BMB NEXUS · GENESIS PLATFORM

AETHERION: A Security Architecture for Local-First Software

Local-first delivery is the cheapest way to ship software and the hardest to protect. AETHERION is the four-layer composition that makes it commercially survivable — entitlement without a server, proprietary code on hardware you do not control, user data you are liable for but cannot reach, and collaboration that does not drag you back into hosting. A reference architecture, with a 650,000-line production system as the proof.

Technical Specification · BMB Nexus
Updated 2026-08-04 · 9 pages
https://genesis.bmbnexus.ai/whitepapers/aetherion

Security Architecture · Local-First · Reference Architecture · Post-Quantum · Software Economics · Threat Modelling

AETHERION: A Security Architecture for Local-First Software

Technical Whitepaper v1.0

Four layers that make shipping to hardware you do not control commercially survivable.

BMB Nexus · Genesis Platform
Specification v1.0 · 2026-08-04 · genesis.bmbnexus.ai/whitepapers/aetherion
Security correspondence: care@bmbnexus.ai

Abstract

Local-first delivery — software that runs entirely on the customer's own machine — has the best economics in the industry and is used by almost nobody who could benefit from it. The reason is not technical difficulty. It is that local delivery removes four protections that server hosting provides implicitly, and each removal is independently sufficient to stop a vendor from shipping. **AETHERION is the composition that restores all four.** It comprises FORTRESS (runtime secrets and data at rest), VEIL (executable integrity on an untrusted host), SERAPHUM (cryptographic entitlement binding with no server in the trust path), and AETHER (post-quantum communications through a relay that holds no keys). This document specifies the architecture as a **reference design rather than a product description**: the blockers it removes, the principles that make the four layers compose rather than merely coexist, the interlocks between them, and the boundary of the whole. A production system of approximately 650,000 shipped lines serves as the reference implementation. As with each layer's own specification, we state precisely what the architecture does not claim — including the one gap that matters most to anyone evaluating it commercially.

Keywords: Local-First Software, Security Architecture, Reference Architecture, Untrusted Host, Entitlement Binding, Post-Quantum Cryptography, Defence in Depth, Software Economics, Fail-Closed Design

---

1. The Local-First Paradox

Consider the economics of shipping software that runs on the customer's machine.

There is no per-user compute cost. There is no scaling curve — the ten-thousandth customer costs the same to serve as the first, which is to say nothing. There is no infrastructure bill that grows with success, no capacity planning, and no single outage that takes every customer offline at once. Margin does not erode as the business grows; it is fixed at the moment of sale.

The customer's position is equally good. Their data never leaves their premises, which resolves most of the compliance conversation before it starts. Latency is local. And the software does not stop working if the vendor is acquired, changes direction, or ceases to exist — **the customer's continued use does not depend on the vendor's**

continued existence.

On both sides of the transaction this is the superior arrangement. It is also rare. The question worth asking is why.

1.1 What hosting silently provides. A server-hosted vendor controls the runtime absolutely. The customer never possesses the software; they possess a session. From that single fact, four protections follow for free — and a vendor who has never shipped locally may not realise they were relying on any of them:

- **Entitlement is trivially enforced.** Access is a database row. There is nothing on the customer's side to modify.
- **Proprietary implementation is never distributed.** The customer receives outputs, never the system that produced them.
- **Data protection is under vendor control.** The vendor is liable for customer data and can actually reach it.
- **Collaboration is native.** Everyone is already connected to the same infrastructure.

1.2 What local delivery removes. Ship the same product locally and all four protections vanish simultaneously. Entitlement logic is now code the customer controls. Proprietary implementation sits on their disk. The vendor remains liable for data on a machine they cannot reach. And any collaborative feature appears to require exactly the infrastructure the model was meant to eliminate.

Each of those is independently sufficient to kill the decision. This is the paradox: the cheapest and most customer-favourable way to ship software is blocked, and it is blocked by security architecture rather than by economics or engineering difficulty.

AETHERION exists to remove all four blockers at once. Removing three is not a partial solution — a vendor whose entitlement can be patched out will not ship, regardless of how well the other three are handled.

---

2. The Four Blockers, Mapped

Each blocker corresponds to one layer. The mapping is deliberate and complete.

