Test Cases for Epic 1 – Enclave-Bound Anonymous Ingress & Mixer Network

Goal Recap: Transactions enter the network completely blind, routed through hardware-attested TEEs with traffic-analysis resistance.

Test Environment Assumptions (Common to All):

- Testnet running with ≥50 relay nodes (multi-vendor TEEs: Intel SGX, AMD SEV-SNP, ARM CCA/TrustZone simulated or real).
- Wallet app (iOS/Android/desktop/web) installed and synced.
- Tools: Wallet GUI/CLI, onion-inspect CLI tool (for parsing serialized onions), tcpdump/Wireshark for packet capture, attestation verifier CLI (vendor-specific: Intel DCAP, AMD VCEK, ARM PSA), node logs viewer, DHT query tool, latency ping script, hardware RNG checker if available.
- Test accounts with balances (>1 test NERV each).
- Debug modes enabled on nodes/wallets for forcing routes and verbose logging.
- All tests assume basic node setup: ./nerv-relay --tee-vendor=sgx --port=4433 etc.
- Record everything in a test log spreadsheet: test ID, date/time, node IPs, tx_hash/onion_hash, screenshots/commands/output snippets, timings.

Reference User Story: ING-01

As a user, I can generate an enclave-routable onion transaction with 3–5 hops

Acceptance Criteria: Transaction <4 KB serialized, contains encrypted payload + 5-layer onion (public keys of TEE relays), signed with Dilithium key.

Test Case ID: TC-ING-01-01

Description: Positive case – Generate and inspect valid default 5-hop onion transaction.

Prerequisites: Wallet synced, balance >0.5 test NERV, DHT refreshed (≥100 attested relays available).

Detailed Test Script:

1. Open wallet app → go to "Send Private" tab.
2. Enter recipient blinded commitment (use test account 2).
3. Enter amount: 0.05 test NERV.
4. Optional metadata: leave blank for minimal size.
5. Select "Max Privacy" (default 5 hops).
6. Tap "Preview Transaction" → note displayed onion size estimate.
7. Tap "Sign & Broadcast" → biometric/PIN confirm.
8. Wallet shows "Submitted" with tx_hash/onion_hash (e.g., 0x123...). Copy it.

9. Export raw serialized onion: Wallet menu → "Export Raw Onion" → save as hex/base64 to onion_positive.hex.
10. Measure size: Command line wc -c onion_positive.hex or ls -lh (convert hex to bytes if needed: size_bytes = hex_length / 2).
11. Inspect with tool: ./onion-inspect onion_positive.hex --verbose.
12. Check output: Exactly 5 layers, each with ML-KEM public key, outer Dilithium signature valid (tool shows "Signature: VALID").
13. Verify signature manually: ./dilithium-verify --pubkey user_pubkey.bin --sig outer_sig --msg outer_blob.
14. Screenshot wallet preview, export, and inspect output. Record fields: hop count, size, sig status.
15. Repeat 5 times, record average size.

Expected Results:

- Serialized size: 3000–3800 bytes (<3888 bytes hard cap).
- Inspect shows 5 distinct relay PKs, layered encryption intact, no plaintext visible.
- Outer signature VALID, signed by user's Dilithium key.

Pass/Fail: Pass if size <4 KB, 5 layers, valid sig.

Troubleshooting: If size over, check metadata; if no relays, force DHT refresh.

Test Case ID: TC-ING-01-02

Description: Configurable hops – Generate 3-hop onion.

Detailed Test Script:

1. In wallet settings → "Privacy Level" → select "Fast (3 hops)".
2. Repeat steps 2–14 from TC-ING-01-01.
3. Inspect: Should show exactly 3 layers.

Expected Results: 3 layers, smaller size (~2200–3000 bytes), valid sig.

Pass/Fail: Pass if matches config.

Test Case ID: TC-ING-01-03

Description: Negative – Oversize rejection.

Detailed Test Script:

1. Use debug wallet mode (if available) or modify to add large metadata (e.g., 2 KB note).
2. Attempt to generate/sign.
3. Observe error message.

Expected Results: Wallet error "Onion exceeds max size (3888 bytes)" before signing. No broadcast.

Pass/Fail: Pass if rejected gracefully.

Reference User Story: ING-02

As a relay node, I can receive, decrypt one layer, attest in TEE, and forward without logging

Acceptance Criteria: Remote attestation report included, measurement hash = known good, no plaintext in logs/RAM.

Test Case ID: TC-ING-02-01

Description: Positive – Relay processes valid onion with attestation.

Prerequisites: Run local relay node: ./nerv-relay --tee-vendor=sgx --debug-log. Force wallet to route through it (debug config: set relay list to your node's IP:port as hop 1).

Detailed Test Script:

1. Start packet capture: sudo tcpdump -i any port 4433 -w relay_capture.pcap.
2. Send private tx from wallet (force route via your relay). Record onion_hash.
3. Monitor relay logs: tail -f relay.log | grep "Processing onion".
4. Look for lines: "Decrypted layer", "Generating attestation", "Forwarding to next_hop".
5. Stop capture after forward observed.
6. Open pcap in Wireshark → filter outbound packets → extract forwarded onion blob + attestation report.
7. Verify attestation: ./attestation-verify --vendor=sgx --report forwarded_report.bin --expected-hash known_good_measurement.txt.
8. Grep logs for sensitive data: grep -i "amount\|balance\|recipient" relay.log (should be empty).
9. If RAM dump tool available (debug mode), check no plaintext.
10. Screenshot logs, Wireshark attestation field, verify output. Record timing (<50 ms processing).
11. Repeat with SEV/TrustZone node if multi-vendor setup.

Expected Results:

- Forwarded packet includes fresh attestation report.
- Measurement hash matches published good hash (no debug mode).
- Logs show processing but NO plaintext/inner data.
- No sensitive strings in logs/RAM.

Pass/Fail: Pass if attestation valid and zero leakage.

Troubleshooting: If no attestation, check TEE sealing; if plaintext, fail security.

Test Case ID: TC-ING-02-02

Description: Cover traffic generation.

Detailed Test Script:

1. Configure relay chaff ratio=5x via admin command.
2. Idle relay 5 minutes → capture outbound traffic.
3. Send 10 real txs spaced 10s.
4. Analyze pcap: Count packets, classify real (from known wallet IP) vs chaff (random destinations).

Expected Results: ~50 chaff packets per real, indistinguishable size (1500 bytes padded).

Pass/Fail: Pass if ratio ≈5x.

Test Case ID: TC-ING-02-03

Description: Negative – Reject invalid onion.

Detailed Test Script:

1. Craft malformed onion (wrong key, bad format) using debug tool.
2. Send directly to relay port (nc or curl).
3. Check logs/response.

Expected Results: Log "Invalid layer format", drop packet, no forward, no crash.

Pass/Fail: Pass.

Reference User Story: ING-03

As the final relay, I can decrypt the inner payload and submit to shard ingress queue

Acceptance Criteria: Payload ZK-wrapped, adds attestation + timestamp, forwards blinded to mempool.

Test Case ID: TC-ING-03-01

Description: End-to-end final relay submission.

Prerequisites: Run local final relay, force wallet route ending there.

Detailed Test Script:

1. Capture on final relay as TC-ING-02-01.

2. Send tx → monitor logs for "Decrypt inner payload", "Verify Halo2 proof", "Add attestation/timestamp", "Push to mempool".
3. Query shard mempool RPC: curl ... nerv_getMempool.
4. Check submitted entry: Blinded tx, chain of 5 attestations, secure timestamp.
5. Verify inner Halo2 proof manually if tool available.

Expected Results: Tx in mempool blinded, full attestation chain, timestamp monotonic.

Pass/Fail: Pass if reaches mempool correctly.

Test Case ID: TC-ING-03-02

Description: Double-spend rejection.

Detailed Test Script:

1. Submit same tx twice (replay onion).
2. Check logs for "Duplicate detected", drop + slashing evidence.

Expected Results: Second dropped, evidence logged.

Pass/Fail: Pass.

Reference User Story: ING-04

As a light client, I can discover 100+ active attested relays via DHT

Acceptance Criteria: Only valid/unexpired attestations, random diverse sample of 5.

Test Case ID: TC-ING-04-01

Description: Discovery and diversity.

Detailed Test Script:

1. Run light wallet → debug → "Refresh Relays".
2. Log/print discovered list (≥100 entries).
3. For each: Check attestation expiry > now + 1h, valid signature.
4. Send tx → log selected 5 relays (IPs, vendors, geo if tagged).
5. Check diversity: No same vendor/IP dominant.

Expected Results: ≥100 valid relays, 5 selected diverse.

Pass/Fail: Pass.

Test Case ID: TC-ING-04-02

Description: Expired relay filtering.

Detailed Test Script:

1. Simulate expired attestation on one relay (debug).
2. Refresh → should exclude it.

Expected Results: Only fresh attestations.

Pass/Fail: Pass.

Reference User Story: ING-05

As the network, I enforce cover traffic & timing obfuscation

Acceptance Criteria: Chaff 1–10×, jitter ±200 ms, fixed padding.

Test Case ID: TC-ING-05-01

Description: Jitter and padding observation.

Prerequisites: Capture on multiple relays.

Detailed Test Script:

1. Configure jitter ±200 ms, padding 1500 bytes.
2. Send 20 txs with exact 500 ms spacing from wallet.
3. Capture outbound inter-packet delays.
4. Plot delays (Excel or script).

Expected Results: Delays normal distribution around 100 ms mean ±200 ms, all packets 1500 bytes.

Pass/Fail: Pass if obfuscated.

Test Case ID: TC-ING-05-02

Description: Configurable chaff.

Detailed Test Script:

1. Set ratio=10x on relay.
2. Idle + low traffic → count chaff vs real.

Expected Results: ~10x chaff.

Pass/Fail: Pass.

Reference User Story: ING-06

As an auditor, I can cryptographically prove no relay ever linked sender → tx

Acceptance Criteria: Replay proves k-anonymity >1,000,000.

