

Ghostdrive: A Peer-to-Peer Private Memory Network

Ghostdrive contributors · ghostdrive.com

October 2026 · version 2.0 (supersedes v1.0, July 2026, and v1.1, October 2026)

Abstract. A person's memory — notes, files, conversations and the context their AI works from — should belong to that person and no one else, yet today it lives in plaintext on servers whose operators can read it, lose it or train on it. We propose a system in which every byte is sealed on the owner's device before it exists anywhere else, and in which a network of untrusted peers stores, relays and proves possession of data it can never read. Keys descend from a 256-bit root that the owner holds as 24 words; files become content-addressed ciphertext blocks that are byte-compatible with IPFS; structured data lives in Automerge documents synchronised as sealed blocks between the owner's devices; and messages between people use a post-quantum hybrid key encapsulation. Durability is purchased through storage deals whose proofs any stranger can verify against a public Merkle root, with Reed–Solomon erasure coding across providers. A recursive zero-knowledge registry gives constant-size, independently verifiable checkpoints for curated data and its lineage. On top of this substrate runs a personal agent that reads only what its owner allows, acts only through explicit approvals with four hard stops no setting can remove, and signs a receipt for every action it takes. Every figure in this paper is reproduced by one command in the open-source implementation.

1. Introduction

Memory has become the most valuable input of personal computing. The assistants people now rely on are only as useful as the context they can see, and that context — mail, calendars, documents, years of notes — is precisely what people should least want to hand to an operator. The prevailing architecture nevertheless stores it in plaintext on someone else's computer. The operator can read it, its terms may permit training on it, and its continued existence depends on a single company's solvency and goodwill.

Decentralised storage networks address availability and censorship, but not confidentiality. Content on IPFS is public by construction; Filecoin proves that some bytes are stored, not that only their owner can read them; Arweave's permanence is incompatible with erasure. End-to-end encrypted cloud drives keep content private but still route every byte through one company and offer no way to verify that data is actually held.

What is needed is a system in which (a) no party other than the owner can read the data, (b) any third party can still verify integrity and storage, (c) the owner's devices stay authoritative and fully functional offline, and (d) software acting on the owner's behalf — increasingly an AI agent — is bound by rules the owner can audit. In this paper we describe such a system. It separates *confidentiality* (client-side encryption under keys derived from the owner's root), *integrity and availability* (content addressing, publicly verifiable storage proofs and erasure coding), *coordination* (CRDT documents synchronised as sealed blocks), and *agency* (an approval layer with signed receipts).

2. Design Overview

Ghostdrive is organised as six layers with strictly downward dependencies (Figure 1). A single device with no network is already a complete encrypted vault; each layer above adds durability, sharing or automation, and degrades gracefully when peers are absent.

L6 Agent	Your Ghost · approvals · hard stops · signed receipts · MCP
L5 Registry	recursive zk proof · evaluator quorum · Base anchor
L4 Durability	segment-Merkle proofs · escrow · Reed-Solomon
L3 Network	libp2p · Noise · DHT · WebRTC · relay · mailboxes
L2 Documents	Automerge CRDTs as sealed blocks · device sync
L1 Data	root · key ring · AES-256-GCM · CIDv1 blocks

Figure 1: Layered architecture. Trust decreases upward: L1 requires trusting only local cryptography; no layer above can read what L1 has sealed.

The design maintains four invariants.

- **I1, zero plaintext egress.** Bytes leave a device only as AES-256-GCM ciphertext under keys derived from the owner's root; keys are never transmitted except sealed to another of the owner's devices or to an invited member.
- **I2, verify, don't trust.** Every block received from any source is re-hashed against its content identifier before use. A malicious peer can waste bandwidth but cannot inject data.
- **I3, the device is the authority.** Every operation completes against local state; synchronisation merges later without a coordinator.
- **I4, no action without consent.** Data reaches a model outside the device only under an explicit grant, and an agent acts in the world only through an approval whose use is recorded in a signed receipt.

