Loggie — Technical Capability Brief

Cryptographic Integrity Infrastructure for Defense and Government Systems

Document Classification: UNCLASSIFIED

Brief Version: 2.0.0

Platform Version: 1.4.0

Date: 2026-02-12

Repository Commit: `9304063`

Build Provenance: Deterministic build from `scripts/build-defense-brief.mjs`

Table of Contents

- A. Executive Summary
- B. System Overview
- C. Architecture
- D. Deployment Models
- E. Data Flow and Trust Boundaries
- F. Security Posture Summary
- G. Cryptographic Primitives
- H. Integration Footprint
- I. Compliance Mapping
- J. Appendix

A. Executive Summary

Loggie is a cryptographic integrity and verification infrastructure for document provenance, authenticated messaging, and verifiable identity management. It provides cryptographic sealing, post-quantum signatures, content-addressed storage, and optional on-chain notarization of integrity proofs. Loggie is deployed as a software layer beneath existing operational systems — it does not replace databases, SIEMs, or records management platforms. It adds an independent, verifiable integrity chain to the records they produce.

Core capability: Loggie enables organizations to create, seal, verify, and audit digital records with cryptographic guarantees of authenticity and integrity — without exposing sensitive content to any external system.

Key properties:

- **No sensitive data egress.** Plaintext content, private keys, and personally identifiable information never leave the customer-controlled environment. Only cryptographic commitments (hashes, signatures, encrypted ciphertext) cross the trust boundary.
- **Post-quantum readiness.** The cryptographic layer implements NIST-standardized post-quantum algorithms (ML-KEM-768 for key encapsulation, ML-DSA-65 for digital signatures) alongside classical primitives (X25519, ECDSA), providing hybrid security against both classical and quantum adversaries.
- **Deployment flexibility.** Loggie operates identically across connected, on-premises, and fully air-gapped environments. Core cryptographic operations require no network connectivity. All deployment modes produce identical integrity guarantees.
- **Verifiable integrity chain.** Documents and messages receive cryptographic seals composed of content hashes, Merkle tree commitments, and digital signatures. These seals support independent, third-party verification without access to the original plaintext.
- **Optional notarization.** Integrity proofs may be optionally anchored to public or private ledgers for tamper-evident timestamping. This capability is additive; the system functions fully without it.
- **Non-repudiation.** Every sealed record carries a Dilithium (ML-DSA-65) digital signature bound to its creator's cryptographic identity. This provides demonstrable authorship that survives independent audit, litigation, and inspector general review.

Loggie comprises 35+ packages organized into five architectural layers, from application interfaces through core cryptography to optional on-chain anchoring. It is designed for integration into enterprise and government systems that require strong data provenance guarantees — including CMMC-scoped environments, supply chain verification workflows, and classified enclave operations.

B. System Overview

What Loggie Is

Loggie is a **cryptographic integrity and verification infrastructure** delivered as a modular, deployable software layer. It provides:

1. **Sealed Envelopes** — End-to-end encrypted message containers with post-quantum key encapsulation and authenticated encryption (AEAD). Each envelope carries a digital signature for sender authentication.
1. **Identity Management** — Self-sovereign identity creation with cryptographic key generation (classical and post-quantum), key rotation, and a keyring architecture that isolates secret material from application code.
1. **Document Integrity** — Content hashing, metadata sanitization, and Merkle tree computation for batched integrity proofs. Documents receive verifiable integrity tags without modification of the original content.
1. **Notarization Anchoring** — Optional submission of Merkle roots and commitment hashes to smart contracts for immutable, timestamped proof of existence.
1. **Decentralized Storage Integration** — Content-addressed storage via IPFS and Filecoin, ensuring that sealed (encrypted) content is stored without reliance on a single storage provider.

What Loggie Is Not

- Loggie is **not a blockchain**. It uses existing networks for optional notarization.
- Loggie is **not a storage platform**. It integrates with existing storage infrastructure.
- Loggie is **not a system of record**. It sits beneath operational systems, adding an independent integrity layer to the records they already produce.
- Loggie does **not transmit plaintext**. All data leaving the customer environment is encrypted or is a cryptographic commitment.
- Loggie does **not require network connectivity** for core cryptographic operations.

Defense-Relevant Use Cases

Operational Domain	Capability	Value
CMMC evidence anchoring	Cryptographic seals on policy documents, access logs, incident reports	Tamper-evident audit trail that survives third-party assessment
Supply chain provenance	Hash-based integrity proofs on firmware, BOMs, supplier attestations	Independent verification without exposing source material
Autonomous mission logs	Sealed envelopes on UAS telemetry, AI decision events, flight records	Non-repudiable mission record with post-quantum signatures
Incident reconstruction	Merkle tree commitments over forensic evidence chains	Litigation-defensible, timestamped proof of evidence integrity
Air-gapped integrity	Full cryptographic operations in SCIFs and classified enclaves	No network dependency; sneakernet-exportable commitment hashes
Authenticated messaging	Post-quantum sealed envelopes between personnel or systems	No intermediary can read or forge; recipient-only decryption

C. Architecture

Loggie is organized into five distinct architectural layers. Each layer has clearly defined responsibilities and trust assumptions.

