte-for-byte against reth's own Merkle-Patricia implementation and against real chain-2800 canonical account and storage roots proven with eth_getProof witnesses (EIP-161 empty-account handling included), which is the execution-and-commit core a block producer needs; the honest remaining pieces for full block production are a sparse post-state trie (the witness plumbing is already in place), the receipts root and header assembly, and a faithful fee model. It is correct today, a genuine parallel EVM rather than a synthetic model. On throughput the result is now a real net speedup, honestly scoped. The dominant cost was not the per-attempt context rebuild (that was fixed first) but two spurious-write hot keys that turned even genuinely-disjoint transactions into all-to-all conflicts: the block coinbase, which revm reads and writes on

every transaction, and any shared contract account, which revm marks touched on any storage write. This is the well-known Block-STM-over-EVM coinbase problem. Bypassing the coinbase read and skipping unchanged-account writes, both proven state-preserving and guarded so they can never commit a wrong state, removes the false conflicts. On a compute-heavy, many-users, disjoint-storage workload (the realistic one-shared-ERC-20-or-AMM, many-callers pattern, roughly 100,000 gas and up per transaction) `aere-parallel-evm` then runs bit-identical to sequential at up to about 7x on sixteen cores, with zero speculative aborts, crossing over sequential at two threads. The result holds under a real fee market as well: with a non-zero basefee and per-transaction priority fees, a commutative-coinbase accumulator (the block reward captured as an order-independent per-transaction delta, materialized lazily only for the rare transaction that actually reads the beneficiary) keeps the coinbase from becoming a shared hot key, so the same compute-heavy workloads still run about 6 to 7x on sixteen cores with zero aborts and bit-identical to sequential revm, coinbase balance included. Stated precisely: this is still a benchmark of the real engine, not yet L1-integrated, and cheap sub-30,000-gas transfers or genuinely-contended workloads do not benefit. A correct engine with a real, honestly-scoped throughput result.

(3) The scheduling proof. `aere-block-stm` is a from-scratch, zero-dependency Rust implementation of the Block-STM optimistic-concurrency algorithm (Gelashvili et al., arXiv:2203.06871, the same design family behind Aptos, Sui, and Monad): multi-version memory, speculative execution, validation, abort-and-retry. It is proven near-linear over a SYNTHETIC state model (balances and storage slots, four transaction kinds), which validates the concurrency control itself; artifact (2) is what marries that control to a real EVM. The near-linear figures below are that synthetic benchmark, not real-EVM throughput.

FIG 2.2

Block-STM speedup versus single-threaded sequential, by workload profile, over the SYNTHETIC scheduling benchmark (16-core box, 20,000-tx batches). Not real-EVM throughput; see artifact (2), aere-parallel-evm, for the real-EVM engine.



Single-machine microbenchmarks of the executor. Not a network throughput number and not a TPS claim.

What it shows. Low- and medium-conflict batches reach roughly 8.6x and 9.3x over single-threaded sequential; a high-conflict batch still reaches about 4.6x. The pathological batch, where every transaction increments the same slot, has no available parallelism, so Block-STM correctly serializes it and runs slightly slower (about 0.73x) because it pays for aborted speculation. That worst case is drawn openly. Every run also checks the parallel state root against a sequential oracle: 6,000 comparisons, zero mismatches.

Three properties make this more than a benchmark toy.

It is self-verifying. There is exactly one implementation of what a transaction does, and it is driven two ways: once by a sequential oracle that applies transactions in strict index order (the definition of the correct answer) and once by the parallel executor. Because both share the same transaction-semantics function, the parallel result can differ from the sequential result only if the concurrency control is wrong. Every run executes the batch both ways and refuses to emit a state root unless the two match, so a concurrency bug cannot silently produce a bad commitment.

Its correctness is proven, not asserted. A harness runs six workload profiles across 200 randomized batches at five thread counts each, 6,000 comparisons in total, including deliberately adversarial fully-conflicting workloads (an all-hot profile and a brutal single-counter profile). The reported result is zero mismatches: the Block-STM committed state

equals the sequential oracle in every case. Because the workloads are generated by a seedable PRNG, any case is exactly replayable.

Its measured speedup is honest, including the cases that look bad. On a 16-core box, with 20,000-transaction batches carrying a realistic simulated per-transaction compute cost, low-conflict and medium-conflict batches reach roughly 8.6x and 9.3x speedup versus single-threaded sequential execution. A high-conflict batch still reaches about 4.6x because only a fraction of its transactions touch the hot slot. A pathological batch in which every transaction increments the same slot has no available parallelism at all; Block-STM correctly serializes it and is actually slightly slower than sequential (about 0.73x) because it pays for aborted speculation. That worst case is stated openly, because no correct parallel executor can beat sequential when there is nothing to parallelize. A separate worker-count scaling run on execution-bound load is near-linear, at 1.94x, 3.61x, 6.19x, and 9.38x on 2, 4, 8, and 16 workers, each result bit-identical to sequential; that is an execution-phase win, and an end-to-end throughput win still needs real execution-heavy load plus a parallel commit. These are single-machine microbenchmarks of the executor. They are not a network throughput number and not a TPS claim.

The concrete place this executor runs is AERE's rollup layer. A TypeScript wrapper in the rollup sequencer serializes the prior L2 state slice and the ordered batch, shells out to the executor, and gets back a keccak256 state root; the `execute` path runs the batch both in parallel and sequentially and throws if they diverge, so a bad root can never reach the chain. That root is what a sequencer proposes on-chain. The rollup stack is AERE's Rollup-as-a-Service system: the canonical factory `AereRaaSFactoryV2` at `0xB7F8c754AC3155197d76f01857172bBd5a5F39Ab` (bonded in WAERE `0x7e84d7d66d5da4cfE46Da67CDEeB05B323e1f5e8`, with fees routed to `AereSink` at `0x69581B86A48161b067Ff4E01544780625B231676`) deploys a fixed per-rollup settlement contract, and the sequencer proposes each epoch's root there under an optimistic challenge window. Because the executor guarantees the parallel result equals sequential execution, the committed root is deterministic and reproducible, which is precisely what makes it safe as a rollup commitment and what would make a fraud proof constructible by re-running the same binary.

Two on-chain anchors accompany the executor. The first is `AereBlockSTMRegistry` at `0x98E2C3e615841919d173D8FF642514c5902E8A28`, an opt-in, advisory-only registry where a contract publishes the storage-slot read and write hints for its own function selectors so a scheduler can reduce aborts. It is metadata only: it holds no funds, changes no execution

semantics, has no admin over anyone else, and a wrong hint costs at most one extra abort. Critically, the registry has no runtime effect on mainnet today, because parallel execution is not mainnet-active; the Block-STM client is implemented with serial-equivalence correctness proven and demonstrated on testnet, and it was not part of the AerePQC mainnet fork, so it is not an unbuilt someday but also not yet on mainnet. The second anchor is a validity anchor. The bounded-VM `AereRollupValidity` at `0x38772063572DF94E90351e44ccbBEefD5F497fbd` verifies SP1 Groth16 validity proofs that AERE's deterministic rollup executor transitioned one state root to another by applying a specific batch, routed through the `SP1VerifierGateway` at `0x9ca479C8c52C0EbB4599319a36a5a017BCC70628`, and its epoch 0 was recorded on-chain from a real proof. It has since been superseded by a full-EVM validity anchor. The step that carries the real proof is `AereEVMValidityBatch` at `0x49D30a5999eA5b7f6eFf12196AB006316683Ff3C`, which recorded and verified a 16-block batch of real chain-2800 blocks (9111008 to 9111023) in a single Groth16 proof through the same gateway, a step up from the earlier single-block `AereEVMValidity` at `0x1f2CB0ebDBb21500e99868DbF1dB3abCcDd7DF26`, both verifying SP1 proofs of a real revm (via rsp / sp1-reth) executing genuine chain-2800 blocks. Both of those are in the canonical registry, and both are superseded there by anchors that bind the proven block to QBFT canonical history: `AereEVMValidityV2` `0x4884ad6617e320735b65D31915d1aBC884768D9B` (2,852 bytes of live code) and `AereEVMValidityBatchV2` `0xe154B8993FB54dB4418e03ef4a04894f29E690aE` (3,425 bytes). The honest reading of that pair is precise: the V2 anchors are the current ones by design, and neither has recorded a proof yet, so the demonstrated batch above remains the strongest thing actually proven on chain.

> **Honest scope of the parallel-execution work**
>
> The Block-STM microbenchmark executor models a bounded set of four transaction kinds (Transfer, Sweep, Increment, and a constant-product AMM swap) over a balance-and-storage map, and its correctness harness proves that specific executor, not arbitrary EVM bytecode. Two things that used to sit here have moved. Validity now proves a real full EVM through `AereEVMValidityBatch` `0x49D30a5999eA5b7f6eFf12196AB006316683Ff3C`, a real revm inside the SP1 zkVM with a 16-block batch of real chain-2800 blocks proved and verified on-chain in one Groth16 proof, a proof-of-approach scaling toward production rather than a live continuous sequencer. Block-STM is implemented in the forked client with serial-equivalence correctness proven and near-linear execution-phase scaling to 9.38x at 16 workers; it remains a testnet demonstration, was not part of the AerePQC mainnet fork, and is not mainnet-active, with throughput tuning ongoing. What remains genuinely ahead is wiring a live mempool into the executor end to end and a general on-chain fraud-proof verifier for the optimistic settlement path.

The honest scope of the parallel-execution work is drawn tightly. The Block-STM microbenchmark executor models a bounded set of four transaction kinds (Transfer, Sweep, Increment, and a constant-product AMM swap) over a balance-and-storage map, and its correctness harness proves that specific executor, not arbitrary EVM bytecode. Two things that used to sit on the roadmap here are now built and proven. First, validity no longer covers only the bounded VM: the full-EVM batch validity anchor `AereEVMValidityBatch` `0x49D30a5999eA5b7f6eFf12196AB006316683Ff3C` verifies SP1 Groth16 proofs of a real revm (via rsp / sp1-reth) executing genuine chain-2800 blocks, and a 16-block batch (blocks 9111008 to 9111023) has been proved and verified on-chain in one Groth16 proof through the SP1 gateway, a step up from the earlier single-block `AereEVMValidity` `0x1f2CB0ebDBb21500e99868DbF1dB3abCcDd7DF26`, so AERE now proves real full-EVM execution rather than a fixed transaction set. The remaining honest scope is that this is a proof-of-approach scaling toward production, not a live continuous sequencer, and transaction density is low because node state is pruned. Second, Block-STM is implemented in AERE's forked execution client with serial-equivalence correctness proven, the parallel result bit-identical to sequential and scaling near-linearly to 9.38x at 16 workers on execution-bound load, and demonstrated on testnet, so moving parallel execution into the base client is no longer a paper item; it was not part of the AerePQC mainnet fork and is not yet mainnet-active, and throughput tuning is ongoing. What remains genuinely ahead is wiring a live mempool into the executor end to end and a general on-chain fraud-proof verifier for the optimistic settlement path. For the full

derivation of the scheduler, the multi-version memory, the correctness harness, the benchmark methodology, and the settlement flow, see the companion specification `spec-parallel-execution`.

Client diversity: a complete second client, proven on an isolated testnet

TESTNET

For most of its life AERE ran a single execution client, Besu, and that was stated plainly as a limitation: a consensus-relevant bug in one client has no independent implementation to cross-check against. That work is now complete on an isolated test network, and the boundary below states exactly what that does and does not mean for mainnet. AERE has a second, independent execution client on a different codebase and language: a patched Nethermind 1.39.0 (a .NET client, versus the Java Besu producers) taught AERE's QBFT header and seal validation rules and wired to the five AerePQC precompiles. It peered with the live validators over devp2p, followed live chain 2800, and independently validated the full chain to genesis with zero invalid seals. It also full-state syncs from AERE's exact genesis rather than merely checking headers: it rebuilds world state and re-executes real chain-2800 blocks to the exact real-chain state roots, block for block. The five post-quantum precompiles execute byte-for-byte identically inside this second client, with 27 of 27 NIST KAT vectors accepting and rejecting correctly.

One result here deserves more emphasis than it has previously been given. Two independently written execution clients now agree on block hash and state root across the post-quantum activation block itself. Besu and Nethermind were compared at five heights spanning block 9,189,161 and produced identical values at every one. A hard fork that changes execution behavior is exactly where a second implementation is most likely to diverge, and the point of having one is to catch precisely that. Very few networks at Aere's stage can make this statement about a fork they have already shipped, and it is checkable by anyone who syncs either client. The correct scope for the claim is verification, not production, and the boundary below states that scope exactly.

On an isolated test network the second client no longer merely follows: it reaches QBFT consensus together with Besu as an equal validator. Both clients propose in turn, exchange Proposal, Prepare, Commit and RoundChange votes over Besu's own `istanbul`/100 devp2p sub-protocol, and every committed block carries a quorum of committed seals signed by both clients' validator keys, rather than one client producing solo while the other follows. Two fixes completed the work. The first is a Besu-style vote

relay: the engine consumed a peer's vote locally but never re-broadcast it, which is harmless on a full mesh but fatal on a sparse one, because a vote on the quorum-critical path died at the non-relaying node and a mixed set stalled whenever Nethermind signatures were needed to reach quorum. Holding the topology fixed and flipping only the relay turned every stall into progress (0 blocks to 26 and 0 to 20 in the two exactly-quorum scenarios), with zero forks. The second is the safety-critical acceptor-side round-change locking: an acceptor now rejects an equivocating proposer that proposes a conflicting value at a later round without a valid round-change certificate. Three distinct equivocation attacks are proven rejected, a legitimate re-proposal is accepted, and a paired control with the rule disabled shows the same attack accepted, which isolates the fix as exactly what prevents the fork. The rules are machine-checked as well as tested: an automated prover confirms that two different blocks can never both reach a commit quorum. A four-validator set at fault tolerance one survives the loss of a validator.

> **The honest boundary on client diversity**
>
> This is not client diversity on the live network, and the phrase "full client diversity" is not claimed for AERE mainnet. Every consensus result above, both clients proposing, the cross-client seal quorums, the vote relay, the acceptor locking and the machine-checked proofs, was obtained on an isolated test network with empty blocks, not on mainnet. On live chain 2800 Besu remains the sole producer of every block; the second client independently re-validates that history, and because it enforces the chain's rules on its own it has also twice stopped at divergences it refused to accept, which is exactly the cross-check a second predicate exists to provide; it produces nothing. Putting a second client into the live validator set is a deliberate, founder-supervised step that has not been taken, precisely because a bug in a producing client can halt or fork a live chain; it stays gated behind extended soaking and review (Chapter 9, Phase 4). Consensus cryptography is hybrid in the producing client in the anchor-certificate sense: every 32nd block must carry at least three valid Falcon-512 validator seals (f+1 of nine) (raised at block 14,961,456, August 21, 2026: the enforced minimum is now six of nine, a full 2f+1 quorum) under its hash, alongside classical secp256k1 ECDSA on every block; the second client independently re-validates the chain, certificates included. So the honest statement today is a single-producer mainnet with a second validating client, and a complete second client proven in consensus on testnet, a real step most single-client networks never take, stated without overclaiming what is live.

Architectural honesty

AERE's architecture is strong where it is standard and modest where it is new. The consensus is a mature BFT engine giving genuine deterministic sub-second finality, and it now runs on nine Foundation-operated validators with f=2 fault tolerance (commit quorum 6-of-9), though on a single producing execution client (Besu, with a second independent client that validates chain 2800 and reaches consensus with Besu on an isolated testnet, but produces no mainnet blocks), and a single operator, so that fault tolerance is against node outages rather than a dishonest operator. The EVM is at honest mainnet parity, which is a feature: no surprises for tooling or auditors. The parallel-execution engine is real, self-verifying, and benchmarked; the base layer executes sequentially on mainnet today, but Block-STM is now implemented in the forked client with serial-equivalence correctness proven and demonstrated on testnet (not part of the AerePQC mainnet fork), and validity is proven for a real full EVM through AereEVMValidity rather than limited to a bounded VM. There has been no external audit, usage is thin, and there is no exchange listing. Each of these limitations is named here on purpose, because for a young chain the accuracy of the description is itself part of the architecture. The subsequent chapters build on this base: Chapter 3 on the post-quantum verifier suite, and the companion specifications on the anti-MEV mempool, account abstraction, parallel execution, identity and compliance, and the fee flywheel.

03

THE DIFFERENTIATOR

On-Chain Post-Quantum Signature Verification

A live suite of six NIST post-quantum verifiers running on AERE's own EVM, now recording end to end on mainnet since the AerePQC fork put the native precompiles live, with the one honest boundary stated plainly: this is verification, not consensus.

Chain 2800 · Besu QBFT · companion deep-dive: `research/pqc-onchain-verification.md`

Final as math, safe when the math changes

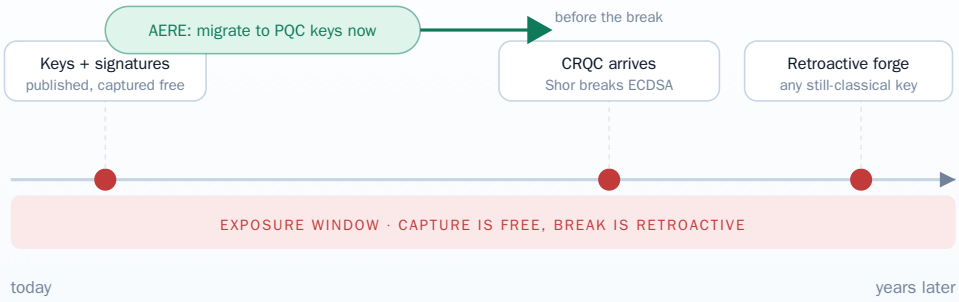
AERE's settlement promise has two halves. The first is that a transaction is final in well under a second, because QBFT gives single-slot BFT finality: settlement is as final as the math that orders it. The second half is what this chapter is about. Settlement has to stay safe when the math underneath today's signatures changes. Almost every account on almost every chain is secured by secp256k1 ECDSA, whose security reduces to the hardness of the discrete-logarithm problem, and Shor's algorithm solves that problem in polynomial time on a sufficiently large quantum computer. AERE's answer is to make the chain able to verify post-quantum signatures on its own EVM, today, so that account and settlement authorization can move onto quantum-resistant keys long before a cryptographically relevant quantum computer exists. Final as math, safe when the math changes.

The threat is timing, not a machine that runs today

The reason to act now is not a quantum computer that can break ECDSA today. It is the asymmetry between when data is captured and when it can be attacked. A public ledger publishes, permanently and to everyone, the two artifacts an attacker wants most: account public keys and the signatures made under them. An adversary can record the chain today and, at any later date when the hardware arrives, recover private keys from the exposed public keys and forge transactions against any account whose key material is already public. For an immutable public ledger that recording step is free and is already happening. Nothing is decrypted here, signatures are public; this is the ledger's analogue of the harvest-now, decrypt-later posture, retroactive forgery from exposed public keys, and it is

why a chain that expects to still be securing value in a decade should be able to verify post-quantum signatures well before it is forced to.

FIG 3.1 Harvest-now, decrypt-later: why the window opens before the machine exists



What it shows. The capture step is available today and permanent; the break can arrive years later and applies retroactively to any account whose key material is already public. AERE's response is to make migration to post-quantum keys possible now, ahead of the break, not to claim a quantum computer exists today.

Post-quantum schemes remove the lever Shor's algorithm pulls. Hash-based signatures (WOTS+, XMSS, SLH-DSA) rest only on the pre-image and collision resistance of a hash function; the best known quantum attack is Grover's, a quadratic speedup that standard parameter sizing absorbs. Lattice signatures (Falcon, ML-DSA) rest on structured-lattice problems for which no efficient quantum algorithm is known. Making the chain able to check these natively is the precondition for the migration.

Why the account and settlement layer, and not consensus

Post-quantum security can enter at two very different layers, and AERE is explicit about which one it has built. Changing consensus so that validators sign blocks with a post-quantum scheme is invasive: it touches the client, the gossip and finality protocols, and every operator. Changing the account and settlement layer is additive: a contract that verifies a Falcon or SLH-DSA signature deploys like any other contract, and a smart account can bind its authorization to a post-quantum key with no protocol change at all. AERE takes the additive path. Everything in this chapter lives above consensus. Consensus itself was changed separately and later, and the design was first proven on an isolated 4-validator QBFT testnet on which every validator Falcon-512-signs each block's commit hash and gossips its seal inside its QBFT Commit message, so each block embeds a gossiped, verified 2f+1 quorum certificate of distinct valid Falcon-512 seals that every node re-

verifies on import, blocking any post-fork block that lacks a valid quorum. Across two healthy runs (229 and 215 blocks) every post-fork block carried a valid 3-of-4 certificate with zero rejects, the full upstream consensus unit-test suite stayed green, the negative tests failed closed, and the block hash stayed byte-identical to upstream, while the classical secp256k1 ECDSA seal stayed the decisive liveness seal throughout. That testnet demonstration was then carried to mainnet 2800 in the anchor form: the seal registry is bound on-chain through an immutable anchor contract (nine entries, checkable with eth_getStorageAt and with the published manifests), and since 2026-08-14 an anchor block (every 32nd) does not finalize without a certificate of at least three valid Falcon-512 validator seals (f+1 of nine) (raised at block 14,961,456, August 21, 2026: the enforced minimum is now six of nine, a full 2f+1 quorum), with classical ECDSA finalizing every block. Correction published 2026-08-19: this document previously stated that from block 14,050,000 every block required a 2f+1 (six-of-nine) post-quantum quorum to finalize. The per-block Falcon quorum rule armed at that height is, in the shipped code, retired in favour of the anchor rules from block 13,014,000, and a re-reading of the code and of the chain (non-anchor blocks carry no Falcon seals) shows it changed no enforcement; that statement is withdrawn. What is enforced is the anchor certificate described here. The honest scope is stated at the front and repeated at the close: application-layer and account-layer quantum resistance is live, consensus is hybrid post-quantum only in the anchor-certificate sense since 2026-08-14, not purely post-quantum, and none of it has yet been audited by a third party.

> **Honest boundary: application and account layer, not consensus**
>
> Everything in this chapter sits above consensus. Consensus became hybrid post-quantum, in the anchor-certificate sense of chapter 3, by separate, explicitly approved activations (the anchor at block 13,014,000, the three-seal minimum from 2026-08-14), delivered through coordinated per-node activation heights rather than a re-genesis; nothing in this chapter is what did that, and nothing here should be read as implying it.

The live verifier suite

The suite spans both families NIST is standardizing for signatures: the stateful and stateless hash-based schemes, and the structured-lattice schemes. Each verifier is an ordinary immutable contract with no owner, no admin, and no upgrade path. Given a public key, a message, and a signature it returns exactly the accept-or-reject decision the

reference implementation returns, no more. Every address below is copied verbatim from the canonical registry (`sdk-js/src/addresses.ts`).

FIG 3.2 Six spec-complete verifiers, live on chain 2800

HASH · ONE-TIME
WOTS+
AerePQCVerifier
0x1cE2949e8cE3f1A77b178aF767a4455c08ec6F82
● RECORDS ON-CHAIN

HASH · MANY-TIME
XMSS-SHA2_10_256
AereXmssVerifier · RFC 8391
0x77b14E264D0bb08d304d4e0E527F0fCdFc88B112
● RECORDS ON-CHAIN

HASH · STATELESS
SLH-DSA-SHA2-128s
AereSphincsVerifier · FIPS 205
0xAfFc9F8d950969b46b54e77758BbFf7e000c87e6
● RECORDS ON-CHAIN

LATTICE · NTRU
Falcon-512
AereFalcon512Verifier
0x4E8e9682329e646784fB3bd01430aA4bA54D8fFC
● RECORDS ON-CHAIN

LATTICE · NTRU
Falcon-1024
AereFalcon1024Verifier
0xF0aFA59BaB2058e4B6e6B424b7f76750F1F66e36
● RECORDS VIA PRECOMPILE

LATTICE · MODULE
ML-DSA-44
AereMLDSA44Verifier · FIPS 204
0xf1F7A6Acd82D5DAf9AF3166a2F736EE52C5F85AE
● RECORDS VIA PRECOMPILE

What it shows. The full suite with each contract's verbatim canonical address, cryptographic family, and honest on-chain status. All six schemes record a result in a mined transaction on mainnet: four record in pure Solidity, and Falcon-1024 and ML-DSA-44 record through the native precompiles that went live with the AerePQC fork (block 9,189,161), for the reason set out in the gas-cap section below.

