AEGIS — Pre-Audit Hardening Checklist

Purpose. The work to do *before* paying for an external audit. Auditors charge the same whether they find issues you already knew about or not, so closing known gaps first converts audit spend into new findings rather than a list you could have written yourself.

Sequencing principle. Cheap-and-serious first (P0), then the issues that require design decisions (P1), then scale and research work (P2). Each item states the concrete fix and how you know it is done.

P0 — Close before showing the code to anyone external

P0-1 · Lock down the local HTTP API DONE

Where: `aegis_app.py` (`Handler`, `start_http`) **Problem:** No authentication, no CSRF token, no `Origin`/`Host` validation. Any website the user visits can POST to `127.0.0.1:8001/api/send`; DNS rebinding is possible. **Fix:** - Generate a random 32-byte session token at engine start; print the UI URL with the token embedded (`http://127.0.0.1:8001/?t=…`). Require it as a `Authorization: Bearer` header on every `/api/*` call. - Reject any request whose `Origin` is present and not `http://127.0.0.1:<port>`, and whose `Host` is not `127.0.0.1:<port>` or `localhost:<port>`. - Add `X-Content-Type-Options: nosniff`, `Content-Security-Policy: default-src 'self'`, and `Cache-Control: no-store` to responses. - Bind explicitly to `127.0.0.1` (already done) and document that `0.0.0.0` must never be used without TLS and real auth. **Done when:** a cross-origin `fetch()` from a test page cannot send a message; requests without the token receive 401; a rebinding test with a spoofed `Host` is rejected. Add regression tests for all three. **Effort:** 0.5–1 day.

P0-2 · Stop storing private keys in plaintext DONE (passphrase vault; OS keystore still recommended)

Where: `persistence.py`, `aegis_app.py`/`aegis_chat.py` (`--state`) **Problem:** Identity and ratchet private keys are written as unencrypted JSON. This also weakens the forward-secrecy claim, because old key material persists. **Fix:** - Encrypt the state file with a key derived from a user passphrase via Argon2id (or scrypt if you must stay stdlib-adjacent), AEAD-sealed with ChaCha20-Poly1305, with a versioned header. - Prefer an OS keystore where available: DPAPI (Windows), Keychain (macOS), libsecret/kwallet (Linux). Fall back to the passphrase file. - Set restrictive file permissions (`0600`; on Windows, an ACL limited to the user). - Delete superseded ratchet keys explicitly rather than leaving them in the serialized blob. **Done when:** the state file is unreadable without the passphrase/keystore; a test asserts no raw key bytes appear in the file; permissions are verified on each supported OS. **Effort:** 2–4 days (keystore integration is the long pole).

P0-3 · Authenticate mailbox tokens and provider polls DONE

Where: `messenger.py::mailbox_token`, `transport.py` (`FRAME_POLL` handling) **Problem:** The token is `H(public AegisID)`, so anyone knowing a user's public ID can compute it and poll the provider — and the provider `pop`s messages on read, so an attacker **steals and destroys** the victim's queued ciphertext. **Fix:** - Establish a per-user secret with the provider at registration; derive tokens as `HMAC(secret, epoch)` so they rotate and are unguessable. - Require the poller to prove knowledge of the secret (challenge–response or a signed poll), not merely present the token. - Do not delete on read: mark delivered and expire on a timer, so a lost connection does not destroy mail. **Done when:** a test shows a third party who knows only the victim's AegisID cannot retrieve or delete their mail; tokens differ across epochs. **Effort:** 2–3 days.

P0-4 · Bound memory and add rate limits DONE (bounds+TTL; per-sender rate limits still TODO)

Where: `transport.py` (`provider_store`, `_handle_conn`, `_out_q`) **Problem:** `provider_store` is an unbounded in-memory dict; relays accept unlimited packets; queues are unbounded. Trivial memory-exhaustion DoS. **Fix:** per-mailbox message cap and byte quota; TTL expiry sweep; global store ceiling with backpressure; per-connection and per-peer packet rate limits; bounded `asyncio.Queue` sizes with explicit drop policy; connection count limits. **Done when:** a flood test (10^5 packets, many mailboxes) leaves memory bounded and the node responsive; limits are configurable and documented. **Effort:** 2–3 days.

P0-5 · Persist sessions in the application

Where: `aegis_app.py` (never calls `persistence.dump_ratchet`) **Problem:** Conversations reset on restart, which users read as data loss and which pushes them toward re-handshaking unnecessarily. **Fix:** persist ratchet state (encrypted, per P0-2) on a debounce after each message and on clean shutdown; restore on start; handle corrupt-state recovery by falling back to a fresh session with a visible UI notice. **Done when:** a restart test continues an existing conversation, including out-of-order handling, with encrypted state on disk. **Effort:** 1–2 days.