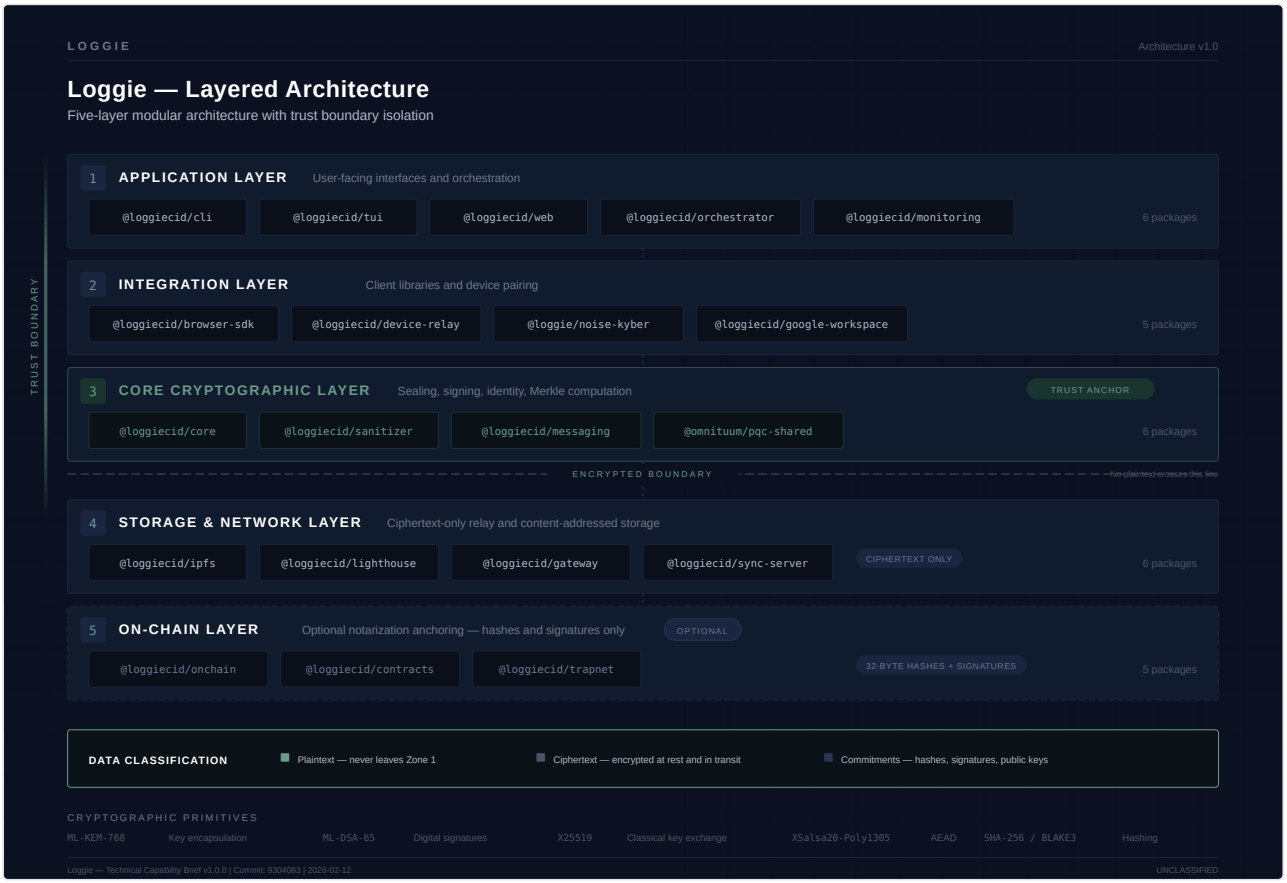

Figure 1: Loggie layered architecture with trust boundary indication.

Layer Descriptions

Layer 1 — Application Layer. Operational interfaces including a command-line tool, a terminal user interface, a web application, and an orchestrator for automated workflows. The CLI supports an 8-layer command structure mirroring the contract architecture (registry, factory, identity, revenue, token, search, developer, governance).

Layer 2 — Integration Layer. Client libraries for embedding Loggie capabilities into external applications. The browser integration provides a membrane pattern that normalizes identity shapes and isolates secret keys behind a keyring interface. Device relay and Noise-Kyber modules enable post-quantum secure device pairing.

Layer 3 — Core Cryptographic Layer. The cryptographic foundation implementing sealing/unsealing, key generation, digital signatures, identity hashing, and envelope construction. The sanitizer handles metadata stripping and Merkle tree computation. Shared cryptographic primitives are provided by the Omnituum PQC library.

Layer 4 — Storage and Network Layer. Integration with decentralized storage providers (IPFS, Filecoin, Lighthouse), self-hosted gateways, and sync servers. All data passing through this layer is encrypted; storage nodes handle ciphertext only.

Layer 5 — On-Chain Layer (Optional). Smart contract interactions for notarization anchoring, identity registration, and inbox management. Only cryptographic commitments (32-byte hashes, signatures) are submitted on-chain. This layer is entirely optional and can be disabled for air-gapped deployments.

Package Count by Layer

Layer	Packages	Core Components
Application	6	CLI, web interface, TUI, orchestrator, node dashboard, monitoring
Integration	5	Browser integration, device relay, Noise-Kyber, extension helpers, Google Workspace
Core Crypto	6	Core engine, sanitizer, messaging, config, system, PQC shared library
Storage / Network	6	IPFS, Lighthouse, Filecoin, gateway, sync server, GraphQL server
On-Chain	5	Notarization, contracts, testnet, ambient, RF vision

D. Deployment Models

Loggie supports three deployment models. Core cryptographic operations — sealing, signing, hashing, and verification — function identically across all three models with no feature degradation.