Blocker	Layer	Mechanism
Entitlement can be modified — no authority to check against	**SERAPHUM**	Lattice key-encapsulation binding entitlement to hardware, with no server in the trust path
Proprietary implementation sits on the customer's disk	**VEIL**	Sealed-build transformation plus binary-level capability removal
Liable for data on a machine you cannot reach	**FORTRESS**	Authenticated encryption at rest, zero-knowledge authentication, live integrity monitoring
Collaboration drags you back into hosting	**AETHER**	Post-quantum sealed transport through a relay that holds no keys and no identity

The fourth deserves emphasis because it is the one most often conceded. Vendors who successfully ship locally frequently reintroduce a central server the moment customers ask for collaboration — and in doing so they rebuild every dependency they had eliminated, because the coordination layer becomes load-bearing. AETHER's relay forwards sealed material it cannot read and holds no key that would let it. **Going multiplayer therefore does not cost the sovereignty the model was built on.**

---

3. Principles

Four independent security systems on one machine are not an architecture; they are four systems on one machine. What makes AETHERION compose is a small set of principles every layer obeys. These are the substance of this document — the individual mechanisms are specified in the individual papers.

3.1 Authorisation to use is not authorisation to dismantle. The central principle, and the one most often collapsed. A user may be fully authorised to operate the software and to access their own data through supported interfaces, while remaining unauthorised — architecturally, not merely by policy — to extract raw key material, instrument the process, read protected implementation, or modify the security boundary. **Protecting the user's data and protecting the system's internals are different promises**, and a coherent architecture keeps both without either implying the other.

> The system trusts the operator to use it. It does not trust any process — including the operator's — to dismantle it.

3.2 Fail closed on integrity; never on availability. An integrity failure must terminate the affected operation rather than degrade to a permissive path. An *availability* failure must do the opposite. A vendor endpoint being unreachable is not evidence that anything is wrong with a customer's install, and treating it as such produces software that disables itself during the vendor's outage — precisely the dependency local-first delivery exists to remove. Conflating these two is, in our assessment, the most common architectural error in licensed local software.

3.3 Ambiguity is never denial. The corollary. A timeout, a network fault, or an unparseable response is an inconclusive result. Only an explicit, authenticated rejection may alter entitlement state. We state this as a principle because we violated it once: a revision of the entitlement path treated an inconclusive result as a denial, and a transient network fault could clear a legitimate activation. A paying customer being locked out by a packet loss is a security failure, not a security success.

3.4 No layer claims another layer's boundary. Each layer states its own limits precisely and names the layer that handles what it does not. A sealing layer that implied it defeated runtime tampering, or a code-integrity layer that implied it protected data at rest, would produce a documentation set in which every layer is secure and no reader can determine what actually guards what. Precision about limits is what makes composition legible.

3.5 Prefer claims a third party can check. Security architecture is unusually prone to unfalsifiable assertion. Wherever a property can be made externally verifiable it should be, and that property should be stated prominently — see §5.3.

3.6 Never report a success you cannot honour. An interface that returns success while failing to perform its function is the most dangerous defect a security system can have, because nothing fails, nothing logs, and trust is fully placed. We hold this as a principle because we shipped an instance of it internally: a sealing routine reported success while producing output it had not sealed. It was caught and corrected before release, and it is the reason every claim across this document set was checked against the implementation before being written.

---

4. The Four Layers

Each layer has its own specification. This section states role and boundary only.

4.1 FORTRESS — The Vault. Runtime secrets, protected state, and user-owned data at rest. Authenticated encryption, zero-knowledge password verification (the password itself is never stored), key material held guarded in memory and destroyed after use, threshold secret sharing with tamper-evident shares, decoy states that convert silent offline probing into a detected event, and a live coherence heartbeat that collapses key material on accumulated anomaly. FORTRESS also owns runtime environment inspection for the platform. → Specification

4.2 VEIL — The Skin. Executable identity and integrity on hardware the operator fully controls. Sealed-build transformation to encrypted bytecode, binary-level removal of the standard instrumentation entry points, disclosure boundaries preventing the system from revealing its own implementation, and fail-closed loading. VEIL does not perform continuous runtime attestation and does not claim to. → Specification

4.3 SERAPHUM — The Seal. Cryptographic binding of entitlement, installation, and machine, with no server in the trust path. A module-lattice key-encapsulation mechanism at NIST Category 1 parameters seals per-install material; the platform's Category 5 post-quantum primitives are specified in the same document. Superseded build attestations remain valid indefinitely, so a customer is never stranded by the vendor's release cadence. → Specification

4.4 AETHER — The Voice. Everything that leaves the endpoint. Hybrid ML-KEM-1024 ⊗ X25519 confidentiality and hybrid Ed25519 ⊕ ML-DSA-87 authentication — NIST Category 5 on both axes — with a continuous ratchet that folds a fresh lattice secret into every step. Endpoints are outbound-only and never open a listening port; the relay forwards opaque ciphertext and holds no key. → Specification

---

5. Composition

The layers are not independent products bundled together. Each is load-bearing for at least one other, and the interlocks are where the architecture's actual strength lives.

