AEGIS — Technical Due Diligence

Document status: pre-audit self-assessment, prepared by the development team. **Subject:** AEGIS reference implementation and AEGIS Desktop prototype. **Version reviewed:** the codebase accompanying this document (2,269 lines of protocol code across 15 modules; 1,487 lines of application, demo and test code; 31 automated tests, all passing).

> **Read this first.** This document is written to be *useful to a reviewer*, not to sell. Every known weakness we are aware of is listed in §6, including issues that a hostile reviewer would otherwise find and hold against us. No independent security audit has been performed. AEGIS must not be deployed to protect people at real risk until §6 items are closed and an external audit is complete.

---

1. What AEGIS is

AEGIS is a secure messaging protocol and reference implementation whose design thesis is that **message-content security is a solved problem, and the remaining hard problems are metadata resistance and trust establishment.** It therefore reuses standard, vetted primitives for confidentiality and invests its originality in composition:

1. **Hybrid post-quantum key agreement and signatures.** X25519 + ML-KEM-768 (FIPS 203) for key encapsulation; Ed25519 + ML-DSA-65 (FIPS 204) for signatures. Security holds if *either* the classical or post-quantum component holds.
2. **A hybrid Double Ratchet.** A KEM ciphertext is carried in each asymmetric ratchet step alongside the classical DH, giving forward secrecy and post-compromise security against a quantum-capable adversary.
3. **A quantified web of trust.** Trust in an unknown key is computed from k vertex-disjoint attestation paths, aggregated with a noisy-OR rule, with membership provable via an RSA accumulator without revealing the trusted set.
4. **A metadata-resistant transport.** Onion routing over the hybrid KEM, constant-rate cover traffic, fixed-size packets, Loopix-style Poisson per-hop mix delays, and provider mailboxes so a recipient is never the final hop.

What is genuinely novel is the *combination*: to our knowledge no shipped system pairs a hybrid PQ ratchet with a quantified, privacy-preserving web of trust on top of a provider-mailbox mixnet. **What is not novel** is any individual primitive — and that is deliberate. We invent no ciphers.

2. Implementation inventory

Module	Lines	Responsibility
`crypto.py`	236	Primitive wrappers; hybrid KEM and signature composition
`identity.py`	101	Self-certifying IDs, prekey bundles
`handshake.py`	135	Signed PQXDH-style handshake (non-repudiable)
`deniable.py`	122	Deniable handshake (implicit DH authentication)
`ratchet.py`	229	Hybrid Double Ratchet, atomic decrypt
`trust.py`	251	RSA accumulator, k-disjoint-path web of trust
`mixnet.py`	175	In-process onion routing and cover traffic model
`groups.py`	105	Sender-key group messaging
`session.py`	74	High-level client API
`persistence.py`	113	Identity and ratchet-state serialization
`wire.py`	122	Fixed-size onion packets, TCP framing
`transport.py`	265	Async TCP mix node, mix delays, provider mailboxes
`messenger.py`	187	End-to-end sessions over the transport
`directory.py`	113	File-backed peer discovery
`verify.py`	40	Safety numbers for out-of-band verification

Applications: `aegis_app.py` (264) + `aegis_webui.html` (362) — desktop app; `aegis_chat.py` (278) — terminal client; `net_demo.py`, `demo.py` — demonstrations.

Dependencies: `cryptography` (X25519, Ed25519, ChaCha20-Poly1305, HKDF, SHA-256), `kyber-py` (ML-KEM-768), `dilithium-py` (ML-DSA-65), `pytest` (test only). See §7 for the supply-chain assessment — **the two PQ dependencies are the single largest supply-chain risk in the project.**

3. Security claims, stated precisely

Claims are stated with their scope. A claim not listed here is not made.

#	Claim	Scope and conditions
C1	Message confidentiality and integrity	Against an adversary who does not hold either endpoint's long-term or session keys. AEAD is ChaCha20-Poly1305.
C2	Forward secrecy	Compromise of current keys does not reveal earlier messages, provided old keys were erased. Key material at rest is now encrypted (G3 resolved); in-memory erasure is still not guaranteed in Python.
C3	Post-compromise security	After a passive compromise ends, security heals once an asymmetric ratchet step completes.
C4	Post-quantum confidentiality	Holds if *either* X25519 or ML-KEM-768 remains unbroken. Addresses harvest-now-decrypt-later.
C5	Post-quantum authentication (signed mode)	Holds if *either* Ed25519 or ML-DSA-65 remains unbroken.
C6	Deniability (deniable mode)	Repudiable authentication; no transcript signature exists. **Rests on the classical X25519 leg only** — see G1.
C7	Sender–recipient unlinkability vs. a global passive observer	Under the stated mixnet assumptions: constant-rate cover traffic, uniform packet size, Poisson mix delays, ≥1 honest relay on the path, and a sufficiently large concurrent user population. **The last condition is not satisfiable at low user counts** — see G2.
C8	Recipient is not the final hop	Provider mailboxes; the penultimate relay sees traffic addressed to a provider. The provider still learns that *its own* user has mail.
C9	Identity binding	AegisIDs are the hash of the identity keys, so a substituted key yields a different ID and a different safety number.