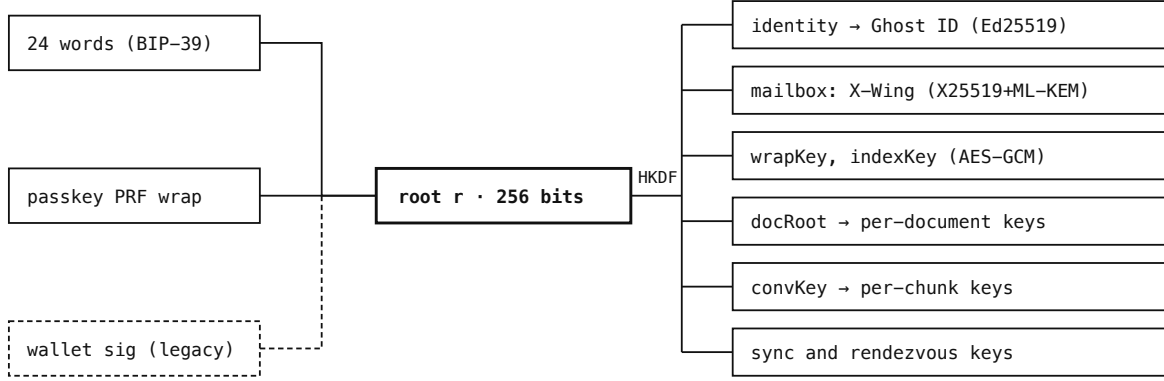
3. Keys and Identity

A vault begins as 32 random bytes, the *root* r . The owner holds r as a BIP-39 phrase of 24 English words (256 bits of entropy plus an 8-bit checksum), which restores the vault on any device. A device may additionally keep r wrapped under a passkey: the WebAuthn PRF extension yields a per-passkey secret that never leaves the authenticator without user verification, from which an AES-GCM key wraps the root. A stolen wrapped root is useless without the passkey. Earlier vaults derived the root from a wallet signature; a rekey procedure moves such vaults to a random root (§10).

All working keys descend from the root by HKDF-SHA256 under a fixed protocol salt and a distinct label per purpose:

$$K_x = \text{HKDF-SHA256}(r, \text{salt} = \text{"drive369/v3"}, \text{info} = x) \quad (1)$$

The salt is fixed rather than bound to a web origin, so moving the application to another domain never changes a key (a defect of v1.0). Figure 2 shows the resulting hierarchy. The *Ghost ID* is the public half of an Ed25519 key derived from r : the same on every device that opens the vault, issued by no server, encoded as 52 base32 characters. The *mailbox key* is an X-Wing key pair (X25519 combined with ML-KEM-768), so that messages stay private unless both the classical and the post-quantum component are broken (§6.3).



Derivation is one-way: nothing on the right reveals anything to its left. The dashed path is the legacy wallet root, moved by rekey.

Figure 2: Key hierarchy. The 24 words and an optional passkey both reach the same root; every working key is derived from it, so all of a person's devices compute identical keys without exchanging them.

4. Encrypted Content Addressing

4.1 Chunking and IPFS-native blocks

Files are split by a Gear rolling hash with FastCDC-style normalisation: minimum 32 KiB, target 128 KiB, maximum 256 KiB minus 28 bytes, the 28 bytes leaving room for the IV and GCM tag. Because boundaries depend on content rather than offsets, an insertion re-chunks only its neighbourhood. Each chunk m_i is encrypted and the ciphertext addressed by its hash, so every block is a standard raw IPFS leaf: any IPFS node, gateway or pinning service can store, serve and verify Ghostdrive blocks under the same identifier without knowing that they are encrypted.

4.2 Convergent keys and deterministic IVs

Under the default convergent policy each chunk receives its own key, and the GCM nonce is derived from the chunk rather than drawn at random:

$$k_i = \text{HMAC}(K_{\text{conv}}, \text{"chunk:"} \parallel \text{SHA-256}(m_i)) \quad (2)$$

$$\text{IV}_i = \text{HMAC}(\text{ivKey}(k_i), \text{SHA-256}(m_i))_{[0..12)} \quad (3)$$

$$c_i = \text{AES-256-GCM}_{\text{enc}(k_i)}(\text{IV}_i, m_i), \quad \text{CID}_i = \text{CIDv1}(\text{raw}, \text{SHA-256}(c_i)) \quad (4)$$

Identical chunks of one owner therefore encrypt to identical blocks in any file or version, which is what deduplicates versions (4.37× over ten edited versions of a document, §12). A given key–IV pair never encrypts different plaintext, so GCM's nonce requirement holds. Owners hold different K_{conv} , so nothing is shared or confirmable across owners. A *random* policy — a fresh key per file and no deduplication — is available and is the one to use for data that must be erasable: convergent keys can always be re-derived from the root, random keys can be destroyed.