P1 — Close before production, requires design decisions

P1-1 · Sign and expire the directory DONE (single-authority)

Where: `directory.py` **Fix:** directory entries signed by the publishing node; documents carry issue/expiry timestamps; clients reject stale or unsigned entries; plan a multi-authority or gossip model so no single writer controls the relay set. **Effort:** 3–5 days for the signed-entry version; longer for consensus.

P1-2 · Delivery semantics

Where: `messenger.py` **Fix:** message IDs, acknowledgements, bounded retry with jitter, duplicate suppression, and an explicit per-message state machine (queued → sent → delivered → failed) surfaced in the UI. Note the tension: read receipts and delivery receipts leak metadata; make them optional and off by default. **Effort:** 1–2 weeks.

P1-3 · Handle glare and session collisions

Where: `handshake.py`, `deniable.py`, `messenger.py` **Fix:** deterministic tie-break (e.g. lower AegisID wins) when both parties initiate simultaneously; define behaviour for duplicate initial messages and replayed prekey bundles; enforce one-time-prekey single use across restarts (currently only in memory). **Effort:** 3–5 days.

P1-4 · Fuzz the parsers

Where: `wire.py`, `messenger.py` (`_load_initial`), `persistence.py`, `ratchet.py` (header parsing) **Fix:** property-based tests (Hypothesis) and a coverage-guided fuzzer (Atheris) against every `from_bytes`/`_unlp` path. These parse attacker-supplied bytes and are the classic source of memory and logic bugs. Assert no unhandled exception type escapes the frame handler. **Effort:** 3–5 days for a first harness; run continuously thereafter.

P1-5 · Migrate to audited native PQ crypto

Where: `crypto.py` **Fix:** replace `kyber-py`/`dilithium-py` with liboqs or PQClean bindings. Validate against NIST KATs. Expect a 10–100× speedup, which unblocks realistic cover-traffic rates and relay throughput. **Why P1 and not P0:** it is a dependency swap behind a stable interface, but the current libraries are explicitly not production-grade or side-channel hardened. **Effort:** 1–2 weeks including test-vector validation and packaging per OS.

P1-6 · Threat-model the endpoint

Fix: define and document what happens on a compromised device: disappearing messages, local database encryption, screen-capture posture, clipboard hygiene, and a documented "we do not defend against endpoint compromise" statement in the product itself, not only in the whitepaper. **Effort:** design work; 1 week to document and implement the cheap parts.

P2 — Scale, research, and product maturity

- **P2-1 · Formal verification.** Model the handshake and ratchet in Tamarin or ProVerif. This is what turns "we argue informally" into a defensible claim, and it is what a serious reviewer will ask for. *4–8 weeks of specialist time.*
- **P2-2 · Post-quantum Sphinx.** Open research. Track the literature; do not promise it.

- **P2-3 · Class-group accumulator** to remove the trusted setup in `trust.py`.
- **P2-4 · Sybil resistance** for the web of trust (proof-of-personhood, stake, or vouching cost).
- **P2-5 · Multi-device and backup.** Users expect both; both are genuinely hard under forward secrecy. Expect a full quarter.
- **P2-6 · Group messaging at scale.** Migrate from sender keys to MLS (RFC 9420) rather than inventing group rekeying.
- **P2-7 · Performance and load testing.** Establish relay capacity, latency under mix delay, and the bandwidth cost per idle user — the last number determines your unit economics.
- **P2-8 · Mobile clients.** See the commercialization document; this is the single largest engineering line item.

Engineering practice to adopt now

- **CI on every commit:** the 25-test suite, plus linting and a type-checker (`mypy --strict` on `aegis/`).
- **Reproducible builds and pinned, hash-verified dependencies** (`pip --require-hashes`). A supply-chain compromise of a PQ library is a total break.
- **Signed release artifacts** and a documented verification procedure.
- **A written security policy** (`SECURITY.md`) with a disclosure address and response commitment, plus a bug-bounty program once audited.
- **Test vectors** published for the hybrid constructions so third parties can validate interoperability.
- **No new claims without a test.** Every entry in the due-diligence claims table should map to at least one automated test.

Suggested order of work

```
Week 1      P0-1   , P0-2           (auth + key storage — DONE)
Week 2      P0-3   , P0-4           (mailbox auth + DoS bounds — DONE)
Week 3      P0-5, P1-4 harness    (persistence + fuzzing)
Weeks 4–6   P1-1, P1-2, P1-3      (directory, delivery, glare)
Weeks 6–8   P1-5                  (native PQ crypto)
Week 9      Freeze, re-test, prepare audit package
Weeks 10+   External audit; remediate findings; publish report
```

A disciplined engineer could complete P0 in about two weeks and P0+P1 in roughly two months. That is the realistic minimum before an audit is worth paying for.