The hash family builds from one primitive. WOTS+ (AerePQCVerifier) is the one-time building block: it splits the 256-bit message hash into 64 base-16 digits plus a three-digit checksum, advances each supplied element to the end of its keccak256 chain, and accepts when the hash of the 67 chain-ends matches the committed public key. The checksum defeats the obvious forgery, since raising any message digit lowers the checksum and reversing that would require inverting keccak256. XMSS-SHA2_10_256 (AereXmssVerifier) is the honest many-time extension, reproducing the RFC 8391 verify path exactly: recover the WOTS+ leaf from the signature, walk a height-10 SHA-256 Merkle

authentication path to a candidate root, and accept when that root equals the published long-term root. SLH-DSA-SHA2-128s (AereSphincsVerifier) is the FIPS 205 standardization of SPHINCS+ in its smallest-signature parameter set, and unlike XMSS it is stateless: a few-times FORS layer selects a hypertree leaf pseudo-randomly, so one key signs an effectively unbounded number of messages with no signer state. The verifier implements FIPS 205 Algorithm 20 end to end, including the one load-bearing subtlety that separates FIPS 205 from round-3 SPHINCS+, the `base_2^12` big-endian FORS index extraction, and the validation discussion below returns to why that detail is provable.

The lattice family carries the differentiated weight. Falcon-512 (AereFalcon512Verifier) performs the full NTRU hash-and-sign check on-chain: it streams SHAKE256 to rejection-sample the challenge polynomial, decodes the 897-byte NIST public key and the 615-byte compressed signature with the reference `comp_decode`, recomputes the short vector by a negacyclic convolution over an in-contract number-theoretic transform, and accepts if and only if the squared l2 norm is at most the acceptance bound. Falcon-1024 (AereFalcon1024Verifier) is structurally identical at ring degree 1024. ML-DSA-44 (AereMLDSA44Verifier) implements the FIPS 204 module-lattice Fiat-Shamir-with-aborts verifier for Dilithium2: it samples the 4x4 matrix directly in the NTT domain via SHAKE128 rejection sampling, recomputes the commitment, and checks the challenge match and the infinity-norm bound.

Two composite primitives turn the verifiers from read-only checks into usable authorization, and are covered below: AereHybridAuth at `0xc20390C9656ECe1AE37603c84E395bC898b3FAA1`, and AerePQCAccountFactory at `0xd5315Ea7caa60d320c4f34b1bEd70dd9cc02CE58`. A precursor lattice core, AereLatticeVerifier at `0x60c06E6A3CC201B46A16650be096C6E45424dfD9`, implements ring arithmetic at a reduced degree (n=74) and is a demonstration core only; the spec-complete Falcon-512 verifier supersedes it for real use.

Why the verifiers can be trusted: KAT and ACVP conformance

A verifier is only as trustworthy as the vectors it is checked against, so each scheme is validated bit-for-bit against official, externally sourced fixtures that are committed to the repository. Validation runs in both directions: the valid vectors must accept, and the crafted-invalid vectors (tampered message, signature component, or key) must reject with the exact accept-or-reject pattern the reference produces. Falcon-512 and Falcon-1024 are checked against the Falcon round-3 NIST submission KAT response files; the Falcon-512 vector 0, for instance, carries a squared norm of 28,308,410 against the bound

34,034,726 and the on-chain verifier returns the same decision as the reference `crypto_sign_open`. ML-DSA-44 is validated against the NIST ACVP ML-DSA-sigVer-FIPS204 vectors, test group 8, where all 15 cases (3 valid plus 12 crafted-invalid) reproduce NIST's expected results, cross-checked by an independent from-scratch Python FIPS 204 verifier that also matches 15 of 15. SLH-DSA-SHA2-128s is validated against ACVP SLH-DSA-sigVer-FIPS205 test group 31, all 14 cases (2 valid plus 12 crafted-invalid). That SLH-DSA cross-check is the most instructive in the suite: the reference C matches 14 of 14 only after correcting the FORS index extraction to the FIPS 205 `base_2^b` decoding, and the unpatched reference matches only 12 of 14, failing exactly the two valid vectors. That isolates the single behavioral change between round-3 SPHINCS+ and standardized SLH-DSA and confirms the on-chain contract implements the standardized form. XMSS is validated against the deterministic vector from the XMSS reference implementation, cross-checked by independent Python and JavaScript reimplementations of the RFC 8391 path. The companion research paper enumerates the fixture files and their SHA-256 digests.

The honest boundary: the EIP-7825 per-transaction gas cap

AERE keeps Fusaka's EIP-7825 cap, which limits any single transaction to 2^{24} = 16,777,216 gas regardless of the block gas limit, deliberately, so the chain stays a functional peer of Ethereum mainnet rather than leaning on an unbounded local gas limit. That cap draws a precise and honest line through the suite. A verifier that records its result on-chain must complete the entire verification, plus the storage write and event, within 16,777,216 gas. A read-only `eth_call` is not a transaction and is not subject to the cap.

Four of the six schemes fit under the cap and record a full verification in an ordinary pure-Solidity transaction. The recorded figures are taken from committed deployment records, not projection: XMSS at 1,561,963 gas, SLH-DSA-SHA2-128s at 1,812,066 gas, and Falcon-512 at 10,492,455 gas, with WOTS+ dominated by a bounded number of short keccak256 chains. Two schemes do not fit in pure Solidity. A state-changing Falcon-1024 verification in Solidity lands at roughly 21.7M gas and ML-DSA-44 at roughly 52.9M gas (52,934,073, local estimate), both above the cap, because in pure Solidity their cost is dominated by SHAKE computed inside the EVM over a larger ring or a rejection-sampled matrix, so their pure-Solidity verifiers stay read-only references reachable via `eth_call` against the same official vectors. The path that removes the limitation, the native precompiles, is now live on mainnet: as of the AerePQC fork (block 9,189,161, 2026-07-12) the native Falcon-1024 and ML-DSA-44 precompiles record the same verification on-

chain within the cap, as the native-precompile section below sets out. This was always a limitation of the pure-Solidity execution budget, not of the cryptography.

FIG 3.3 Verify-and-record gas: pure Solidity vs the native precompile, against the 16,777,216 cap

What it shows. Bars are drawn on one linear gas axis, so the precompile results are the near-invisible green slivers at the left edge, which is the point: Falcon-1024 drops from roughly 21.7M gas to 145,496 gas, and ML-DSA-44 from 52,934,073 gas to 351,050 gas. Both Solidity bars cross the dashed EIP-7825 cap; both precompile results sit under 2.1% of it. Falcon-512 already records in pure Solidity at 10,492,455 gas, under the cap. Precompile figures are measured on mainnet 2800, live since the AerePQC fork (block 9,189,161).

From verifier to wallet: hybrid auth and a post-quantum smart account

The verifiers are not left as read-only curiosities. AereHybridAuth (`0xc20390C9656ECe1AE37603c84E395bC898b3FAA1`) authorizes an action only if both a secp256k1 ECDSA signature and a NIST Falcon-512 signature verify over the same 32-byte hash, with the Falcon leg delegated to the live Falcon-512 verifier. This is the conservative migration posture: an account stays as safe as classical ECDSA against today's adversaries while adding a post-quantum requirement, so that neither a broken curve nor a broken lattice alone suffices to forge. Its live `authorize` transaction used 10,299,873 gas, under the cap. AerePQCAccountFactory (`0xd5315Ea7caa60d320c4f34b1bEd70dd9cc02CE58`) goes further: it is a CREATE2 factory that deploys an ERC-4337 v0.7 smart account whose sole owner is a

Falcon-512 public key with no classical ECDSA fallback, where both the `validateUserOp` path and the EIP-1271 `isValidSignature` path are decided by the live verifier. A sample account owned by a real Falcon-512 key sits at `0xa42a5e7F72E46BadC11367650Ec34D676194326f`, and a full user operation through it used 10,278,313 gas, also recorded on-chain and under the cap. That is the practical result of Phase 1: not just a verifier that can check a post-quantum signature, but a wallet whose authorization is quantum-resistant end to end at the account layer.

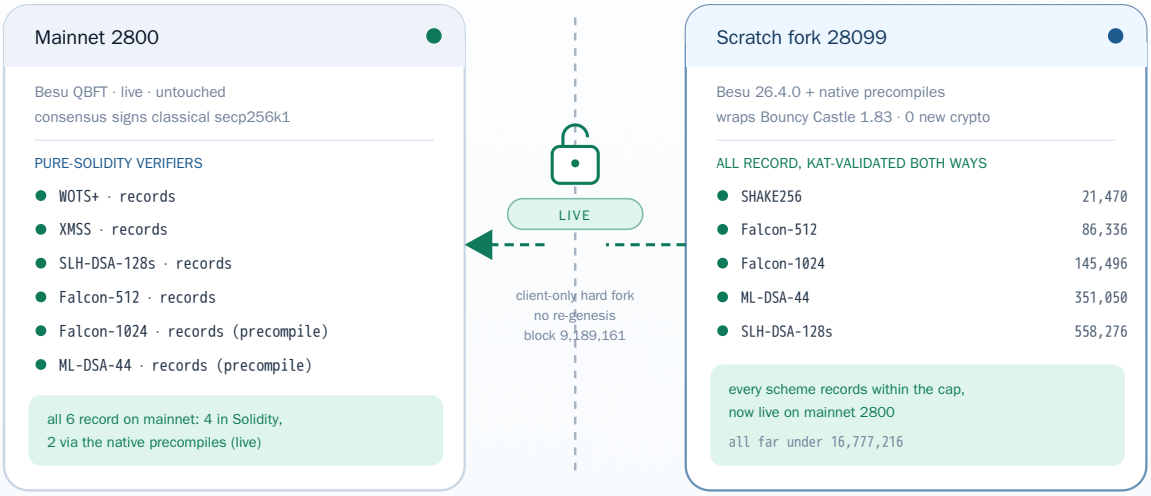
The native precompile path: live on mainnet as of the AerePQC fork

The reason Falcon-1024 and ML-DSA-44 do not fit in pure Solidity is almost entirely the cost of computing SHAKE and SHA-256-family sponge functions inside the EVM. Fusaka already demonstrates the fix for a different primitive: RIP-7951 exposes secp256r1 verification as a native precompile at a fixed cost roughly two orders of magnitude below the Solidity equivalent. The same approach applied to the extendable-output functions and the lattice inner loops moves every scheme in this suite under the cap.

That workstream is built, proven, and now live on mainnet. AERE forked Hyperledger Besu v26.4.0 and added native precompiles that wrap the audited Bouncy Castle 1.83 BCPQC verifiers already on the client classpath, introducing zero new dependencies and zero hand-rolled cryptography. The precompiles for SHAKE256, Falcon-512, Falcon-1024, ML-DSA-44, and SLH-DSA-SHA2-128s were validated bit-for-bit against the same official NIST KAT and ACVP fixtures the pure-Solidity verifiers pass, in both directions, including all the negative vectors (41 of 41 accept and reject cases), first on an isolated scratch chain (chain ID 28099) and then activated on mainnet 2800 by the AerePQC fork at block 9,189,161 (2026-07-12), where they now live at the reserved band 0x...0AE1 to 0x...0AE5. The measured full verify-and-record transactions, in gasUsed, are: Falcon-512 86,336, Falcon-1024 145,496, ML-DSA-44 351,050, SLH-DSA-SHA2-128s 558,276, and SHAKE256 21,470, all far under the 16,777,216 cap. The two schemes that were view-only in pure Solidity now record on-chain with wide margin: Falcon-1024 drops from roughly 21.7M gas to 145,496 gas (0.87% of the cap) and ML-DSA-44 from roughly 52.9M gas to 351,050 gas (2.09% of the cap), while Falcon-512 falls from 10,492,455 gas to 86,336.

FIG 3.4

Fork-precompile architecture: the AerePQC fork activated the native precompiles on mainnet 2800



What it shows. The precompiles were built and KAT-validated on an isolated scratch fork (chain ID 28099), then activated on mainnet 2800 by the AerePQC fork at block 9,189,161 (2026-07-12), a coordinated client-only hard fork with no re-genesis that added behavior only at the otherwise-empty precompile addresses. The gate is drawn open on purpose: the precompiles are live. An independent external audit of the client changes is still outstanding (see Chapter 9).

Live on mainnet as of the AerePQC fork

The candor here is deliberate and non-negotiable. The precompiles were proven first on an isolated scratch fork and are now live on AERE mainnet chain 2800: the AerePQC fork activated them at block 9,189,161 (2026-07-12) as a coordinated client-only hard fork with no re-genesis and no state migration, adding precompile behavior only at otherwise-empty addresses and also activating the extended EIP-2935 block-hash lookback (an 8,191-block window) in the same fork. One honest caveat belongs here rather than in a footnote: between blocks 9,182,380 and 9,189,160 the chain deviates from EIP-2935 as written, because Besu skips the history write on proof-of-authority chains lacking the three request-contract addresses. It is deterministic, it is upstream Besu behavior rather than a local patch, and it is identical across the clients we have compared, but an integrator reading the specification would not predict it, so the range is published in Chapter 9.4. The fork added the precompiles and EIP-2935 only; it did not change consensus, and Block-STM parallel execution was not part of it and remains a testnet demonstration. The accurate statement about mainnet now is that all six schemes record on-chain: four in pure Solidity, and Falcon-1024 and ML-DSA-44 through the native precompiles. Since 2026-08-14 AERE mainnet consensus is hybrid in a precise and limited sense: every 32nd block (an anchor block) does not finalize without a certificate of at least three valid Falcon-512 validator seals (f+1 of nine) (raised at block 14,961,456, August 21, 2026: the enforced minimum is now six of nine, a full 2f+1 quorum) under its block hash, with classical ECDSA (secp256k1) QBFT finalizing every block; post-quantum verification is also available as on-chain precompiles. The one caveat that stays: an independent external audit of the client changes is still outstanding (see the roadmap in Chapter 9).

Crypto-agility: swapping a scheme with no redeploy LIVE

"Safe when the math changes" is only credible if a scheme can be replaced without rewriting every contract that depends on it, so AERE makes that concrete with two small deployed contracts. `AereCryptoRegistry` (`0xaE6fC596bb3eCcbf5c5D02D67B0Ef065b3Afbaa5`) is an on-chain algorithm registry that maps an `algorithmId` to a record of `{verifier, status, gas, successor}`, and its `resolveActive()` view follows the successor chain so a caller always lands on the currently active verifier for a scheme. When a scheme is deprecated the registry owner marks it and points its successor at the replacement, and every reader that resolves through the registry migrates on its next call. Naming that owner correctly matters, because this is the authority the whole agility story rests on: measured on 2026-08-01, `owner()` on both the registry and the authorizer returns an operational deployment account rather than the Foundation account. Both are scheduled to move to governance, and until they do, the scheme-swap authority sits with an operational key rather than with

the Foundation or with a vote. `AereHybridAuthorizer` (`0x168F2A6a3071e7654CF1784a6f5d7BC8e1a582E0`) routes every signature check through `resolveActive`, so swapping the verifier behind a post-quantum scheme migrates all of its consumers at once, with no redeploy on their side.

The boundary is drawn exactly. Routing to a scheme that is already registered, or to a new contract verifier, needs no fork, because it is a registry write the Foundation can make. Adding a brand-new native precompile still requires a supervised client hard fork, of the kind that activated the PQC precompiles at block 9,189,161. Crypto-agility here means the account and application layer can adopt a new or successor scheme without redeploying its consumers, not that consensus or the client can be re-parameterized from a contract call.

Limitations, stated plainly

An on-chain post-quantum claim is easy to overstate, so the boundaries are drawn explicitly.

> Above consensus, not in it. The capabilities in this chapter are at the application and account layer. Consensus is hybrid post-quantum in the anchor-certificate sense since 2026-08-14, by a separate activation; nothing in this chapter is what did that.

> Centralized network today. The network runs nine validators under a single operator, and one execution client (Besu), with only two-fault Byzantine tolerance at n=9 (quorum 6-of-9), and the roadmap phases are the plan to change each of those facts in order.

> Validated, not externally audited. The verifiers are validated against official NIST vectors and cross-checked by independent reimplementations, which is strong evidence of functional correctness, but they have not undergone a third-party security audit yet; KAT conformance establishes that the accept-or-reject decision matches the reference on the tested vectors, not that every malformed-encoding edge case is handled.

> Thin usage. These are recently deployed primitives exercised by genuine on-chain transactions and `eth_call` results, not production load.

> Scheme-specific caveats. The XMSS verifier checks a signature against a given root; it does not and cannot enforce the signer's one-time-per-leaf state, so signing security still depends on the off-chain signer never reusing a leaf index. Only the smallest parameter sets are on-chain for ML-DSA (44) and SLH-DSA (SHA2-128s), and only the internal interfaces are verified, not the external-context or pre-

hash wrappers. The AereLatticeVerifier core is a reduced-degree (n=74) demonstration, not a spec-complete scheme.

> The breadth claim, hedged not headlined
>
> With those caveats stated honestly, and to the best of our present knowledge, we are not aware of any other public chain that verifies all of these on-chain. The full derivation, the fixture digests, the per-scheme gas accounting, and the precompile receipts are in the companion research paper, `research/pqc-onchain-verification.md`.

Chapter 3 contract registry (verbatim, chain 2800)

SCHEME	CONTRACT	ADDRESS	FAM
WOTS+ (one-time)	AerePQCVerifier	0x1cE2949e8cE3f1A77b178aF767a4455c08ec6F82	Ha
XMSS-SHA2_10_256	AereXmssVerifier	0x77b14E264D0bb08d304d4e0E527F0fCdFc88B112	Ha
SLH-DSA-SHA2-128s	AereSphincsVerifier	0xAfFc9F8d950969b46b54e77758BbFf7e000c87e6	Ha
Falcon-512	AereFalcon512Verifier	0x4E8e9682329e646784fB3bd01430aA4bA54D8fFC	Lat
Falcon-1024	AereFalcon1024Verifier	0xF0aFA59BaB2058e4B6e6B424b7f76750F1F66e36	Lat
ML-DSA-44 / Dilithium2	AereMLDSA44Verifier	0xf1F7A6Acd82D5DAf9AF3166a2F736EE52C5F85AE	Lat
Hybrid ECDSA + Falcon-512	AereHybridAuth	0xc20390C9656ECe1AE37603c84E395bC898b3FAA1	Co
PQC smart-account factory	AerePQCAccountFactory	0xd5315Ea7caa60d320c4f34b1bEd70dd9cc02CE58	Co

Every address above is copied verbatim from the canonical registry `sdk-js/src/addresses.ts`. AereLatticeVerifier (`0x60c06E6A3CC201B46A16650be096C6E45424dfD9`) is a reduced-degree (n=74) demonstration core, superseded by AereFalcon512Verifier for real use.

04 VERIFIABILITY

Zero-Knowledge and Verifiability

A multi-prover, route-by-selector, verify-then-record surface on mainnet 2800: two zkVMs, a KZG path, a generated Halo2 verifier, real recursion, and five application verifiers, each with its honest edge.

Chain 2800 · companion deep-dive: `research/specs/spec-zk-stack.md`

If the AERE thesis is "final as math, safe when the math changes," this is the chapter where "final as math" stops being a slogan and becomes a contract call. A zero-knowledge proof lets one party convince another that a computation ran correctly and produced a claimed output, without the verifier re-running that computation and often without the prover revealing its inputs. AERE hosts a working surface of these verifiers directly on mainnet 2800, so an ordinary account can hand proof bytes to an ordinary EVM contract, receive a cryptographic accept-or-reject verdict, and have that verdict recorded permanently. The full engineering treatment, with every verification path, trust boundary, and measured cost, lives in the companion document `research/specs/spec-zk-stack.md`; this chapter summarizes what the stack is, what each piece actually proves, and where the honest edges are.

Four framings hold everything below in place. First, this is an application and account layer capability, not a consensus feature. None of the verifiers here change how AERE produces or finalizes blocks; consensus is hybrid post-quantum in the anchor-certificate sense (chapter 3) by a separate mechanism, and these verifiers are not it. The verifiers are ordinary contracts that add verifiable computation on top of the chain, not inside consensus. Second, the network is small and centralized today: nine validators under a single operator, one client (Besu), no external audit, and thin on-chain usage of these verifiers (single-digit proof records for most of them). That is the baseline against which the engineering should be read, not a footnote to it. Third, addresses are printed verbatim from the canonical registry `sdk-js/src/addresses.ts`. An earlier edition of this chapter withheld the hex for several verifiers that were then awaiting promotion into that registry, and that is out of date: measured on 2026-08-01, the KZG verifier, the Halo2 verifier, the zkML verifier, the recursion aggregator and the state-root anchor are all in the canonical registry and all carry live code, so their addresses are printed below. Where the registry marks an

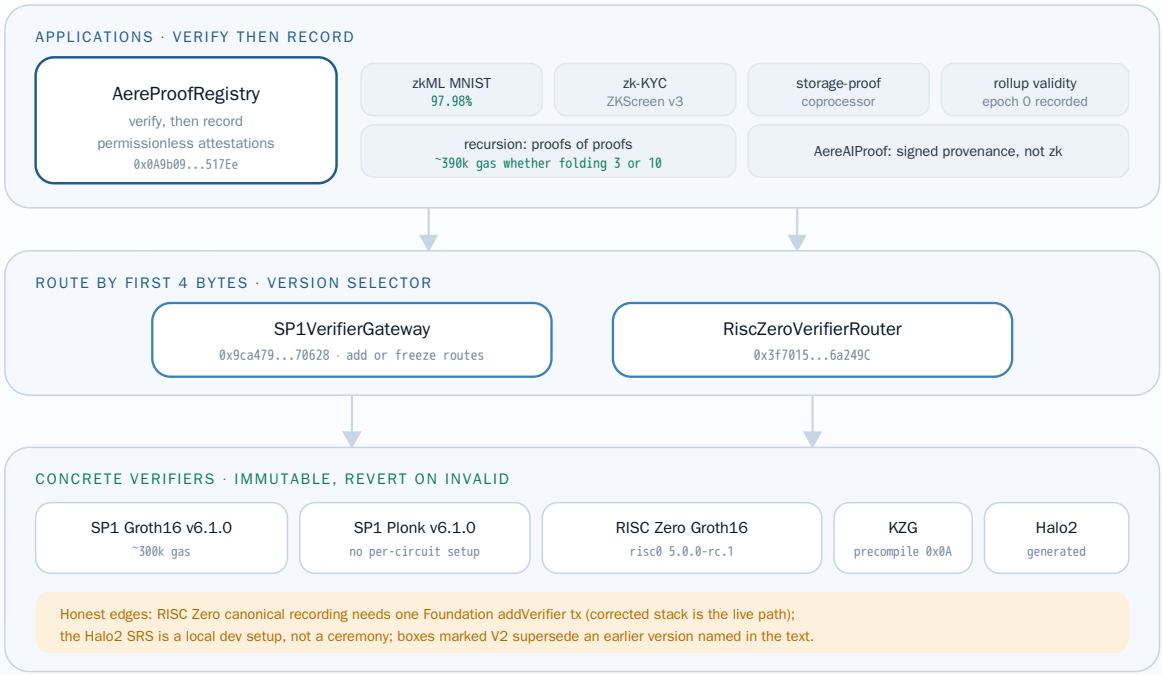
address as superseded, this chapter says so and names the replacement instead of quietly citing the old one. Fourth, the honest breadth claim is narrow: AERE hosts a multi-prover verification surface (SP1, RISC Zero, a raw KZG precompile path, and a generated Halo2 verifier) behind a route-by-selector gateway with a permissionless attestation registry on top. That is unusual to have all in one place. It is not a claim that no other chain can verify any one of these.