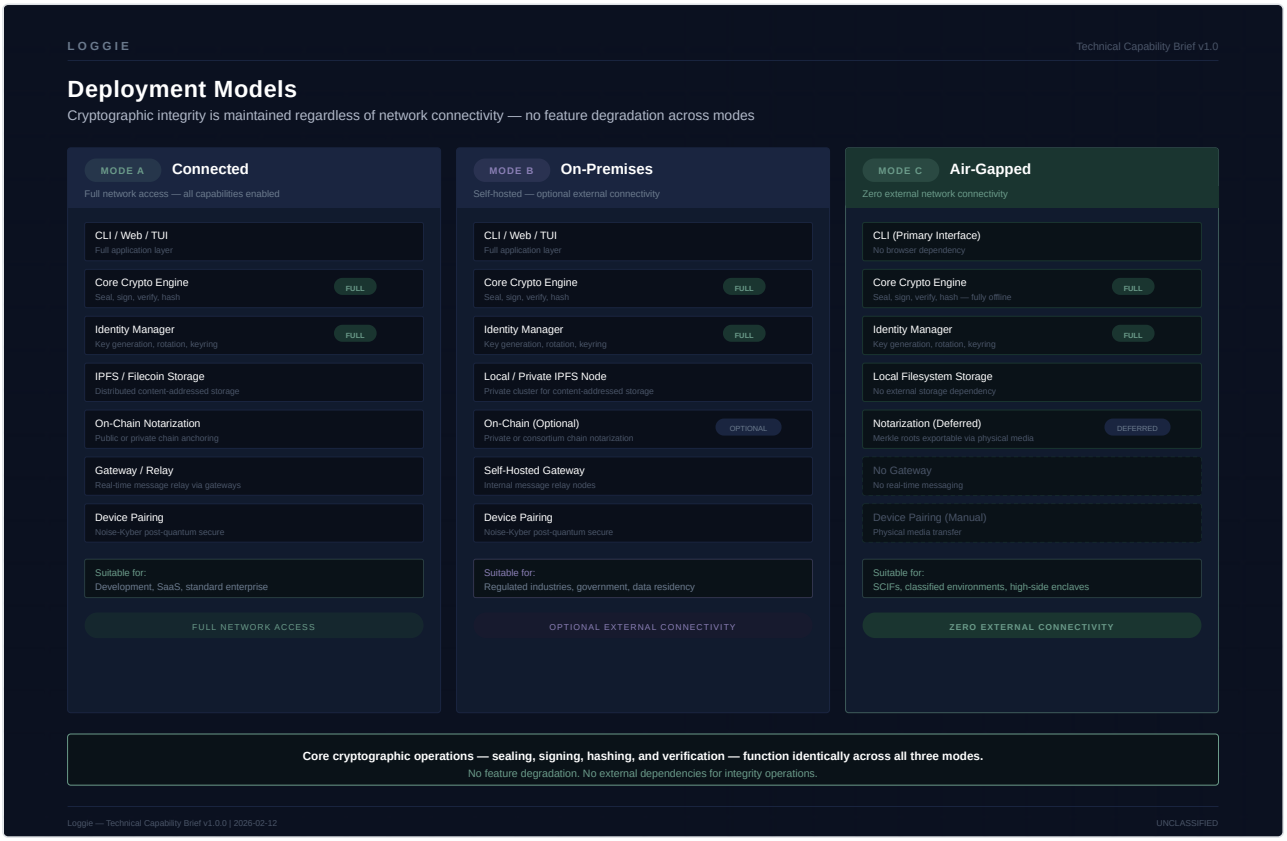

Figure 2: Deployment models. Cryptographic integrity is maintained regardless of network connectivity.

Mode A: Connected

Full network access with all capabilities enabled. Suitable for development, SaaS deployments, and standard enterprise use.

- Real-time message relay via gateways
- On-chain notarization with public or private networks
- IPFS/Filecoin distributed storage
- Device sync and pairing

Mode B: On-Premises

Self-hosted infrastructure with optional external connectivity. Suitable for regulated industries, government systems, and enterprise environments with data residency requirements.

- Private IPFS cluster for content-addressed storage
- Self-hosted gateway nodes for message relay
- Optional connection to private or consortium chain for notarization
- Full cryptographic operations locally

Mode C: Air-Gapped

Zero external network connectivity. Suitable for SCIFs, classified environments, and high-side enclaves.

- CLI-primary interface (no browser dependency)
- Local filesystem storage

- All cryptographic operations fully offline
- Merkle roots and commitment hashes exportable via physical media (sneakernet) for external verification or later anchoring
- No dependency on external key servers, certificate authorities, or network services

Deployment Decision Matrix

Requirement	Connected	On-Premises	Air-Gapped
Cryptographic sealing	Full	Full	Full
Digital signatures	Full	Full	Full
Integrity verification	Full	Full	Full
Real-time messaging	Yes	Self-hosted	No
On-chain notarization	Public chain	Private chain	Deferred
Content-addressed storage	IPFS/Filecoin	Private IPFS	Local FS
Device pairing	Noise-Kyber	Noise-Kyber	Manual