4.3 Manifests and capabilities

A file's manifest lists its chunk identifiers, sizes and keys, together with its name, type and plaintext hash; above 2,000 chunks the list moves into sealed chunk-list blocks, allowing files of roughly 500 GB. The manifest is itself an encrypted block, so storage nodes see only opaque blocks of similar size (Figure 3). Sharing is a capability string `drive369:v3:<manifestCID>:<key>`: whoever holds it can read that file and nothing else. Reading verifies every block (I2) and then the plaintext hash.

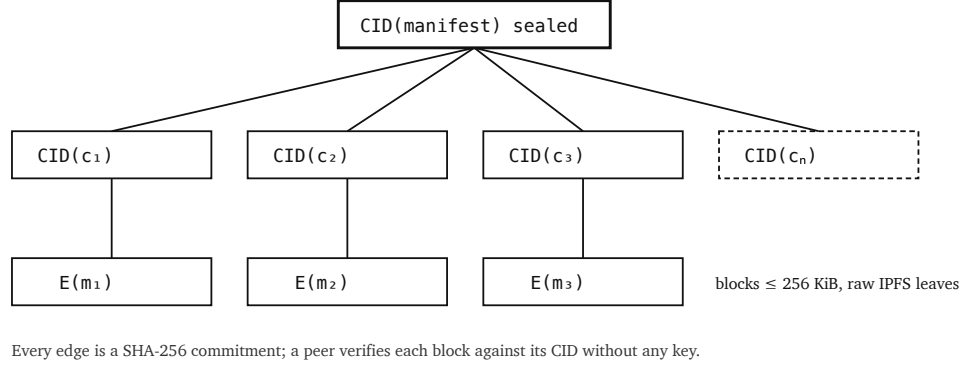


Figure 3: Sealing and content addressing. Leaves are ciphertext blocks; the manifest that links them is itself sealed, so the network learns neither content nor file structure.

5. Documents and Synchronisation

Structured data — profiles, notes, threads, goals, settings, the agent's state — lives in Automerge documents. A change is serialised incrementally, sealed under a per-document key and stored as an ordinary block; every N changes a full snapshot replaces the incremental chain. The per-document key is

$$\text{key}_d = \text{HMAC}(K_{\text{docRoot}}, \text{"doc:"} \| d) \quad (5)$$

so each document's blocks have their own key and any peer may store or relay them. The small head record listing a document's snapshot and changes is sealed and kept on the device. The owner's devices find one another at a rendezvous key in the DHT that is derived from the vault and rotates daily, exchange head records over a channel sealed with a key every device derives independently, fetch the blocks they lack by CID from whichever peer holds them, and merge (Figure 4). Concurrent offline edits merge deterministically without a coordinator, which is what makes the system local-first (I3). A query index (SQLite compiled to WebAssembly over the origin-private file system) is derived from these documents, sealed at rest, and never treated as the source of truth.

Groups are shared vaults: a group is a random root of its own, delivered to each member inside a sealed message. Every member derives the same group key ring and meets the others at the group's own rendezvous keys. Removing a member requires a new root, because what a member has already received remains readable to them; the implementation rotates the group to a new root and re-shares it with the remaining members.

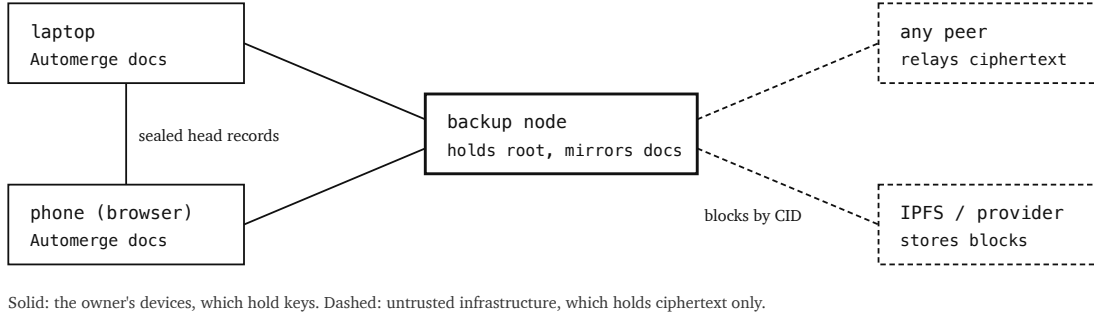


Figure 4: Synchronisation between an owner's devices. Head records travel sealed; blocks may come from anyone because each is verified against its CID.