Test Case ID: TC-ING-06-01

Description: Run auditor tool on traffic.

Detailed Test Script:

1. Capture multi-relay traffic (1-hour high volume).
2. Run ./unlinkability-auditor --replay capture.pcap --attestations all_reports/.
3. Review metrics: k-anonymity, correlation coefficients.

Expected Results: k >1,000,000 passive, >100,000 active; Pearson <0.01.

Pass/Fail: Pass.

Test Case ID: TC-ING-06-02

Description: ProVerif formal check (if model available).

Detailed Test Script:

1. Run provided ProVerif model on protocol spec.
2. Check for unlinkability proof.

Expected Results: ProVerif outputs "proven".

Pass/Fail: Pass.

Test Cases for Epic 2 – Verifiable Delay Witnesses & Blind Validation

Goal Recap: Any user can prove a tx is canonical with a ~1-2 KB witness, no full chain download needed.

Test Environment Assumptions (Common to All):

- Testnet running with ≥50 full nodes (shards active, consensus operational from Epic 1 & 4).
- Wallet app (mobile/desktop/web) fully synced, with private tx support.
- Tools: Wallet GUI/CLI, vdw-verify CLI tool, Halo2 verifier binary (for all platforms), Wireshark/tcpdump for network capture, node RPC client (e.g., curl or custom script), Arweave/IPFS query tools, benchmark apps (Xcode Instruments for iOS, Android Profiler, Chrome DevTools).

- Test accounts with balances, multiple private txs sent in advance.
- Reorg simulation tool (debug mode on testnet nodes to force shallow reorgs).
- All tests assume a successful private tx submission (from Epic 1) that reaches finality.

I'll provide **more detailed scripts** than Epic 1: each step includes exact commands (where applicable), screenshots/notes expectations, data to record, and troubleshooting tips for junior testers.

Reference User Story: BV-01

As a shard node, I can generate a 1–2 KB VDW for any committed tx

Acceptance Criteria: Contains tx_hash, shard_id, lattice_height, embedding_delta_root, tee_sig; size ≤1800 bytes.

Test Case ID: TC-BV-01-01

Description: Positive case – Successful VDW generation after tx finality.

Prerequisites: Run a full shard node locally (with TEE enabled). Send a private tx visible in your node's mempool. Wait for finality (~2s median).

Detailed Test Script:

1. Start your shard node in debug mode: ./nerv-node --dev --tee-debug (logs all events verbosely).
2. Send a private test tx from wallet (amount 0.01 test NERV to another test account). Record the tx_hash (displayed in wallet as "Pending tx_hash: 0xabc...").
3. Monitor node logs: tail -f node.log | grep "Finality achieved". Note the block/lattice height when finality occurs.
4. Once finalised, query node RPC for VDW: curl -X POST http://localhost:9933 -H "Content-Type: application/json" --data '{"jsonrpc":"2.0","method":"nerv_getVDW","params":["0xabc..."],"id":1}'.
5. Save response VDW blob to file: copy the "result" hex/base64 to vdw_positive.bin.
6. Measure size: ls -lh vdw_positive.bin or wc -c vdw_positive.bin.
7. Parse VDW with debug tool (if available): ./vdw-inspect vdw_positive.bin – check fields: tx_hash matches, shard_id correct (from node config), lattice_height matches finality log, inclusion_proof present (≤750 bytes), embedding_root (32 bytes), TEE attestation report + Dilithium sig (96 bytes).
8. Record all fields in test log spreadsheet (columns: field name, value, expected).

Expected Results:

- Size: 1200–1600 bytes (average ~1400).
- All required fields present and correctly populated.
- No errors in node logs during generation.

- Generation logged: "VDW generated for tx_hash in <200 ms".

Pass/Fail: Pass if size ≤1800 bytes and all fields valid.

Troubleshooting: If no VDW, check finality reached (force batch if needed via debug RPC).

Test Case ID: TC-BV-01-02

Description: Boundary case – Maximum size VDW with batched/large delta proof.

Detailed Test Script:

1. Send 256 private txs in quick succession (scripted via wallet SDK or CLI loop) to force a full batch.
2. Wait for batch finality.
3. Query VDW for one tx in the batch (use any tx_hash from the batch).
4. Save and measure as in TC-BV-01-01 step 5–7.
5. Compare inclusion_proof size (should be closer to ≤750 bytes max).

Expected Results: Size ≤1800 bytes even for batched tx. Proof component larger but total capped.

Pass/Fail: Pass if ≤1800 bytes.

Test Case ID: TC-BV-01-03

Description: Negative case – No VDW for uncommitted tx.

Detailed Test Script:

1. Send tx but prevent finality (pause consensus on testnet cluster if possible, or use invalid tx).
2. Attempt RPC query for VDW immediately.
3. Check response error code.

Expected Results: Error "VDW not available: tx not finalised" or similar. No blob returned.

Pass/Fail: Pass if graceful error.

Reference User Story: BV-02

As a light client, I can verify a VDW in <80 ms on iPhone 15

Acceptance Criteria: Uses Halo2 recursive verification, offline after receipt.

Test Case ID: TC-BV-02-01

Description: Performance & correctness – Offline verification on mobile (iOS).

Prerequisites: Valid VDW from TC-BV-01-01 saved. iPhone 15 test device with wallet app installed. Device offline (airplane mode).

Detailed Test Script:

1. Transfer vdw_positive.bin to iPhone (AirDrop or file share).
2. Open wallet app → "My Proofs" or "Transaction History" → select the tx → "View Proof".
3. App auto-loads cached VDW (or import manually if needed).
4. Tap "Verify Offline".
5. Use Xcode Instruments (connect device via USB): Start "Time Profiler" trace before tap, stop after result.
6. Note verification time (app displays ms).
7. Repeat 10 times, record average/p95.
8. Check result screen: Green "Confirmed Forever" badge, details (height, root hash).
9. Export verification log if app has debug mode.

Expected Results:

- Verification time: average <70 ms, p95 <80 ms.
- Result: VALID, with all four steps passed (TEE attestation → Dilithium → Halo2 delta → root reconciliation).
- Works fully offline (no network error).

Pass/Fail: Pass if <80 ms and VALID.

Troubleshooting: If slow, check background processes; restart device.

Test Case ID: TC-BV-02-02

Description: Cross-platform verification benchmark.

Detailed Test Script:

1. Use same VDW blob on: Android Pixel 9 (Kotlin app + JNI), Web (Chrome on laptop, Wasm verifier), Desktop (Rust CLI).
2. For each: Run verifier 20 times, record timings with platform tools (Android Profiler, Chrome Performance tab, CLI --benchmark).
3. Compile results in spreadsheet.

Expected Results: All platforms <80 ms (web/desktop faster, ~50 ms).

Pass/Fail: Pass if all meet threshold.

Test Case ID: TC-BV-02-03

Description: Negative – Invalid VDW detection.

Detailed Test Script:

1. Manually tamper VDW blob (flip one bit in proof or sig using hex editor).
2. Import tampered file to wallet on iPhone.
3. Attempt offline verify.

Expected Results: Red error "INVALID: Halo2 proof failed" or "Signature invalid". Detailed error per step.

Pass/Fail: Pass if detects all tamper types (test 5 variants: proof, sig, attestation, root).

Test Case ID: TC-BV-02-04

Description: TEE-accelerated path (if device supports).

Detailed Test Script:

1. On device with Secure Enclave (iPhone), enable "Use hardware acceleration" in wallet settings.
2. Verify same VDW.
3. Compare time vs software path.

Expected Results: Faster (~30-50% reduction), attestation shows hardware use. Graceful fallback if disabled.

Pass/Fail: Pass if faster and correct.

Reference User Story: BV-03

As a node, I serve VDWs over HTTP/3 with range-proof caching

Acceptance Criteria: CDN-compatible, 5-year archival guarantee.

Test Case ID: TC-BV-03-01

Description: Public serving & partial download.

Prerequisites: Full node running with public endpoint exposed. Known tx_hash with VDW.

Detailed Test Script:

1. Query full VDW: curl -v --http3 https://node-testnet.nerv.net/vdw/0xabc... > vdw_full.bin.
2. Check headers: Cache-Control, ETag present.
3. Partial request: curl -v --http3 -H "Range: bytes=0-500" https://node-testnet.nerv.net/vdw/0xabc... > partial.bin.
4. Verify partial matches first 501 bytes of full.
5. Wait 35s, query Arweave: use arweave.net/tx/<predicted_txid> or search by metadata.

6. Query IPFS gateway: ipfs.io/ipfs/<cid>.

Expected Results: Full download succeeds, partial supported (206 Partial Content), pinned on Arweave/IPFS within 30-60s.

Pass/Fail: Pass if all access methods work.

Test Case ID: TC-BV-03-02

Description: Historical bucket after simulated aging.

Detailed Test Script:

1. Use old tx (from testnet archive, >1 simulated year).
2. Query VDW – should serve with Merkle proof (<50 KB extra).

Expected Results: Valid with small extra proof.

Pass/Fail: Pass if retrievable.

Reference User Story: BV-04

As a wallet, I can request and permanently cache VDWs for my txs

Acceptance Criteria: Offline validation works forever.

Test Case ID: TC-BV-04-01

Description: Auto-fetch & cache flow.

Detailed Test Script:

1. Send tx from wallet, note tx_hash.
2. Wait for confirmation (wallet shows "Pending" → "Confirmed").
3. Background fetch: check wallet logs/debug for "VDW fetched".
4. Go offline (airplane mode).
5. View tx history → proof should load instantly from cache.
6. Export proof: QR, file, base64 – scan/open on another device to verify.

Expected Results: VDW cached locally (encrypted), permanent, exportable.

Pass/Fail: Pass if offline proof works.

Test Case ID: TC-BV-04-02

Description: Manual fetch & long-term cache.

Detailed Test Script:

1. Delete cache (wallet setting).
2. Online: manual "Refresh Proofs".
3. Verify fetch time <5s.
4. Offline test as above.

Expected Results: Re-fetches successfully, caches permanently.

Pass/Fail: Pass.