One shape: route by selector, verify then record

Every proof system in the stack follows the same two-layer pattern, and understanding it once explains the whole surface. The lower layer is a concrete verifier: an SP1 Groth16 or Plonk verifier, a RISC Zero Groth16 verifier, a generated Halo2 verifier, or the EIP-4844 point-evaluation precompile. Each takes proof bytes plus public inputs and either returns cleanly or reverts. The canonical SP1 and RISC Zero verifiers follow the Succinct and RISC Zero convention of returning nothing and reverting on an invalid proof, so AERE's wrapper contracts consistently catch that revert and translate it into a typed error, and a caller never mistakes empty return data for a pass.

FIG 4.1

The verification surface: applications verify then record, routers route by 4-byte selector, concrete verifiers stay immutable



What it shows. Application verifiers and the attestation registry hold only a router address, so a new prover version plugs in upstream with a single owner call and no downstream migration. Each concrete verifier a route points at is immutable, so what a frozen route computes cannot change. The honest limit, measured on the live bytecode on 2026-08-01, is that the route owner also chooses which contract a selector resolves to: the SP1 gateway carries addRoute and freezeRoute, the RISC Zero router carries addVerifier and removeVerifier, and the routes in use are not frozen today. The guarantee therefore holds against the verifier a route already points at, not against the party that owns the route. The amber strip lists the honest edges carried verbatim in the sections below.

The upper layer routes and attests. Concrete verifiers sit behind routers keyed on the first four bytes of the proof, a version selector. `SP1VerifierGateway` `0x9ca479C8c52C0EbB4599319a36a5a017BCC70628` routes an SP1 proof to the correct SP1 verifier version by its leading selector and refuses frozen routes. `RiscZeroVerifierRouter` `0x3f7015BC3290e63F7EC68ecF769b00aB296a249C` routes a RISC Zero seal the same way. Because routing is by selector, a new prover version is added with a single owner call on the router, and downstream application contracts that hold only the router address need no migration when a new prover ships. AERE's application verifiers all hold the gateway or router address rather than a concrete verifier, precisely for that reason. On top of the

routers, `AereProofRegistry` `0x0A9b09677DbE995ACfC0A28F0033e68F068517Ee` records each successful verification as a permanent, event-indexed attestation with permissionless program registration. The mechanism is real and the usage is thin, so the number is published beside it: measured on 2026-08-01 the canonical registry holds one registered program and one recorded proof.

The provers: SP1 and RISC Zero

SP1 (Succinct Labs) is a RISC-V zkVM. A guest Rust program compiles to an ELF, and its verification key binds that exact ELF; a proof then attests that the guest ran to completion on some private witness and committed a specific public output. Groth16 gives constant-size proofs at roughly 300k gas to verify; Plonk avoids a per-circuit trusted setup at somewhat higher cost. Both route through the same gateway above, with the production Groth16 verifier at `SP1VerifierGroth16_v6_1_0` `0xb5456d48bFdA70635c13b6CBE1Ad0310Dc0171aD`, the retained prior version at `SP1VerifierGroth16_v6_0_0` `0xa9BD3020bC9a9614F9e2BC1618153c9fB1890ca6`, and Plonk at `SP1VerifierPlonk_v6_1_0` `0x24a7a85E6D9A2b120F2730880bE7283dFB14d29B`. Every AERE application verifier in this stack that says "SP1" verifies through this one gateway, whose owner adds or freezes routes. Each concrete verifier behind a route is immutable, so what a frozen route computes cannot change; what the owner does control is which contract a selector resolves to, and the SP1 routes in use are not frozen today.

RISC Zero is a second RISC-V zkVM, included so the stack is genuinely multi-prover rather than single-vendor. A receipt attests that a guest with a given image ID produced a journal with a given digest. This is also where the stack is most honest about its own history. The originally deployed RISC Zero verifier, retained on chain as `RiscZeroGroth16Verifier_DEPRECATED` `0x95cB30f3bdb3187f39203A9907bf707Aef07a1FD`, was bootstrapped with a non-canonical control root that matches no released risc0 version, so it could never route a genuine receipt and every real seal reverted. The fix deployed a corrected verifier, `RiscZeroGroth16Verifier` `0xb6fD00D88Bf8B08d6371d2D98E0e239B89Ab5B9D`, carrying the genuine control root and selector for risc0 5.0.0-rc.1. A self-owned corrected stack, `RiscZeroVerifierRouter_Corrected` `0x62b96F7211F832f47d8Fc0D8317D5867B0Af43E5` and `AereProofRegistry_Corrected` `0x174F616E2048A71408E2491791ef77cd6913bbEf`, was deployed at the time to record a real RISC Zero factorization proof, and it remains on chain. Two corrections to the earlier edition of this text belong here in plain sight rather than in a footnote. First, the canonical `addVerifier` transaction described previously as staged and

unsigned was in fact executed: measured on 2026-08-01, the canonical router `0x3f7015BC3290e63F7EC68ecF769b00aB296a249C` resolves selector `0xef6cb709` to the corrected verifier `0xb6fD00D88Bf8B08d6371d2D98E0e239B89Ab5B9D` (4,664 bytes of live code), and `AereProofRegistry.risc0Router()` returns that same canonical router, so canonical RISC Zero recording is wired and live rather than roadmap. Second, that router, the registry and the SP1 gateway were described as Foundation-owned; `owner()` on all three does not return the Foundation account. They sit today under an operational deployment account and are scheduled to move to governance, which is tracked in Chapter 9. Recording the bug rather than quietly redeploying is deliberate, and the full trace lives in the spec.

The attestation registry

`AereProofRegistry` is the single log that both verifies and records proofs from SP1 and RISC Zero. It verifies first (delegating to the gateway or router, which revert on an invalid proof), then writes a record and emits an event indexed by program, so off-chain consumers can subscribe to exactly the program they care about. Program registration is permissionless: anyone registers their own SP1 verification key or RISC Zero image ID under a human-readable name. The Foundation can disable a malicious program but cannot disable a legitimate one, and disabling never retroactively invalidates a proof already recorded. View-only twins give the verdict without writing a record, for callers that need the answer but not a permanent attestation. Because the registry holds the gateway and router rather than concrete verifiers, a future prover version plugs in upstream with no registry migration.

Commitment primitives: KZG and Halo2

Two more verifiers extend the surface from zkVM proofs to polynomial-commitment proofs. `AereKZGVerifier` `0x6596307BD8f54d9A91FE364EBC3e594F200AC862` (1,988 bytes of live code, in the canonical registry as of 2026-08-01) forwards a 192-byte input to the EIP-4844 point-evaluation precompile at address `0x0A` and proves that a blob polynomial committed to by a 48-byte KZG commitment evaluates to a claimed value at a claimed point. It enforces both the precompile success flag and the exact canonical 64-byte return, so no empty account or stray contract can satisfy it by chance, then records the opening append-only with no owner and no admin. The honest scope is that this verifies a KZG opening; it does not by itself prove that a particular blob was posted for a particular transaction, and QBFT

produces no blobs, so this is a verification primitive available to applications rather than a data-availability layer the chain itself uses.

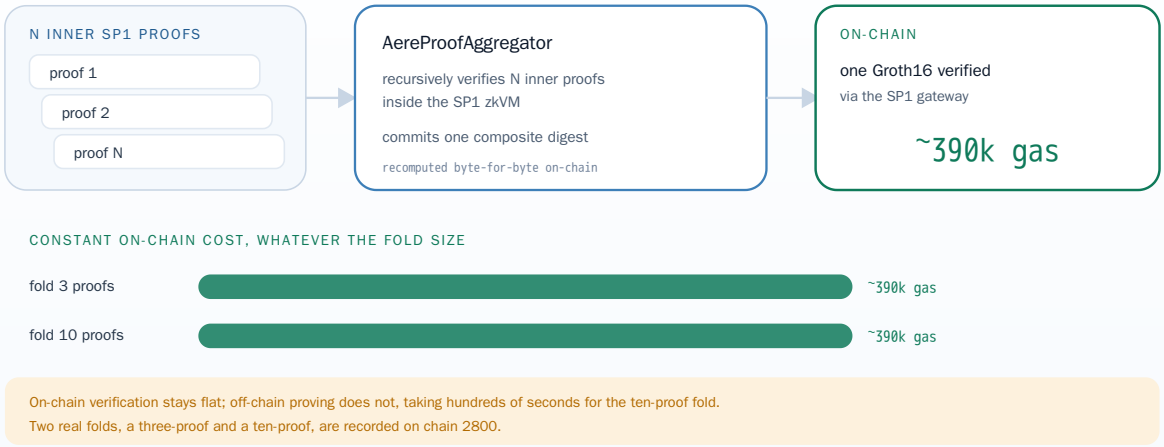
`AereHalo2CubicVerifier` `0x414Cfe640B2770856d3D7262a7a3729f5c07dC55` (4,804 bytes of live code, in the canonical registry as of 2026-08-01) with its companion `AereHalo2ProofAnchor` (deployed, address in `contracts/deployments/halo2-cubic.json`, and the one contract in this group still absent from the canonical registry) verifies a real Halo2 proof for a small standard-PLONK circuit, generated by the privacy-scaling-explorations Solidity verifier generator with the verifying key embedded in bytecode, and the anchor records each accepted proof append-only. The load-bearing caveat here is stated plainly: the KZG structured reference string was generated locally on the AERE build server, not from a public multi-party trusted-setup ceremony. That is sufficient to demonstrate that AERE can host and verify a real Halo2 proof on chain. It is not production-grade for a value-bearing circuit, where a ceremony SRS would be required, and replacing the dev SRS is an explicit roadmap item before any Halo2 circuit secures value.

Recursion: proofs of proofs

`AereProofAggregatorV2` `0x944dE720D6C9696Bdf10b6E2F00dd21ABB1A2b4C` (4,663 bytes of live code) is where the stack demonstrates recursion. It supersedes the first aggregator `0x6a260238890E740dB12b371E0C5d17a2470F84C5`, which the canonical registry now marks as deprecated because it counted the folded inner program keys without requiring them to be registered, so a fold containing an unregistered inner program was recorded rather than rejected. V2 requires every folded inner key to be an allowed program and reverts otherwise. Both contracts are live; the earlier one is left on chain and holds no funds. An aggregation guest recursively verifies N inner SP1 proofs inside the zkVM and commits a single composite digest binding every inner verification key and public-values hash. The on-chain contract recomputes that composite byte-for-byte from the supplied inner arrays, requires it to equal the proof's public values, and then verifies one Groth16 proof through the SP1 gateway. Two real aggregations are recorded on the first aggregator, a three-proof fold of distinct statements and a ten-proof scale fold, with the recognized inner-program count matching in each case, and V2 carries one recorded aggregation of its own (measured 2026-08-01: two records on the deprecated contract, one on V2). The measured point is that verifying the aggregate on chain costs roughly 390k gas whether folding three inner proofs or ten, so verification cost stays essentially flat as the folded set grows. The proving side is not free, and the artifacts record it honestly (hundreds of

seconds of off-chain proving for the scale fold), but constant on-chain verification cost is exactly what recursion buys, and it is the mechanism by which many statements can settle behind one succinct check.

FIG 4.2 Recursion: many proofs fold into one, and the on-chain verification cost stays flat

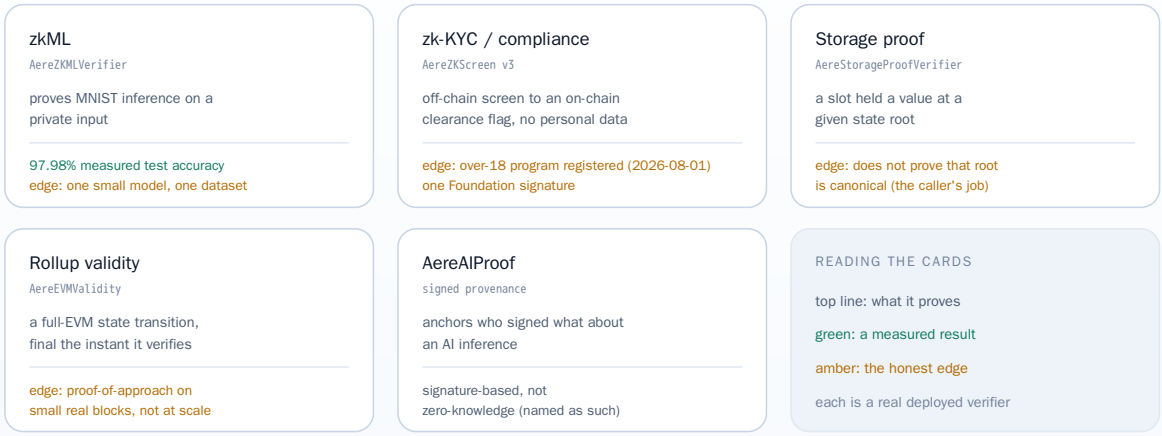


What it shows. An aggregation guest recursively verifies N inner SP1 proofs inside the zkVM and commits a single composite digest; the chain recomputes that digest byte-for-byte and verifies one Groth16 proof. On-chain cost stays essentially flat at roughly 390k gas whether the fold binds three inner proofs or ten, which is exactly what recursion buys. The proving side is not free, and the artifacts record that honestly.

Applied verifiability

The primitives above become useful through five application verifiers, each proving something a settlement layer actually needs.

FIG 4.3 The five application verifiers, each with the honest edge it carries



What it shows. The five application verifiers gathered in one view, each with the honest edge stated verbatim in the sections above. Four are zero-knowledge; `AereAIProof` is signature-based provenance and is labelled as such so the distinction is never blurred. Green marks a measured result, amber the honest edge.

zkML. `AereZKMLVerifier` `0xf1BF15d5018a35D21FBB4Ec868f062DF7C06783c` (3,658 bytes of live code, in the canonical registry) proves that a machine-learning model classified a private input without revealing that input. The reference model is a genuinely trained, quantized MNIST digit classifier whose forward pass runs inside the SP1 zkVM on a private image, exposing only the predicted digit, with the weight hash recomputed inside the zkVM. One boundary was previously stated too broadly and matters to integrators: the recording path `verifyAndRecord` checks that the model is registered and binds the chain id, the caller and the model hash, while the plain `verify` view checks only the proof and the input length and records nothing. Only the recording path supports the stronger statement that the classification came from the registered model. Its measured accuracy, recorded on-chain for transparency, is 97.98 percent on the MNIST test set. The honest scope is that this proves inference integrity for one small model on one dataset. It is a real measured classifier, not a hand-picked toy, and it makes no claim about large models.

zk-KYC and zk-compliance. `AereZKScreen` `0x3A097A459FD26aC79573aCB5adB51430e473C2f1` proves that a user passed an off-chain compliance screen inside the zkVM and anchors only a clearance flag, so no personal data touches the chain. Its soundness core is a root-binding check: each program binds a Foundation-set authorized root, and the proof must match it, which is what stops a prover from proving membership in an allowlist of their own making. The version history in the registry records that earlier deployments had exactly that

class of hole, and the current v3 is the fix, a candid detail preserved rather than erased. Alongside it, `AereCompliancePoolSP1Verifier` `0xE2D3fa91b680E835c971761ba75Fde0204AEF95E` is the production SP1 verifier for a compliance-preserving shielded withdrawal, consumed by `AereCompliancePoolV2` `0xB144c923572E5Ac1B6B961C4ccfec36917173465`, which currently holds zero deposits. Grouped with these for completeness is `AereAIProof` `0xFf92c669AbF4C1DAE31eBFCC017764036d9D97e6`, which is signature-based provenance, not zero-knowledge: it anchors signed statements about AI inferences and proves who signed what, not a hidden computation. It is named as such so the distinction is never blurred.

Storage-proof coprocessor. `AereStorageProofVerifierV2` `0x487021f9aE018657B7335ADC6Fc254717751a1Dd` (3,160 bytes of live code) proves that, at a given state root, a specific account's storage slot held a specific value. It supersedes the first storage-proof verifier `0xF9a1A183bEb3147D88dA5927301683fEbFb9362E`, which the canonical registry marks as deprecated because anchored and unanchored submissions shared one record key, so a record established through the anchored path could be displaced by one submitted through the unanchored path. V2 keeps the two apart and its canonical read returns the anchored record only. The earlier contract is immutable, holds no funds and is left on chain; do not read it as canonical. The SP1 guest RLP-decodes and hash-chains the real Merkle-Patricia proof of AERE's state trie, walking from state root to storage root to value with no Foundation-seeded side tree. A first real proof was recorded on chain against the earlier contract, and during the migration that same proof was replayed onto V2 and accepted by its verification path. The precise reading, because it is the whole point of the fix: replaying it through the unbound path left it in the submitter's own namespace, so V2's canonical read still reports nothing, exactly as designed. V2 holds no anchored record today, and establishing one needs a fresh proof anchored inside the anchor's block-hash reach. The primitive carries the single most important caveat honestly: the verifier proves the value against the given root, but it does not prove that the root is the canonical root of that block. That canonicity is the caller's job. The companion `AereStateRootAnchorV2` `0x0b959402283c0D2B9db4FfD7823f99782462e488` (2,652 bytes of live code, in the canonical registry, and the only writer whose submissions the verifier treats as anchored) is the building block that closes that gap, recovering a block hash strictly from the `BLOCKHASH` opcode. Chain 2800 served `BLOCKHASH` for the last 256 blocks, and the EIP-2935 history contract that extends that lookback to an 8191-block window is live on mainnet, deployed with the AerePQC fork (block 9,189,161, 2026-07-12) alongside the post-quantum precompiles. Measured on 2026-08-01 it carries 83 bytes of code and the window was walked end to end: offsets of 1, 255 and 8,190 blocks behind the head each return the

canonical block hash reported by `eth_getBlockByNumber`, and 8,191 blocks back is the last that answers. One honest gap remains and is named rather than implied: the consumer contract that would use that window, `AereHistoryStateRootAnchorV2` `0x4C1F9daD515c7990b65F9833960CA4315216330C`, reads back an immutable `VERIFIER()` of `0xF9a1A183bEb3147D88dA5927301683fEbFb9362E`, the deprecated verifier above, so the extended-lookback path is deployed but not yet wired to the corrected verifier. Repointing it needs a fresh deployment, and until then the trustless canonicity available in practice is the 256-block anchor path, not the 8191-block one.

Rollup validity anchor. AERE records SP1 Groth16 validity proofs that its deterministic executor transitioned one state root to the next by applying a committed batch, and unlike an optimistic rollup with a challenge window, an epoch recorded here is final the instant the proof verifies. The first anchor, the bounded-VM `AereRollupValidity` `0x38772063572DF94E90351e44ccbBEefD5F497fbd`, recorded its epoch 0 from a real proof over a fixed four-kind transaction set. It has since been superseded by a full-EVM validity anchor, `AereEVMValidity` `0x1f2CB0ebDBb21500e99868DbF1dB3abCcDd7DF26`, which verifies SP1 proofs of a real revm (via rsp / sp1-reth) executing genuine chain-2800 blocks; two real blocks have been proved and verified on-chain through the SP1 gateway. This is the most literal expression of "final as math" in the stack, and it now proves real full-EVM execution rather than a bounded VM. The current anchors bind that proof to the canonical chain: `AereEVMValidityV2` `0x4884ad6617e320735b65D31915d1aBC884768D9B` verifies an SP1 proof that a block's EVM execution is valid and ties it to the QBFT-canonical block at that height, and `AereEVMValidityBatchV2` `0xe154B8993FB54dB4418e03ef4a04894f29E690aE` extends the same check across a batch of consecutive blocks. The honest remaining scope is that AereEVMValidity is a proof-of-approach on small real blocks today; scaling it to full-throughput production rollup batches is ongoing, and nothing built on it should imply a production-scale validity rollup yet.

> **Honest scope of the whole surface**
>
> What it is not, stated so nothing reads as spin: it is not a consensus feature, it does not run on an audited or decentralized network yet, and its Halo2 SRS is a dev setup. Two former boundaries have moved: its rollup validity proof now covers a real full EVM through `AereEVMValidity` rather than a bounded VM, a proof-of-approach on small real blocks still being scaled to production batches; and trustless storage-proof canonicity beyond the 256-block window is addressed by AERE's extended EIP-2935 lookback to an 8191-block window, now live on mainnet as of the AerePQC fork (block 9,189,161).

Honest scope and what is next

Taken together, the stack is a route-by-selector, verify-then-record surface spanning two zkVMs, a KZG precompile path, a generated Halo2 verifier, real recursion, and five application verifiers, with a permissionless attestation registry binding them. What it is not, stated so nothing reads as spin: it is not a consensus feature, it does not run on an audited or decentralized network yet, and its Halo2 SRS is a dev setup. Two boundaries that used to sit here have moved: its rollup validity proof now covers a real full EVM (a real revm inside the SP1 zkVM, verified on mainnet through `AereEVMValidity`) rather than a bounded VM, as a proof-of-approach on small real blocks that is still being scaled to production batches; and trustless storage-proof canonicity beyond the 256-block window is addressed by AERE's extended EIP-2935 lookback to an 8191-block window, now live on mainnet as of the AerePQC fork (block 9,189,161). Two more items that used to head the roadmap are done and are removed rather than left to age: the canonical registry can already record RISC Zero proofs, because the router transaction described earlier as unsigned was executed and the route resolves live, and the verifiers that were awaiting promotion are in `sdk-js/src/addresses.ts`, with only `AereHalo2ProofAnchor` still outside it. The near-term roadmap is what is left: promote that last anchor into the registry, replace the Halo2 dev SRS with a ceremony SRS before any value-bearing circuit, repoint the 8191-block history anchor at the corrected storage-proof verifier, commission the first external audit of the now-live native PQC client changes, and scale the full-EVM validity prover from small real blocks to production rollup batches. Every measured figure quoted above is a single-run value drawn from a repository artifact rather than a benchmarked average, and every claim here is meant to be checked against the source, not taken on trust. That is the point of a verifiability stack, and it is the same discipline the rest of this whitepaper holds itself to: final as math, and honest about which math is not finished yet.

05

THE FIRST FIVE MINUTES

Accounts and Onboarding

An account you unlock with a fingerprint and no seed phrase, a first transaction you pay for with no native AERE, and an honest map of which parts are live, source-complete, or roadmap.

Chain 2800 · companion deep-dive: `spec-account-abstraction`

The hardest problem in crypto is not throughput or finality. It is the first five minutes. A new user is handed twelve random words, told that losing them means losing everything and that sharing them means the same, asked to buy a native token from somewhere else before they can do anything at all, and then left to sign an opaque hex blob with no way to know what it does. AERE treats that experience as a bug to be engineered away, not a rite of passage. The account layer is where the network's thesis, final as math, safe when the math changes, has to meet an actual human with a phone. This chapter describes how a person gets an AERE account with no seed phrase, transacts without holding any native AERE, upgrades an existing wallet in place, hardens it with session keys and social recovery, gives a non-fungible token its own wallet, and, at the far edge of the roadmap, binds authorization to a post-quantum key. The companion specification `spec-account-abstraction` carries the byte-level detail; here we summarize what is live, what is source-complete but not yet deployed, and where the seams honestly are.

What the chain gives the wallet layer

Two protocol features do the heavy lifting, and both arrive from the network's EVM ruleset, Pectra plus Fusaka, which targets functional parity with Ethereum mainnet.

The first is the secp256r1 (P-256) verification precompile introduced by RIP-7951, present unconditionally at address `0x100` from the Fusaka activation block onward. P-256 is the curve that every consumer secure enclave already speaks: Apple Face ID and Touch ID, Windows Hello, Android biometric keystores, hardware keys such as YubiKey, and the EU Digital Identity Wallet. Before this precompile, checking one of those signatures inside a contract meant running the curve arithmetic in Solidity, which is prohibitively expensive. With it, the check is a single native call cheap enough to run on every transaction. That one

change is what makes a passkey a first-class on-chain signer, and it is what lets a user authorize account operations with a fingerprint instead of a mnemonic.