6. Network

6.1 Transports and block exchange

Nodes run libp2p with the Noise handshake (X25519, ChaCha20-Poly1305) and Yamux multiplexing over TCP and WebSockets; browsers reach each other over direct WebRTC, signalled through any node acting as a circuit relay. Peer and provider discovery uses a Kademlia DHT. Blocks move over `/ghostdrive/block/1.1.0`: one length-prefixed want-list of up to 64 identifiers per stream, answered in order, the requester re-hashing every block. Readers become seeders: every fetched block is re-announced, and the blocks of one file are striped across the peers that hold them by rendezvous hashing, with the others as fallbacks.

6.2 Encryption in depth

Every connection is encrypted by Noise, but confidentiality does not depend on it: the payload of every block, document and message is already sealed by I1 before a connection exists. A relay therefore forwards ciphertext inside ciphertext. What the network still observes is metadata — which peers talk, when, and how much — and which ciphertext identifiers a peer requests (Figure 5).

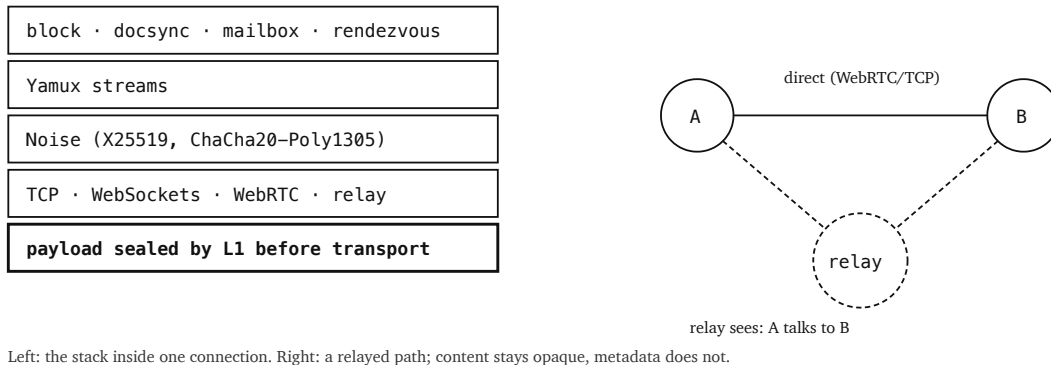


Figure 5: Network layers. Transport encryption protects metadata of a session; content confidentiality is established beforehand by the data layer.

6.3 Messages and mailboxes

People address each other by Ghost ID. A sender encapsulates to the recipient's X-Wing public key, derives an AES-256-GCM key from the shared secret, encrypts the message and signs the envelope with its Ghost ID key:

$$(\kappa, s) = \text{XWing.Encaps}(\text{pk}_{\text{to}}), \quad k = \text{HKDF}(s), \quad \sigma = \text{Ed25519}_{\text{sk}_{\text{from}}}(\text{aad} \parallel \text{iv} \parallel \text{ct}) \quad (6)$$

A message recorded today is not opened by a future quantum computer unless ML-KEM-768 is also broken, and file capabilities shared in messages inherit this protection. When the recipient is offline, the sender's node or a relay keeps the sealed envelope and delivers it later; nodes hold ciphertext with a time-to-live and nothing else.

7. Durability: Storage Proofs and Erasure Coding

7.1 Proofs anyone can verify

Availability is purchased, not assumed. An owner opens a storage deal committing to a *storage root*: a Merkle tree over fixed 1 KiB segments of the deal's ciphertext blocks, with domain-separated hashing and odd nodes promoted unchanged:

$$\text{leaf}(s) = \text{SHA-256}(0x00\|s), \quad \text{node}(l, r) = \text{SHA-256}(0x01\|l\|r) \quad (7)$$