Reference User Story: BV-05

As the protocol, I guarantee VDW non-repudiability even if shard reorgs

Acceptance Criteria: Reorgs invalidate old VDWs and issue new ones automatically.

Test Case ID: TC-BV-05-01

Description: Reorg handling – Shallow reorg (<10 confirmations).

Prerequisites: Testnet with reorg tool (force fork on cluster).

Detailed Test Script:

1. Send tx, get original VDW (cache it).
2. Trigger shallow reorg (depth 5-8) including the tx.
3. Wallet detects (push or poll).
4. New VDW auto-fetched, old marked "Revoked".
5. Verify both: old returns REORG_INVALID, new VALID against new root.

Expected Results: Automatic replacement, revocation proof via Merkle tree.

Pass/Fail: Pass if seamless update.

Test Case ID: TC-BV-05-02

Description: Deep reorg safety (rare case).

Detailed Test Script:

1. Simulate deep reorg (>10 confirmations, probabilistic negligible).
2. Check wallet alerts/push "Proof updated due to reorg".

Expected Results: Replacement issued, no data loss.

Pass/Fail: Pass.

Test Cases for Epic 3 – Neural State Embeddings & Homomorphic Updates

Goal Recap: Replace Merkle trees with 512-byte AI-generated latent vectors that are cryptographically verifiable, homomorphically updatable, recursively provable, and irreversibly private.

Important Notes for Testers:

- This is the most technically complex epic. Tests require ZK/ML knowledge and specialised tools.
- Many tests need a running full node with TEE + Halo2 prover enabled, plus access to the LatentLedger circuit verifier.
- Tools required:
    - latentledger-inspect CLI (parse embeddings/deltas)
    - halo2-verifier binary (mobile/desktop/Wasm)
    - embedding-reconstruct debug tool (decoder for round-trip tests)
    - Model viewer (PyTorch or ONNX for initial model)
    - Constraint counter tool (halo2-constraint-count)
    - TEE logs + attestation verifier
- All timings on reference hardware: Apple M2 Ultra desktop, iPhone 15 mobile, NVIDIA H100 for proving (if available).
- Record everything: constraint counts, proof sizes, reconstruction errors (must be exactly 0), timings.

Reference User Story: EMB-01

Define and implement the base "LatentLedger" circuit (Halo2) that maps arbitrary state → 512-byte embedding

Acceptance Criteria: Round-trip reconstruction error = 0 for balances up to 2^{256}.

Test Case ID: TC-EMB-01-01

Description: Positive – Full encoder/decoder round-trip on small state (100 accounts).

Prerequisites: Full node running with LatentLedger circuit loaded (trusted setup parameters pinned). Generate synthetic state with 100 accounts (balances 1–1000 test NERV).

Detailed Test Script:

1. On node, trigger manual re-embedding: RPC ./nerv-rpc reembed_shard --shard=0 --force.
2. Node logs "Running encoder in TEE" → "New embedding root: 0xabc...". Capture full 512-byte embedding (log or RPC get_current_embedding). Save to embedding_small.bin.
3. Export private state trie (debug RPC export_state_shard 0 > state_small.json).
4. Run decoder locally: ./embedding-reconstruct --embedding embedding_small.bin --decoder-params params.bin --output reconstructed.json.

5. Compare files: diff state_small.json reconstructed.json or script hash comparison.
6. Check reconstruction error log: must show "Max error: 0.000000000" (exact integer match).
7. Verify embedding size: wc -c embedding_small.bin → exactly 512 bytes.
8. Screenshot node logs, diff output, error = 0.
9. Repeat with balances up to 2^{128} (use big-int script to generate).

Expected Results:

- Exact match (diff empty).
- Error exactly 0 (no floating-point drift).
- Embedding fixed 512 bytes.

Pass/Fail: Pass if round-trip perfect.

Troubleshooting: If error >0, check quantization or fixed-point format.

Test Case ID: TC-EMB-01-02

Description: Large state round-trip (10,000 accounts).

Detailed Test Script:

1. Generate large synthetic state (script provided).
2. Force re-embedding.
3. Repeat steps 2–8 from TC-EMB-01-01.
4. Measure encoder time in TEE logs (<5s target).

Expected Results: Still exact reconstruction, time reasonable.

Pass/Fail: Pass.

Test Case ID: TC-EMB-01-03

Description: Constraint count verification.

Detailed Test Script:

1. Run ./halo2-constraint-count --circuit latentledger_encoder.
2. Record total constraints.

Expected Results: ≤8.2M constraints (current target ≤7.914M).

Pass/Fail: Pass if within budget.

Reference User Story: EMB-02

Implement homomorphic addition/subtraction on embeddings

Acceptance Criteria: embed(A + Δ) = embed(A) + homomorphic_delta(Δ) with |error| ≤10^{-9}.

Test Case ID: TC-EMB-02-01

Description: Single transfer homomorphism.

Prerequisites: Current embedding from live shard.

Detailed Test Script:

1. From wallet, prepare private transfer (0.1 test NERV). Do NOT submit yet.
2. Export client-side delta: Wallet debug → "Export Delta Vector" → delta_single.bin (512 bytes).
3. Manually apply: ./embedding-math add current_embedding.bin delta_single.bin new_expected.bin.
4. Submit tx → wait finality → get new canonical embedding embedding_after.bin.
5. Compare: diff new_expected.bin embedding_after.bin or numerical ||diff|| <10^{-9}.
6. Run Halo2 HomomorphicUpdate verifier: ./halo2-verifier --circuit homomorphic_update --public-inputs current+delta --proof node_provided_proof.
7. Record error bound from verifier log.
8. Screenshot comparisons and verifier output.

Expected Results:

- Embeddings match within 10^{-9} (practically identical in fixed-point).
- Proof verifies successfully (<25 ms).

Pass/Fail: Pass if error ≤10^{-9} and proof valid.

Test Case ID: TC-EMB-02-02

Description: Batched homomorphism (256 transfers).

Detailed Test Script:

1. Script 256 simultaneous transfers (different amounts/accounts).
2. Wallet/SDK aggregates into single batched delta.
3. Export aggregated delta_batch.bin.
4. Apply manually to current embedding.
5. Submit batch → finality → compare new embedding.

Expected Results: Still ≤10^{-9} error (≤256×10^{-9} total bound). Single proof for batch.

Pass/Fail: Pass.

Test Case ID: TC-EMB-02-03

Description: Negative – Invalid delta rejected.

Detailed Test Script:

1. Tamper delta vector (flip bit).
2. Attempt to use in debug submission.
3. Node rejects with Halo2 proof failure.

Expected Results: Rejection before consensus.

Pass/Fail: Pass.

Reference User Story: EMB-03

Train initial 24-layer transformer compressor on synthetic tx dataset

Acceptance Criteria: Compression ≥800×, model ≤8 MB.

Test Case ID: TC-EMB-03-01

Description: Model size and compression ratio.

Detailed Test Script:

1. Download initial model checkpoint (from repo or on-chain IPFS).
2. Measure: ls -lh initial_encoder.onnx or .pt.
3. Run compression benchmark: ./benchmark-compression --state-size 100MB --embedding embedding.bin.
4. Record ratio (state bytes / 512).

Expected Results: Model ≤8.2 MB (quantised), ratio ≥800× (ideally 900×+).

Pass/Fail: Pass.

Test Case ID: TC-EMB-03-02

Description: Circuit transcription fidelity.

Detailed Test Script:

1. Run inference with PyTorch model on test state.
2. Run same in Halo2 simulated mode.
3. Compare outputs element-wise.

Expected Results: No accuracy loss vs floating-point.

Pass/Fail: Pass.

Reference User Story: EMB-04

Implement embedding inclusion proof (tx → embedding path)

Acceptance Criteria: Proof size ≤800 bytes.

Test Case ID: TC-EMB-04-01

Description: Inclusion proof generation and verification.

Detailed Test Script:

1. Send private tx → finality.
2. Query inclusion proof via RPC or from VDW (contains delta path).
3. Save proof inclusion_proof.bin.
4. Measure size: wc -c.
5. Verify offline: ./halo2-verifier --circuit embedding_inclusion --proof inclusion_proof.bin --public tx_hash+old_root+new_root.
6. Time verification on iPhone 15 (<60 ms).

Expected Results: Proof ≤800 bytes, verifies successfully.

Pass/Fail: Pass.

Test Case ID: TC-EMB-04-02

Description: Aggregated proof for 256 txs.

Detailed Test Script:

1. Batch scenario → single aggregated inclusion proof.
2. Verify covers all txs (check public inputs).

Expected Results: One small proof for batch.

Pass/Fail: Pass.

Reference User Story: EMB-05

Nodes periodically re-embed entire shard state inside TEE and publish new root

Acceptance Criteria: Every 1000 txs or 10s.

Test Case ID: TC-EMB-05-01

Description: Periodic re-embedding trigger.

Detailed Test Script:

1. Monitor node logs during traffic.
2. Send 1001 txs rapidly.
3. Check log for "Re-embedding triggered" at ~1000.
4. Verify new root committed in consensus (neural votes agree).
5. Check TEE attestation on re-embedding event.

Expected Results: Triggers correctly, new root with valid attestation.

Pass/Fail: Pass.

Test Case ID: TC-EMB-05-02

Description: Emergency slow-path fallback.

Detailed Test Script:

1. Simulate model divergence (debug inject bad encoder).
2. Node detects $>10^{-6}$ error.
3. Logs "Falling back to Merkle root for epoch".

Expected Results: Automatic fallback, alert raised.

Pass/Fail: Pass.

Test Cases for Epic 4 – AI-Native Optimistic Consensus

Goal Recap: Achieve sub-second finality via neural voting + cryptographic fallback, with reputation weighting and Monte-Carlo disputes.