The second is EIP-7702, which lets an existing externally owned account sign an authorization that makes it execute a designated contract's code while remaining controlled by its original key. This is the upgrade-in-place primitive: a plain wallet can gain batching, session keys, and passkey support without moving a single token to a new address.

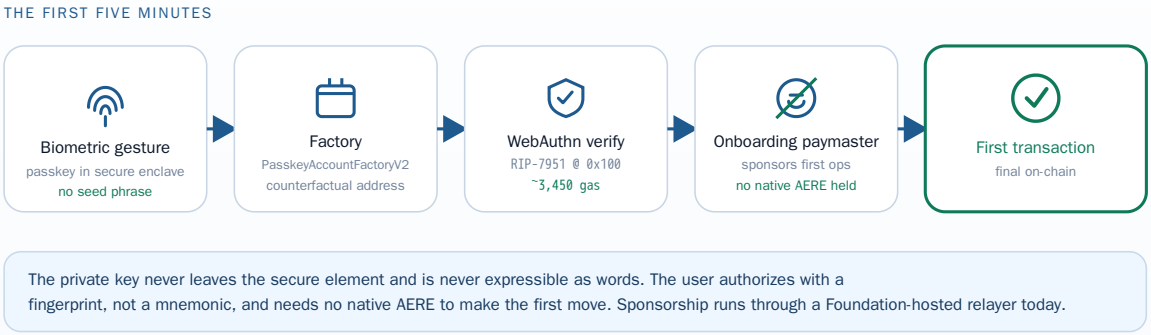
One constraint from the chain shapes the designs below and is worth stating up front. AERE's own EntryPoint keeps a strictly sequential, single-dimension nonce per sender rather than the two-dimensional nonce keys canonical ERC-4337 uses to route between signature validators. Accounts that want to switch validators therefore route on a signature prefix byte instead of a nonce key. It is a small thing, but it is the kind of honest seam this document prefers to name rather than paper over.

Passkey smart accounts (live) LIVE

The production onboarding path is a passkey smart account: a contract wallet whose owners can be WebAuthn P-256 keys, ordinary key-based accounts, or a mix of both. A user creates one with the same biometric gesture they already use to unlock their phone. There is no seed phrase to record, because the private key never leaves the device's secure element and is never expressible as words.

FIG 5.1

Onboarding without a seed phrase and without native AERE: passkey to a sponsored first transaction



The private key never leaves the secure element and is never expressible as words. The user authorizes with a fingerprint, not a mnemonic, and needs no native AERE to make the first move. Sponsorship runs through a Foundation-hosted relayer today.

What it shows. The two protocol primitives (the RIP-7951 P-256 precompile at `0x100` and the paymaster stack) combine so a person creates an account with a biometric gesture and transacts with no native token. The counterfactual factory means funds can be sent to a wallet before it is deployed. The honest seam: sponsorship and relaying currently depend on a Foundation-hosted relayer, and the two-EntryPoint split described below is real.

Accounts on this line are minted by `AerePasskeyAccountFactoryV2` at `0x5FFa9a6487DA4641a1A1e7900ff2bD4525D34fdA`, a deterministic factory that can compute an account's address before it exists, so funds can be sent to a wallet that has not yet been deployed. Each account is pinned to the account-facing EntryPoint, `AereEntryPointV2` at `0x8D6f40598d552fF0Cb358b6012cF4227B86aF770`. The original single-owner factory, `AerePasskeyAccountFactory` at `0xfB0eF980667A79Fe1AB69c5f2d512118F1B30739`, remains on chain for its first demo account only. One correction belongs here in plain sight: our own canonical registry now flags the live V2 factory as flawed in the signature-validation path of the accounts it produces, and says new wallets should not be created on it. A corrected factory, `AerePasskeyAccountFactoryV2Fixed` `0x8FA2B45D63EBaa4551A55d7f33a82d597e423428`, is deployed with 9,675 bytes of live code and is inert until a supervised swap points the product surfaces at it. Until that swap lands, read the passkey wallet line as in migration rather than as settled, and treat the address above as the historical factory rather than as a recommendation.

Under the hood, WebAuthn verification is a precompile-only port of Coinbase's audited `webauthn-sol` library. It rejects malleable signatures, confirms the assertion is a genuine `webauthn.get` response, reconstructs the challenge and binds it to the on-chain operation, requires a verified user gesture (a biometric or PIN, not a bare presence tap), and finally

calls the `0x100` precompile. The ownership model stores each owner as raw bytes, a key-based account or a P-256 public key, and every owner-management action must arrive as an authenticated self-call, so no outside party can add or remove owners. Any single owner can manage the set, and the set can never be emptied. That is the account's recovery story in its shipped form: keep at least one owner, and pre-register a backup key, for example a hardware wallet, so that losing the primary passkey does not lose the account. Guardian-based recovery is a separate module discussed further down.

An account can authorize a call three ways, all resolving through the same signature-verification routine: the standard ERC-4337 path driven by a bundler, a direct passkey path where the Foundation relayer (or the user) submits the call without a bundler, and EIP-1271, which lets the wallet produce contract signatures accepted by Permit2, OpenSea, Snapshot, and ordinary dapp logins. Each path binds its signature to this account, this chain, and this exact call, so a signature can never be replayed elsewhere.

The EntryPoint, stated honestly

AERE actually ships two EntryPoint contracts, and integrators should know the difference. `AereEntryPointV2` is the account-facing coordinator: it runs the canonical validate-then-execute batch flow, computes the standard operation hash bound to the chain, keeps per-account deposits, and emits the standard events that bundlers and indexers subscribe to. It is shaped so external bundlers such as Pimlico, Stackup, or ZeroDev can target it, while AERE runs its own relayer during bootstrap. Alongside it, a lightweight `AereEntryPoint` at `0x19773ba45287A64B05d0BCBD59D1371BF51Bd5D2` carries the deposit-and-stake surface that the paymasters bind to.

> **A named seam: two EntryPoints**
>
> The honest consequence, spelled out in the companion spec, is that the account-facing EntryPoint charges a sponsoring paymaster's deposit but does not itself invoke that paymaster's policy checks; gasless sponsorship therefore currently runs through the relayer plus lightweight-EntryPoint path where the policy hook is actually called. Unifying the two into a single EntryPoint that validates both account and paymaster in one batch is a tracked roadmap item. We would rather document the seam than imply a byte-for-byte reimplementation that does not yet exist.

Gas that feels like it is not there

"Gasless" on AERE means the user needs no native AERE to transact. Fees on the network are negligible to begin with, and the paymaster stack removes even that friction by letting the Foundation, a dapp, the user's own stake, or an ordinary token cover the cost. Four paymasters are live, all sharing a common base and each enforcing its own sponsorship policy.

FIRST TOUCH

Onboarding

`AereOnboardingPaymaster`

`0x4058E406475Dbed7056Aee0c808f293F05fEa879`

Sponsors a hard-capped number of ops per address under a sitewide daily ceiling for Sybil resistance.

● LIVE

DAPP-FUNDED

App factory

`AereAppPaymasterFactory`

`0xEC22603E8712cBc5c31E53370D10f1a80CcB4DF0`

Any dapp permissionlessly deploys and funds its own paymaster from its own treasury.

● LIVE

PAY-IN-TOKEN

Token V2

`AereTokenPaymasterV2`

`0x217f56a5b0C7f35abe4D2fff924A6c13B85d7243`

Settle fees in a whitelisted token at an oracle price with a bounded markup.

● LIVE

STAKE-FOR-TX

Stake quota V2

`AereStakeQuotaPaymasterV2`

`0xE50464ca7E8E7F542D1816B3172a2330cFE384E8`

A user's effective AERE stake buys a daily quota of sponsored operations that resets each day.

● LIVE

`AereOnboardingPaymaster` at `0x4058E406475Dbed7056Aee0c808f293F05fEa879` is the first touch. It sponsors a small, hard-capped number of operations per address under a sitewide daily ceiling for Sybil resistance, letting a fixed and modest Foundation budget carry a large number of first-time users through their opening interactions before they hold anything at all. `AereAppPaymasterFactory` at `0xEC22603E8712cBc5c31E53370D10f1a80CcB4DF0` lets any dapp permissionlessly deploy and fund its own paymaster from its own treasury, the standard pattern for a game or marketplace that wants to pay its users' fees. `AereTokenPaymasterV2` at `0x217f56a5b0C7f35abe4D2fff924A6c13B85d7243` lets a user settle fees in a whitelisted token instead of native AERE, converting at an oracle price with a bounded markup. `AereStakeQuotaPaymasterV2` at `0xE50464ca7E8E7F542D1816B3172a2330cFE384E8` offers a stake-for-transactions model: a user's effective AERE stake, read live from `AereStakingV2` at `0x1D95eF6D17aeAB732dF914Ba2d018c270BC155FC` and `AereLockedStaking` at `0x21108c28A849b05aE6b7a3a5bc435C9Bc897E7Ad`, buys a daily quota of sponsored operations that resets each day.

Both V2 token and stake-quota paymasters are hardened redeploys that fixed confirmed bugs in their V1 predecessors, and both V1 contracts, along with the legacy passkey factory, remain on chain but deprecated. Integrators must point at the V2 addresses above. The companion spec documents each fix; the shape of the honesty is that these contracts were reviewed, found wanting, and replaced in the open rather than quietly.

Upgrading a wallet in place (EIP-7702, live) LIVE

Not everyone wants a new address. A user who already holds an ordinary key-based wallet can keep it and gain smart-account behavior by delegating to `AereDelegate7702V2` at `0xC6d18e0Ce1B6467d952e31315505E0190a4A3310`, the audited replacement for the earlier V1 at `0x5673D92080efbd0987402E9335c14200d0a5EaeF`. The V1 is deprecated and should not be delegated to, because a session key on it could escape its intended scope; V2 closes that gap. Because EIP-7702 transmits only code and not storage, every account delegating to this one shared implementation gets independent storage at its own address and can re-point or revoke its delegation at any time with no migration. The delegate gives that wallet atomic batching (many actions in one confirmation), an optional registered passkey for relayer-paid calls, and scoped session keys with an expiry, a target and selector allowlist, a per-key spend cap, and a kill switch. A companion index, `AereDelegationRegistry` at `0x6c25c07D134713b6C2F8E19D807423f022903D63`, is a purely informational, permissionless directory that dapps read to show a "smart-wallet enabled" badge; it holds no funds and has authority over nothing.

Modular accounts, session keys, and social recovery (in development)

IN DEVELOPMENT

The next-generation account line is source-complete but not yet on mainnet, and this document will not print an address for a contract that does not exist on chain 2800. It comprises an ERC-7579 modular account with pluggable validator and executor modules, two generations of a session-key validator, and a social-recovery module. The session-key validators grant a scoped key a time window, a spend cap, and an allowlist of permitted calls, and the V2 iteration closes a confirmed cap-bypass and forbids a session key from ever calling account-admin functions back into its own account. The social-recovery module is the guardian-based flow the live passkey account deliberately lacks: an M-of-N guardian set can, after a timelock and a threshold of approvals, rotate the account's owner, with the current owner able to cancel at any time before execution. When this line is audited and

deployed, its addresses will enter the canonical registry and this chapter will be updated to describe it as live. Until then it is honestly labeled in development.

Token-bound accounts (ERC-6551, live) LIVE

ERC-6551 gives every non-fungible token its own wallet. These are account primitives, not tokens; they mint nothing and have no supply. The `ERC6551Registry` at `0x7fFdA0AcDeB919938dB91dbEa779D841c833EF68` is the verbatim reference registry, a stateless, owner-less factory that deploys a deterministic account bound to a specific token. The implementation those accounts point at is `AereTokenBoundAccount` at `0xBc5e24180f3F75DC6b1965E4aaD3D4F184b6F9E2`, whose sole source of authority is whoever currently holds the bound token, read live. The holder can execute calls and produce EIP-1271 signatures on the account's behalf, and because the account also supports the ERC-4337 path, a paymaster can sponsor the token wallet's fees too. A live demonstration binds token number one of `AereNFT` at `0x3f9A9D9CAB005327869396C69bE226ef98039f1c` to the account at `0x82D24cC4E09CaBfD9B00233438B8B6321CBC13Dd`. The practical effect is that a character, a membership, or a real-world-asset token can own and move its own portfolio, and transferring the token transfers the whole wallet with it.

A post-quantum wallet (live, honest scope) LIVE

At the far edge of the account roadmap sits the most experimental primitive in the stack and the one whose scope must be stated most carefully. `AerePQCAccountFactory` at `0xd5315Ea7caa60d320c4f34b1bEd70dd9cc02CE58` creates a working ERC-4337 smart account whose sole owner is a NIST Falcon-512 lattice public key, with no classical fallback. Every authorization, whether a bundled operation, a contract-signature check, or a direct call, is decided by the live on-chain `AereFalcon512Verifier` at `0x4E8e9682329e646784fB3bd01430aA4bA54D8fFC`, which runs real lattice cryptography rather than a stub. A sample account owned by a genuine Falcon-512 key lives at `0xa42a5e7F72E46BadC11367650Ec34D676194326f`, and both a contract-signature acceptance and a full operation through the account-facing EntryPoint were demonstrated end to end.

The honest scope is exactly this. The key is Falcon-512, NIST security level 1, not Falcon-1024. A single post-quantum authorization is genuinely heavy: it consumes on the order of ten million units of on-chain execution work, well under the network's per-transaction ceiling of 16,777,216 units, but far above an ordinary transfer. That cost is why the two

largest verifiers in AERE's broader suite, Falcon-1024 and ML-DSA-44, were read-only views on mainnet, rather than usable account owners, until the native precompiles arrived. A native precompile that makes lattice verification cheap was validated against every NIST known-answer test on an isolated Besu scratch fork (chain 28099), where a full Falcon-512 verify-and-record costs only tens of thousands of units, and it is now live on AERE mainnet: the AerePQC client fork activated the five native PQC precompiles at block 9,189,161 (2026-07-12), so a Falcon-512 verify-and-record costs a fixed precompile fee on chain 2800 today, and the two heaviest schemes, Falcon-1024 and ML-DSA-44, moved from read-only views to on-chain records. Crucially, this post-quantum property lives at the account and application layer. It secures how this wallet authorizes operations. It does not touch consensus, which is hybrid post-quantum in the anchor-certificate sense by a separate mechanism. On the breadth of the verifier suite this account draws on, we are not aware of any other public chain that verifies all of these on-chain, and that is the claim, stated with exactly that hedge and nothing stronger. The research paper `pqc-onchain-verification` carries the full analysis.

Honest limitations

This layer is real and on chain, and a serious reader should weigh it against what is equally true today. AERE runs nine QBFT validators under a single operator (f=2, quorum 6-of-9), on one client (Besu), with no external security audit of these contracts and thin real-world usage; the wallet layer inherits that trust profile, and sponsorship and relaying currently depend on a Foundation-hosted relayer. The two-EntryPoint split described above is a genuine seam, and its unification is future work. The account-facing EntryPoint implements a compatible subset of the canonical fee economics, sufficient for the hosted relayer rather than a drop-in reimplementation. The live passkey account's recovery is "keep or add a backup owner"; guardian-based social recovery is written but not yet deployed. The modular ERC-7579 line carries no mainnet address and must not be presented as if it did. The post-quantum account is quantum-resistant at the account layer, level 1, and the native precompile that makes it efficient is now live on mainnet via the AerePQC fork (block 9,189,161), so a Falcon-512 authentication costs a fixed precompile fee rather than roughly ten million units of Solidity work. And several deprecated V1 contracts remain on chain at their old addresses; integrators must use the canonical V2 addresses named here. None of this is fatal, and all of it is the point: the account layer is built to be honest about the math it depends on, and safe to change as that math changes.

06

FAIR ORDERING

Fair Ordering and Interoperability

Two opt-in anti-MEV subsystems and two interop stacks, all at the application and account layer, with the exact line drawn between what is verified on-chain and where the trust is still human.

Chain 2800 · companion deep-dive: `spec-antimev-mempool`

Two questions decide whether a settlement layer is fair to the people who use it. Who controls the order in which transactions execute, and what happens when value has to cross a boundary between chains. Both are ordering problems. Maximal Extractable Value (MEV) is the profit a party earns from controlling ordering inside one chain. Interoperability is the problem of preserving a user's intent as it moves across chains, where a second ordering authority on the far side can extract the same way. AERE addresses both, and this chapter describes what is built, what is demonstrated, and what is still a trust assumption, without rounding any of it up.

An honest banner, first

> **Application layer, not base-consensus MEV resistance**
>
> Everything in this chapter lives at the application and account layer. None of it changes AERE's base consensus. AERE runs Hyperledger Besu QBFT with nine validators under one operator and one client, and validators sign classical secp256k1 QBFT messages to order the underlying transactions. A QBFT chain cannot acquire a protocol-level encrypted mempool or an enshrined proposer-builder separation without a client fork, and AERE has not forked its client for this. So what follows is not base-consensus MEV resistance and not enshrined PBS. It is a set of opt-in smart-contract and off-chain subsystems that a user or an application can choose to route through, and that protect what they can protect precisely.

Everything in this chapter lives at the application and account layer. None of it changes AERE's base consensus. AERE runs Hyperledger Besu QBFT with nine validators under one operator and one client, and validators sign classical secp256k1 QBFT messages to order the underlying transactions. A QBFT chain cannot acquire a protocol-level encrypted

mempool or an enshrined proposer-builder separation without a client fork, and AERE has not forked its client for this. So what follows is not base-consensus MEV resistance and not enshrined PBS. It is a set of opt-in smart-contract and off-chain subsystems that a user or an application can choose to route through, and that protect what they can protect precisely. Stating that boundary plainly is the point. For a young chain with a small validator set, credibility comes from scoping the claim correctly, not from dressing app-layer tooling as a consensus property.

The deep mechanics of the anti-MEV design, including the on-chain pairing algebra, the epoch state machine, and the audit-fix history, live in the companion specification `spec-antimev-mempool`. This chapter summarizes and surfaces; the spec is the reference.

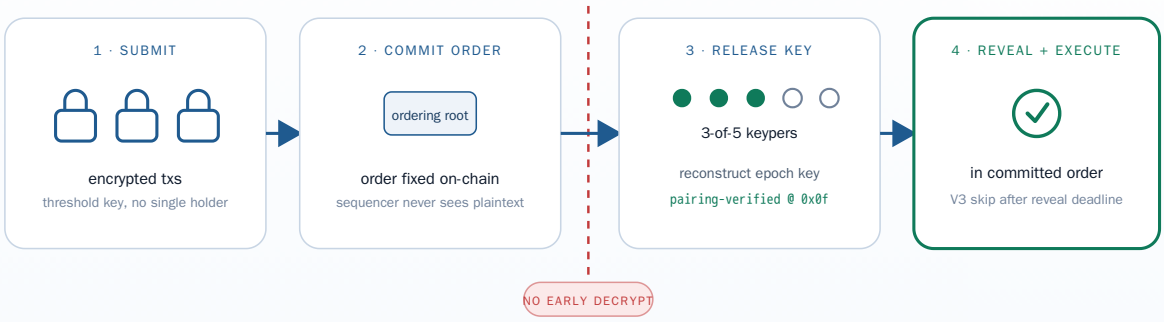
Where MEV lives on AERE

MEV is extracted in the gap between when a transaction is broadcast and when it is finally included and ordered. A party that can see pending transaction contents and influence their order can front-run, back-run, and sandwich a user's trade. On AERE the ordering authority inside that gap is the QBFT block proposer for ordinary transactions and, for the batch-settlement path, the solver that assembles a batch. AERE attacks the gap from two directions at once. Hide the contents, so whoever fixes the order cannot see what they are ordering. Remove the intra-batch ordering surface, so a set of trades clears together in one atomic transaction with no slot to insert a sandwich into. The two subsystems are conceptually a pipeline and are independent contracts today. They are described in turn.

Direction one: a threshold-encrypted mempool

The first subsystem is a Shutter-style encrypted mempool. The idea is simple to state and hard to make trustworthy. Users submit their transactions encrypted, under a threshold key that no single party holds. The sequencer commits an ordering while the contents are still ciphertext, so the order cannot depend on plaintext it never saw. Only after ordering is fixed does a committee of keypers release enough shares to reconstruct the epoch decryption key, at which point the transactions are revealed and executed strictly in the committed order. Reordering after the fact is impossible, because the ordering root is already on-chain and reveals must follow it.

FIG 6.1 Threshold-encrypted mempool: order is fixed on ciphertext before any key is released



What it shows. The ordering is committed on-chain while the transactions are still ciphertext, so the order cannot depend on plaintext the sequencer never saw. Submitting a decryption share reverts until that commit exists, which is the structural no-early-decrypt guarantee. On-chain, the epoch key is pairing-verified through the EIP-2537 precompile at `0x0f`; the final byte-mask XOR is an off-chain step. This is a trusted-dealer, single-coordinator proof-of-concept, outside the canonical registry.

AERE built this in three stages. The V0 contract, `AereShutterMempool`, is a code-only reference that establishes the commit and reveal skeleton with a placeholder decryption key. It is orchestration plumbing, not cryptography, and it exists in the repository only to be superseded. It carries no deployment record.

The real cryptography arrives in V2 and V3, `AereShutterMempoolV2` and `AereShutterMempoolV3`. These implement Boldyreva threshold BLS over the BLS12-381 curve and verify it on-chain using the EIP-2537 pairing-check precompile at address `0x0f`, which is live on chain 2800 under the Pectra ruleset. The contract checks two pairing equalities through that single precompile: that each keyper's share is consistent with its published verifiable-secret-sharing commitment, and that the reconstructed epoch decryption key is the genuine threshold signature over the epoch tag. A malformed share or a forged key fails its pairing and reverts, so a keyper cannot post an inconsistent share and no decryption key can be accepted early or by a minority. The no-early-decrypt guarantee is enforced structurally: submitting a decryption share reverts until the sequencer has already committed the ordering. V3 adds one thing V2 lacked, a liveness escape hatch. In V2 a single committed position whose opening was never revealed would stall every later position in the epoch forever. V3 stamps a reveal deadline at finalization (sixty seconds in the proof-of-concept) and lets anyone skip a genuinely stuck head

position once that window closes, while keeping reveal itself permissionless and never deadline-gated, so a late-but-honest opener can still land. Because a well-formed ciphertext embeds its own opening, any holder of the reconstructed key can reveal any position, which is what makes skip safe: it can only ever fire on a position nobody could reveal, never to censor a revealable one.

Both V2 and V3 were deployed to chain 2800 and exercised end-to-end, with a three-of-five keyper committee, a finalized epoch, an ordering root over committed envelopes, and reveal-execute transactions walking the order in sequence. The companion spec transcribes the epoch identifiers, decryption-key hashes, and transaction hashes verbatim from the deployment artifacts for anyone who wants to re-verify.

> **A proof-of-concept, with the caveats it carries**
>
> The V2 and V3 contracts are a proof-of-concept. They are not registered in the canonical SDK address registry, and they carry no production-traffic claim. The proof-of-concept relies on a trusted-dealer Shamir setup, meaning one party transiently holds the master secret while sharing it, which is the single most important cryptographic caveat and a thing a live product must not ship on. The symmetric unmasking step happens off-chain. The coordinator and sequencer default to a single address in the proof-of-concept. Keyper misbehavior can be flagged on-chain through an accountability primitive that reverts on false accusations, but it is detection only, with no economic slashing yet.