5.1 SERAPHUM ⇄ VEIL. Cryptographic entitlement binding inside an editable artefact is theatre — an attacker edits the branch that consults it. A sealed artefact with no entitlement binding grants everyone everything. **Neither layer is meaningful alone; together they make entitlement both cryptographic and non-trivially editable.**

5.2 VEIL ⇄ FORTRESS. The distributed artefact is re-keyed at first unlock to material derived within FORTRESS, so the shipped artefact and the key that opens it thereafter are not the same thing. Possessing the distribution is not sufficient. Conversely, FORTRESS's protections would be moot if the code enforcing them could be freely modified — VEIL is what makes FORTRESS's controls durable rather than advisory.

5.3 The externally verifiable property. One AETHERION property can be checked by a third party on a shipped binary with no vendor cooperation: the **fuse state** of the runtime, readable with the standard published tooling. An evaluator can confirm that runtime option injection, alternative execution modes, and remote debugging are disabled, and that package integrity is enforced — in about a minute, without trusting a sentence of this document. In a field where anti-tamper claims are almost universally unfalsifiable, we regard this as the single most important sentence available to us, and we would rather a reader start there than with any architectural diagram.

5.4 AETHER ⇄ the endpoint layers. End-to-end encryption is only as meaningful as the integrity of the endpoints. AETHER's guarantees presuppose that the endpoint performing the sealing is the endpoint the vendor shipped — which is VEIL's function — and that the keys it uses are protected at rest, which is FORTRESS's. A messaging layer atop an unprotected endpoint secures the wire and nothing else.

---

6. What the Architecture Costs

Security architectures are usually presented without their bill. Ours has one.

6.1 A one-time cost at boot, not a continuous tax. Key derivation and artefact decryption occur at startup. They are measurable, they are bounded, and they do not recur while the software runs. It is worth separating this cleanly from continuous overhead, because the two are routinely conflated — including by us. Runtime heaviness in our own product was attributed to encryption for months before measurement identified an unrelated rendering cost as the actual source. **Ask of any performance claim: is this a one-time spike or a continuous tax?** Cryptography is usually the former, and it is a convenient thing to blame for the latter.

6.2 Operational discipline. A sealed build is not a build configuration; it is a pipeline with failure modes of its own. An asset that does not survive transformation is absent at runtime with no compile-time warning. We have shipped releases that passed continuous integration while a required runtime asset had been removed by an unrelated cleanup step. The mitigation is not cleverness — it is extracting the distributed package and inspecting it before release, every release. **A green build is a claim; an extracted package is evidence.**

6.3 Loss of central remediation. This is the real trade, and it should be stated rather than buried. A hosted vendor can patch every customer in minutes. A local-first vendor cannot. The architecture must therefore be conservative by default, fail closed on integrity, and never require a network round trip to keep working. **You exchange the ability to fix everything instantly for the property that nothing breaks when you disappear.** That is a genuine trade, and it favours the customer.

---

7. Reference Implementation

AETHERION is a reference architecture, but it is not a proposal. It is described from a system in production.

The reference implementation is ARIA, a local-first cognitive platform of approximately **650,000 shipped lines** across roughly 1,500 source files, distributed as a desktop binary. Every layer described here is applied to it: the runtime ships as encrypted bytecode with **exactly one readable source file**; entitlement is sealed cryptographically against the machine; user data resides in authenticated-encrypted local databases with checksum verification on every write; and peer-to-peer collaboration runs through a relay that cannot read what it forwards.

Two properties are worth citing because they are unusual and measurable. The system performs semantic retrieval over the user's own data with **zero external API calls for embeddings** — the model runs locally, so no query text leaves the machine even for indexing. And all encrypted domain databases can be exported to plain, portable formats with a raw key and no dependence on the security layer: **the escape hatch is deliberate, because "owned" without the ability to leave is a slogan.**

The point of citing a reference implementation is falsifiability. An architecture that has never survived contact with a shipped product, real customers, and its own failure modes is a proposal. §6.2 exists because ours did not survive contact cleanly, and we would rather publish the resulting discipline than the impression that it went smoothly.

---

8. Trust Boundary

AETHERION holds under the following conditions.