Important Notes for Testers:

- This epic depends on Epics 1–3 (onion ingress, embeddings, homomorphic updates).
- Tests require a multi-node testnet (≥500 nodes recommended for realistic latency/finality).
- Tools needed:
    - Node CLI/RPC (e.g., nerv-rpc, curl JSON-RPC)
    - Neural vote inspector (./neural-vote-inspect)
    - Distilled predictor model verifier (./predictor-verify)
    - BLS verifier (bls-verify)
    - Monte-Carlo simulator (debug tool)
    - Latency measurement script (multi-ping)
    - Chaos injection tools (tc netem, node killer scripts)
- Reference hardware: Validators on AWS i4i/m6i, geographic diversity (5+ continents).

- Record: Finality timings (probabilistic/crypto), vote agreement %, challenge rate, slash events.

Reference User Story: CON-01

Nodes broadcast predicted post-tx embedding + BLS partial signature

Acceptance Criteria: Prediction with distilled model, message ≤256 bytes, TEE attestation.

Test Case ID: TC-CON-01-01

Description: Positive – Neural vote broadcast after batch.

Prerequisites: Run 3+ validator nodes locally or on testnet, with distilled model loaded. Force a batch of 10 txs.

Detailed Test Script:

1. Start packet capture on validator: sudo tcpdump -i any port 30333 -w votes_capture.pcap (P2P gossip port).
2. Send 10 private txs rapidly to trigger batch execution.
3. Monitor node logs: tail -f validator1.log | grep "Predicting next embedding".
4. Look for: "Running distilled model in TEE", "Predicted hash: 0xabc...", "Broadcasting neural vote".
5. Extract vote message from pcap or gossip log: Save as neural_vote.bin.
6. Inspect: ./neural-vote-inspect neural_vote.bin --verbose.
7. Check fields: node_id, predicted_hash (32 bytes), partial_BLS_sig (48 bytes), reputation_score, TEE attestation report.
8. Measure size: wc -c neural_vote.bin → ≤256 bytes.
9. Verify attestation: ./attestation-verify --report vote_attestation.bin --expected-model-hash known_distilled.txt.
10. Verify partial BLS: ./bls-verify --partial --pubkey validator_pubkey --msg predicted_hash --sig partial_sig.
11. Screenshot logs, inspect output, size. Repeat on 3 nodes.

Expected Results:

- Message ≤256 bytes.
- Valid fresh attestation proving correct model hash.
- Partial BLS valid.
- Prediction runs <8 ms (log timestamp).

Pass/Fail: Pass if all fields valid and size met.

Troubleshooting: If no vote, check batch finality; if wrong model, reload weights.

Test Case ID: TC-CON-01-02

Description: Prediction accuracy across nodes.

Detailed Test Script:

1. After same batch, collect predicted hashes from 10+ nodes (RPC get_latest_votes).
2. Count unique hashes and agreement %.

Expected Results: ≥99% nodes predict identical hash.

Pass/Fail: Pass.

Reference User Story: CON-02

Implement 67% embedding hash quorum → instant probabilistic finality

Acceptance Criteria: Weighted (stake × reputation) ≥67% → finality in ~600 ms.

Test Case ID: TC-CON-02-01

Description: Normal fast-path finality.

Prerequisites: Testnet with ≥200 nodes, varied stake/reputation.

Detailed Test Script:

1. Use timing script: ./measure-finality --tx-count 100 --interval 1s.
2. Send burst of txs.
3. Script records: tx submit time → probabilistic finality event time.
4. Check wallet/node logs for "Probabilistic finality at height X".
5. Query RPC for quorum details: agreement %, weighted sum.
6. Repeat 100 txs, compute median/p95 latency.
7. Screenshot Grafana/finaltiy dashboard if available.

Expected Results:

- Median ≤600 ms probabilistic finality.
- ≥67% weighted agreement logged.
- Full BLS threshold sig aggregated.

Pass/Fail: Pass if median ≤600 ms and quorum met.

Test Case ID: TC-CON-02-02

Description: Gossip acceleration bandwidth savings.

Detailed Test Script:

1. Capture gossip traffic during burst (tcpdump).
2. Compare fast-path (only votes) vs lazy full tx fetch.

Expected Results: Bandwidth reduction ≈40× during normal operation.

Pass/Fail: Pass.

Reference User Story: CON-03

Implement challenge phase with Monte-Carlo dispute resolution in TEE

Acceptance Criteria: Resolves <650 ms, slash losing side.

Test Case ID: TC-CON-03-01

Description: Trigger and resolve challenge.

Prerequisites: Debug mode to force disagreement (inject wrong prediction on 20% nodes).

Detailed Test Script:

1. Configure 20% nodes with bad distilled model (debug flag).
2. Send batch → majority predicts correct, minority wrong.
3. Monitor logs for "Challenge opened" within 800 ms.
4. Select challenger node → bond posted.
5. Watch "Monte-Carlo dispute: selecting 32 TEEs", "Running 10,000 simulations".
6. Time from challenge to resolution (log timestamps).
7. Check outcome: Majority root wins, minority slashed (0.5–5% stake).
8. Verify slash evidence on-chain.
9. Screenshot challenge flow logs, timings.

Expected Results:

- Challenge opened ≤800 ms.
- Resolution ≤650 ms.
- Correct root finalised, slash applied.

Pass/Fail: Pass if resolved correctly and fast.

Test Case ID: TC-CON-03-02

Description: No unnecessary challenges (honest network).

Detailed Test Script:

1. Run 10,000 txs on honest testnet.
2. Count challenge events.

Expected Results: <0.01% blocks challenged.

Pass/Fail: Pass.

Reference User Story: CON-04

Implement reputation score updated via federated learning gradients

Acceptance Criteria: Multiplicative weight (stake × reputation).

Test Case ID: TC-CON-04-01

Description: Reputation update and weighting.

Detailed Test Script:

1. Monitor validator reputation via RPC get_validator_status validator_id.
2. Submit high-quality gradients (honest node).
3. After epoch, check reputation increase.
4. Simulate low contribution → reputation decay.
5. Check voting weight = stake × reputation in vote messages.
6. Force reputation <0.1 → weight 0, node ignored.

Expected Results: Reputation updates every ~10 min, multiplicative weighting applied.

Pass/Fail: Pass.

Test Case ID: TC-CON-04-02

Description: Equivocation slashing.

Detailed Test Script:

1. Force node to double-vote (debug).
2. Detection → immediate 50% reputation burn + jail.

Expected Results: Slash executed.

Pass/Fail: Pass.

Reference User Story: CON-05

Achieve median 600 ms probabilistic finality, 1.8 s cryptographic

Acceptance Criteria: Measured on 500-node global testnet.

Test Case ID: TC-CON-05-01

Description: Global finality benchmark.

Prerequisites: 500-node testnet (5 continents).

Detailed Test Script:

1. Deploy chaos (5–10% packet loss, 100–300 ms latency).
2. Run ./stress-finality --duration 24h --tps 5000.
3. Collect probabilistic and crypto finality timings.
4. Generate percentiles (median, p95, p99).
5. Check correctness under 33% byzantine (inject faulty nodes).

Expected Results:

- Probabilistic median ≤600 ms.
- Crypto ≤1.8 s.
- 99.999% correct under faults.

Pass/Fail: Pass if KPIs met.

Test Case ID: TC-CON-05-02

Description: Attack simulation suite.

Detailed Test Script:

1. Run 10,000 scripted attacks (eclipse, partition, byzantine votes).
2. Measure liveness/safety.

Expected Results: System remains live and safe.

Pass/Fail: Pass.

Test Cases for Epic 5 – Dynamic Neural Sharding

Goal Recap: Auto-scale to 1000+ shards with zero manual config. Shards split/merge like cells based on AI-predicted load, with embedding bisection and erasure-coded replication.

Important Notes for Testers:

- This epic builds on Epics 1–4 (ingress, VDWs, embeddings, consensus).
- Tests require a multi-shard testnet (start with 100–200 shards, ≥500 nodes global).
- Tools needed:
    - Shard dashboard (real-time visualisation: shard count, TPS per shard, splits/merges)
    - LSTM predictor verifier (./lstm-predictor-verify)
    - Embedding bisection tool (./embedding-bisect)

    - Reed-Solomon codec tester (./rs-test)
    - Chaos tools (node killer, partition injector, load generator)
    - RPC commands: get_shard_metrics, get_shard_tree, force_split_proposal (debug)
- Reference: Genetic placement optimiser logs, repair timings.
- Record: Split/merge times, prediction accuracy, recovery from 40% loss, cross-shard latency.

Reference User Story: SHD-01

Implement shard load predictor (LSTM on tx rate, size, gas)

Acceptance Criteria: Predicts overload 15 s ahead with >95% accuracy.

Test Case ID: TC-SHD-01-01

Description: Positive – Accurate overload prediction under ramping load.

Prerequisites: Testnet with 1 active shard, load generator script. LSTM model loaded in all nodes.

Detailed Test Script:

1. Start observability: Open shard dashboard → select shard 0 → monitor "Overload Probability" graph.
2. Run load generator: ./load-gen --shard 0 --ramp-start 500tps --ramp-end 2000tps --duration 300s.
3. Simultaneously monitor node logs on 5 validators: tail -f node.log | grep "LSTM prediction".
4. Record every 5s: current TPS (from dashboard), predicted overload prob for +15s.
5. At exact time when actual TPS crosses overload threshold (e.g., 1200 TPS sustained), note if prediction >0.92 at t-15s.
6. Collect 50 such overload events (repeat ramp 10 times).
7. Export logs → spreadsheet: columns (timestamp, actual TPS, predicted prob at t-15s, correct prediction Y/N).
8. Calculate accuracy: % of events where prob >0.92 exactly 15s (±2s) before overload.
9. Screenshot dashboard during one ramp, log snippet of prediction.

Expected Results:

- Accuracy ≥95% (ideally >98%).
- Inference time <4 ms per prediction (log timestamp diff).
- Model size <1.2 MB (verify ls -lh lstm_model.bin).

Pass/Fail: Pass if accuracy ≥95%.

Troubleshooting: If low accuracy, retrain LSTM via federated update (Epic 6) or check metric collection.

Test Case ID: TC-SHD-01-02

Description: False positive rate under stable load.