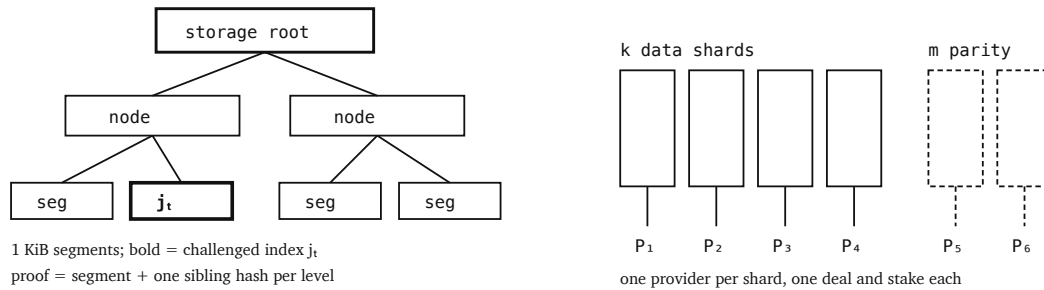
In each epoch e , the deal's c challenge indices (2 to 64, default 32) are derived from a beacon β unknown before the epoch starts — the epoch-start block hash mixed with the signature of a fixed drand quicknet round, verified on-chain through the EIP-2537 precompiles — so neither the provider nor the chain's sequencer can predict or choose them:

$$j_t = \text{SHA-256}(\beta\|\text{deal}\|e\|t) \bmod S, \quad 0 \leq t < c \quad (8)$$

where S is the number of segments. The provider answers with those segments and their Merkle paths; the StorageEscrow contract verifies them itself and pays for the epoch at once. Two consecutive missed epochs forfeit the provider's stake and refund every unpaid epoch to the owner. A verifier needs only the root and the segment count — no keys and no blocks. Providers earn on data they can never read.

7.2 Erasure coding across providers

A proof shows that *someone* holds the data; it cannot show that three providers hold three distinct copies. Ghostdrive therefore stripes a deal's blocks into groups of k and adds m Reed–Solomon parity shards over $\text{GF}(2^8)$ with a Cauchy matrix, which is maximum-distance separable: any k of the $k + m$ shards rebuild the rest. Shard i of every stripe goes to provider i , each under its own deal and stake, so distinct providers become a requirement on separate stakes rather than a hope. Parity of ciphertext is itself ciphertext; every rebuilt block is checked against its CID (Figure 6).



Left: an epoch challenge against the public root. Right: one stripe of RS(4,2); any 4 of the 6 providers rebuild it.

Figure 6: Durability. Storage proofs check possession against a root anyone can hold; erasure coding turns distinct providers into an enforced requirement.

8. Verifiable Registry and Lineage

Some data must be provably curated: datasets, model cards, creative works whose quality was assessed and whose existence at a time matters. Ghostdrive keeps such *golden entries* in a registry committed by a recursive proof (o1js, Kimchi over the Pasta curves):

$$\text{head}_i = \text{Poseidon}(\text{head}_{i-1}, H(\text{entry}_i), \text{score}_i) \quad (9)$$

For every entry the circuit enforces $85 \leq \text{score}_i \leq 100$, valid Schnorr signatures over $(\text{domain}, H(\text{entry}_i), \text{score}_i)$ from at least two of three committee evaluators fixed by hash in the public state, and correct folding. Each step verifies the previous proof inside the circuit, so the final proof covers the whole history at constant size (Figure 7). A rebase step makes each checkpoint carry the previous checkpoint as its base.

The RegistryAnchor contract accepts a checkpoint $(\text{head}, \text{count}, \text{base}, \text{proofHash})$ only if its base equals the latest anchored checkpoint, giving an append-only, ordered, timestamped history. A light client rejects any anchor whose verification-key hash differs from the one pinned in the application, fetches the proof by CID (the proof hash is exactly a raw block digest), checks its public input against the anchor and verifies it. The proof attests that a quorum of named evaluators signed each score; whether their judgment is good remains a governance question, not a cryptographic one.

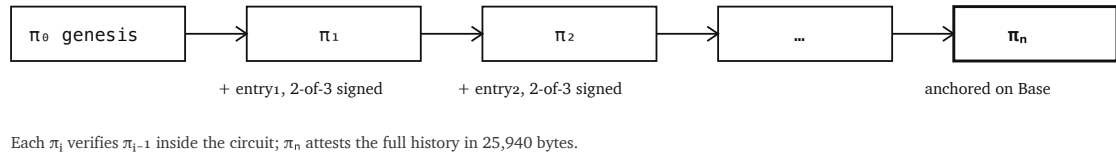


Figure 7: Recursive folding. The prover pays per entry; every verifier pays a constant 348 ms (§12).

Each model version may record the registry head it was trained against and the hash of its parent weights. This is a *tamper-evident lineage record*: it shows which registry state a model claims and lets the claim be checked against the anchor. It does not prove that training consumed that data. Proof of training remains an open problem whose realistic paths — deterministic re-execution, attested training in trusted execution environments, zero-knowledge proofs for small adapters — this paper does not claim (v1.0 did; the claim is withdrawn).