E. Data Flow and Trust Boundaries

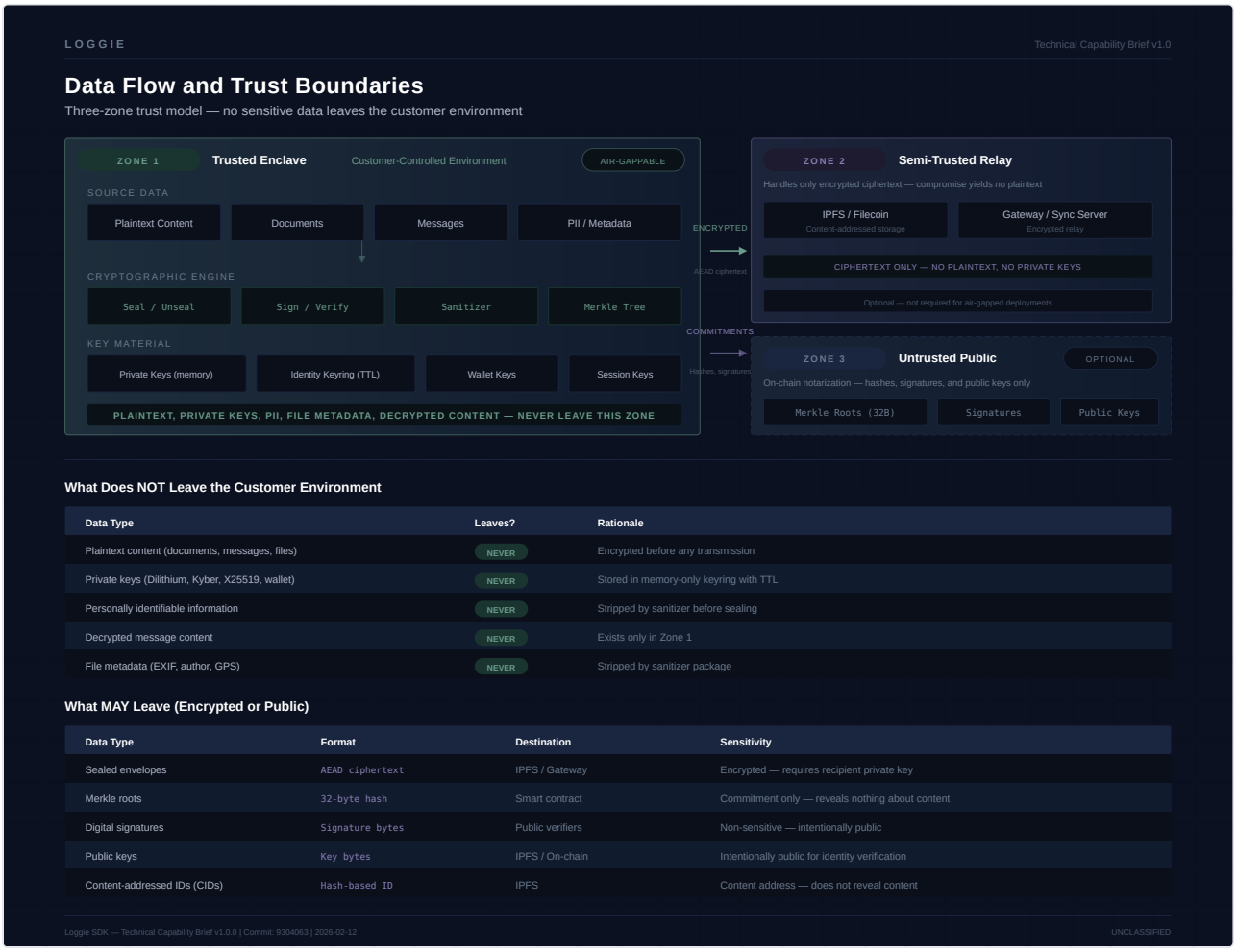

Figure 3: Data flow classification. No sensitive data leaves the customer environment.