Now the honesty that has to travel with all of that. The V2 and V3 contracts are a proof-of-concept. They are not registered in the canonical SDK address registry, and they carry no production-traffic claim. Their addresses come only from repository deployment artifacts and are labeled as such; this document refers to them by name rather than promoting their hex to canonical status. The proof-of-concept relies on a trusted-dealer Shamir setup, meaning one party transiently holds the master secret while sharing it, which is the single most important cryptographic caveat and a thing a live product must not ship on. The symmetric unmasking step happens off-chain, because the pairing precompile returns only a boolean; on-chain, decryption-key correctness is fully pairing-verified, so decryption is deterministic and cannot be produced before the threshold acts, but the final byte-mask XOR is an off-chain step. The coordinator and sequencer default to a single address in the proof-of-concept. Keyper misbehavior can be flagged on-chain through an accountability primitive that reverts on false accusations, but it is detection only, with no economic slashing yet. In short, the encrypted mempool demonstrates that the pairing-verified

threshold path works on AERE. It does not yet demonstrate a decentralized, dealer-free, slashing-backed production system, and the roadmap names each of those gaps as an item to close before promotion.

Direction two: batch-atomic settlement LIVE

The second subsystem is a batch-auction settlement layer, and unlike the mempool it is in the canonical registry and live. It settles trades through a sealed batch with a single uniform clearing price, and it is three contracts:

- `AereSettlement` at `0x9C2957b1622567B4802E4AFd4c42FB2ec70dE875`
- `AereSolverRegistry` at `0xDBD29332a9993d2816EF0bD240288E03a8103f3B`
- `AereVaultRelayer` at `0x9FCA122e87E36D7cba20DbEA7b9b7354A4Cece91`

The flow is that users sign EIP-712 order structs off-chain, gasless, each carrying the tokens to sell and buy, the exact sell amount, the user's own minimum buy amount, a deadline, and a replay nonce. An authorized solver collects open orders and submits the whole batch atomically through a single `settle` call. For each order the contract checks the deadline, requires that the solver's promised delivery meets or exceeds the user's own minimum, recovers and verifies the EIP-712 signature against the stated user, marks the order filled so it cannot replay, pulls the sell amount through the vault relayer, and pushes the bought amount to the user. If any order in the loop reverts, the entire batch reverts. Allowances live on the relayer rather than the settlement contract, a two-contract split that lets the settlement logic be upgraded without users re-approving their tokens. Solver access is an owner-curated allowlist in the registry, with a single bootstrap solver today.

The MEV protection here is atomicity. Because every order in a batch shares one transaction, no actor can wedge a sandwich between two of them, and every filled order is guaranteed to deliver at least the floor its signer chose or the batch does not go through. Signature and replay safety are enforced per order. Those three guarantees, atomic execution, per-order limit protection, and replay safety, are real on-chain invariants that hold today.

The critical honesty is about what the contract does not enforce, because its own docstring is more generous than its code. It does not verify a uniform clearing price on-chain. It checks each order against that order's own floor only, so a solver could in principle fill different users of the same pair at different effective prices as long as each clears its own

limit; uniform-price fairness is a solver behavior, not a contract invariant. There is no surplus capture and no protocol fee: any price improvement above a user's floor accrues wherever the solver directs it, and there is no on-chain wiring from this contract to the AereSink flywheel router at `0x69581B86A48161b067Ff4E01544780625B231676` for surplus or MEV redistribution today. Partial fills, contract-signature (ERC-1271) orders, and dynamic fees are explicitly omitted and deferred. And the operational layer that would make this a live product, a dedicated solver service, an order-book API, and an intent toggle in the swap interface, is roadmap rather than deployed; the contracts are live, the surrounding service is in development. So the honest security posture is this: atomicity plus your own signed limit protect you from the worst MEV, and above that floor you are trusting a permissioned solver, one bootstrapped solver at that, for fair uniform pricing and surplus handling. That is a meaningful and deliberate distinction from a fully trust-minimized, permissionless batch-auction settlement, and it is stated rather than glossed.

How the two relate today

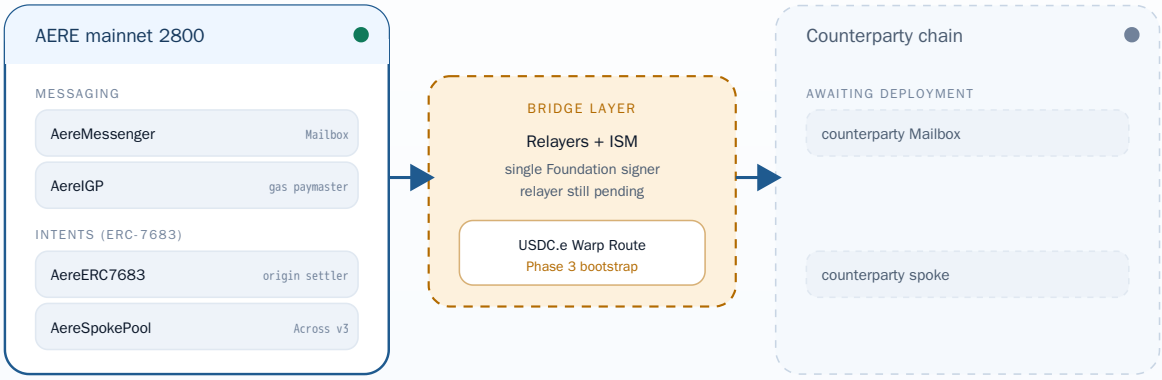
The intended composition is a pipeline: the encrypted mempool blinds the ordering party to order flow, and batch settlement removes the intra-batch ordering surface, so decrypted orders from an epoch would be handed to a settlement executor and cleared as one batch. That integration does not exist on-chain today. The mempool contracts expose an optional executor hook on reveal, but the settlement contract does not implement that interface and no adapter connects them, so a user currently uses one subsystem or the other, not a wired flow. Building that mempool-to-settlement adapter is a named roadmap item, not a shipped feature, and it is described as such.

Interoperability: messages and intents

Fair ordering does not stop at the chain boundary. When value crosses chains, a second ordering authority on the destination can extract exactly as a local proposer would, so AERE's interop layer is built on the same intent-centric philosophy: let the user express what they want, and let a competitive filler deliver it, rather than exposing a raw sequence of steps for someone to reorder. AERE deploys two complementary, interface-faithful interop stacks.

FIG 6.2

Interop stacks: Hyperlane-compatible messaging and an ERC-7683 intent layer, awaiting a live relayer network



Endpoints are deployed and interface-faithful. What moves value, the relayers, the interchain security modules, and the counterparty deployments, is not yet a live decentralized network.

What it shows. The AERE-side endpoints (messaging and the ERC-7683 intent layer) are deployed and faithful to their interfaces. The bridge layer that actually moves value is drawn dashed and amber because it is not yet a live, decentralized network: the existing path runs through a single Foundation signer with a relayer still pending. The first real cross-chain use is the modest USDC.e Warp Route bootstrap that unlocks the home order book and the RWA lending markets in Phase 3.

The messaging layer is Hyperlane-compatible. `AereMessenger` at `0xe54c2329f0786CFE3420c566B646148D25477325` is a Mailbox-compatible cross-chain message bus, and `AereIGP` at `0x61B48615F490A23945988c92835eF35fdD86E837` is the Interchain Gas Paymaster that receives AERE to fund outbound message gas. On top of that sits an intent layer aligned with the emerging ERC-7683 standard: `AereERC7683` at `0x67Fb9830e3a2BC06cEb641cfF3beD87b273ccb29` is an IOriginSettler for gasless cross-chain intents, and `AereSpokePool` at `0xCAB1DBA5f6F06198000C20a974d675f1B3181AbD` is an Across-v3-compatible spoke handling deposit, fill, and settlement. Both of those first versions are marked deprecated in our own canonical registry after an internal audit found a settlement-accounting defect, and neither should be integrated against. The corrected replacements are deployed and carry live code: `AereSpokePoolV2Corrected` `0xbEF9BF0D22dd00DBEdA262C3Cd3178f08C623d8d` (6,317 bytes) and `AereERC7683V2Corrected` `0x8eC4e7F01790C04D450b45d399E71DFd381560e3` (7,931 bytes). They are fresh and inert: no remote spoke is enrolled and no value has been migrated onto them, so the honest status of the intent lane today is built and corrected but not carrying traffic. Together these let a user sign an intent on one side and have a filler satisfy it on the other, with the origin

settler and spoke enforcing the settlement conditions rather than trusting the filler's ordering.

The honesty here mirrors the MEV subsystems. These endpoints are deployed and faithful to their respective interfaces, but an interop stack is only as live as its off-chain security. The relayers, the interchain security modules that validate inbound messages, and the counterparty-chain deployments are what actually move value, and those are not yet stood up as a live, decentralized network; the existing bridge path runs through a single Foundation signer with a relayer still pending. The first real cross-chain use is concrete and modest: bridging USDC.e onto chain 2800 through a Hyperlane Warp Route is the Phase 3 bootstrap that unlocks the home order book and the real-world-asset lending markets. Until that bridge is run, the interop contracts are correct endpoints waiting for their network, not a live cross-chain product, and they are presented that way.

Native interop: an on-chain zk Ethereum light client LIVE

The stronger interop primitive does not wait for a trusted relayer at all: it verifies Ethereum's own consensus from inside a contract on chain 2800. `AereZkEthLightClientV2` at `0x6f01ac96a28D0d670f94A41F323f198be14a17a8` (1,452 bytes of live code) checks live Ethereum Electra sync-committee finality on-chain by verifying a succinct SP1 Groth16 proof rather than trusting a relayer's assertion. It supersedes the first client at `0x2a2b6D936002A62adb4bb1d131007ec564429E9d`, which our own canonical registry marks deprecated because its public `verify` view ran the proof without binding the committed committee to the client's own trusted state, so it was sound as a state machine but unsafe if read as a finality oracle. V2 routes both paths through one binding check. The history below was made on the first client and is real; measured 2026-08-01 that client reports one update and finalized slot 14,753,792, while V2 reports zero updates, so the lineage was carried over at deployment and no new proof has been submitted since. Finality advanced end-to-end through the contract, from slot 14745696 to slot 14753792, in about 297k gas, comfortably under AERE's per-transaction gas cap. Inside the SP1 zkVM the guest proves the full update: 512 sync-committee pubkey decompressions, at least 342 of 512 participation, the BLS12-381 aggregate signature over the Electra signing root, and the finalized-header and next-committee SSZ branches; the on-chain contract does only the cheap Groth16 verify through the live SP1 gateway at `0x9ca479C8c52C0EbB4599319a36a5a017BCC70628`, binds the public values to stored state, then advances finality and rotates the trusted committee. The first real proof, of the period

1800 to 1801 update, was generated on a dedicated non-infra prover in roughly 24 minutes and submitted at block 9,337,697 in 296,898 gas, taking the update count from zero to one; both tampered proofs and tampered public values revert. Around it sit the outbound primitives `AereOutboundOutbox` at `0xd43FeacbbdDc5ff7cE4A72C726f3FBD204ef7936` and `AereOutboundVerifierV2` at `0x08b68bd553116Dffb99E648cb764AA93930da96F`, which admits those messages on a zk proof. The original verifier at `0x8E893686b6B2509C5f7Fe477CE530Bc868B0074e` was deliberately left inert and is now deprecated: its `programVKey` reads `0x0` on-chain, which is fail-closed, so every proof submitted to it reverts, and that value is immutable, so it can never be switched on. It was forgeable by design. Its guest took the validator set as a private, prover-supplied input and committed no validator-set root, so an attacker could list their own keys, self-seal a quorum and produce a valid proof for an arbitrary message. The rebuilt guest commits that root and pins the chain id to a constant, and V2 binds the committed root to an immutable anchor over AERE's real validator set, rejecting anything else, exactly as `AereZkQbftLightClient` does. Both directions are proven on mainnet: a genuine Outbox message was delivered, and a forged one, self-sealed by an attacker's own keys over a fabricated header, was rejected even though the live SP1 verifier confirms that proof is cryptographically valid. The flaw was found by code review, was never exploited and was never exploitable, since the contract was fail-closed throughout. The earlier header-relay `AereEthLightClient` at `0x9A9147236a47aE05eeEF692Bac985706f49E177b` is superseded by the zk client above. The outbound direction now also has its own succinct primitive: `AereZkQbftLightClient` at `0xCaDA54FAb6E7AE311d240Cf0C2Df45e974156488`, the canonical deployment, bound on-chain to the real SP1 program vkey and to AERE's seven-validator anchor as it stood at bootstrap (bootstrapped at block 9,312,565, the first block sealed by that set; the live validator set has been nine since 2026-08-09), so an external verifier could advance a trust-minimized view of AERE's OWN QBFT finality for blocks sealed by that seven-validator set by verifying a single Groth16 proof that a chain-2800 block carries a 2f+1 committed-seal quorum; since the set grew to nine on 2026-08-09 this deployment cannot verify current blocks until it is redeployed against the nine-validator anchor. This is proven live end to end: a real Groth16 proof that chain-2800 block 9,989,005 carries a 5-of-7 committed-seal quorum under the validator set live at that height was verified on-chain in 313,462 gas, advancing the client's tracked finalized head to that block; the proof was generated on a dedicated non-infra prover. Tampered proofs, tampered public values, stale-anchor values and replays all revert. The earlier client at `0xc9A2DCaeD0Ceb2B400Dd705a75cbabEBf5aBB1c2` is deprecated and must not be used: it anchored the historical three-validator set, so it is frozen at block 9,241,622 and can

never verify current finality; its own single advance was genuine at the time, so it is stale rather than fake. Honest boundary: this is an interop-layer contract that does not change AERE consensus, and it is trust-minimized not trustless, with classical ECDSA seals and a BN254 Groth16 wrap, so not quantum-safe; all nine validators are Foundation-operated, so the honest-majority premise is an operator assumption rather than an economic one. The validator-set anchor is immutable and does not track rotation, so a validator-set change requires a fresh deployment, which is exactly why the older client was replaced rather than updated.

> **The honest boundary on the zk light client**
>
> This is trust-minimised, not trustless. It inherits the security of at least two-thirds of Ethereum's 512-member sync committee, the same honest-majority assumption Ethereum light clients rely on, rather than a single relayer's word. It is also not quantum-safe, because the BLS12-381 sync-committee signatures and the Groth16 proof system over BN254 are classical. It advances real Ethereum finality on-chain today; it has not had an external audit, which is still pending.

Threshold authorization: classical and post-quantum committees LIVE

AERE ships on-chain t-of-n committee authorization in two registries. `AereThresholdRegistry` at `0x875BA0dbA1806Ad9aE57627d705ecA12139D1EF4` governs threshold-ECDSA committees, and `AereThresholdPQCRegistry` at `0x8Fbfe1C72E8c83ca2a0c475ce3648D47bCC8643c` is post-quantum: it verifies that at least *t* distinct committee members each supplied a valid PQC signature, checked on-chain through the live AerePQC precompiles. Concretely, `registerCommittee` publishes a committee over one of those precompiles (0x0AE1 Falcon-512, 0x0AE2 Falcon-1024, 0x0AE3 ML-DSA-44, 0x0AE4 SLH-DSA-128s), and `authorize` clears only when at least *t* distinct members each supply a valid signature over a domain-separated, strictly-increasing-nonce challenge, so duplicate or out-of-range indices revert and replay is impossible. Because each leg is an independent single-signer verify, the cost is *t* verifications, and under AERE's 16,777,216-gas per-transaction cap any realistic custody quorum such as 3-of-5, 5-of-9, or 11-of-21 fits with wide margin. The contract holds no funds and has no admin; it is deployed and currently inert, with no committees registered yet, 18 of 18 local tests passing (including a check that duplicate committee keys are rejected at registration, so distinctness is enforced by key and not merely by index), and an external audit pending. The honest boundary is that the post-quantum registry is a

threshold multisig of independent PQC signatures, not a single-aggregate threshold-PQC signature, and it authorizes at the application layer; consensus itself is hybrid post-quantum in the anchor-certificate sense, as described in chapter 3, and this registry is not what does that.

Building directly on that registry, a non-custodial smart account, `AereThresholdAccount`, has been implemented and formally verified. It is an ERC-4337 account whose owner is a t-of-n post-quantum committee, so no single key can move its funds and there is no admin path, intended for custody where a bank or an agent should never be able to act alone. It routes every signature through the same audited precompile-verification code as the registry above, and an automated prover confirms its core invariants: below the threshold it never authorizes, a rejected or duplicated signature is never counted, its two authorization paths are domain-separated, and its replay barrier holds. It is built, unit-tested, property-fuzzed, and machine-checked, with matching TypeScript and Python client libraries, and its CREATE2 factory is now live on AERE mainnet at `0x69734E4044B1C5943B9256A73De41B101BFA2633` (chain 2800), deployed after the full self-audit above. It is the natural bridge from threshold verification to threshold custody.

What this chapter claims, and what it does not

The claim is precise. AERE has built, and in the encrypted-mempool case demonstrated end-to-end on chain 2800, two opt-in anti-MEV subsystems and two interop stacks, all at the application and account layer. The batch-settlement contracts are live and canonical and give real atomicity, per-order limit, and replay guarantees. The encrypted mempool proves a pairing-verified threshold path on-chain but remains a trusted-dealer, single-coordinator proof-of-concept outside the canonical registry. The interop endpoints are interface-faithful and deployed but await a live relayer network and the USDC.e bootstrap. None of it is base-consensus MEV resistance, and none of it is decentralized yet. The roadmap that closes these gaps, a distributed key generation to remove the dealer, keyper slashing, permissionless bonded solvers, on-chain uniform-price verification with surplus routed to AereSink, the mempool-to-settlement adapter, and a live interop relayer set, is laid out in the companion spec and the phase roadmap. This is the AERE thesis applied to ordering and interop: final as math where the math is already verified on-chain, and honest about every place the trust is still human. Final as math, safe when the math changes.

07

PROVABLE COMPLIANCE

Identity, Compliance, and Privacy

A compliance stack built to be provable rather than promised, with a clear map of what is cryptographically trustless, what is optimistic, and what rests on a named issuer.

Chain 2800 · companion deep-dive: `spec-identity-compliance`

AERE treats compliance the way it treats settlement: as something that should be provable rather than promised. The chain's thesis, final as math and safe when the math changes, has a quieter corollary at the regulatory layer. A compliance claim is only worth as much as the proof behind it, and a proof is only honest if it states plainly what it does and does not establish. This chapter describes AERE's identity and compliance stack in that spirit. The companion deep dive, the identity and compliance specification (`spec-identity-compliance`), carries the contract-by-contract mechanics; here we summarize the design and, more importantly, we are candid about where the trust actually sits.

Three facts frame everything below, and we state them before the design rather than after it.

First, every piece of this stack lives at the application and account layer. None of it touches consensus. Validators sign ordinary QBFT messages over secp256k1. The identity and compliance contracts are plain EVM contracts that applications opt into, so nothing here changes the base protocol's security or its guarantees.

Second, AERE today is a Foundation-operated network with a small validator set, one client implementation, and a single operator. The stack has not had an external security audit. Several of these contracts have little or no live usage, and the compliance privacy pool holds zero deposits at the time of writing. We treat those as facts to disclose, not as things to bury.

Third, a single distinction runs through the entire stack and deserves to be named at the top: the difference between an inclusion proof and an absence proof. Most of the cryptography here proves that something is in a set. An address is on a sanctions list. A credential is in an issuer's tree. A note belongs to an attested clean set. Proving that

something is not in a set is a fundamentally harder primitive, and wherever we lack it we say so rather than implying otherwise.

> **Inclusion proof, not absence proof**
>
> Most of the cryptography here proves that something is in a set: an address is on a sanctions list, a credential is in an issuer's tree, a note belongs to an attested clean set. Proving that something is not in a set is a fundamentally harder primitive, and wherever AERE lacks it, this chapter says so rather than implying otherwise. Presenting the sanctions registry alone as clean-gating would overstate it, and we do not.

On-chain identity

The base identity layer is `AereIdentity` at `0x658dD2CD1F798AAb19fEc8FF69A270B2d192CaD1`, a lightweight on-chain registry compatible in spirit with W3C DID and ERC-1056. It exposes three primitives. Any address can publish a pointer to a content-addressed profile document, so only the pointer and a timestamp live on-chain. The Foundation maintains an allowlist of trusted attestors, meaning KYC providers, employers, or communities that are permitted to issue credentials. An attestor then issues claims keyed by a claim-type tag such as an age-over-18 tag or a KYC-tier tag, with an optional hash of an off-chain evidence document, an expiry, and issuer-only revocation. A reader checks whether a subject holds a given claim, from a given attestor, that is unexpired and unrevoked.

The honest scope of this contract matters. Claims are plaintext booleans on-chain. The registry reveals that a particular address holds a particular claim type from a particular attestor. That is a transparency-positive and privacy-negative design. It is appropriate for public credentials such as DAO membership or a verified-email flag, and it is the wrong tool for anything a user would reasonably want kept private, like residency or an exact KYC tier. Sensitive attributes belong on the zero-knowledge path described next, not here. For structured, schema-versioned regulator judgments, the newer successor is `AereAttestationGateway` at `0x9bdacA8dfF39Fc688e8D3c4bbA13bCFC0580c325`, which records append-only schemas and issuer-authorized attestations, supports validity windows and bounded inheritance so a revoked parent invalidates its children, and again stores only hashes rather than personal data. `AereIdentity` remains the simple key-value layer beneath it.

AI-native identity: post-quantum keys and agent DIDs LIVE

The identity layer is built for machine actors as well as people, because an autonomous agent that settles value needs an identity that outlives any single signing key and survives a cryptographic break. Two deployed contracts give it that. `AerePQCKeyRegistry` at `0x1eCa3c5ADcBD0b22636D8672b00faC6D89363691` is a post-quantum key registry: an account binds a PQC public key to itself and proves possession on-chain, so a counterparty can check that a given address genuinely controls the post-quantum key it claims. `AereAgentDID` at `0xce641d7d7C10553D82b06B7C21d423550e7522C5` builds an agent identity on top of that. The durable root of the identity is a Falcon post-quantum key, while the agent authorizes day-to-day actions with revocable secp256k1 session keys that it can rotate quickly. The split is deliberate: a slow, quantum-resistant root that rarely moves, and fast, disposable session keys for operational signing, so a leaked session key never puts the identity itself at risk.

This is what makes the identity AI-native. An autonomous agent registers a Falcon-rooted DID, mints short-lived session keys for a task, and settles through AERE's machine-account rail, the AERE402 payment facilitator and the `AereAgent` registry, with each payment authorized by a session key that can be revoked the moment the task ends. The honest scope is the same as the rest of this chapter. These are deployed application-layer contracts with thin usage, not an audited or high-load system, and none of them changes consensus, which still signs classical secp256k1 QBFT.

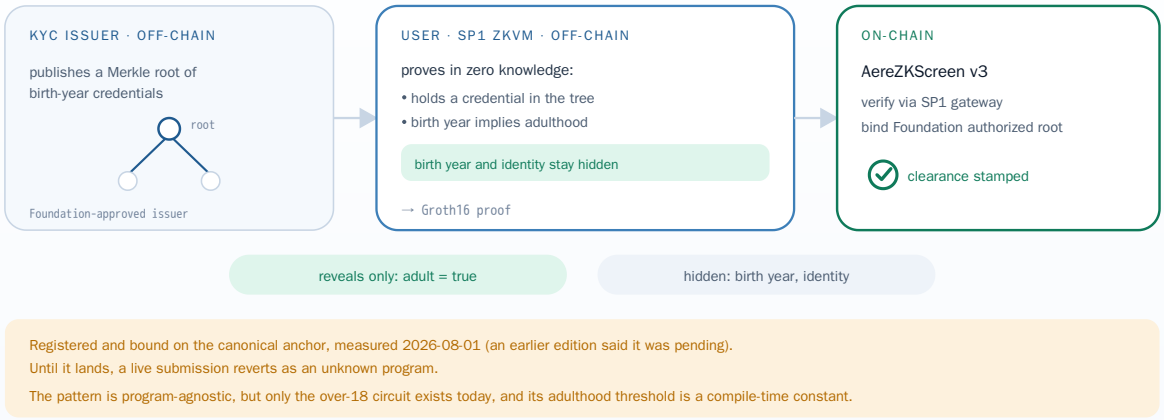
Zero-knowledge KYC attribute proofs

When a user needs to prove a fact about themselves without revealing the underlying data, the anchor is `AereZKScreen` at `0x3A097A459FD26aC79573aCB5adB51430e473C2f1`, the version 3 deployment. This is a compliance-preserving anchor. It records that a user passed an off-chain screen without any of the source data touching the chain. The Foundation registers programs, where each program is one compliance statement identified by the verification key of a specific zero-knowledge circuit. A user runs the corresponding SP1 zkVM program off-chain, produces a Groth16 proof, and submits it. The proof is checked on-chain through the shared SP1 verifier gateway at `0x9ca479C8c52C0EbB4599319a36a5a017BCC70628`, and on success the contract stamps a clearance timestamp that applications can gate against with a freshness window.