Explicitly not claimed: anonymity against an adversary who controls all relays on a path; resistance to long-horizon statistical intersection attacks; protection of a compromised endpoint; availability under active attack; resistance to traffic confirmation when the user population is small.

4. Threat model

In scope. Global passive network observer; malicious relays (up to all but one on a path, for unlinkability); a malicious peer; an adversary recording traffic today to decrypt after a cryptographically relevant quantum computer exists; a network attacker attempting MITM at session setup.

Out of scope. Endpoint compromise (malware, coerced device unlock, screen capture); the user's own operating system and browser; physical coercion; traffic confirmation when so few users are online that cover traffic provides no cover; denial of service; and — importantly — **the mixnet operator's ability to observe that a given user is online at all.**

5. Verification status

- **31 automated tests, all passing**, covering: hybrid primitives; handshake agreement; ratchet forward secrecy, out-of-order delivery and atomic decrypt on forged headers; k-disjoint-path trust scoring; accumulator membership proofs; group sender keys; deniable handshake including a failed impersonation; identity/ratchet persistence across restart; fixed-size onion construction; live TCP delivery with uniform injection rates and uniform packet sizes; end-to-end conversation over the mixnet; provider-mailbox routing; and safety numbers including MITM detection.
- **Independently reproduced** on a separate machine (Windows, Python 3.14, `cryptography` 50.0.0, `kyber-py` 1.2.0, `dilithium-py` 1.4.0).
- **Not done:** formal protocol verification, adversarial testing, fuzzing, side-channel analysis, performance testing at scale, or any external review.

6. Known gaps register

Severity reflects impact **if the system were deployed today**. This is the section a reviewer should read most carefully.

Critical — G3 and G4 are now resolved (see); G1 and G2 remain

G1 — Deniability rests on a classical assumption. The deniable handshake authenticates via an X25519 DH. Against a quantum adversary, confidentiality survives (the PQ KEMs still apply) but the *deniability* property does not. A fully post-quantum deniable AKE is an open research problem. *Mitigation:* document the mode's limits; offer signed mode where non-repudiation is acceptable; track the literature.

G2 — Anonymity requires a crowd that does not yet exist. Cover traffic and mixing only hide a user among *other simultaneous users*. With a handful of users, a global observer can trivially correlate. This is inherent to mixnets, not a bug, but it means **the privacy claim is weakest exactly when a product is newest.** *Mitigation:* be explicit in marketing; consider bootstrap relays that generate baseline traffic; do not claim anonymity below a stated population threshold.

G3 — Encrypted-at-rest key storage. RESOLVED (passphrase vault). Identity state files are now sealed with a key derived from a user passphrase via scrypt (stdlib) and encrypted with ChaCha20-Poly1305; the app and CLI take a `--passphrase` (or `AEGIS_PASSPHRASE`, or an interactive prompt) whenever `--state` is used, and legacy plaintext files are transparently migrated on first open. Verified by tests: no private-key bytes appear in the sealed file, a wrong passphrase fails, and the AegisID is stable across reload. *Residual:* an OS keystore (DPAPI/Keychain/libsecret) and in-memory zeroization are still recommended for production (Python cannot guarantee erasure), and ratchet-session state at rest should use the same vault once P0-5 lands; POSIX file mode is set to 0600 but a Windows ACL is not yet applied.

G4 — Local HTTP API authentication. **RESOLVED.** Previously the API had zero authentication or origin checking. It now requires a per-session bearer token (32 bytes, printed once in the launch URL, compared in constant time) on every `/api/*` call, and rejects any request whose `Host` is not our loopback endpoint (defeating DNS rebinding) or whose `Origin` is cross-site (defeating CSRF). Security headers (`nosniff`, `no-referrer`, `no-store`, a restrictive CSP) are set on all responses. Verified by an end-to-end test (missing/wrong token → 401; bad origin/host → 403) and unit tests on the decision helpers. *Residual:* on Windows the state-file permission restriction relies on an ACL not yet applied (see G3); the token appears in the URL, so `no-referrer` is set to avoid leakage.

High — G5 and G7 resolved; G6 partially resolved

G5 — Authenticated mailbox retrieval. **RESOLVED.** The mailbox token is now bound to the recipient's signing key (`H(domain ‖ signing_pub)`), and retrieval is a challenge–response: the provider issues a random nonce and releases mail only after the caller signs `(nonce ‖ token)` with the matching hybrid identity key. Knowing a user's public ID (hence their token) no longer allows stealing or deleting their mail. Verified by a test where an attacker with the victim's public key but a different signing key is refused and the mail is preserved for the real owner. *Residual:* per-epoch token rotation and unlinkability of the token from the identity at the provider are still future work.