Three-Zone Trust Model

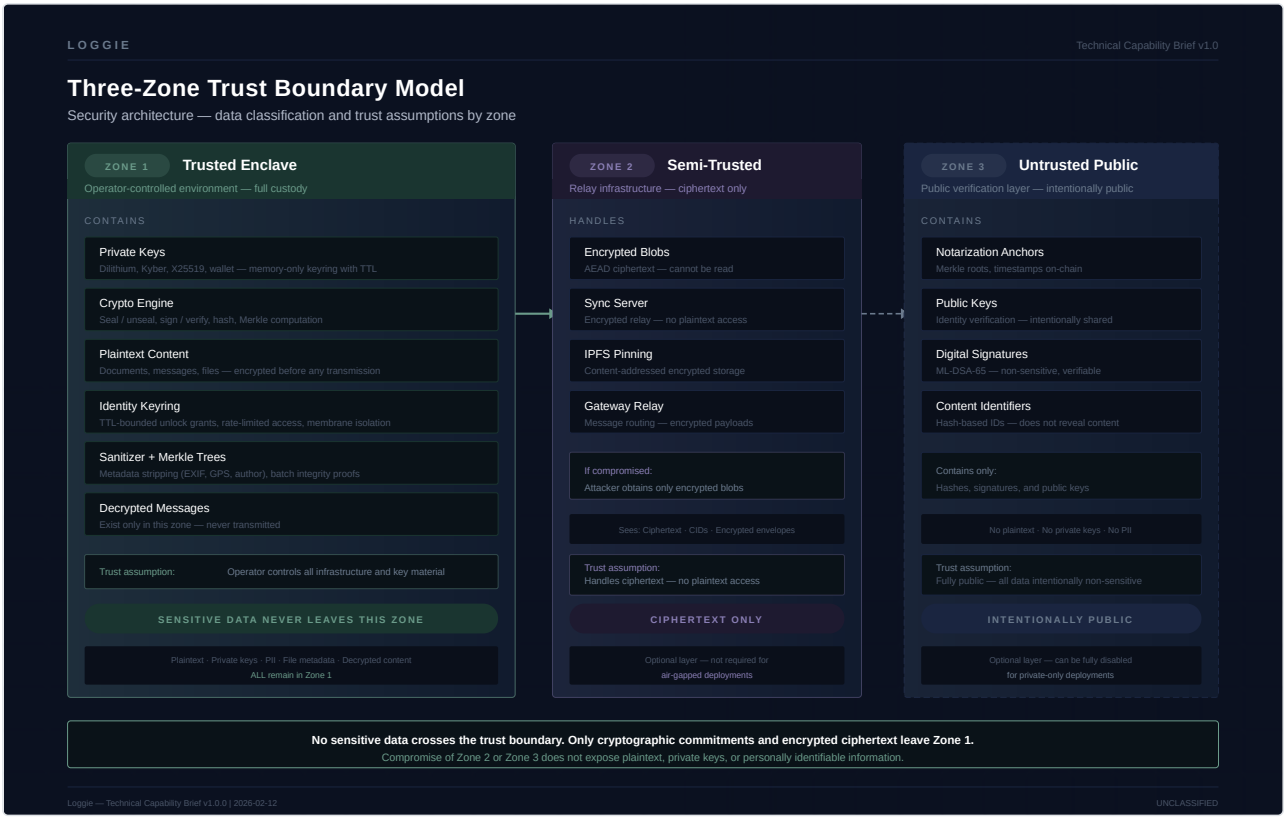

Figure 4: Three-zone trust architecture.

Zone 1 — Trusted Enclave (Customer-Controlled). Contains all sensitive material: private keys, plaintext content, decrypted messages, and the cryptographic engine. The operator has full control. This zone never transmits sensitive data.

Zone 2 — Semi-Trusted Relay. Optional relay infrastructure (gateways, sync servers, IPFS nodes) that handles only encrypted ciphertext. Compromise of Zone 2 does not expose plaintext or private keys.

Zone 3 — Untrusted Public. Public verification layer containing only hashes, signatures, and public keys. All data in Zone 3 is intentionally public and non-sensitive.

What Does NOT Leave the Customer Environment

Data Type	Leaves?	Rationale
Plaintext content (documents, messages, files)	**NEVER**	Encrypted before any transmission
Private keys (Dilithium, Kyber, X25519, wallet)	**NEVER**	Stored in memory-only keyring with TTL
Personally identifiable information	**NEVER**	Stripped by sanitizer before sealing
Decrypted message content	**NEVER**	Exists only in Zone 1
File metadata (EXIF, author, GPS, etc.)	**NEVER**	Stripped by sanitizer package

What MAY Leave (Encrypted or Public)

Data Type	Format	Destination	Sensitivity
Sealed envelopes	AEAD ciphertext	IPFS / Gateway	Encrypted; cannot be read without recipient's private key
Merkle roots	32-byte hash	Smart contract	Commitment only; reveals nothing about content
Digital signatures	Signature bytes	Public verifiers	Non-sensitive; intentionally public
Public keys	Key bytes	IPFS / On-chain	Intentionally public for identity verification
Content-addressed IDs (CIDs)	Hash-based ID	IPFS	Content address; does not reveal content

F. Security Posture Summary

Threat Model Considerations

- **Harvest-now-decrypt-later:** Mitigated by hybrid post-quantum encryption (ML-KEM-768 + X25519). Even if ciphertext is captured today, it resists decryption by future quantum computers.
- **Compromised relay/storage:** Gateway and IPFS nodes see only encrypted blobs. Compromise yields no plaintext.
- **Key exfiltration:** Private keys exist only in memory-only keyring with automatic TTL-based cleanup (5-minute default). Keys are never persisted to localStorage or sessionStorage.
- **Identity impersonation:** Identity documents are cryptographically signed. Verification uses ML-DSA-65 (post-quantum) signatures that can be independently validated.
- **Metadata leakage:** The sanitizer strips EXIF, author, GPS, and other metadata fields before content leaves the environment.
- **Man-in-the-middle:** Sealed envelopes include authenticated encryption (XSalsa20-Poly1305 AEAD). Tampering breaks the authentication tag.