Detailed Test Script:

1. Run constant 800 TPS for 30 minutes.
2. Count predictions >0.92.

Expected Results: <1% false positives.

Pass/Fail: Pass.

Reference User Story: SHD-02

Implement live shard split protocol (state embedding bisect)

Acceptance Criteria: Split completes in <4 s, no downtime.

Test Case ID: TC-SHD-02-01

Description: Positive – Trigger and complete automatic split.

Prerequisites: Single overloaded shard, ≥200 nodes ready to adopt children.

Detailed Test Script:

1. Induce overload (load gen >1200 TPS sustained).
2. Watch dashboard for "SplitProposal broadcast".
3. Monitor logs on multiple nodes: "Co-signing SplitProposal", "67% quorum reached".
4. Time from proposal to "Child shards active" (dashboard shows 2 new shards).
5. Check new shard_ids, parent relationship in shard tree view.
6. Verify no tx loss: Compare pre-split mempool + in-flight txs vs post-split in children.
7. Run ./embedding-bisect --parent embedding_parent.bin --seed correct_seed --output childA childB.
8. Compare generated children vs canonical new embeddings (RPC).
9. Verify Halo2 proof of bisection preservation (error ≤10^{-9}).
10. Screenshot timeline: proposal → quorum → children active. Record exact split time.
11. Repeat 20 splits.

Expected Results:

- Split time ≤4 s (median ≤3.4 s).
- Embeddings match deterministic bisection.

- Zero downtime/tx loss.
- DHT/routing updates instant (<800 ms global gossip).

Pass/Fail: Pass if all splits <4 s and correct.

Test Case ID: TC-SHD-02-02

Description: Rollback on failed quorum.

Detailed Test Script:

1. Simulate 40% nodes reject proposal (debug).
2. Check logs "Quorum failed → revert to parent embedding".

Expected Results: Clean revert, no fork.

Pass/Fail: Pass.

Reference User Story: SHD-03

Implement shard merge when underutilized

Acceptance Criteria: Automatic merge <6 s when siblings <10 TPS for 10 min.

Test Case ID: TC-SHD-03-01

Description: Automatic merge of idle siblings.

Detailed Test Script:

1. Force split → get two child shards.
2. Reduce load to <10 TPS total for 12 minutes.
3. Monitor dashboard for "Merge proposal automatic".
4. Time from trigger to "Parent shard restored".
5. Verify reverse bisection produces identical parent embedding.
6. Check retired child shard_ids purged.

Expected Results: Merge completes <6 s, state lossless.

Pass/Fail: Pass.

Reference User Story: SHD-04

Implement AI-driven erasure coding placement (5–7 replicas)

Acceptance Criteria: Survives 40% node loss, repair <8 s.

Test Case ID: TC-SHD-04-01

Description: Fault tolerance – 40% node loss recovery.

Detailed Test Script:

1. Baseline: Record all shard embeddings replicated (7 replicas via dashboard).
2. Script mass kill: Randomly terminate 40% of nodes instantly.
3. Monitor "Missing chunk detected" → "Repair initiated".
4. Time per shard repair completion.
5. Verify reconstructed embeddings match originals (hash comparison).
6. Check no downtime (finality continues).
7. Screenshot before/after replica map, repair logs.

Expected Results:

- Full recovery <30 s global.
- Per-shard repair <8 s.
- Reed-Solomon reconstruction correct.

Pass/Fail: Pass.

Test Case ID: TC-SHD-04-02

Description: Geo-aware placement constraints.

Detailed Test Script:

1. Check validator metadata (regions).
2. Verify no shard has >2 replicas in same AWS region.
3. Force violation (debug) → optimiser corrects.

Expected Results: Constraints enforced.

Pass/Fail: Pass.

Test Case ID: TC-SHD-04-03

Description: Genetic optimiser latency improvement.

Detailed Test Script:

1. Baseline cross-shard latency (ping tool).
2. Run optimiser 30 min.
3. Measure improvement.

Expected Results: Cross-shard p95 ≤350 ms (from ~800 ms).

Pass/Fail: Pass.

Reference User Story: SHD-05 (Cross-cutting Performance)

Observed Performance: 1.1M+ TPS sustained, dynamic shard count

Test Case ID: TC-SHD-05-01

Description: End-to-end dynamic scaling under real traffic.

Detailed Test Script:

1. Start with 100 shards, mixed load (payments, DeFi-like contracts).
2. Ramp global TPS from 200k → 1.5M over 30 min.
3. Watch dashboard: auto-splits → peak shard count.
4. Then reduce load → observe merges.
5. Record timeline video/screenshots: shard count graph, TPS graph, split/merge events.
6. Verify no sustained TPS drop during transitions.

Expected Results:

- Scales to >1M TPS sustained.
- Shard count dynamically 300 → 1200 → back.
- All splits/merges correct.

Pass/Fail: Pass.

Test Case ID: TC-SHD-05-02

Description: Chaos with dynamic sharding.

Detailed Test Script:

1. Combine 40% kills + load ramps + network partitions.
2. Run 2 hours continuous.

Expected Results: System self-heals, maintains TPS.

Pass/Fail: Pass.

Test Cases for Epic 6 – Useful-Work Economy & Federated Learning

Goal Recap: Nodes earn by improving the collective AI (encoder/decoder models) via federated learning. Rewards proportional to Shapley-value contribution, with differential privacy and secure aggregation in TEEs.

Important Notes for Testers:

- This epic integrates with Epics 3–5 (embeddings, consensus, sharding).

- Tests require a testnet with ≥1000 nodes capable of gradient submission (simulate if needed).
- Tools needed:
    - Gradient inspector (./gradient-inspect)
    - Secure aggregation verifier (debug TEE tool)
    - Shapley approximation calculator (./shapley-approx)
    - DP verifier (Opacus-style auditor)
    - On-chain reward checker (RPC get_gradient_rewards)
    - Model improvement tracker (loss/homomorphism error dashboard)
- Privacy: All gradients encrypted; no individual leakage.
- Record: Aggregation time, Shapley shares, reward distribution, privacy budget (ε), model improvement per epoch.

Reference User Story: UW-01

Nodes submit encrypted gradients after every 1000 txs

Acceptance Criteria: DP-SGD, secure aggregation, dropout tolerance.

Test Case ID: TC-UW-01-01

Description: Positive – Gradient generation and submission.

Prerequisites: Run validator node with local training enabled. Generate 1001 txs in shard.

Detailed Test Script:

1. Monitor node logs: tail -f node.log | grep "Gradient hook".
2. After ~1000 txs, look for "Running local DP-SGD step", "Clipping norms", "Adding Gaussian noise σ=0.5".
3. Capture generated gradient: Log shows "Encrypted gradient size: ~XXX KB".
4. Export gradient blob (debug RPC export_last_gradient > grad_node1.bin).
5. Measure size: ls -lh grad_node1.bin → ≤256 KB.
6. Inspect: ./gradient-inspect grad_node1.bin --metadata → check model version, node_id, quantised int8, encryption (Dilithium/ML-KEM).
7. Verify DP noise: Run auditor ./dp-auditor --gradient grad_node1.bin --sigma 0.5 → confirms clipping and noise.
8. Check submission: Log "Submitted encrypted gradient to aggregator".
9. Repeat on 5 nodes, confirm all submit within 15s window.
10. Screenshot logs, inspect output.

Expected Results:

- Gradient ≤240 KB average.
- DP parameters correct (σ=0.5).
- Encrypted, submitted successfully.

Pass/Fail: Pass if all nodes submit valid DP gradients.

Troubleshooting: If no hook, check tx count; if too large, check quantisation.

Test Case ID: TC-UW-01-02

Description: Dropout tolerance (70% participation).

Detailed Test Script:

1. Simulate 30% nodes offline/no gradient.
2. Check aggregation succeeds with remaining 70%.

Expected Results: Aggregation completes, late gradients ignored.

Pass/Fail: Pass.

Reference User Story: UW-02

Parameter server (run in TEE cluster) aggregates and publishes new model version

Acceptance Criteria: Every 10 minutes, improvement test, rollback if needed.

Test Case ID: TC-UW-02-01

Description: Secure aggregation and model update.

Prerequisites: TEE cluster running (16–32 instances). ≥500 gradients submitted.

Detailed Test Script:

1. Monitor parameter server logs (one instance): tail -f pserver.log | grep "Aggregation round".
2. Time from first gradient to "Aggregation complete".
3. Check "Decrypt aggregate in TEE", "Average weights".
4. Run hold-out test: Log "Validation loss improvement: X.XXX".
5. If >0.003, "Publishing new model version Y".
6. Verify new model on-chain: RPC get_current_model_cid → new IPFS hash.
7. Download and compare size (<8 MB).
8. Verify attestation on aggregation result.
9. Screenshot timeline logs, improvement value.

Expected Results:

- Aggregation <8 s for 10k nodes.
- New model published every ~10 min if improvement.

Pass/Fail: Pass.

Test Case ID: TC-UW-02-02

Description: Automatic rollback on bad update.

Detailed Test Script:

1. Poison 10% gradients (debug).
2. Aggregation produces worse loss.
3. Log "Validation failed → rollback to previous".

Expected Results: Reverts within 2 min.

Pass/Fail: Pass.

Reference User Story: UW-03

Implement on-chain model registry with staking governance

Acceptance Criteria: Immutable versions, 7-day delay.

Test Case ID: TC-UW-03-01

Description: Model registry update with Halo2 proof.

Detailed Test Script:

1. Propose new model (good improvement).
2. Check on-chain proposal includes Halo2 proof of homomorphism preservation.
3. Verify proof: ./halo2-verifier --circuit model_upgrade --proof proposal_proof.bin.
4. Governance vote simulation → after delay, activated.

Expected Results: Proof valid, delay enforced.

Pass/Fail: Pass.

Reference User Story: UW-04

Pay gradient contributors proportional to model improvement (Shapley-value approximation)

Acceptance Criteria: On-chain micro-payments, anti-gaming.

Test Case ID: TC-UW-04-01

Description: Shapley calculation and rewards.