The flagship program is an over-18 proof, and it is worth describing precisely because precision is where honesty lives. A Foundation-approved KYC issuer publishes a Merkle root of birth-year credentials for the people it has verified. The circuit proves, in zero knowledge, that the prover holds a credential in that issuer's tree and that the birth year implies adulthood, without revealing the birth year. The adulthood threshold is a compile-time constant baked into the verification key, which makes the program a dated attribute that must be recompiled to advance the year. That is a deliberate tradeoff, chosen so a stale program cannot silently drift rather than to hide a limitation. The circuit and its proof pipeline are real and have been run end-to-end. An earlier edition of this chapter said the on-chain registration of that program was still pending a Foundation signature and that submitting a proof would revert as an unknown program. That is out of date and the correction runs in our favour, so it is stated as plainly as the caveat was: measured on 2026-08-01 on the live `AereZKScreen` `0x3A097A459FD26aC79573aCB5adB51430e473C2f1`, the over-18 program reads back registered, with its verification key matching the queried key and a non-zero authorized root set, so the program is registered and bound rather than unknown. What remains honest to say is that on-chain usage is thin: this is a live capability with almost no traffic, not a widely used one.

Two further points keep this honest. The contract is program-agnostic, so an EU-jurisdiction or MiCA-eligibility proof, an accredited-investor proof, and an OFAC-screen proof all follow the identical pattern. Only the over-18 circuit exists in the repository today, though. The others are archetypes on the same rail and should be read as roadmap, not as live capabilities. And the version 3 anchor exists because the first two deployments got a soundness detail wrong: an earlier version let a prover supply the very Merkle root the screen was checked against, so anyone could prove membership in a set of their own construction. Version 3 closes that by binding each program to a Foundation-set authorized root and rejecting any proof that does not match it. We surface that history rather than quietly retiring the dead versions, because a compliance layer that hides its own bug fixes has not earned trust. The specification also instructs every consumer to treat a clearance as necessary but not sufficient, combining it with a live sanctions check for sanctions-sensitive flows.

FIG 7.1 The over-18 attribute proof: prove adulthood without revealing the birth year



Registered and bound on the canonical anchor, measured 2026-08-01 (an earlier edition said it was pending).
Until it lands, a live submission reverts as an unknown program.
The pattern is program-agnostic, but only the over-18 circuit exists today, and its adulthood threshold is a compile-time constant.

What it shows. The flagship zero-knowledge attribute proof. A Foundation-approved issuer publishes a Merkle root of birth-year credentials; the user proves off-chain that they hold a credential in that tree and that their birth year implies adulthood, revealing neither the birth year nor their identity; the on-chain anchor verifies the proof against a Foundation-set authorized root and stamps a clearance. The pattern is program-agnostic, but only the over-18 circuit exists today and its on-chain registration is pending one Foundation signature.

The OFAC sanctions registry

`AereSanctionsRegistry` at `0xb7d235718D99560F6EA4Fc5eAea2F8a306A3Cacf` is a read-only, public-good sanctions registry. There is no fee, no discretionary interception, and no custody. The design is a daily cycle: an off-chain ingester reads the official OFAC Specially Designated Nationals list, extracts crypto addresses where present, builds a Merkle tree, and the Foundation publishes the root. The live state is smaller than the design and is published here rather than left implicit. Measured on 2026-08-01, the contract holds exactly one snapshot, epoch 20641, proposed on 2026-07-07 and still in the proposed state with no attestation recorded and the neighbouring epochs empty. No daily cycle is running today. Because the epoch was never attested, a consumer must not read a negative answer from this registry as evidence that an address is not sanctioned; until an epoch is attested, treat the registry as not yet carrying a usable list. The lifecycle per epoch is optimistic and mirrors the network's oracle pattern. A snapshot is proposed with a link to the canonical OFAC source and an optional pinned copy of the full list, anyone may challenge it within a 24-hour window, the Foundation may dismiss a baseless challenge, and after the window elapses without an open challenge anyone may attest the epoch, which freezes the root. History is append-only. There is deliberately no path to remove an address, because

removing one would imply the Foundation is making a sanctions decision, when in fact it only mirrors the canonical list. Additional list sources such as EU, UK, and UN consolidated lists have reserved identifiers, but Phase 1 ships OFAC SDN only.

Here the inclusion-versus-absence distinction becomes the whole story. The registry answers whether an address is on the list, and it answers that with a real Merkle inclusion proof. It cannot prove that an address is clean. A standard Merkle root supports membership proofs only, and proving non-membership would require a sorted or sparse tree, which this is not. So "this address is sanctioned" can be shown trustlessly given the proof, while "this address is clean" reduces to the far weaker statement that no valid inclusion proof was supplied, which depends on the caller actually performing the lookup instead of skipping it. For flows that genuinely need a positive clean assertion, the correct composition is with a positive-allowlist primitive, either the compliance pool's clean-set proof or a future OFAC-screen zero-knowledge program. Presenting the sanctions registry alone as clean-gating would overstate it, and we do not.

One adjacent contract deserves an honesty note. `ChainalysisOracleWrapper` at `0x1B7Be82C80f368f75Cb3807B1bc05E86A498f85c` gives applications a stable interface that forwards to a Chainalysis-style upstream oracle. No such oracle is natively deployed on AERE, so until the Foundation configures an upstream the wrapper returns false for every address and must not be relied on as a sanctions source. It is scaffolding for a future integration, and we label it as such. The real on-chain sanctions signal today is the Merkle registry above. A related signaling surface, `AereForensicEventRegistry` at `0x4a7526A068e5DDE9788f6571E4A99095b14C6fff`, lets detection bots post structured alerts about suspicious activity. It has no custody and no ability to intercept transactions, and in Phase 1 the reads that consumers gate on count only Foundation-confirmed alerts, so the trust anchor there is again the Foundation.

Travel-rule primitives

The FATF Travel Rule requires regulated institutions to exchange identity payloads for transfers above a jurisdictional threshold. Those payloads are confidential and are exchanged off-chain over a message bus. AERE's role is narrow and deliberately so. `AereTravelRuleHashRegistry` at `0xcF0E2e010E6e4506672019b1e570874AEeBC4c84` anchors only a commitment. The originating institution commits a hash of the standardized identity payload along with the counterparty address and the threshold, and the beneficiary later acknowledges receipt or disputes a mismatch. No payload data and no identity is ever

stored on-chain. The anchor proves that a specific payload existed at a specific block, that the log is append-only, and that a regulator can audit the exchange without the chain ever revealing its contents.

This contract has no admin, no interception, and no custody, which makes it genuinely trust-minimized in the narrow sense that it is just an append-only commitment log. The counterparties self-identify by their sending address, and the registry does not itself vouch that a committing address is a licensed institution. That verification is off-chain and can be layered through the attestation gateway. It is a minimal viable Travel Rule anchor, honestly scoped as Phase 1.

The compliant privacy pool

The stack's most ambitious primitive is `AereCompliancePoolV2` at `0xB144c923572E5Ac1B6B961C4ccfec36917173465`, a compliant privacy pool denominated in WAERE (`0x7e84d7d66d5da4cfE46Da67CDEeB05B323e1f5e8`) with proofs verified through its dedicated adapter at `0xE2D3fa91b680E835c971761ba75Fde0204AEF95E`. It is a Privacy-Pools construction in the Soleimani and Buterin lineage, hybridized with a Travel Rule hook. At its core is an append-only Merkle tree of deposit commitments with a rolling recent-root history, so a withdrawal can prove against any recent deposit root. A deposit binds the note to its depositor and records an off-chain Travel Rule commitment, tying the pool back to the anchor described above. A withdrawal submits a zero-knowledge proof that the note exists, that its nullifier is unspent, and that the recipient and fee parameters are bound so a front-runner cannot substitute them.

What makes it compliant, and what makes it honest, is the same mechanism. To withdraw, a user proves that their note is a member of an association root, a Foundation-published Merkle root over the subset of deposits judged clean. This is the correct way to build compliant privacy: the user demonstrates that they are in the good set, rather than making the impossible-to-verify claim that they are not in some bad set. The tradeoff is explicit and we state it. The clean set is Foundation-curated, so a user whose legitimate deposit is not yet included cannot withdraw with privacy until it is. That is a real centralization point. To bound it, association roots go through an optimistic, bonded lifecycle: the Foundation proposes a root, anyone may challenge it with a bond, and the resolution fails safe. One comparison in the earlier edition of this text was wrong and is withdrawn: the sanctions registry runs the same optimistic shape but takes no bond at all, since its challenge entry point is non-payable and holds no balance, so only the compliance pool lifecycle is bonded.

If the Foundation dismisses a baseless challenge within the window the bond is burned to preserve the anti-grief property, and if the Foundation fails to act, anyone can finalize the challenge as upheld, which refunds the honest challenger and permanently rejects the questioned root. That upheld-reclaim path exists because the first pool version could trap an honest challenger's bond forever, a bug we fixed rather than concealed.

Two properties are worth stating plainly. The pool has no admin withdrawal. There is no function by which the Foundation or anyone else can pull user funds, and the compliance surface is limited to curating the clean set and, through the optimistic lifecycle, to a liveness assumption that the Foundation dismisses baseless challenges in time. And the pool is fresh. It holds zero deposits today. We present it as a working, tested primitive with no real usage yet, because that is exactly what it is.

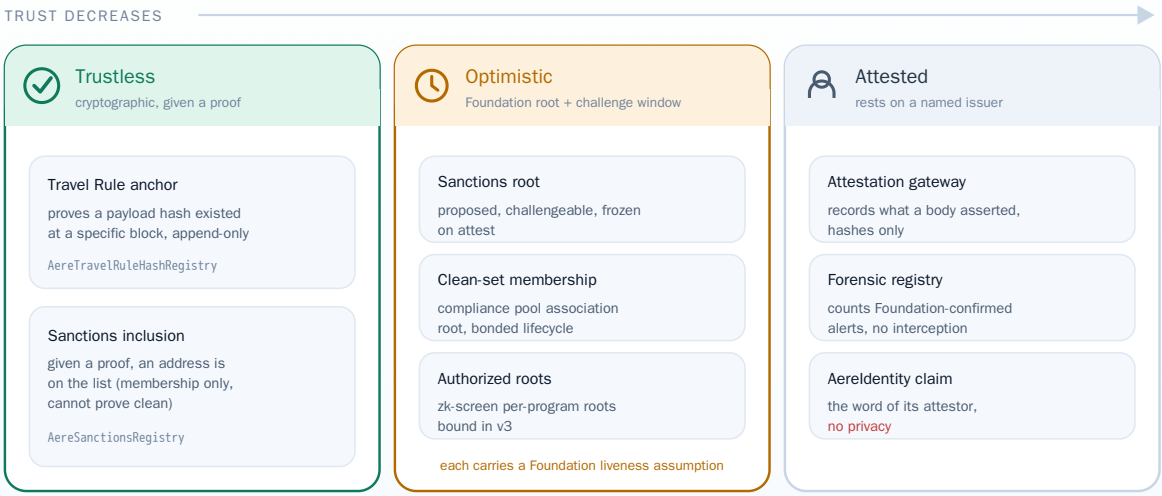
FIG 7.2 The compliant privacy pool: prove membership in a clean set, not the absence of guilt

What it shows. The compliant privacy pool. Deposits enter an append-only commitment tree; a withdrawal proves membership in a Foundation-published clean-set (association) root rather than the impossible claim of not being in some bad set. The clean set is curated, which is a named centralization point bounded by a bonded challenge window; there is no admin withdrawal, and the pool holds zero deposits today.

What is trustless, what is optimistic, and what is attested

The single most useful thing a reader can take from this chapter is a clear map of where trust lives, so we gather it in one place.

FIG 7.3 Where the trust sits: three tiers from cryptographically trustless to a named issuer's word



Foundation-dependent surfaces, collected: the sanctions root, the clean-set root, the per-program authorized roots, the attestation schemas and authorized attestors, the pool's compliance-provider allowlist, and the forensic alert confirmations. Optimistic windows reduce this trust, they do not eliminate it.

What it shows. The stack sorted by where trust sits. Only two statements are cryptographically trustless. The optimistic tier rests on a Foundation-published root plus a challenge window and a liveness assumption. The attested tier is exactly the word of a named issuer. Naming this honestly is treated as an asset: as the validator set widens and an audit lands, these are the surfaces that will decentralize.

Cryptographically trustless statements are few and precise: the Travel Rule anchor proves a payload hash existed at a block, and the sanctions registry proves, given a proof, that an address is on the list. Optimistic statements rest on a Foundation-published root plus a challenge window: the sanctions root itself, the compliance pool's clean-set membership, and the zero-knowledge screen's per-program authorized roots all fall here, and each carries a Foundation liveness assumption. Plain attestations rest on a named issuer: the attestation gateway records what a body asserted, the forensic registry counts what the Foundation confirmed, and an `AereIdentity` claim is exactly the word of its attestor with no privacy.

The pieces that ultimately depend on the Foundation, collected so none is buried, are the sanctions root, the clean-set association root, the per-program authorized roots, the attestation schemas and their authorized attestors, the pool's compliance-provider allowlist, and the forensic alert confirmations. The optimistic challenge windows reduce this trust but do not eliminate it, and they assume the Foundation stays live. We consider naming this an asset rather than a weakness. A compliance stack that pretends to be trustless when it is optimistic is worse than one that is honest about where the trust sits, and as the validator set widens and an external audit lands, these are precisely the

surfaces that will decentralize. Final as math where the math is real, and honest about the human roots where it is not yet. For the full contract mechanics, audit history, and the trust table in tabular form, see the companion identity and compliance specification.

08 THE ECONOMY

Token Economics and the Flywheel

A supply that is fixed and unmintable, an immutable three-bucket revenue router that no admin can change, real yield instead of issuance, and a candid account of what is still reserve-funded.

Companion: `research/specs/spec-flywheel-economics.md` · addresses from `sdk-js/src/addresses.ts`

Companion deep-dive: `research/specs/spec-flywheel-economics.md`. Source of truth for addresses: `sdk-js/src/addresses.ts`. This chapter summarizes the design and surfaces the honest caveats; the spec holds the parameter-by-parameter detail.

AERE's economic design rests on a single, unfashionable commitment: the money supply is fixed and no contract in the system can mint. Everything else, the burn, the flywheel, the staking yields, is built on top of that constraint rather than around it. This is the economic expression of the network thesis, final as math, safe when the math changes. A fixed cap is a promise that can be verified in one on-chain read, and it is a promise that never needs to be renegotiated.

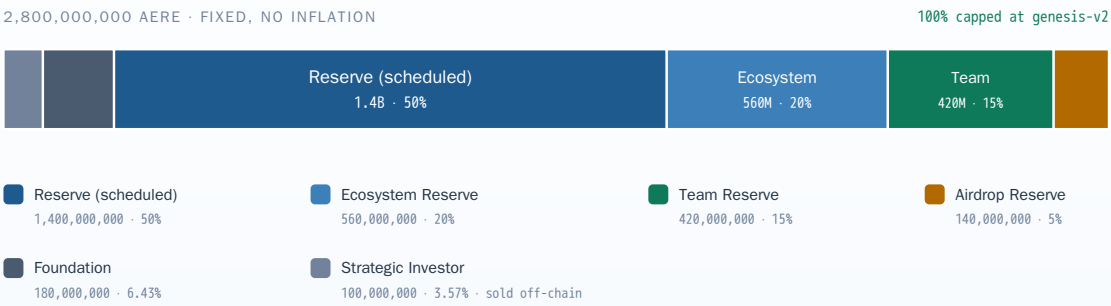
A fixed supply, verifiable and capped

Total supply is 2,800,000,000 AERE, capped at genesis-v2 on 2026-05-07 and allocated at genesis across six wallets: 100M to the Strategic Investor (sold off-chain), 180M to the Foundation as chain admin and contract owner, 1.4B to the Reserve (scheduled), 560M to the Ecosystem Reserve, 420M to the Team Reserve, and 140M to the Airdrop Reserve. Those are the genesis allocations, and a reader who checks the balances today will not find that split intact, so the measured position is published here rather than left to be discovered. Measured at the head on 2026-08-01, the six genesis addresses hold 2,239,999,990.70 AERE between them: Strategic Investor 100,000,000, Foundation 179,999,989.999, Reserve 1,400,000,000, Ecosystem Reserve 0.699996, Team Reserve 420,000,000, Airdrop Reserve 140,000,000. The Ecosystem allocation no longer sits at its genesis address; 559,999,999 AERE of it are held at a single separate account on chain. The remaining 10.30 AERE of the difference reconciles as the fee spend of these six wallets since genesis, which is a reconciliation of the two columns rather than a traced sum of transactions, so it is marked NOT MEASURED as an independent figure. That

account is not published here, and establishing and documenting its custody is an open item listed in the risks and limitations section of Chapter 9. Nothing was minted and nothing was destroyed by that move: the cap is unchanged and none of the mechanisms described below mint tokens. Native AERE is the gas and value token; `WAERE` (`0x7e84d7d66d5da4cfE46Da67CDEeB05B323e1f5e8`) is its 1:1 ERC-20 wrapper on the WETH9 pattern, and every ERC-20-facing mechanism in the economy (the sink, the staking receipt) operates on WAERE. The economic thesis is monotone: the burn path only ever removes AERE against this fixed cap, and no path re-mints. Every Ownable contract in the economy chapter that answers `owner()` returns the Foundation-controlled account (`0x0243A4f47D44b40b65D33f20329dE20D00c6f3C3`, a single-key EOA, not a threshold multisig; a Safe multisig is roadmap), which was re-measured on 2026-08-01 and holds; across the whole canonical registry the picture is mixed, and the count is published in the governance chapter. The load-bearing flywheel contracts, by contrast, have no owner at all.

FIG 8.1

Fixed supply: 2,800,000,000 AERE across six genesis wallets, capped at genesis-v2, never inflated



What it shows. The whole 2.8B supply, set at genesis-v2 (2026-05-07) and never increased. No mechanism in the system mints tokens; the burn path only ever removes AERE against this cap. The Foundation holds 180M as chain admin and contract owner, but the load-bearing flywheel contracts have no owner at all.

The burn: 37.5% of the validator reward

AERE's QBFT chain does not run EIP-1559 protocol-level base-fee burning. Being precise about this matters for credibility. The widely quoted "37.5% burn" is not a consensus rule; it is applied to validator coinbase rewards by a splitter contract that validators (or their forwarder daemons) call each block, operationalizing the whitepaper's fee-burn intent at the coinbase layer rather than inside consensus.

The original `AereCoinbaseSplitter` (`0xb4b0eCe9011613A5b84248a9B42a0f309E6F01Ec`) performs a two-way split: it forwards `burnBps / 10000` of the coinbase to the burn vault and rebates the remainder to the validator, atomically, with cumulative counters that let any explorer verify the burned total in a single read. Its successor `AereCoinbaseSplitterV2` (`0x8C1A48eFA57b66fEE743A00E3899c29ad3Fd27b4`) adds a third slice: 37.5% burn, 15% routed to the flywheel sink, and the derived 47.5% rebated to the validator, so a fraction of every block reward begins compounding for stakers instead of returning entirely to the block producer.

A word on the word "sealed," because honesty here is worth more than a stronger-sounding claim. The 37.5% burn rate is not immutable. It is an owner-adjustable parameter, bounded inside `[0, 5000]` basis points (a hard 50% cap that gives headroom while bounding abuse). What is truly sealed in AERE's economy is not the burn rate but the router it feeds, `AereSink`, which has no owner and no setters at all. The burn rate is a Foundation-tunable dial with a fixed ceiling; the flywheel split beneath it is set in stone.

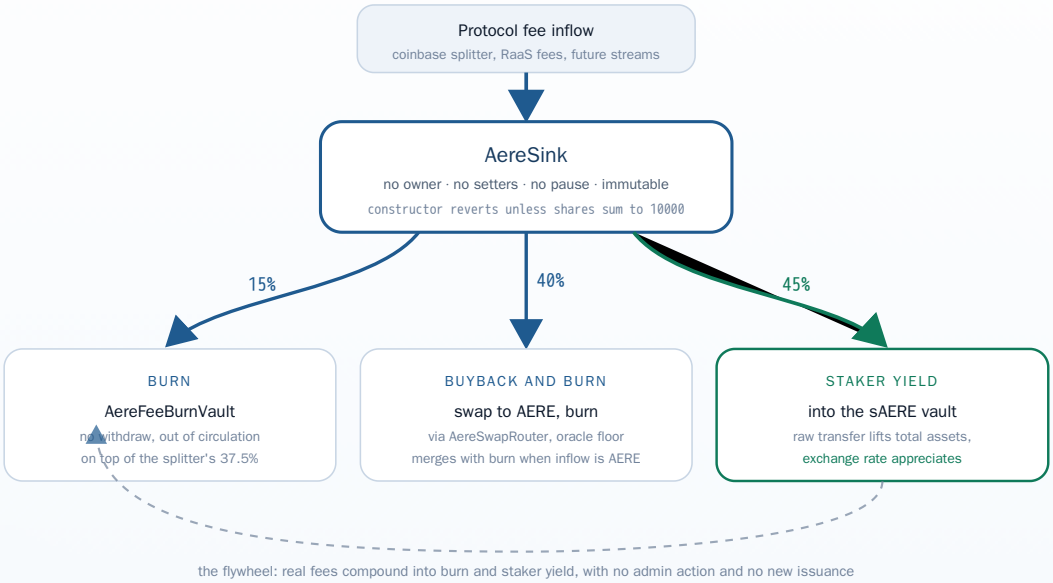
The burn endpoint itself is `AereFeeBurnVault` (`0x696afDF4f814e6Fd6aa45CE14C498ed9375fB2c6`), a stateless contract with no owner, no admin, no withdraw function, and no escape hatch. AERE that arrives there is permanently removed from circulation because nothing can move it out. There are two honest ways to describe that state: value on a no-withdraw contract is out of circulation the instant it arrives, and value becomes literally unrecoverable at the zero address once anyone calls the permissionless `sweepToZero()`. Both counters are exposed on-chain, so the reader can pick the definition they mean and verify either.

The immutable flywheel: AereSink

`AereSink` (`0x69581B86A48161b067Ff4E01544780625B231676`) is the credibility anchor of the whole economy. It is the single entry point for any protocol fee stream that should accrue to AERE holders, and it has no owner, no admin, no setters, no pause, and no upgrade proxy. It splits every inflow across three buckets whose recipients and basis points are fixed as `immutable` at deploy; the constructor reverts unless the three shares sum to exactly 10000. The deployed configuration is 15 / 40 / 45:

FIG 8.2

The AereSink flywheel: one immutable, ownerless router splits every fee inflow 15 / 40 / 45



What it shows. Every inflow is split on a fixed 15 / 40 / 45 basis that is set as `immutable` at deploy. There is no `setBucket`, no `setRecipient`, no `setRouter`. BURN and BUYBACK both destroy AERE; STAKER-YIELD lifts the sAERE exchange rate. The absence of an admin key is what makes the 15 / 40 / 45 promise worth believing.