Key Custody Model

Key Type	Algorithm	Storage Location	Persistence	Access Pattern
Wallet key	ECDSA (secp256k1)	Hardware wallet or encrypted store	Persistent	Transaction signing only
Dilithium signing key	ML-DSA-65	Encrypted local store	Persistent (encrypted)	Message signing
Kyber decryption key	ML-KEM-768	Encrypted local store	Persistent (encrypted)	Message decryption
X25519 key	Curve25519	Encrypted local store	Persistent (encrypted)	Classical key agreement
Session keys	XSalsa20-Poly1305	Memory only	Ephemeral	Per-message encryption

Identity Security Controls

The identity management system implements a **membrane architecture** (`docs/identity-membrane.md`) that enforces:

- Secret keys are never placed on identity objects passed to UI code
- A keyring abstraction mediates all secret key access
- Unlock grants are memory-only with 5-minute TTL and automatic cleanup
- Rate limiting on unlock attempts (5 attempts per 5-minute window)
- EIP-191 style unlock messages bound to identity CID, origin, and timestamp

Auditability

- All sealed envelopes carry version identifiers (`loggie.seal.v1.1`)
- Merkle trees use golden test vectors for regression testing
- Protocol invariants are documented and verified (`docs/protocol/merkle-shapes.md`)
- Anti-regression lint gates block dangerous patterns in CI

G. Cryptographic Primitives

All cryptographic primitives listed below are verified against source code. References point to actual implementation files.

Algorithms in Use

Primitive	Algorithm	Standard	Implementation	Purpose
Post-quantum KEM	ML-KEM-768 (Kyber)	NIST FIPS 203	`core/src/crypto/pqc/kyber.ts` via `@omnituum/pqc-shared`	Key encapsulation for message encryption
Post-quantum signature	ML-DSA-65 (Dilithium)	NIST FIPS 204	`core/src/crypto/pqc/dilithium.ts` via `@omnituum/pqc-shared`	Message and identity signing
Classical key agreement	X25519	RFC 7748	`core/src/crypto/x25519/` via `@noble/curves`	Classical key exchange (hybrid with Kyber)
Classical signature	Ed25519	RFC 8032	`core/src/crypto/sign.ts` via `@noble/curves`	Deterministic key derivation from seeds
Authenticated encryption	XSalsa20-Poly1305	NaCl secretbox	`core/src/crypto/nacl/seal.ts` via `@omnituum/pqc-shared`	Symmetric AEAD for message content
Authenticated encryption	AES-256-GCM	NIST SP 800-38D	`core/src/crypto/backup/encrypt.ts` via WebCrypto / Node.js `crypto`	Backup encryption (identity keys, mnemonics)
Authenticated encryption	XChaCha20-Poly1305	draft-irtf-cfrg-xchacha	`noise-kyber/src/noise.ts` via `@omnituum/pqc-shared`	Noise protocol handshake and device sync
Hash (content integrity)	SHA-256	FIPS 180-4	`core/src/crypto/primitives.ts` via `@omnituum/pqc-shared`	Leaf hashing in batch Merkle trees, identity hashing, general content hashing
Hash (Merkle nodes)	BLAKE3	BLAKE3 spec	`sanitizer/src/hash/merkle.ts` via `@omnituum/pqc-shared`	Internal node hashing in Merkle trees, folder integrity, Noise transcript
Hash (on-chain compat)	Keccak-256	Ethereum Yellow Paper	`onchain/src/files-inbox.ts` via `ethers`	Hash chain for on-chain file inbox
Key derivation	HKDF-SHA-256	RFC 5869	`core/src/crypto/primitives.ts` via `@omnituum/pqc-shared`	Deriving sub-keys from shared secrets, hybrid KDF mixing
Passphrase KDF (CLI)	scrypt	RFC 7914	`core/src/crypto/kdf-params.ts` via Node.js `crypto`	Memory-hard passphrase-to-key derivation for backups (N=16384, r=8, p=1)
Passphrase KDF (browser)	PBKDF2-SHA-256	NIST SP 800-132	`core/src/crypto/kdf-params.ts` via WebCrypto	Browser passphrase-to-key derivation for backups (100,000 iterations)
Wallet signatures	ECDSA (secp256k1)	SEC 2	`ethers` library, `@noble/secp256k1`	Ethereum wallet transaction signing
Noise protocol	Noise XX + ML-KEM	Noise Framework	`noise-kyber/` package	Post-quantum secure device pairing (`Noise_XX_25519+Kyber768_ChaChaPoly_BLAKE3`)

Sealed Envelope Cryptographic Structure (v1.1)

The `SealedEnvelopeV1_1` format (`core/src/types/seal-v1.1.ts`) specifies:

```
Envelope version:    loggie.seal.v1.1
KEM:                 ML-KEM-768 (Kyber768)
Signature:           ML-DSA-65 (Dilithium3)
AEAD:                XSalsa20-Poly1305
Hybrid mode:         Kyber + X25519 (dual key wrapping)
```

Each sealed envelope contains:

1. **Sender identity** with Dilithium public key

2. **Per-recipient key wraps** (both Kyber KEM ciphertext and X25519 ephemeral key)
3. **AEAD-encrypted content** (XSalsa20-Poly1305 with random 24-byte nonce)
4. **Dilithium signature** covering envelope + recipients + metadata
5. **Threading metadata** (message ID, thread ID, reply chain)

Merkle Tree Implementations

Loggie contains four distinct Merkle implementations (documented in `docs/protocol/merkle-shapes.md`):

Tree	Leaf Hash	Node Hash	On-Chain	Golden Vectors
BatchSigningMerkle	SHA-256(canonical JSON)	BLAKE3	Yes	`fixtures/merkle/batch-signing/`
FolderMerkle	BLAKE3(UTF-8(CID))	BLAKE3	Yes	`fixtures/merkle/folder/`
FilesInboxHashChain	keccak256	Sequential accumulator	Yes	Documented
ConversationIntegrity	SHA-256	SHA-256	No (local)	Advisory only