Prerequisites: Successful aggregation round with 1000+ gradients.

Detailed Test Script:

1. After model update, query RPC get_shapley_contributions --round X.
2. Export table: node_id, shapley_value, reward_NERV.
3. Sum rewards ≈ emission slice for round.
4. Check top contributors have higher shapley (correlate with honest gradients).
5. Verify on-chain mint/transfer to nodes.
6. Screenshot contribution leaderboard.

Expected Results: Rewards proportional, total matches emission.

Pass/Fail: Pass.

Test Case ID: TC-UW-04-02

Description: Anti-gaming – Clone detection and slash.

Detailed Test Script:

1. Submit 100 identical gradients (Sybil debug).
2. Check detection "Similarity clustering → slash clones".
3. Slashed nodes lose rewards.

Expected Results: >99% detection, slash applied.

Pass/Fail: Pass.

Test Case ID: TC-UW-04-03

Description: Long-term economic simulation.

Detailed Test Script:

1. Run provided simulator ./economic-sim --nodes 100k --years 5.
2. Check inflation <4%/year, useful-work dominates.

Expected Results: Matches tokenomics model.

Pass/Fail: Pass.

Cross-Cutting Tests for Epic 6

Test Case ID: TC-UW-06-01

Description: End-to-end federated improvement over 90 days (simulated).

Detailed Test Script:

1. Run accelerated testnet (fast epochs).
2. Track weekly: homomorphism error reduction, compression ratio.
3. Verify measurable improvement.

Expected Results: Steady AI enhancement.

Pass/Fail: Pass.

Test Case ID: TC-UW-06-02

Description: Privacy proof.

Detailed Test Script:

1. Run formal auditor on pipeline.
2. Confirm (ε≈1.2, δ=1e-8) per 30 days.

Expected Results: Bounds met.

Pass/Fail: Pass.

Test Cases for Epic 7 – Quantum-Resistant Cryptography Suite

Goal Recap: Day-1 post-quantum security – zero legacy elliptic curves in critical paths. All signatures Dilithium-3, KEM ML-KEM-768, agility framework.

Important Notes for Testers:

- This epic is mostly independent but integrates everywhere (signatures in blocks, votes, attestations, onions, VDWs; KEM in onion layers, QUIC, TEE channels).
- Tools needed:
    - dilithium-verify, ml-kem-encaps/decaps, sphincs-verify CLIs
    - liboqs tester (./oqs-test all)
    - Key migration tool (./pq-migrate-wallet)
    - Constant-time analyser (dudect, ctgrind, FlowTracker)
    - Benchmark suite (./pq-bench)
    - Governance proposal simulator
- Reference: NIST KAT vectors, Wycheproof test suite.
- Record: Key/sig sizes, verify timings, handshake latency impact, migration success.

Reference User Story: QR-01

Replace all ECDSA/EdDSA with CRYSTALS-Dilithium-3

Acceptance Criteria: Full drop-in, optimised, constant-time.

Test Case ID: TC-QR-01-01

Description: Positive – Dilithium signature in critical paths (block header, neural vote, VDW).

Prerequisites: Node running with pq_crypto=true. Generate a block/finalise batch.

Detailed Test Script:

1. Monitor node logs during normal operation: tail -f node.log | grep "Signing with Dilithium".
2. Capture a block header (RPC get_block height).
3. Extract signature field → save block_sig.bin.
4. Extract public key from validator set.
5. Verify: ./dilithium-verify --level 3 --pubkey val_pubkey.bin --msg block_header_no_sig --sig block_sig.bin.
6. Measure size: wc -c block_sig.bin → ~3297 bytes.
7. Repeat for neural vote message and VDW signature.
8. Check no secp256k1/ed25519 mentions: grep -r "secp256k1\|ed25519" src/ | wc -l → 0 in critical paths.
9. Screenshot verify output, sizes.

Expected Results:

- All verifications VALID.
- Sig size ~3.3 KB, pk ~1.8 KB.

Pass/Fail: Pass if all critical signatures use Dilithium and verify.

Test Case ID: TC-QR-01-02

Description: Performance benchmarks.

Detailed Test Script:

1. Run ./pq-bench --primitive dilithium3 --iterations 10000.
2. Record verify time on Ice Lake CPU and Apple M2.
3. Repeat sign time.

Expected Results:

- Verify <60 µs Ice Lake, <90 µs M2.

Pass/Fail: Pass if thresholds met.

Test Case ID: TC-QR-01-03

Description: Constant-time hardening.

Detailed Test Script:

1. Run dudect: ./dudect dilithium-sign.
2. Run ctgrind/valgrind on test binary.
3. Check FlowTracker report.

Expected Results: No timing leakage (t-test p>0.01).

Pass/Fail: Pass.

Reference User Story: QR-02

Implement ML-KEM-768 for all enclave-to-enclave and node-to-node key exchange

Acceptance Criteria: Hybrid with X25519 optional.

Test Case ID: TC-QR-02-01

Description: Positive – ML-KEM in onion routing and QUIC.

Prerequisites: Capture traffic during tx submission.

Detailed Test Script:

1. Send private tx → capture onion layers (wallet export or relay debug).
2. Inspect outer/inner layers: ./ml-kem-inspect layer1.bin → ciphertext ~1088 bytes.
3. Verify encapsulation uses ML-KEM-768 pk.
4. Check QUIC initial packet (tcpdump): ./quic-inspect capture.pcap → KEM ID for ML-KEM.
5. Optional hybrid: Enable hybrid mode → check both KEMs negotiated.
6. Screenshot inspect output, ciphertext sizes.

Expected Results:

- All hop encryptions ML-KEM-768.
- Ciphertext ~1088 bytes.

Pass/Fail: Pass.

Test Case ID: TC-QR-02-02

Description: Handshake latency impact.

Detailed Test Script:

1. Benchmark classical X25519 handshake vs PQ vs hybrid (script ./handshake-bench).
2. Run over real networks (4G/5G/satellite sim).

Expected Results: <+12 ms added latency.

Pass/Fail: Pass.

Reference User Story: QR-03

Implement SPHINCS+ as stateless backup

Acceptance Criteria: For cold wallets, hedged mode optional.

Test Case ID: TC-QR-03-01

Description: SPHINCS+ cold key usage.

Detailed Test Script:

1. Generate SPHINCS+ keypair (HSM simulated).
2. Sign genesis-level tx or recovery message.
3. Verify: ./sphincs-verify --variant SHA256-192s-robust --pubkey cold_pub --msg msg --sig sig.bin.
4. Measure sig size ~41 KB, verify time <1.2 s desktop.

Expected Results: Valid, large sig as expected.

Pass/Fail: Pass.

Test Case ID: TC-QR-03-02

Description: Hedged signing toggle.

Detailed Test Script:

1. In wallet settings → "Quantum doomsday mode".
2. Sign high-value tx → produces Dilithium + SPHINCS+ co-sig.

Expected Results: Both sigs present and valid.

Pass/Fail: Pass.

Reference User Story: QR-04

Implement migration path and cryptographic agility

Acceptance Criteria: CryptoVersion enum, 180-day upgrades, legacy verify 24 months.

Test Case ID: TC-QR-04-01

Description: Agility framework and version tagging.

Detailed Test Script:

1. Inspect signed messages (block, vote): Check CryptoVersion field = 1 (Dilithium-3 + ML-KEM).
2. Simulate future upgrade proposal: Governance tool → new version 2 (e.g., Dilithium-5).
3. Check 180-day vote + 90-day migration delay enforced.

Expected Results: Version tagged, delay respected.

Pass/Fail: Pass.

Test Case ID: TC-QR-04-02

Description: Wallet key migration.

Detailed Test Script:

1. Old wallet with legacy keys (simulate).
2. Run ./pq-migrate-wallet old_wallet.json.
3. One-click batch tx upgrades all keys to PQ.
4. Verify offline after migration.

Expected Results: All keys Dilithium/ML-KEM, zero downtime.

Pass/Fail: Pass.

Test Case ID: TC-QR-04-03

Description: Legacy verification support.

Detailed Test Script:

1. Submit message signed with old ECDSA (debug).
2. Node verifies for 24 months post-launch.
3. After simulated disable → rejects legacy.

Expected Results: Graceful phase-out.

Pass/Fail: Pass.

Cross-Cutting Tests for Epic 7

Test Case ID: TC-QR-07-01

Description: Full NIST/Wycheproof vector suite.

Detailed Test Script:

1. Run ./oqs-test --kat --wycheproof.
2. CI must pass 100%.

Expected Results: All vectors pass.

Pass/Fail: Pass.

Test Case ID: TC-QR-07-02

Description: Quantum threat monitoring dashboard.

Detailed Test Script:

1. Open public dashboard.
2. Check alerts for qubit milestones, new papers.

Expected Results: Live updates.

Pass/Fail: Pass.

Test Case ID: TC-QR-07-03

Description: Emergency quantum-break playbook.

Detailed Test Script:

1. Simulate break → activate SPHINCS+ everywhere.
2. Measure activation time on staging.

Expected Results: <72 h full switch.

Pass/Fail: Pass.

Test Cases for Epic 8 – Extreme Scalability Layer

Goal Recap: Achieve 500k+ TPS per node, >1M+ network-wide via parallel shards, native account abstraction, gasless onboarding, MEV elimination, and recursive proof compression.

Important Notes for Testers:

- This epic requires a fully functional testnet with dynamic sharding (Epic 5), embeddings (Epic 3), and consensus (Epic 4).
- Tests need high-performance hardware: 64–128 core validators (AWS i4i.32xlarge, Hetzner AX162, Mac Studio M2 Ultra).
- Tools needed:
    - Load generator (./tps-blaster --parallel-shards)
    - Parallel execution monitor (dashboard: shards active, CPU per shard)

    - Paymaster simulator (./paymaster-fund)
    - Fee predictor API tester (curl fee-predictor.api)
    - Proof size measurer (./proof-stats)
    - MEV extractor simulator (./mev-sim)
- Record: TPS per node/global, CPU utilisation, cross-shard latency, proof sizes, gasless success rate.