- **Secrets remain on the machine that generated them.** There is no escrow. The vendor cannot recover material it never held, which is a limitation and is intended as one.
- **Vendor infrastructure is not trusted.** Relays forward ciphertext they cannot open. Infrastructure is a router, not a party.
- **The network is assumed hostile.** Transport security is hygiene, not a boundary.
- **Vendor availability is not required.** Unreachable endpoints degrade to deferred state, never to denial. The software continues to function.
- **The operator is trusted to use, not to dismantle** (§3.1).

---

9. What AETHERION Does Not Claim

9.1 It does not claim that extraction is impossible. Client-side protection is properly described as resistance, detection, and rapid termination — never mathematical impossibility. An adversary with full machine control, sufficient expertise, and unlimited time is not provably excluded. Any local-first architecture claiming otherwise is misrepresenting the problem.

9.2 It does not defend a machine already compromised at runtime. An adversary with privileged execution inside the process while it is legitimately unlocked is inside the boundary.

9.3 It does not protect against the operator's own decisions. Disclosed credentials and authorised-but-misunderstood actions defeat the architecture by legitimate means.

9.4 Post-quantum protection is not uniform across layers, and the difference is deliberate. Communications operate at **NIST Category 5** on both confidentiality and authentication. The entitlement seal operates at **Category 1**. The reasoning is that the seal protects a machine-scoped binding containing no user content, while communications carry user content exposed to harvest-now-decrypt-later recording. **A single "post-quantum, Level 5" claim across the whole platform would be inaccurate**, and we state the asymmetry here so that no reader has to discover it themselves. Migration of the seal to Category 5 parameters is under evaluation.

9.5 It has not undergone independent third-party audit. This is the most important disclosure in the document and the one an evaluator should weigh most heavily. Individual layers are validated against official test vectors where applicable, adversarially tested internally, and one property is externally verifiable (§5.3). None of that is equivalent to independent review. We treat the absence of an audit as the gating item for any commercial adoption of this architecture beyond our own product, and we consider a vendor who claims otherwise to be telling you something useful about their standards.

---

10. Provenance and Attribution

AETHERION is designed, implemented, and specified by **BMB Nexus**. The four layer specifications, the reference implementation, and the cryptographic implementations described in them are our own work, developed from published standards rather than adapted from existing products.

10.1 Why attribution matters for a security layer specifically. A reader evaluating a security architecture is entitled to know who stands behind its claims and where to send a finding. An unattributed security document is not a neutral one — it is a document with nobody accountable for it. Every specification in this set carries its publisher, its version, its canonical location, and a security correspondence address, and our disclosure contact is published at `/.well-known/security.txt` per RFC 9116.

10.2 The mark. Where AETHERION protects a product, the appropriate attribution is **“Secured by AETHERION”**. The architecture name is what carries meaning to an end user; it identifies the specific protections described in these documents, which they can read and evaluate. Naming the implementing organisation instead would tell that user about a company rather than about the guarantees they are being offered.

10.3 What the mark asserts. It asserts that the four layers are present and composed as specified here — not merely that some encryption is in use. A product employing one layer in isolation is not AETHERION-protected, and the distinction matters: as §1 and §5 argue, the architecture's value lies in the composition. Three layers out of four leaves a blocker standing.

---

11. Status

Deployed. All four layers are in production in the reference implementation and applied to every shipped release.

Specified. Each layer has a public specification stating its own mechanisms, validation, and limits. This document specifies only their composition.

Open. Independent third-party review (§9.5); evaluation of Category 5 parameters for the entitlement seal (§9.4).

On how this document is written. Every claim was checked against the implementation before it was written down. Where a mechanism lives in one layer, it is credited to that layer rather than to the architecture generally. Where a protection is build-time rather than continuous, it is described as build-time. Claims that could not be verified were removed rather than softened — including several that would have read well.

Appendix: The Four Layers

AETHERION is four layers with four responsibilities and four specifications. Each states its own boundary so that no layer makes vague claims on behalf of the others.

LAYER SPECIFICATIONS

FORTRESS — The Vault

Runtime secrets, protected state, user-owned data at rest. Owns runtime environment inspection.

VEIL — The Skin

Executable identity and integrity on hardware the operator fully controls.

SERAPHUM — The Seal

Binds entitlement, installation, and machine cryptographically, with no server in the trust path.

AETHER — The Voice

Every message, call, and file that leaves the endpoint, sealed post-quantum on both axes.

This document specifies composition only. For mechanisms, parameters, validation regimes, and per-layer limits, read the layer specifications above — each is written to be evaluated on its own terms.

Cite as: BMB Nexus. “AETHERION: A Security Architecture for Local-First Software.” Genesis Platform Technical Specification v1.0, 4 August 2026. https://genesis.bmbnexus.ai/whitepapers/aetherion