Library Dependencies for Cryptography

Library	Version	Provides
`@omnituum/pqc-shared`	0.3.0	Unified PQC layer: ML-KEM-768, ML-DSA-65, SHA-256, BLAKE3, HKDF, secretbox, XChaCha20-Poly1305
`@noble/post-quantum`	(via pqc-shared)	Dilithium and Kyber reference implementations
`@noble/curves`	~2.0.0	X25519, Ed25519, secp256k1
`@noble/secp256k1`	3.0.0	Pure JS secp256k1 ECDSA
`tweetnacl`	1.0.3	XSalsa20-Poly1305 secretbox
`kyber-crystals`	1.0.7	ML-KEM-768 (used in Noise protocol device pairing)
`ethers`	6.15.0	ECDSA wallet operations, keccak256, message hashing
`bls-eth-wasm`	1.4.0	BLS signature support
Node.js `crypto`	(built-in)	AES-256-GCM, scrypt, PBKDF2 (platform-specific paths)
WebCrypto	(built-in)	AES-256-GCM, PBKDF2 (browser-specific paths)

H. Integration Footprint

Deployment Packages

Loggie is published under the `@loggiecid` scope (core packages) and `@loggie` scope (extension packages). Deployment complexity scales with operational requirements.

Minimum deployment (cryptographic operations only):

```
@loggiecid/core          — Seal, unseal, sign, verify, identity, hash
@omnituum/pqc-shared     — Post-quantum primitives (transitive dependency)
```

Standard deployment (with browser-based operator interface):

```
@loggiecid/core          — Core cryptographic operations
@loggiecid/browser-sdk   — Browser identity management, keyring
@loggiecid/config        — Configuration management
```

Full deployment (all capabilities):

```
@loggiecid/core          — Core cryptographic engine
@loggiecid/browser-sdk   — Browser interface with identity membrane
@loggiecid/messaging     — Message handling
@loggiecid/sanitizer     — Metadata stripping, Merkle trees
@loggiecid/onchain       — On-chain notarization
@loggiecid/ipfs          — IPFS storage integration
@loggiecid/gateway       — Gateway relay
```

API Surface Summary

Module	Key Exports	Purpose
`core/seal`	`seal()`, `unseal()`	Encrypt and decrypt message envelopes
`core/crypto/pqc`	`dilithiumGenerateKeypair()`, `dilithiumSign()`, `dilithiumVerify()`	Post-quantum signatures
`core/crypto/pqc`	`generateKyberKeypair()`, `kyberEncapsulate()`, `kyberDecapsulate()`	Post-quantum key encapsulation
`core/identity`	`createIdentity()`, `verifyIdentity()`, `computeCanonicalHash()`	Identity lifecycle
`core/threading`	`extractThreads()`, `buildInboxView()`	Message thread management
`sanitizer`	`sanitize()`, `computeMerkleRoot()`, `generateProof()`	Content sanitization, Merkle proofs
`browser-sdk`	`getSelectedIdentity()`, `keyring`, `tryNormalizeIdentity()`	Browser identity membrane
`onchain`	`anchor()`, `verify()`	On-chain notarization

System Requirements

Requirement	Specification
Runtime	Node.js 18+ or modern browser (ES2020+)
Package manager	pnpm (workspace protocol)
TypeScript	5.0+
Network	Optional (air-gapped mode supported)
Hardware	No special requirements; post-quantum operations complete in <20ms
External services	None required; IPFS, chain endpoints are optional

I. Compliance Mapping

> **Disclaimer:** Loggie does not claim compliance certification. The following mapping identifies capabilities that align with specific control families. Actual compliance requires organizational policies, procedures, and independent assessment beyond the scope of this software.

NIST SP 800-171 Rev. 2 — CUI Protection Alignment

Control Family	Relevant Capability	Implementation Evidence	Notes
3.1 Access Control	Keyring with TTL-bounded unlock grants; rate-limited access	`core/src/identity/crypto-store.ts`, `browser-sdk/src/identity/keyring.ts`	Covers key access; organizational AC policies required
3.5 Identification and Authentication	Cryptographic identity with ML-DSA-65 signatures; identity hash verification	`core/src/identity/`, `core/src/crypto/pqc/dilithium.ts`	Self-sovereign identity model
3.8 Media Protection	Metadata sanitization; encrypted storage; no plaintext egress	`sanitizer/`, `core/src/seal.ts`	Covers data-at-rest and data-in-transit
3.13 System and Communications Protection	End-to-end encryption (hybrid PQ); authenticated encryption (AEAD); trust boundary enforcement	`core/src/seal-v1.1.ts`, `core/src/crypto/nacl/`	Covers SC-8, SC-12, SC-13 equivalents
3.14 System and Information Integrity	Merkle tree integrity proofs; golden test vectors; anti-regression CI gates	`sanitizer/src/hash/merkle.ts`, `docs/protocol/merkle-shapes.md`	Verifiable integrity chain

NIST SP 800-53 Rev. 5 — Control Family Mapping

Control	Description	Loggie Capability	Status
SC-8	Transmission Confidentiality	Sealed envelopes with hybrid PQ encryption	Implemented
SC-12	Cryptographic Key Establishment	ML-KEM-768 + X25519 hybrid key exchange	Implemented
SC-13	Cryptographic Protection	NIST-approved algorithms (FIPS 203, 204)	Implemented
SC-28	Protection of Information at Rest	Encrypted local storage, memory-only keyring	Implemented
SI-7	Software, Firmware, and Information Integrity	Merkle tree proofs, content hashing, sanitization	Implemented
IA-5	Authenticator Management	Key generation, rotation, TTL-based expiry	Implemented
AU-10	Non-Repudiation	Dilithium digital signatures on all sealed envelopes	Implemented