Reference User Story: SCL-01

Parallel execution engine across 500 shards

Acceptance Criteria: Single 64-core node >180k TPS local, cross-shard ≤1.1s p99.

Test Case ID: TC-SCL-01-01

Description: Positive – Local TPS on single high-core validator.

Prerequisites: Single powerful node (64+ cores), 500 simulated shards (local mode).

Detailed Test Script:

1. Start node in benchmark mode: ./nerv-node --local-shards 500 --parallel-exec.
2. Monitor dashboard → confirm 500 shards active, memory isolation (cgroups).
3. Run local load gen: ./tps-blaster --local --target-tps 200000 --duration 300s --tx-type simple-transfer.
4. Watch real-time metrics: TPS counter, CPU % per core (htop or dashboard).
5. Check logs for "Parallel execution: X txs/sec across Y shards".
6. Record peak sustained TPS (5-min window).
7. Verify no OOM or single-shard bottleneck (one shard overload test → others continue).
8. Screenshot dashboard TPS graph, CPU balanced across NUMA nodes.
9. Repeat on reference hardware list.

Expected Results:

- Sustained ≥180k TPS local.
- CPU 85–95% balanced.
- No cross-shard blocking (all intra-shard txs).

Pass/Fail: Pass if ≥180k TPS sustained.

Test Case ID: TC-SCL-01-02

Description: Cross-shard latency under load.

Detailed Test Script:

1. Global testnet → generate 50% cross-shard txs.

2. Measure p95 finality for cross-shard vs intra.
3. Use tracer txs with timestamps.

Expected Results: Cross-shard p99 ≤1.1s.

Pass/Fail: Pass.

Reference User Story: SCL-02

Native account abstraction & fee sponsorship

Acceptance Criteria: Users pay zero gas if sponsored, paymaster marketplace.

Test Case ID: TC-SCL-02-01

Description: Gasless onboarding flow.

Prerequisites: Paymasters pre-funded on testnet. New wallet (no balance).

Detailed Test Script:

1. Install fresh wallet → no seed funding.
2. Enable "Gasless mode" (default on).
3. Send private tx (0.05 test NERV).
4. Wallet shows "Sponsored by Paymaster X – 0 gas paid".
5. Tx confirms → balance updated.
6. Check paymaster logs: sponsorship recorded.
7. Repeat 100 txs from new wallets.
8. Verify >95% sponsored.
9. Screenshot wallet UX, sponsorship toast.

Expected Results: Zero user gas, seamless.

Pass/Fail: Pass.

Test Case ID: TC-SCL-02-02

Description: Paymaster competition.

Detailed Test Script:

1. Query marketplace contract → top 10 paymasters.
2. Check coverage stats on dashboard.

Expected Results: >95% new-user txs sponsored.

Pass/Fail: Pass.

Reference User Story: SCL-03

AI fee predictor API

Acceptance Criteria: Exact fee 10s ahead >99.9% accuracy, zero MEV.

Test Case ID: TC-SCL-03-01

Description: Fee prediction accuracy.

Detailed Test Script:

1. Wallet shows "Exact fee: 0.00007 NERV" pre-sign.
2. Record predicted vs actual (post-tx receipt).
3. Run 10,000 txs under varying congestion.
4. Calculate overpay rate.

Expected Results: >99.9% exact or user never overpays.

Pass/Fail: Pass.

Test Case ID: TC-SCL-03-02

Description: MEV elimination.

Detailed Test Script:

1. Run MEV extractor sim ./mev-sim --txs 100000.
2. Check extractor profit.

Expected Results: Profit = 0 (priority only by fee paid).

Pass/Fail: Pass.

Test Case ID: TC-SCL-03-03

Description: Fee smoothing refunds.

Detailed Test Script:

1. Pay during peak → slight overpay.
2. Congestion drops → check automatic micro-refund <10 min.

Expected Results: Refund received.

Pass/Fail: Pass.

Reference User Story: SCL-04

Recursive embedding compression → 900× smaller proofs

Acceptance Criteria: Private transfer proof ~420 bytes.

Test Case ID: TC-SCL-04-01

Description: Proof size reduction.

Detailed Test Script:

1. Send private transfer → get VDW/inclusion proof.
2. Measure total proof component: ./proof-stats vdw.bin.
3. Compare vs baseline zkEVM (e.g., Polygon reference).

Expected Results: Average ≤420 bytes private tx proof, overall 900× compression.

Pass/Fail: Pass.

Test Case ID: TC-SCL-04-02

Description: Mobile verification with folding.

Detailed Test Script:

1. On iPhone 15 → verify aggregated proof.
2. Time <60 ms.

Expected Results: Fast with Nova fallback.

Pass/Fail: Pass.

Test Case ID: TC-SCL-04-03

Description: Aggregation node efficiency.

Detailed Test Script:

1. 10,000 proofs → one aggregated.
2. Daily sync data ≤80 KB.

Expected Results: Achieved.

Pass/Fail: Pass.

Cross-Cutting Tests for Epic 8

Test Case ID: TC-SCL-08-01

Description: 1M+ TPS global stress test.

Detailed Test Script:

1. 7-day continuous run, real mixed load (DeFi, NFT mints, games).
2. 5 continents, video + explorer proof.

Expected Results: >1M TPS sustained, no crashes.

Pass/Fail: Pass.

Test Case ID: TC-SCL-08-02

Description: Multi-hardware benchmark.

Detailed Test Script:

1. Run on AWS, Hetzner, MacStudio, mobile light client sync.

Expected Results: All >500k TPS/node where applicable.

Pass/Fail: Pass.

Test Case ID: TC-SCL-08-03

Description: Scalability dashboard live checks.

Detailed Test Script:

1. Open public explorer → verify live TPS, shard count, fee graph.

Expected Results: Accurate real-time.

Test Cases for Epic 9 – Developer Experience & SDK

Goal Recap: Make building on NERV feel like writing a React app – one-click deploys, multi-language SDKs, AI auditor, light clients everywhere.

Important Notes for Testers:

- Focus on developer and end-user tools: SDKs, CLI, wallets, documentation.
- Tools needed:
    - All SDKs installed (npm i @hln/sdk, pip install hln-sdk, etc.)
    - hln CLI binary (macOS/Linux/Windows)
    - VS Code with NERV extension
    - Mobile wallets (iOS/Android)
    - Web light client demo page

    - AI auditor CLI (hln audit contract.move)
    - Testnet faucet API
- Record: Deployment times, compile/proof sizes, audit detection rate, light client sync times, UX screenshots.

Reference User Story: DX-01

Release Rust + TypeScript/JavaScript, Python, Go, Swift, Kotlin SDK with built-in ZK & embedding generation

Acceptance Criteria: Identical APIs, auto-paymaster, 500 shared e2e tests.

Test Case ID: TC-DX-01-01

Description: Positive – Send private transfer via TypeScript SDK (most common).

Prerequisites: Node.js installed, testnet RPC URL.

Detailed Test Script:

1. Create new project: mkdir test-hln && cd test-hln && npm init -y && npm i @hln/sdk.

Create send.ts:
TypeScript
```
import { HLNWallet, PrivateTransfer } from '@hln/sdk';

const wallet = new HLNWallet('your_test_mnemonic');

await wallet.sync();

const tx = new PrivateTransfer('blinded_recipient_commitment', 0.05);

const receipt = await wallet.send(tx);
```

2. console.log('VDW:', receipt.vdw);
3. Run ts-node send.ts.
4. Check console: tx submitted, VDW received automatically.
5. Verify no manual onion/embedding code needed.
6. Check auto-paymaster: Log "Sponsored by paymaster".
7. Screenshot code + output.

Expected Results: Tx succeeds, zero gas, full privacy automatic.

Pass/Fail: Pass.

Test Case ID: TC-DX-01-02

Description: Cross-language parity (Python example).

Detailed Test Script:

1. pip install hln-sdk.
2. Write identical script in Python (same API calls).
3. Run → same result.

Expected Results: Identical behaviour across languages.

Pass/Fail: Pass.

Test Case ID: TC-DX-01-03

Description: Shared e2e test suite.

Detailed Test Script:

1. Clone repo → run npm run test:e2e (or equivalent).
2. All 500 tests pass across languages.

Expected Results: 100% pass.

Pass/Fail: Pass.

Reference User Story: DX-02

“hln deploy” CLI that compiles, proves, and deploys private contracts in one command

Acceptance Criteria: <6s to mainnet, prints QR for contract address.

Test Case ID: TC-DX-02-01

Description: One-command private contract deploy.

Prerequisites: hln CLI installed, wallet configured.

Detailed Test Script:

1. Create folder my-private-token.
2. Write simple Move contract with #[private] fields.
3. Run hln deploy ./my-private-token.
4. Timer start → end.
5. CLI outputs: compile success, proof generated (~X KB), tx submitted, contract address QR code (displayed in terminal).
6. Scan QR → opens explorer with contract.
7. Screenshot entire CLI output + QR.

Expected Results:

- Total time <6s.
- QR scannable, address correct.

Pass/Fail: Pass.

Test Case ID: TC-DX-02-02

Description: VS Code integration.

Detailed Test Script:

1. Open contract in VS Code with NERV extension.
2. Hover → see proof size estimate.
3. Right-click → "Deploy to HLN" → one-click.

Expected Results: Deploys instantly.

Pass/Fail: Pass.

Reference User Story: DX-03

One-click light client for React Native & Web

Acceptance Criteria: <8s first sync, works offline forever after.

Test Case ID: TC-DX-03-01

Description: Mobile light client sync and offline use.

Prerequisites: Fresh wallet install on iPhone/Android.

Detailed Test Script:

1. Open app → first launch.
2. Timer: from open to "Synced" (embedding root headers).
3. Go airplane mode.
4. View balance, send tx (queued), view old proofs.
5. All work without network error.
6. Screenshot sync timer, offline screens.

Expected Results:

- First sync <8s.
- Full offline functionality.

Pass/Fail: Pass.

Test Case ID: TC-DX-03-02

Description: Web embeddable light client.