G6 — Resource bounds. **PARTIALLY RESOLVED.** Provider storage now enforces a global mailbox cap, a per-mailbox message cap (shedding oldest), and TTL expiry via a periodic sweep; outbound queues are bounded with a drop policy; and concurrent inbound connections are capped. Verified by a bounds test. *Residual:* per-sender rate limiting and admission control (proof-of-work or tokens) are not yet implemented, so a high-volume sender can still consume a share of capacity.

G7 — Signed, expiring directory entries. **RESOLVED (single-authority).** Each directory entry is now signed by the publishing identity over a canonical encoding of all its fields, carries issued/expiry timestamps, and is re-published within its TTL; readers reject entries that are unsigned, tampered, key-mismatched, or expired. Verified by tests (tamper, missing signature, and expiry are all rejected). *Residual:* this defends integrity and freshness, not availability — a folder-writer can still delete files or withhold entries; production needs a gossiped or multi-authority directory.

G8 — No replay protection at the provider layer, and no delivery semantics. There are no acknowledgements, retries, ordering guarantees, or duplicate suppression above the ratchet. Messages can be silently lost.

Medium

G9 — Size uniformity is not full Sphinx. Packets are padded and re-padded to a constant size, which defeats size correlation, but the construction lacks Sphinx's exact per-hop bitwise unlinkability and reply blocks. Post-quantum Sphinx is an open problem.

G10 — RSA accumulator uses a trusted setup. `trust.py` generates a 2048-bit modulus and discards the factors; anyone retaining them could forge membership proofs. *Fix:* class groups (no trusted setup) or a multi-party ceremony.

G11 — Performance. Pure-Python ML-KEM measures ≈6.1 ms per encapsulation and ≈8.2 ms per decapsulation in our environment, so a 3-hop onion costs ≈18 ms of CPU to build. This forced cover-traffic intervals ≥0.2 s in testing and will not support a large relay at scale. *Fix:* native libraries (liboqs / PQClean bindings) — likely a 10–100× improvement.

G12 — No session persistence in the application. `aegis_app.py` never calls the persistence layer for ratchet state (confirmed), so conversations reset when the engine restarts. The capability exists in `persistence.py` but is unused.

G13 — Sybil resistance. k-disjoint paths raise the cost of forging trust but do not stop a determined attacker from farming identities.

G14 — No glare handling. Simultaneous session initiation by both parties is not resolved deterministically.

Lower

G15 — No multi-device support; no message backup; no group rekeying in the networked layer; no media/attachments; MAX_SKIP=1000 skipped keys is a tunable memory-vs-usability DoS surface; no push notifications (architecturally awkward, since push providers observe delivery).

7. Supply chain and IP

PQ dependencies are the key risk. `kyber-py` and `dilithium-py` are educational/reference-quality pure-Python implementations. They are readable and correct-by-inspection for our purposes, but they are **not constant-time, not side-channel hardened, and not audited for production use.** Any production build must migrate to a maintained native implementation (liboqs, PQClean, AWS-LC, or the `cryptography` library's PQ support as it matures) and re-validate all test vectors. `cryptography` itself is well-maintained and widely audited.

Standards alignment. ML-KEM-768 and ML-DSA-65 are the NIST-standardized FIPS 203/204 algorithms — the right long-term choices. The hybrid construction follows current best practice (do not abandon classical security).

Licensing. Dependency licenses must be inventoried before distribution (`cryptography` is Apache-2.0/BSD dual; the PQ libraries are permissive but must be confirmed per-version). No third-party code was copied into AEGIS.

Patents/novelty. The composition is plausibly novel; the primitives are not. Prior art is dense (Signal's X3DH/Double Ratchet, PQXDH, Loopix, Nym, Sphinx, accumulator literature). Patentability is doubtful and, in this market, likely counterproductive. *This*

is not legal advice — retain counsel before any publication or filing decision, since disclosure timing affects rights.

8. Reviewer's shortlist

If a reviewer has one hour, we suggest:

1. `aegis_app.py` request handling — review the G4 fix (`_reject`, `host_ok`, `origin_ok`, `token_ok`) for bypasses.
2. `transport.py` poll handling (`_verify_poll`, `poll_provider`) — review the G5 challenge–response for bypasses.
3. `persistence.py` — review the G3 vault (`seal`/`unseal`, scrypt parameters, migration path).
4. `ratchet.py::decrypt` — verify the snapshot/restore rollback genuinely prevents state corruption from forged headers.
5. `crypto.py` — verify the hybrid KDF binds both shared secrets correctly and that the composition is not weakened by the combiner.

9. Bottom line

AEGIS is a **credible, working research prototype with an unusually complete implementation** for its stage: the protocol runs end-to-end over real sockets, the metadata defences are implemented rather than described, and the test suite covers the claims. It is **not production software.** The path to production runs through §6: G3 and G4 (the cheap, serious issues) are now fixed with tests; G5, G6 (partial) and G7 are also fixed; G8 (delivery semantics) and parser fuzzing are next, then an external audit, then the native-crypto migration in G11.

Any statement that AEGIS "protects" users should be withheld until that work is done and independently reviewed.