- BURN (15%) goes to `AereFeeBurnVault`, on top of the splitter's 37.5%.
- BUYBACK-and-burn (40%) is designed to swap non-AERE inflows to AERE through a DEX router and burn the proceeds. Measured on the live sink on 2026-08-01, this bucket is not wired: `DEX_ROUTER()` reads `0x0000000000000000000000000000000000000001` and `ORACLE()` reads the zero address, both immutable, so the deployed sink is not connected to `AereSwapRouter` `0x7526B2E5526EfA84018378b60F2844Dad77523D8` or to any price oracle, and the buyback path cannot run for a non-AERE token on this instance. The 15/40/45 split itself is real and reads back from the contract as 1500, 4000 and 4500 basis points. When the inflow is already AERE or WAERE, which is the normal case since the splitter wraps before flushing, BURN and BUYBACK merge into one transfer to the burn vault, because buying AERE with AERE is a no-op.
- STAKER-YIELD (45%) is transferred into the sAERE vault. A raw transfer into an ERC-4626 vault lifts total assets without minting shares, so the exchange rate appreciates for every existing holder. This is the wire that connects protocol fees to stakers.

The sink is engineered to be un-gameable at the edges as well as the center. Its swap design computes the minimum-output floor from an external oracle price rather than from pool reserves, because pool reserves can be manipulated by the same actor in the same transaction, and the interface is compatible with `AereLendingOracle` at `0xc0f18A567067F1B84BDf75eDEbFaDdCBb70A4C49`. That protection is a property of the design and not of the live instance: as measured above, the deployed sink holds the zero address as its oracle and address one as its router, both immutable, so no swap path is active on it and the floor described here is not doing any work today. Rewiring means a fresh sink deployment, and it is a roadmap item rather than an owner call. A failed swap does not revert the whole flush; the tokens rest as dust and anyone can re-flush them later through the permissionless `sweepDust`. And crediting the staker bucket triggers a best-effort `sync()` on the vault so the arrival begins dripping immediately rather than jumping the price for a sandwicher to race.

> ☆ The load-bearing claim: an ownerless split
>
> Immutability is the entire point. There is no `setBucket`, no `setRecipient`, no `setRouter`. If the router ever needs replacing, the sink is redeployed and fee sources are rewired in the open; the split itself can never be changed under holders. That absence of an admin key is what makes the 15 / 40 / 45 promise worth believing.

Immutability is the entire point. There is no `setBucket`, no `setRecipient`, no `setRouter`. If the router ever needs replacing, the sink is redeployed and fee sources are rewired in the open; the split itself can never be changed under holders. That absence of an admin key is what makes the 15 / 40 / 45 promise worth believing. See the companion spec for the sandwich-resistance and dust-recovery detail.

sAERE: real yield, not issuance

`sAERE` (`0xA2125bE9C6fd4196D9F94757Df18B3a2A5e650b0`) is the liquid-staking receipt, an OpenZeppelin ERC-4626 vault over WAERE with no owner, no admin, no pause, and no parameter setters. Depositors lock WAERE and receive sAERE; the WAERE-per-sAERE rate drifts up as the sink's STAKER-YIELD bucket routes fees in. This is real yield in the strict sense. It comes from protocol fee inflows, not from token issuance, and no one chooses whether, when, or how much; once funds arrive they vest mechanically. There is no fixed APR on sAERE. The rate is whatever the fee flow produces.

Two safety properties matter economically. Yield is not recognized as a step function the instant the sink transfers AERE in; arrivals vest linearly over a seven-day drip, so share price grows continuously rather than in block-sized jumps. That linear drip is precisely what defeats the flash-loan sandwich of deposit, flush the sink, and redeem in one block, since the price cannot jump within a block. And the classic ERC-4626 first-depositor inflation attack is neutralized by a virtual-share decimals offset, backed by a dead-shares seed at deploy. The honest caveat is that sAERE only appreciates if fees actually flow; with thin current usage, realized yield to date is minimal and should be read live from the exchange rate, not assumed. The mechanism is real and admin-free; the volume is early.

Staking products: an honest subsidy with a runway

AERE offers three staking surfaces, and they are frequently and wrongly conflated. Only sAERE is admin-free real yield. The other two pay a stated rate, and that rate is a reserve-funded subsidy, not perpetual issuance.

Delegated validator staking, `AereStakingV2` (`0x1D95eF6D17aeAB732dF914Ba2d018c270BC155FC`), lets holders delegate native AERE to validators and earn an APR while validators post a self-stake bond and take a commission. The default reward rate is 8% (800 bps), owner-settable within `[1, 10000]` bps; accrual is timestamp-based and so immune to block-time changes, with a 7-day unbonding queue and a 5% slash rate on validator self-stake. One limit of the deployed code is stated here rather than left for a reader to find: the self-stake is only reachable by the slash path while the validator is registered as active, because deregistration moves the whole self-stake into the unbonding queue and the slash path does not reach that queue. The penalty is therefore weaker than the phrase slashable bond suggests, and making it unconditional needs a redeploy, which is a roadmap item. In context, measured 2026-08-01, the contract reports zero total staked and holds a zero balance, so nothing is delegated or bonded on it today. This is a bug-fix redeploy of a deprecated V1; use only V2.

Fixed-term locked staking, `AereLockedStaking` (`0x21108c28A849b05aE6b7a3a5bc435C9Bc897E7Ad`), pays a fixed APY in AERE at maturity across four tiers: 30 days at 10%, 90 days at 15%, 180 days at 22%, and 365 days at 30%. Principal is protected by construction; a withdrawal reverts unless the contract balance covers all locked principal plus the maturity, so one staker's payout can never drain another's principal. Early exit forfeits reward and returns principal only.

LOCKED 30 DAYS	LOCKED 90 DAYS	LOCKED 180 DAYS	LOCKED 365 DAYS
10% APY	15% APY	22% APY	30% APY

> **Staking APRs are a reserve-funded subsidy, not issuance**
>
> Because supply is capped at 2.8B and there is no consensus-level reward minting, both staking pools are paid from Foundation-seeded balances drawn out of the genesis reserves. So the 8% APR and the 10-to-30% locked APYs are subsidies with a runway, not native issuance and not perpetual rates. They are safe (principal is protected and the contracts cannot pay what they do not hold) but they are finite. The intended end-state is that the fee flywheel, not the reserve, becomes the funding source.

The honesty that must accompany those numbers is the crux of a fixed-supply chain. Because supply is capped at 2.8B and there is no consensus-level reward minting, both staking pools are paid from Foundation-seeded balances drawn out of the genesis reserves. `AereStakingV2` reverts a claim with "pool insufficient (needs foundation top-up)" if its balance cannot cover the reward, and `AereLockedStaking` simply cannot pay a maturity the reserve does not cover until it is topped up. So the 8% APR and the 10-to-30% locked APYs are subsidies with a runway, not native issuance and not perpetual rates. They are safe (principal is protected and the contracts cannot pay what they do not hold) but they are finite. The intended end-state is that the fee flywheel, not the reserve, becomes the funding source, at which point yield and burn grow together as real usage grows, without any parameter change and without any admin action, because the sink has no admin.

A fourth surface, `AereGovernanceStaked` (`0x8D77C888e439C4fADb2e23F1567a0A1965F80bCb`), is not a yield product; it reads staking state to give delegated stake voting power, with a 7-day unbonding period that makes vote-borrowing impractical.

No adaptive issuance, by design

This is the deliberate line in AERE's economics, and it is worth stating without hedging: there is no adaptive, algorithmic, or discretionary issuance, and there never will be under this design. Supply is fixed at 2.8B. The chain does not inflate to pay stakers, does not mint to hit a staking-ratio target, and does not run a monetary-policy dial that a committee can turn. Adaptive issuance is explicitly off-strategy and out of scope. That choice has a real cost, which the design accepts openly: it makes staking APR a bounded subsidy rather than a self-refilling faucet, and it puts the burden of long-run sustainability on genuine fee

revenue routed through the sink. AERE takes that trade on purpose. A holder can verify the cap once and never worry that it moves; an inflating chain asks its holders to trust a policy that can change. Fixed supply is the money-side meaning of the thesis, safe when the math changes, because the one number that governs dilution simply does not change.

What the flywheel does per block, worked

Putting the pieces together, take a validator coinbase reward `R` under the V2 defaults and the sink's 15 / 40 / 45 split. The splitter burns 0.375 R directly, routes 0.15 R into the sink, and rebates 0.475 R to the validator. Inside the sink, BURN (15%) and BUYBACK (40%) are both AERE-destroying, so 0.55 of the 0.15 R slice, which is 0.0825 R, is burned, while STAKER-YIELD (45%) sends 0.0675 R to sAERE holders. Aggregating, 0.4575 R is burned, 0.0675 R accrues to stakers, and 0.475 R is rebated to the validator. The V2 source comment cites roughly 52.5% as "protocol-directed" (the full non-rebated share); the stricter and more honest figure for actual token destruction is 45.75%, since 6.75% becomes staker yield rather than being burned. Both are stated so a reader can choose the definition they mean. The same `AereSink` is the intended router for other fee streams beyond the coinbase, so broadening the set of contracts that flush into it is the primary lever that makes sAERE yield and the burn self-sustaining, and it requires no change to the immutable sink.

What R actually is today: zero

Every formula above is stated in terms of the validator coinbase reward R. The honest and load-bearing fact is that R is currently zero on mainnet 2800, so 0.4575 R is also zero, and the burn destroys effectively nothing. A reader who takes only one number from this chapter should take that one.

There are two independent causes, and either alone would be sufficient. First, the genesis QBFT configuration declares no `blockreward` key, and Besu therefore issues no block subsidy to the proposer. Second, `londonBlock` is 0, so EIP-1559 is active from genesis and the 1 Gwei base fee is destroyed by the protocol rather than credited to the coinbase account. What reaches a proposer is only the priority tip, and at present transaction volume that rounds to nothing. Because the two causes are independent, fixing one would not by itself start the flywheel.

This is measurable by anyone in two RPC calls against the public endpoint, with no privileged access. Sampling the nine validator accounts at the chain head, then re-sampling after the head advanced 17 blocks, returns byte-identical balances for all nine. Lifetime burned, read as the balance of `AereFeeBurnVault` `0x696afDF4f814e6Fd6aa45CE14C498ed9375fB2c6`, is 137,352,594,046,167,719 wei, which is 0.13735259 AERE against a fixed supply of 2,800,000,000 AERE, or roughly one part in twenty billion.

We state this plainly because the alternative is worse. A burn mechanism that is live, immutable, ownerless and internally reviewed (not externally audited), but currently burning nil, is a true and defensible thing to have built. A project that publishes its own zero is more credible than one that lets a percentage imply a quantity. The mechanism is correct and it is already deployed; what is missing is fee volume, and fee volume is a demand problem, not a contract problem. The number rises when real usage arrives, and until it does this document will keep quoting it as it is.

Any figure in this section can be re-derived at any time. Today the re-derivation runs through Foundation-operated endpoints; run your own node against the public genesis if you do not want to trust ours.

BURNED PER REWARD R	TO STAKERS	REBATED TO VALIDATOR	TOKEN DESTRUCTION
0.4575 R	0.0675 R	0.475 R	45.75% the honest figure

Honest limitations

The mechanisms above are admin-minimized, reentrancy-guarded, and verifiable from source, but the economy is early and the reader deserves the caveats plainly:

> **Realized volume is thin.** The chain runs with nine validators under one operator, and one client (Besu). Burn and yield to date depend on validators actually running the coinbase-forwarder daemon and on real fee flow. This chapter quotes parameters and formulas, not lifetime-burned or TVL or yield figures; those are to be read live from the contracts' on-chain counters.

> **The V2 three-way split is an operational state, not a code invariant.** The sink slice only routes once the Foundation has wired `AereCoinbaseSplitterV2` to `AereSink` through the contract's seven-day

> sink-rotation timelock. Until then V2 behaves like the two-way V1. Confirm on-chain before quoting the 45.75% figure as live.

> Staking APRs are reserve-funded. No consensus-level minting exists; the delegated and locked pools pay from Foundation-seeded balances out of the fixed 2.8B. This is on-strategy (no adaptive issuance) but it means the APR is bounded by the reserve until fee flow replaces it.

> No external audit. Every fix in the economy so far comes from internal audit rounds. The contracts are admin-minimized and guarded, but they have not had a third-party audit.

The design goal is not to hide any of this. It is to make the parts that must be trusted as small as possible, an immutable sink and a no-withdraw burn vault, and to make everything else verifiable in a single on-chain read. Fixed supply, a sealed flywheel split, real yield instead of issuance: final as math, safe when the math changes.

09

GOVERNANCE AND PATH

Governance, Roadmap, and Limitations

How AERE is governed today and where that is not yet binding, the ordered and gated path from a single-operator network to a credibly neutral one, and a consolidated account of what is still weak.

Chain 2800 · companion deep-dives: `docs/ROADMAP-PHASES.md`, `spec-flywheel-economics.md`, `research/pqc-onchain-verification.md`

This chapter states three things plainly: how AERE is governed today and where that governance is not yet binding, the ordered path from what runs on mainnet now to a credibly neutral settlement layer, and a consolidated honest account of what is still weak. For a young chain, candour is a credibility asset, so nothing here is softened. The companion deep-dives are `docs/ROADMAP-PHASES.md` (the phase-by-phase gate list), `research/specs/spec-flywheel-economics.md` (the economic mechanisms cited below), and `research/pqc-onchain-verification.md` (the post-quantum verifier suite). Every address in this chapter is copied verbatim from the canonical registry `sdk-js/src/addresses.ts`. Contracts that are planned but not yet deployed are named by artifact only and are never given a fake address; the Governor and the Timelock, once in that category, are now deployed and addressed in the governance section, but own nothing yet.

9.1 Governance today, and the binding gap

AERE has an on-chain governance surface and a candid gap between it and real control. Both matter, so both are stated.

The parameters that exist

The stake-weighted governance contract `AereGovernanceStaked` `0x8D77C888e439C4fADb2e23F1567a0A1965F80bCb` reads voting power directly from the staking system: a holder's weight equals the total native AERE it has delegated across validators in `AereStakingV2` `0x1D95eF6D17aeAB732dF914Ba2d018c270BC155FC`, a stake-to-vote model. Its live parameters, verifiable in source, are a proposal threshold of 100,000 AERE, a quorum of 20 percent of total staked AERE, a voting period of 7 days, and an execution delay of 2 days. The 7-day unbonding period on the staking contract is what makes vote-borrowing impractical: stake cannot be flash-loaned in to swing a vote and withdrawn the same block.

These figures are documented in the flywheel-economics spec and are read straight from the deployed bytecode, not projected.

PROPOSAL THRESHOLD	QUORUM	VOTING PERIOD	EXECUTION DELAY
100,000 AERE	20% of staked	7 days	2 days

Why that surface is advisory, not binding

Here is the honest part. Those parameters describe a vote, but a passed vote does not today move the protocol by itself. The legacy `AereGovernance` contract executes proposals only against itself and is owned by the Foundation. `AereGovernanceStaked` tallies stake weight but does not own any protocol contract, so it cannot, on its own, change a parameter or perform an upgrade. Almost every `Ownable` protocol contract on chain 2800 is controlled by the Foundation-controlled account `0x0243A4f47D44b40b65D33f20329dE20D00c6f3C3` (a single-key EOA, not a threshold multisig; a Safe multisig is roadmap). The word every was too strong and is corrected here with the measurement behind it, dated 2026-08-11: of the canonical registry contracts, 79 answer `owner()`; 61 return the Foundation account and 18 return other single-key operational accounts. Those 18 include contracts this document treats as load bearing, among them the proof router, the proof registry and the cryptographic-agility registry. They are scheduled to move to governance together with the rest, and the transfer is tracked in the roadmap below. In practice this means governance is advisory: the community can signal, but a Foundation signature, or in those 18 cases an operational one, is what actually executes. Describing this as decentralized governance would be false, so we do not.

> **Governance is advisory until ownership moves to a Timelock**
>
> Almost every `Ownable` protocol contract on chain 2800 is controlled by the Foundation-controlled account, and the measured exceptions are named in the section above: 61 of the 79 registry contracts that answer `owner()` return the Foundation account, and 18 sit under other single-key accounts pending transfer to governance. The stake-weighted vote can signal, but an owner signature is what actually executes. Describing this as decentralized governance would be false, so we do not. The counterweight is real: the money flywheel and the verifiers already have no admin key at all.

Where control is already relinquished

The counterweight, and it is a real one, is that the pieces of the system where an admin key would be most dangerous have no admin key at all. The revenue router `AereSink` `0x69581B86A48161b067Ff4E01544780625B231676` has no owner, no setters, no pause, and no upgrade proxy; its 15 / 40 / 45 split across burn, buyback-and-burn, and staker-yield is constructor-set and immutable, so no vote and no Foundation signature can ever change how fees are divided. The liquid-staking receipt `sAERE` `0xA2125bE9C6fd4196D9F94757Df18B3a2A5e650b0` and the burn endpoint `AereFeeBurnVault` `0x696afDF4f814e6Fd6aa45CE14C498ed9375fB2c6` are likewise admin-free, as are the post-quantum verifiers. So the correct statement is precise: AERE's money flywheel is already ungovernable by anyone, while its parameter and upgrade authority still rests with the Foundation. The roadmap closes that second gap deliberately, and only after the network is hardened enough for it to be safe.

ALREADY OWNERLESS

- AereSink revenue router, 15 / 40 / 45 constructor-set and immutable
- sAERE liquid-staking receipt, no admin, no setters
- AereFeeBurnVault burn endpoint, no withdraw
- The post-quantum verifiers, pure immutable logic

STILL FOUNDATION-CONTROLLED

- Most Ownable protocol contracts, owned by the Foundation-controlled account (a single-key EOA today; a Safe multisig is roadmap), with 18 measured exceptions still under other single-key operational accounts and scheduled to move with the rest
- Parameter and upgrade authority rests with the Foundation
- The stake-weighted vote is advisory, not binding
- Governor and Timelock deployed (addresses in the governance section) but owning nothing yet, so governance is advisory until the ownership transfer

The path to bind

Binding governance means two concrete steps, both listed as roadmap in `docs/ROADMAP-PHASES.md` and neither yet performed. First, a Governor plus a Timelock. Both now exist on chain 2800 (AereGovernorV2 and its timelock, addressed in the governance section below) but they own nothing yet, so they bind nothing. Second, transfer ownership of the protocol contracts from the Foundation-controlled account to that Timelock, so that a parameter change or upgrade requires a passed on-chain vote followed by a timelock delay rather

than a Foundation signature. That ownership transfer is the exact moment the Foundation's unilateral control ends. It is the single largest and most irreversible founder signature in the entire plan, and by design it comes late: after the validator set has grown and after the first external audit, never before.

9.2 The roadmap, phase by phase

The roadmap is written to be verifiable rather than aspirational. Every live capability names the contract that backs it; every capability that is not yet live is labelled in development or roadmap and states the one honest dependency that unlocks it. The dependency vocabulary is deliberately blunt: code, money, people, or a founder signature.

FIG 9.1

The path, phase by phase, with state badges and the one honest dependency that unlocks each

1

The live foundation LIVE

The network that is running: 0.5s single-slot finality, Ethereum-parity EVM, fixed 2.8B supply, the full deployed stack, and the on-chain post-quantum verifier suite.

DEPENDENCY: NONE, IT EXISTS

2

PQC to the protocol layer, first decentralization, first audit

IN DEVELOPMENT

Native PQC precompiles and EIP-2935 lookback are now live on mainnet via the AerePQC fork (block 9,189,161), and the validator set has grown from six to nine (deeper fault tolerance, f=2 with a quorum of six of nine); still ahead, the first external audit.

CODE PEOPLE MONEY FOUNDER SIGNATURE

3

Ecosystem, liquidity, real usage, and a listing ROADMAP

Bridge USDC.e, open the home CLOB with a seeded liquidity ladder, and light up the RWA lending and intent rails; a centralized-exchange listing follows later.

REAL USAGE ORGANIC MAKERS AND USERS

4

21 validators, client-diversity posture, and binding governance

ROADMAP

Grow the set from 9 to 21 under a Validator Charter, hold an honest single-client posture until a shadow-net interop run proves otherwise, deploy the Governor and Timelock, and transfer ownership to it.

PEOPLE CODE THE OWNERSHIP-TRANSFER SIGNATURE

5 **Credibly neutral, quantum-durable settlement** END-STATE

Not a new build: the condition in which the earlier phases are all true at once, precompiles audited and live, an independent 21-node set, binding governance, and real fees flowing continuously into the immutable sink.

CULMINATION OF ALL ABOVE

What it shows. Each phase is coloured by state (green live, amber in development, grey roadmap) and carries the blunt dependency vocabulary that unlocks it: code, money, people, or a founder signature. None of these milestones is reached by announcement. The detailed, verbatim account of each phase follows.

Phase 1: the live foundation (shipped, dependency: none, it exists) LIVE

Phase 1 is the network that is running, not a promise. Consensus is Hyperledger Besu QBFT with 0.5-second blocks, halved from one second mid-chain at block 2,137,652, and single-slot BFT finality, so a transaction is final in well under a second. The EVM tracks Ethereum's ruleset with Pectra activated at block 2,075,341 and Fusaka at block 2,106,597, giving functional parity with Ethereum mainnet, including the RIP-7951 P-256 precompile at `0x100`, the EIP-2537 BLS precompiles, EIP-7702 delegation, and the EIP-7825 per-transaction gas cap of 16,777,216, which AERE keeps deliberately. Supply is fixed at 2,800,000,000 AERE, capped at genesis-v2 on 2026-05-07 across six genesis wallets, with no new issuance in any mechanism.

The deployed, Foundation-owned stack includes the wrapped token WAERE `0x7e84d7d66d5da4cfE46Da67CDEeB05B323e1f5e8`, the staking pool `AereStakingV2`, the immutable flywheel `sAERE` feeding from `AereSink`, the lending engine proven end to end on its `AereLendingMarket` proof market `0x2C2d39dB711C0A33De04Dc74b1E22f4760FD4bb0`, a multi-prover zk-verifier stack routing through the SP1 gateway `0x9ca479C8c52C0EbB4599319a36a5a017BCC70628`, and the rollup validity anchors, from the bounded-VM `AereRollupValidity` `0x38772063572DF94E90351e44ccbBEefD5F497fbd`, whose epoch 0 was recorded from a real SP1 Groth16 proof, up to the full-EVM `AereEVMValidityBatch` `0x49D30a5999eA5b7f6eFf12196AB006316683Ff3C`, which proved and verified a 16-block batch of real chain-2800 blocks in a single Groth16 proof through the SP1 gateway, a proof-of-approach on real blocks not yet at production rollup cadence. That contract is superseded in the canonical registry by the canonical-bound `AereEVMValidityBatchV2` `0xe154B8993FB54dB4418e03ef4a04894f29E690aE`, which is deployed and has not yet recorded a proof, so the batch above is still the strongest recorded result. The genuinely differentiated

piece is the post-quantum verifier suite running on-chain at the application layer: WOTS+, XMSS, Falcon-512, and SLH-DSA-SHA2-128s all record on-chain, while Falcon-1024 `0xF0aFA59BaB2058e4B6e6B424b7f76750F1F66e36` and ML-DSA-44 `0xf1F7A6Acd82D5DAf9AF3166a2F736EE52C5F85AE` now record on-chain through the native precompiles that went live with the AerePQC fork (block 9,189,161, 2026-07-12), since a full record transaction for each exceeds the 2^{24} cap in pure Solidity. We are not aware of any other public chain that verifies all of these on-chain. The honest scope limit is stated at the front: this is application and account layer post-quantum security. Nothing in Phase 1 made consensus post-quantum; consensus became hybrid post-quantum in the anchor-certificate sense later, at block 13,014,000 and on 2026-08-14, by a separate activation, and any claim that Phase 1 did it would be false.

Phase 2: PQC to the protocol layer, first decentralization, first audit IN DEVELOPMENT

State: partly shipped. The AerePQC client fork is live on mainnet (native PQC precompiles plus EIP-2935, block 9,189,161), so the post-quantum precompiles and the extended block-hash lookback are done; what remains is decentralization and the first external audit. Dependency for the rest: people (independent operators), plus money (an audit), plus a founder signature (validator votes). Phase 2 raises the credibility floor in three ways.