Detailed Test Script:

1. Open demo page with <script src="hln-light.js"></script>.
2. Embedded widget syncs, shows balance.

Expected Results: Works in iframe, <8s sync.

Pass/Fail: Pass.

Test Case ID: TC-DX-03-03

Description: Deep links and QR.

Detailed Test Script:

1. Generate hln://pay/... link.
2. Click/scan → wallet opens → tx ready <800 ms.

Expected Results: Instant open.

Pass/Fail: Pass.

Reference User Story: DX-04

AI auditor flags 95% of reentrancy, overflow, DoS bugs at compile time

Acceptance Criteria: "Fix this for me" button, auto-fix 80%+ common bugs.

Test Case ID: TC-DX-04-01

Description: AI audit detection rate.

Detailed Test Script:

1. Prepare 100 vulnerable contracts (known reentrancy, overflow, etc.).
2. Run hln audit contract.move.
3. Count flagged vs actual vulnerabilities.

Expected Results: ≥95% detection with explanation.

Pass/Fail: Pass.

Test Case ID: TC-DX-04-02

Description: Auto-fix functionality.

Detailed Test Script:

1. Load vulnerable contract in IDE.
2. AI flags issue → "Fix this for me".
3. Click → patched version suggested.
4. Approve → clean contract.

Expected Results: ≥80% common bugs auto-fixed correctly.

Pass/Fail: Pass.

Bonus User-Facing Stories

Test Case ID: TC-DX-05-01

Description: Natural private send UX.

Detailed Test Script:

1. Recipient shows QR "Send me $10 privately".
2. Sender scans → amount prefilled, no address visible.

Expected Results: Seamless.

Pass/Fail: Pass.

Test Case ID: TC-DX-06-01

Description: Private ENS resolution.

Detailed Test Script:

1. Register vitalik.hln → blinded commitment.
2. Resolve → only owner sees link.

Expected Results: No public linkage.

Pass/Fail: Pass.

Test Case ID: TC-DX-07-01

Description: Privacy-preserving bridge.

Detailed Test Script:

1. Bridge ETH → private HLN.
2. One-click, <15s, no address reuse.

Expected Results: Private on arrival.

Pass/Fail: Pass.

Test Case ID: TC-DX-08-01

Description: Private payment push notifications.

Detailed Test Script:

1. Receive tx → push "You received money".
2. Open → reveal after biometric.

Expected Results: No leak pre-unlock.

Pass/Fail: Pass.

Cross-Cutting Tests for Epic 9

Test Case ID: TC-DX-09-01

Description: Documentation and playground.

Detailed Test Script:

1. Visit docs.hln.net → interactive examples work.

Expected Results: Clear, functional.

Pass/Fail: Pass.

Test Case ID: TC-DX-09-02

Description: Bug bounty program live.

Detailed Test Script:

1. Check Immunefi page → $10M pool active.

Expected Results: Live at testnet.

Pass/Fail: Pass.

Test Cases for Epic 10 – Testnet → Mainnet Launch Sequence

Goal Recap: Safe, audited, incentivized rollout culminating in mainnet genesis (June 2028). This epic covers the final pre-launch phases: internal devnet, public testnet, audits, bug bounties, genesis ceremony, and Day 0–30 monitoring.

Important Notes for Testers:

- These tests span months/years – many are milestone/gating checks, not daily scripts.
- Coordination required: Core tech, ops, security, foundation, auditors, community.
- Tools needed:
    - Chaos injector suite (node killer, netem, byzantine actors)
    - Audit tracker (spreadsheet/Notion)
    - Bug bounty dashboard (Immunefi)
    - Genesis verifier tool (./genesis-verify file.json)
    - War room monitoring (Grafana, PagerDuty)
    - Livestream setup
- Record: Reports, videos, signed letters, metrics screenshots, timelines.

Reference User Story: LCH-01

Internal devnet (100 nodes) – all epics above

Acceptance Criteria: 30-day continuous run, ≥300k TPS, zero crashes, all VDWs valid.

Test Case ID: TC-LCH-01-01

Description: Devnet-Alpha stability and performance.

Prerequisites: 100–200 core team nodes deployed (diverse hardware/OS/geo).

Detailed Test Script:

1. Launch devnet: Ops script ./deploy-devnet-alpha --nodes 150.
2. Confirm all epics integrated (check feature flags).
3. Start chaos baseline: 10% random node restarts daily.
4. Run continuous load: ./tps-blaster --global --target 400000tps --duration 30days.
5. Monitor Grafana: TPS, crashes, finality, VDW validity rate.
6. Daily check: Zero unplanned crashes, all VDWs verify offline.
7. Week 4: Inject 20% byzantine + partitions.
8. Final report: Export 30-day metrics, video summary.
9. Screenshot key dashboards weekly.

Expected Results:

- ≥300k TPS sustained.
- Zero critical crashes.
- Self-healing under chaos.

Pass/Fail: Pass if 30 days complete with metrics met.

Gate: Blocks public testnet.

Test Case ID: TC-LCH-01-02

Description: 168-hour full chaos campaign.

Detailed Test Script:

1. Intensify chaos: 50% byzantine, full partitions, latency spikes.
2. Run 168h non-stop.
3. Log every recovery time.

Expected Results: Self-heal within SLA, published report.

Pass/Fail: Pass.

Reference User Story: LCH-02

Public testnet with 10k nodes & real token rewards

Acceptance Criteria: ≥10k nodes week 1, ≥100k DAW by week 8, staged TPS ramps.

Test Case ID: TC-LCH-02-01

Description: Testnet-Omega launch and adoption.

Detailed Test Script:

1. Announce launch → open faucet + rewards.
2. Weekly monitor: Node count (explorer), DAW (wallet analytics).
3. Week 1 target: ≥10k nodes.
4. Run staged ramps: Week 2 500k TPS, Week 4 1M, Week 6 1.5M.
5. Community scripts for dApp deployment.
6. Screenshot weekly metrics, leaderboard.

Expected Results: Adoption targets hit, no rollback needed.

Pass/Fail: Pass.

Test Case ID: TC-LCH-02-02

Description: Genesis rehearsal on testnet.

Detailed Test Script:

1. Full dry-run of mainnet ceremony.
2. Livestream, record.
3. Verify 100% success.

Expected Results: Flawless execution.

Pass/Fail: Pass.

Reference User Story: LCH-03

Four independent security audits

Acceptance Criteria: All clean or low-only findings.

Test Case ID: TC-LCH-03-01

Description: Audit coordination and fixes.

Detailed Test Script:

1. Commission audits (Trail of Bits, Kudelski, Runtime Verification, academic).
2. Provide full source, binaries, threat models.
3. Weekly sync meetings.
4. Track findings in shared sheet: ID, severity, status, fix commit.
5. Re-audit rounds until zero critical/high.
6. Collect final clean reports + attestation letters.
7. Publish all reports publicly.

Expected Results: Four clean final reports.

Pass/Fail: Pass (mandatory gate for genesis).

Reference User Story: LCH-04

Bug bounty up to $10M+

Acceptance Criteria: Live 90 days pre-genesis, valid criticals paid.

Test Case ID: TC-LCH-04-01

Description: Bounty program operation.

Detailed Test Script:

1. Launch on Immunefi ($5M critical, $10M quantum-break).
2. Monitor submissions weekly.
3. Triage valid criticals → pay out transparently.
4. Zero unresolved criticals at genesis.

Expected Results: Program active, payouts public.

Pass/Fail: Pass.

Test Case ID: TC-LCH-04-02

Description: Red-team pentest.

Detailed Test Script:

1. Engage external red team (insider access).
2. Final report clean/cosmetic.

Expected Results: No high findings.

Pass/Fail: Pass.

Reference User Story: LCH-05

Genesis with 5-year token emission schedule & useful-work rewards

Acceptance Criteria: Transparent ceremony, ≥25k validators ready.

Test Case ID: TC-LCH-05-01

Description: Genesis allocation ceremony.

Detailed Test Script:

1. Multi-sig + TEE + HSM setup.
2. Generate 4096 PQ key shares.
3. Livestream ceremony.
4. Publish genesis file 14 days pre-launch (IPFS/Arweave).

Expected Results: Successful, verifiable.

Pass/Fail: Pass.

Test Case ID: TC-LCH-05-02

Description: Validator onboarding.

Detailed Test Script:

1. Open portal → monitor sign-ups.
2. Verify ≥25k independent validators (stake + TEE).

Expected Results: Target reached.

Pass/Fail: Pass.

Reference User Story: LCH-06

Mainnet launch – Day 1 target 100k TPS

Acceptance Criteria: Block 0 produced, war room 7 days, listings live.

Test Case ID: TC-LCH-06-01

Description: Genesis Day execution.

Detailed Test Script:

1. UTC 14:00 → start countdown livestream.
2. Monitor Block 0 production.
3. Verify embedding root matches published genesis.
4. Confirm ≥15 dApps, ≥3 Tier-1 CEX listings.

Expected Results: Smooth launch.

Pass/Fail: Pass.

Test Case ID: TC-LCH-06-02

Description: Day 0–7 war room.

Detailed Test Script:

1. 24/7 monitoring, hotfix readiness.
2. No rollback needed.

Expected Results: Stable first week.

Pass/Fail: Pass.

Test Case ID: TC-LCH-06-03

Description: Post-launch 30-day report.

Detailed Test Script:

1. Compile metrics, incidents.
2. Publish T+30.

Expected Results: Transparent summary.

Pass/Fail: Pass.

Cross-Cutting Final Safeguards

Test Case ID: TC-LCH-FINAL-01

Description: Independent genesis verification.

Detailed Test Script:

1. Multiple teams run ./genesis-verify genesis.json.
2. Signed letters T-7 days.

Expected Results: All confirm integrity.

Pass/Fail: Pass.

Test Case ID: TC-LCH-FINAL-02

Description: Disaster recovery drill.

Detailed Test Script:

1. Restore from genesis + snapshots.
2. Measure RTO <4 hours.

Expected Results: Successful.

Pass/Fail: Pass.