CMMC Level 2 Alignment

CMMC Practice	Loggie Relevance
AC.L2-3.1.1 Authorized access	Keyring membrane enforces access control to cryptographic material
SC.L2-3.13.1 Boundary protection	Three-zone trust model; no sensitive data crosses boundary
SC.L2-3.13.8 CUI encryption	Hybrid PQ encryption for all data in transit
SC.L2-3.13.11 CUI confidentiality	XSalsa20-Poly1305 AEAD; content encrypted at rest
SI.L2-3.14.1 Flaw remediation	Anti-regression CI gates; golden test vectors

Mapping Approach — Areas Requiring Organizational Completion

The following control areas are not addressable by software alone and require organizational policies and procedures:

- **TODO:** Physical security controls (PE family) — Require facility-level implementation
- **TODO:** Personnel security (PS family) — Require HR policies
- **TODO:** Incident response (IR family) — Require organizational IR plans; Loggie provides audit logging hooks
- **TODO:** Risk assessment (RA family) — Require organizational risk framework
- **TODO:** Security assessment (CA family) — Require third-party assessment of deployed system
- **TODO:** Configuration management (CM family) — Loggie provides deterministic builds; organizational CM policy required

J. Appendix

Version and Provenance

Field	Value
Brief version	2.0.0
Platform version	1.4.0
Repository	loggiecid-sdk
Git commit	`9304063`
Build date	2026-02-12
Build tool	`scripts/build-defense-brief.mjs` (Playwright PDF renderer)
Build command	`pnpm brief:build:defense`

Deterministic Build Notes

This document is generated deterministically from the source tree:

1. Markdown source at `docs/briefs/loggie-defense-capability-brief.md`
2. SVG diagrams at `docs/briefs/assets/`
3. Git commit hash and build timestamp injected at build time
4. PDF rendered via Playwright (Chromium headless) for consistent typography
5. Output placed at `dist/briefs/loggie-defense-capability-brief.pdf`

Rebuilding from the same commit with `pnpm brief:build:defense` produces a document with identical content (timestamps will differ).

Source File References

Cryptographic primitive implementations referenced in this document:

Reference	File Path
Dilithium (ML-DSA-65)	`packages/core/src/crypto/pqc/dilithium.ts`
Kyber (ML-KEM-768)	`packages/core/src/crypto/pqc/kyber.ts`
X25519 key agreement	`packages/core/src/crypto/x25519/`
NaCl secretbox (XSalsa20-Poly1305)	`packages/core/src/crypto/nacl/seal.ts`
SHA-256 / HKDF-SHA-256	`packages/core/src/crypto/primitives.ts`
Merkle trees (BLAKE3 / SHA-256)	`packages/sanitizer/src/hash/merkle.ts`
Sealed envelope v1.1	`packages/core/src/seal-v1.1.ts`
Envelope types	`packages/core/src/types/seal-v1.1.ts`
Identity canonical hash	`packages/core/src/identity/canonical.ts`
Identity membrane	`packages/browser-sdk/src/identity/keyring.ts`
Merkle shape documentation	`docs/protocol/merkle-shapes.md`
Golden test vectors	`packages/cli/fixtures/merkle/`
Noise-Kyber device pairing	`packages/noise-kyber/`

Glossary

Term	Definition
AEAD	Authenticated Encryption with Associated Data
CID	Content Identifier (IPFS content-addressed hash)
CMMC	Cybersecurity Maturity Model Certification
CUI	Controlled Unclassified Information
HKDF	HMAC-based Key Derivation Function
ISSM	Information System Security Manager
KEM	Key Encapsulation Mechanism
Merkle root	Single hash representing the integrity of an ordered set of items
ML-DSA	Module-Lattice Digital Signature Algorithm (NIST FIPS 204; formerly Dilithium)
ML-KEM	Module-Lattice Key Encapsulation Mechanism (NIST FIPS 203; formerly Kyber)
PQ / PQC	Post-Quantum Cryptography
SCIF	Sensitive Compartmented Information Facility
Sealed envelope	Encrypted, signed message container (Loggie-specific format)
XSalsa20-Poly1305	Stream cipher + MAC combination used in NaCl/libsodium secretbox

Companion Documents

This brief is part of a coordinated collateral stack for defense engagement:

Document	Purpose	Audience
Loggie Defense Relevance Brief	Operational use case articulation across four domains	Program managers, mission owners
Loggie Security FAQ	Conversational security assurance addressing contractor concerns	ISSMs, security reviewers
Loggie CMMC Pilot SOW	90-day scoped proof-of-concept deployment	Contracting officers, program leads
Loggie Architecture Diagram	Two-zone deployment architecture (enclave + optional notarization)	Technical evaluators, system architects
Loggie Defense Pitch Deck	Visual overview for first-meeting context setting	All stakeholders
This Technical Capability Brief	Full technical depth with compliance mapping	Technical evaluators, security reviewers
Executive Capability Brief (8-page)	Trimmed version for compliance buyer first handoff	Compliance leads, decision makers

---

End of Technical Capability Brief