The first is native PQC precompiles, now live. The pure-Solidity verifiers are already heavily optimized, but the residual cost is intrinsic, thousands of SHAKE256 permutations per verify, so the fix is native code. The AerePQC fork adds five native precompiles at the reserved band 0x...0AE1 to 0x...0AE5, one each for SHAKE256, Falcon-512, Falcon-1024, ML-DSA-44, and SLH-DSA-SHA2-128s, wrapping an audited native library (Bouncy Castle 1.83) inside the client and validated bit-for-bit against the same NIST KAT and ACVP vectors the on-chain Solidity verifiers already pass (41 of 41 accept and reject cases). Proven first on an isolated Besu 26.4.0 scratch fork (chain 28099), they were activated on mainnet 2800 by the AerePQC fork at block 9,189,161 (2026-07-12): all KATs pass and measured verify-and-record gas is 86,336 for Falcon-512, 145,496 for Falcon-1024, 351,050 for ML-DSA-44, 558,276 for SLH-DSA-128s, and 21,470 for SHAKE256, all comfortably under the 16,777,216 cap. Activation was a coordinated flag-day hard fork with no re-genesis and no state migration, adding behavior only at otherwise-empty addresses, and the same fork also activated the extended EIP-2935 block-hash lookback (an 8191-block window). The fork added the precompiles and EIP-2935 only; it did not change consensus, and Block-STM parallel execution was not part of it and remains a testnet demonstration. Beyond this application and execution layer capability, consensus

itself carries a post-quantum checkpoint: since 2026-08-14 no anchor block (every 32nd) finalizes without at least three valid Falcon-512 validator seals (f+1 of nine) (raised at block 14,961,456, August 21, 2026: the enforced minimum is now six of nine, a full 2f+1 quorum) under its hash, with classical ECDSA finalizing every block. Separately, a validator finality-attestation layer has been demonstrated on an isolated 4-validator testnet as a gossiped, verified 2f+1 Falcon-512 quorum certificate, every validator co-signing each block and every node re-verifying it on import, blocking after a fork; across two healthy runs (229 and 215 blocks) every post-fork block carried a valid 3-of-4 certificate alongside the decisive ECDSA seal with zero rejects and no halt; that demonstration was later carried to mainnet in the anchor-certificate form described in chapter 3, not as a per-block quorum. The measured numbers and the KAT methodology are detailed in `research/pqc-onchain-verification.md`.

The second was decentralization to seven validators, reached on 2026-07-12, and to nine on 2026-08-09, through one-at-a-time header-voting admission with zero downtime. At n=9 the tolerated Byzantine faults f = floor((9-1)/3) = 2, the credible minimum for a public BFT network, so the chain survives up to two simultaneous validator failures. The set is still Foundation-operated. Each new operator must be an independent legal entity on independent infrastructure, in a different jurisdiction; only such admissions, none of which has happened yet, would remove the single-operator-can-halt objection. It does not by itself make the network decentralized, and it will not be described that way until the Phase 4 targets are met and independently verifiable.

The third is the first external audit. AERE has had no external audit of its contracts or client changes; the numbered findings already fixed in the registry are internal audit discipline, not an independent check. The native PQC precompiles are now live on mainnet through the AerePQC fork, but that activation was a founder decision rather than the product of an independent review, and Besu still produces every mainnet block, so no second producing client is there to catch a consensus bug at runtime, so Phase 2 commissions a real external audit of the now-live native PQC code path and the rest of the stack.

Phase 3: ecosystem, liquidity, real usage, and a listing ROADMAP

State: roadmap, code largely ready. Dependency: real usage, organic makers and users, who cannot be manufactured. AERE trades first on its own native on-chain order book; a centralized-exchange listing follows later.

The native CLOB code (`AereCoreBookV0`, named by artifact because nothing is listed and no market address exists) is written and internally tested, but the market map is empty and the trade UI stays in demo mode until it is populated. Going live means bridging USDC.e onto chain 2800 through a Hyperlane Warp Route, deploying the WAERE/USDC.e market with immutable tick, lot, and taker-fee parameters, and seeding a two-sided liquidity ladder for that market. From day one every taker fee routes into `AereSink`, so trading volume becomes fuel for the token's own burn-and-yield flywheel. The RWA lending markets and the intent and compliance rails are coded and waiting on the same USDC.e bootstrap. A centralized-exchange listing follows later, once the home order book has real, organic trading depth. The honest cost of that patience is the absence of a fiat on-ramp in 2026. The public answer is simple and true: the price of AERE forms on AERE.

Phase 4: 21 validators, client-diversity posture, and binding governance ROADMAP

State: roadmap. Dependency: people (15 or more independent operators and a governance quorum), plus code (a Governor and Timelock; the second-client interop run is now proven on an isolated testnet, so what remains is the supervised live admission rather than the interop work itself), plus a founder signature (the ownership transfer, the single largest irreversible action in the plan). Phase 4 is where the Foundation stops being able to act unilaterally.

Twelve further one-at-a-time admissions lift the set from 9 to 21 validators, raising f to 6 at n=21 with a quorum of 14 of 21. The published distribution targets are concrete: at least 15 distinct legal operators, no operator running more than two validators, infrastructure and network paths independent across operators, and at least five jurisdictions, gated by a published Validator Charter and a charter admissions vote so no single party, the Foundation included, can seat or unseat a validator. Client diversity is stated without exaggeration. For a QBFT chain it is genuinely limited, and the second client is no longer a research item: a patched Nethermind has validated and followed live chain 2800; as of 2026-08-10 it is stalled behind the head and the gap is published rather than hidden, and on an isolated testnet it reaches QBFT consensus with Besu as an equal validator, with both clients proposing and every committed block carrying a cross-client seal quorum (Chapter 2). What remains is the live step, and it is deliberately the slow one, because a bug in a producing client can halt or fork a live chain: extended soaking, review, and a founder-supervised admission of the first mixed-client mainnet validator. Until that happens, the honest statement is a single-producer mainnet on Besu, mitigated by independent operators, staggered patch windows, and never upgrading all validators at

once, with a second validating client on mainnet and cross-client consensus proven on testnet. It is not client-diverse today. Binding governance, described in Section 9.1, is the third part: deploy the Governor and Timelock, then transfer ownership of the protocol contracts to the Timelock. That transfer follows, and does not precede, both the validator expansion and the external audit.

Phase 5: credibly neutral, quantum-durable settlement END-STATE

State: the end-state the earlier phases converge on. Dependency: the culmination of all of the above, mostly people and sustained real usage plus the founder relinquishing unilateral control. Phase 5 is not a new build. It is the condition in which the earlier phases are all true at once, defined concretely so it can be checked rather than asserted: the native PQC precompiles are audited and active on mainnet through a completed flag-day fork, with the permanent honest caveat that this is application and account layer quantum resistance and consensus still signs classical secp256k1; the validator set is 21 nodes across at least 15 operators, five jurisdictions, and eight networks, tolerating six simultaneous faults; the Governor and Timelock own the protocol contracts and the Foundation is one voice among many; and real fee streams flow continuously into `AereSink`, whose immutable split burns AERE and lifts the `sAERE` exchange rate, with extra burn accruing to `AereFeeBurnVault`. The mechanism is immutable and live today; what Phase 5 adds is the volume that makes it matter. This is the destination the brand thesis names: final as math, safe when the math changes.

Cross-phase dependency summary

PHASE	PRIMARY UNLOCK	HONEST DEPENDENCY
1	Live 0.5s-finality chain, full stack, on-chain PQC verifier suite	● NONE, IT EXISTS
2	Native PQC precompiles (fork, live), nine validators (f=2, quorum 6-of-9), first external audit	code + people + money + founder signature
3	Home CLOB liquidity, real RWA markets, native order-book trading	real usage + organic makers and users
4	9 to 21 validators, client-diversity posture, Governor/Timelock ownership	people + code + the ownership-transfer signature

PHASE	PRIMARY UNLOCK	HONEST DEPENDENCY
5	Precompiles live on mainnet, independent set, binding governance, self-sustaining flywheel	culmination of all above

9.3 Engineering direction: distinct before fast

The roadmap above answers a question about state: what runs, what is gated, and on what. This section answers a different one, about ordering. AERE's architecture is settled. Consensus is settled, the post-quantum verification layer is settled, the EVM ruleset is settled. What is left is a long list of things that could be built next, and the only decision that matters is which of them deserve a year. That decision is made by a single published criterion, written here so it can be held against us.

The criterion. Every candidate is sorted into one of two piles: work that makes AERE distinct, and work that makes AERE faster. Anything in the second pile can be built by anyone with money and engineers. Anything in the first pile cannot. When the two compete for the same engineering quarter, the first pile wins. Verkle tries, a QUIC transport, io_uring storage, NUMA-aware scheduling, SIMD hashing and predictive caches are each a genuine improvement, and not one of them changes the category the network competes in. A year spent there produces a faster client, not a different one. This section exists so that year is spent deliberately, and so a reader can check later whether it was.

The three pieces of work ordered first

1. The post-quantum seal quorum becomes part of the header commitment. Chapter 2 describes the research demonstration in which every validator on an isolated test network post-quantum signs each block's commit hash and gossips that seal inside its consensus message, so each block embeds a quorum certificate that every node re-verifies on import. The first ordered piece of work makes that certificate part of what the block header itself commits to, through a fixed-size digest carried inside the header data that is already covered by the block hash, so the certificate is bound by the same hash that binds everything else in the header. That is the step which turns post-quantum evidence from something a block carries into something a block's identity depends on. It is an execution item rather than a research item: the design is fixed, the guards are implemented and compile and pass together in a real client, each guard is proven able to fail under a negative control with the guard removed, the second client accepts the header element at compile time and at run time on real headers, and the activation sequence has been

rehearsed on a test network. In the roadmap's own blunt vocabulary: the header anchor went live at block 13,014,000 and its enforced minimum has been three seals (f+1 of nine) since 2026-08-14, raised to six of nine, a full 2f+1 quorum, at block 14,961,456 (2026-08-21), so an anchor block without that certificate does not finalize, with classical secp256k1 ECDSA finalizing every block. Correction published 2026-08-19: this document previously stated that from block 14,050,000 every block required a 2f+1 (six-of-nine) post-quantum quorum to finalize. The per-block Falcon quorum rule armed at that height is, in the shipped code, retired in favour of the anchor rules from block 13,014,000, and a re-reading of the code and of the chain (non-anchor blocks carry no Falcon seals) shows it changed no enforcement; that statement is withdrawn. What is enforced is the anchor certificate described here.

2. From an L1 EVM to an L1 execution kernel. Today the EVM is the execution layer. The target is an execution kernel that owns state access, scheduling, commitment and proving, with the EVM as its first virtual machine rather than its only one, so that adding a second machine, RISC-V being the obvious candidate, is an additive change rather than a rewrite. Two things are declared immovable in advance and by design: consensus does not change, and the state model does not change. This is the same additive discipline Chapter 3 applies to post-quantum cryptography, applied to execution. It is deliberately written as an architecture document before it is written as code, and the document is not permitted to pass without answering three questions. Where exactly is the boundary, and what interface crosses it. What is declared unchangeable, named explicitly rather than assumed. And which three measurements decide whether the work was worth doing, given as numbers with a stated method rather than as adjectives. A kernel is rewritten once. If the boundary is drawn in the wrong place, nobody finds out for a year, which is precisely why the document comes first.

3. A cryptographic abstraction layer. Today, algorithms are bound to the protocol. The target is protocol to interface to algorithm, so that a primitive or a parameter set can be replaced by registering a new verifier behind an unchanged interface instead of by editing the protocol. This is insurance, not polish. Cryptographic recommendations move: standards bodies revise parameter sets, and the industry has already leaned toward the larger Falcon parameters, which is why AERE's native precompile is Falcon-1024. On the day a primitive is superseded, the whole difference between a week of work and six months of work sits in this layer. It is inexpensive to build before it is needed and expensive to build after, which is why it is third rather than tenth.

The queue, sorted by the same criterion

The rest of the engineering programme is published below in the two piles, in order. Nothing here is a claim about what exists; Section 9.2 is the account of what exists. This is an ordering, and it deliberately carries no dates.

WORK THAT MAKES AERE DISTINCT	WHY IT IS IN THIS PILE
Post-quantum consensus, end to end	Validator signatures, consensus messages, peer authentication, slashing evidence, checkpoint signatures, light-client proofs and bridge signatures, all post-quantum natively rather than merely prepared for. This is the direct continuation of the first ordered item above, and it is the one line of work that would make the network's security assumption different in kind from every other chain's.
Native proof of execution	Chapter 4 describes the validity anchors that already record real proofs of real chain-2800 execution. The distinct version of that work is a chain where every block produces a proof of its own execution by default rather than by opt-in. It changes what the chain guarantees, not how quickly it does it.
The execution kernel	See item 2 above. It is the only item on the whole list that changes what the network is rather than how fast it runs.
The cryptographic abstraction layer	See item 3 above. Agility under a change of primitive is a property no amount of later optimization can retrofit cheaply.

WORK THAT MAKES AERE FASTER	SCOPE
Parallel execution, second generation	Conflict prediction, an explicit dependency graph, parallel scheduling and a proof obligation on top of the Block-STM work in Chapter 2.
Trie engine, second generation	Verkle commitments, SIMD hashing, cache locality, compressed witnesses, parallel hashing and incremental commitment.
Fully parallel commit	Execution, state update, trie update, hashing, witness generation and proof generation all overlapped rather than staged.
Adaptive QBFT	Adaptive timeouts, proposer selection, message batching, peer selection and retransmission.

WORK THAT MAKES AERE FASTER	SCOPE
Post-quantum engine, second generation	SIMD and AVX512 paths, GPU and FPGA offload, batching, a verification cache and speculative verification.
Network stack	QUIC, multipath, adaptive gossip, erasure coding, congestion-aware routing, prioritization and binary propagation.
Scheduler	NUMA-aware, CPU-topology-aware, SIMD-aware and cache-aware, on an allocation-free hot path.
Native block builder	Ordering by throughput, trie locality, cache locality, dependency structure and verification cost.
State cache	Multi-layer, predictive, trie-aware and witness-aware.
RPC engine	Asynchronous, zero-copy, binary and streaming.
Validator runtime	Lock-free and wait-free where the algorithm permits, allocation-free and copy-free on the critical path.
Storage layer	io_uring, direct I/O, huge pages, asynchronous writes and adaptive compaction.
Signature aggregation	Aggregation applied wherever the protocol allows it rather than only where it is conventional.
Light client	Fewer proofs, less data and fewer hashes for the same assurance.
Conflict-aware mempool	Ordering by conflict, dependency, priority and locality so the executor receives work it can parallelize.
Economic verification	Model checking, SMT solving, and invariance and simulation proofs for the fee and burn mechanisms described in Chapter 8.

Two rules that govern the whole list

Formal verification is aimed, not sprayed. The target is complete coverage of the invariants whose failure would halt the chain or lose user funds, not complete coverage of lines of code. Verifying everything sounds stronger and is weaker in practice, because it consumes exactly the engineers and months the distinct pile needs. Stating the target as invariant

coverage makes it a decision that can be argued with rather than an omission that has to be discovered.

One at a time, and proven. Twenty simultaneous work sites are not a programme, and a document reporting twenty things in progress says nothing reliable about any of them. Each item above is finished only when it carries a measurement and a negative control, meaning its own proof has been shown capable of turning red when the property it checks is removed. A check that has never failed cannot be trusted. That rule is the same one the verification section of this document applies to itself.

9.4 Risks and limitations

This chapter consolidates the weaknesses named throughout the whitepaper into one place, in plain language, so a reader never has to reconstruct them from footnotes.

> Centralization and fault tolerance. AERE today is a permissioned, single-operator network: nine validators, all keys held and run by the Foundation. With n=9 the tolerated Byzantine fault count is two (quorum 6-of-9), so the chain survives two simultaneous validator outages, but all nine are one operator. Stated in the standard unit, the Nakamoto coefficient is 1: one party can halt the chain, because that party is every party. Fault tolerance of two is a property of the algorithm, not of the operator set, and the two should not be confused. This is the most material limitation, and the completed Phase 2 step to nine and Phase 4 (to 21) exist to remove it in order.

> One producing client. The network produces every block with a single client (Besu). A second, independent client (a patched Nethermind) has validated and followed live chain 2800, which is verification-layer diversity when it is caught up; as of 2026-08-10 it is stalled behind the head, and it reaches consensus with Besu on an isolated testnet, but it produces no mainnet blocks. The verification-layer result is real and worth its exact weight: the two clients agree on block hash and state root at five heights spanning the post-quantum activation block 9,189,161, which is where a divergence would be most likely and most damaging. That is genuine cross-implementation assurance on the live chain. It is not production diversity, and the difference is that a bug in the sole producing client still has nothing to stop it from being written into a block. For a QBFT chain the live step is genuinely hard and founder-supervised, so this is stated as a mitigated single-producer posture, never as full client diversity.

> No external audit. Neither the contracts nor the client changes have had a third-party audit. The numbered findings already fixed and redeployed in the registry are the product of internal audit rounds, which is groundwork, not an independent verdict. The native PQC precompiles are already live

on mainnet via the AerePQC fork, so the first external audit that Phase 2 commissions is an independent review of code now in production rather than a gate before activation.

Governance is not yet binding. Most `Ownable` protocol contracts are owned by the Foundation-controlled account `0x0243A4f47D44b40b65D33f20329dE20D00c6f3C3` (a single-key EOA, not a threshold multisig; a Safe multisig is roadmap); measured on 2026-08-11, 61 of the 79 registry contracts that answer `owner()` return that account and 18 return other single-key accounts, and not one is owned by a timelock or a governor contract. The Governor and the Timelock are no longer pending deployment: `AereGovernorV2` `0x03251BD9A115385B76220b17914f56eAC0773047` carries 16,616 bytes of live code, its vote source `AereGovLockV2` `0x547F58B087BEbB1746d6159070E755b868C27731` carries 10,240, and the timelock `0x9d618c5AB5f40c7187fc8D644d1aC9Bae0805f85` carries 7,427 and reports a minimum delay of 172,800 seconds, which is 48 hours. The stake-weighted vote stays advisory because ownership has not been transferred to that stack, not because the stack is unbuilt. The mitigating fact is that the money flywheel (`AereSink`, `sAERE`, `AereFeeBurnVault`) and the verifiers already have no admin key at all.

Thin usage, and subsidy-funded yield. Realized throughput and fee flow are early. Because supply is fixed at 2.8B with no consensus-level minting, staking yields, the 8 percent default delegated APR on `AereStakingV2` and the 10 to 30 percent locked APYs, are paid from Foundation-seeded genesis reserves, not from issuance. They are a subsidy with a runway, safe because principal is protected, but they are self-sustaining only once real fee revenue routed through `AereSink` replaces the reserve. Burn and yield figures should be read live from the on-chain counters, never assumed. AERE remains a fixed-supply chain: there is no adaptive issuance and no new token.

No fiat on-ramp, no listing. There is no fiat gateway and no CEX listing in 2026. A retail user who does not already hold a wallet, AERE, and bridged USDC.e cannot yet reach the market. This sequencing is deliberate: the price of AERE forms on its own on-chain order book first, with a centralized-exchange listing to follow later.

Post-quantum scope. The post-quantum property is at the application and execution layer, not consensus. On mainnet 2800, Falcon-1024 and ML-DSA-44 record on-chain through the native precompiles that went live with the AerePQC fork (block 9,189,161, 2026-07-12); their pure-Solidity verifiers stay read-only references because a full record transaction exceeds the 2^{24} cap in pure Solidity. Since 2026-08-14 AERE mainnet consensus is hybrid in a precise and limited sense: every 32nd block (an anchor block) does not finalize without a certificate of at least three valid Falcon-512 validator seals (f+1 of nine) (raised at block 14,961,456, August 21, 2026: the enforced minimum is now six of nine, a full 2f+1 quorum) under its block hash, with classical ECDSA (secp256k1) QBFT finalizing every block; the flip was a separate, explicitly approved decision, delivered through

coordinated per-node activation heights rather than a re-genesis, exactly as this document said it would be.

Validator economics are currently nil, and so is the burn. Validator coinbase revenue on mainnet 2800 is zero, for two independent reasons: genesis declares no `blockreward`, and the 1 Gwei base fee is destroyed by EIP-1559 rather than paid to the proposer. Because the 37.5% burn is a share of that reward, it currently burns effectively nothing: 0.13735259 AERE over the chain's lifetime, against 2,800,000,000 AERE of supply. **Aere is not deflationary today.** The mechanism is live, immutable and ownerless and will burn in proportion to real fee flow when there is real fee flow; that is a demand problem, not a contract problem. Today the network is secured by an operator who is paid nothing by the protocol, which is sustainable only while that operator is also the Foundation. Chapter 6 gives the measurement and the two RPC calls that reproduce it.

A documented deviation from written EIP-2935. Anyone running an Aere node should know this before they debug it. Between blocks 9,182,380 and 9,189,160 chain 2800 deviates from the EIP-2935 specification as written, because Besu skips the history-window write on proof-of-authority chains that lack the three request-contract addresses. The behavior is Besu's, not a local patch, and it is deterministic and identical across the clients we have compared, so it does not threaten consensus. It does mean that a historical block-hash lookup into that window can return a result that the written specification would not predict. We publish the exact range rather than let an integrator discover it as a mystery.

Key custody is a single point of failure. Signing authority for chain 2800 is concentrated: the same operator holds the validator signing keys and the Foundation account, and neither hardware custody nor a threshold scheme is in use for them yet. The consequence is the honest one to state, so it is stated: losing or compromising that custody is losing or compromising the network's authority, and hardening it is a Phase 2 item rather than a solved problem. Measured on 2026-08-01, the Foundation address is a single key rather than a multisig, holds 179,999,989.99 AERE, and, re-measured on 2026-08-11, is returned by `owner()` on 61 of the 79 canonical-registry contracts that answer it; no contract on chain is currently owned by a timelock. The specific arrangement of key storage is deliberately not described here, because that detail helps an attacker and helps a reader not at all. This is stated in the same document that describes the governance roadmap because the gap between the two is the point.

Part of the genesis supply sits outside its genesis wallet. Measured on 2026-08-01, the six genesis addresses hold 2,239,999,990.70 AERE against a 2,800,000,000 cap. The Ecosystem Reserve allocation was moved off its genesis address, so 559,999,999 AERE of the difference is held at a single separate account and the rest is the fee spend of those wallets since genesis. The cap is unchanged, nothing was minted and nothing was destroyed. What is open is documentation: bringing that balance under the same governance handover as the rest of the protocol, and publishing an

attestation for it, is an unfinished item and is listed here rather than left for a reader to find by adding up the table in Chapter 8.

Test coverage is partial, and here is the number. Contract test coverage is 36.44% of statements, 28.86% of branches and 32.14% of functions, with 161 of 270 contract files at zero coverage. The full suite runs end to end for the first time at 2,055 passing, 8 failing and 23 pending across 175 files. The core economic contracts are the well-covered ones and the untested majority is largely peripheral, but 36% is 36% and it is published rather than rounded into a claim about rigor.

Throughput framing. The 273,000 TPS figure that appears in AERE's design materials is a design ceiling derived from the architecture, not a measured mainnet result. It is never presented as observed throughput.

Every one of these is a fact the roadmap is built to change, in a named order, behind a named gate. None of them is reached by announcement. They are reached by shipping the code, running the audit, recruiting the operators, and casting the votes, and each milestone is written so it reads as accurate only when it is independently verifiable on-chain.

 Whitepaper Chain ID 2800 · 0xAF0 Status 2026-08-23

This page and its Markdown master `docs/WHITEPAPER-V2.md` have drifted apart and the footer used to claim they had not. Measured on 2026-08-02: this page carries 103 distinct contract addresses, the Markdown master 93, the print copy 92, and ten addresses appear here that are absent from the master, so regenerating the master over this page would delete published sections. Merging the two is an open item. Every contract address here is copied verbatim from the canonical registry `sdk-js/src/addresses.ts`, and where the registry marks an address as superseded this document says so and names the replacement. Every claim in this document is meant to be checked on-chain rather than taken on faith.