9. Private Memory and the Personal Agent

9.1 Memory and consent

Notes, threads and text files are split into passages of about 600 characters and embedded on the device with the multilingual-e5-small model (384 dimensions, int8). The resulting index lives in memory while the vault is open and is sealed as one snapshot at rest; search is exact cosine similarity over unit vectors, a few milliseconds for 10,000 passages. A backup node may hold its own index of the same vault. Nothing reaches a model outside the device without a grant in the owner's sealed consent document: grants name a source, a purpose (ai:local, ai:cloud, train:personal, train:collective) and optionally an audience; the default is deny, cloud use implies local use, and nothing implies collective training. Every use is logged.

Inference runs where the owner chooses: in the tab (Qwen 3.5 0.8B over WebGPU, 63 tokens/s on an Apple M5 Pro), on the owner's desktop or node (4B at 27 tokens/s, 9B at 16 tokens/s on the same machine), or through a key the owner provides. These are the only figures in this paper not produced by the benchmark command; they come from the measurements recorded in the implementation's local-AI design notes.

9.2 The agent and its approvals

Each person may run one agent, *their Ghost*, whose entire state — profile, standing instructions, goals, tasks, reminders, approval rules — is one sealed document the person can read and edit. Its always-on part runs on the person's own backup node on a 30-second heartbeat: it runs due tasks and reminders, reads connected accounts (a calendar's iCal feed, Notion pages shared with an integration, Google Calendar through an OAuth flow with PKCE run on the node itself, a Google Takeout mailbox imported and sealed locally, tools exposed by local MCP servers), and prepares actions.

Every action outside the agent's own thread passes one decision function (Figure 8). Let a be an action, $H(a)$ the set of hard stops it triggers, and $\rho_{\text{kind}(a)}$ an "always allow" rule for its kind with a daily count u_{today} and limit L (default 10):

$$\text{decide}(a) = \begin{cases} \text{deny} & \text{if the kill switch is set} \\ \text{ask, in the application only} & \text{if } H(a) \neq \emptyset \\ \text{allow} & \text{if } \rho_{\text{kind}(a)} \text{ is active and } u_{\text{today}} < L \\ \text{ask} & \text{otherwise} \end{cases} \quad (10)$$

The four hard stops are computed from what an action *does*, not what it is called: any non-zero amount makes it a payment; any recipient the owner has not approved by hand makes it a send to a new recipient; any secret field makes it a sign-in; any deletion whose target is not inside the vault is a deletion outside the vault. Hard stops are never covered by a rule, may be approved only in the application with the vault unlocked, never by a tap in a messaging app, and are re-checked by the node immediately before execution. The kill switch — one button, a command, or `/stop` from the owner's phone — halts every task and action at once, revokes every rule and denies whatever waits; only the application can resume.

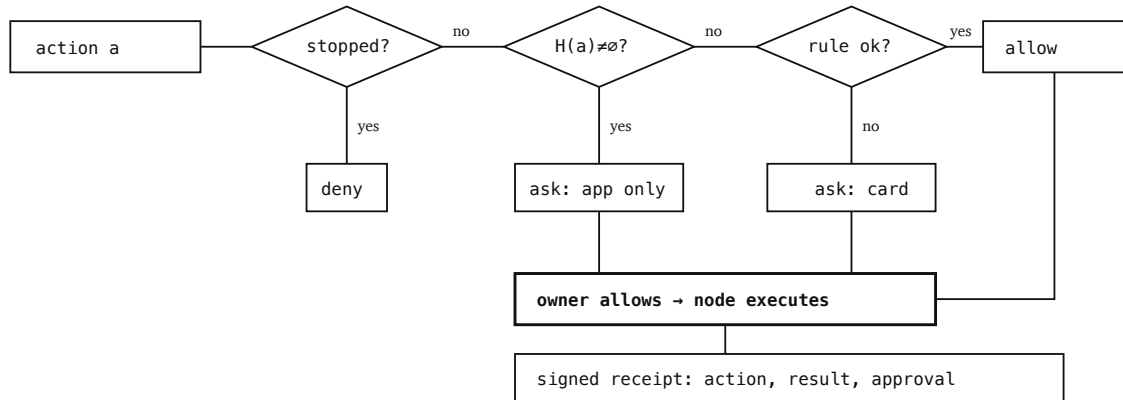


Figure 8: The approval path of every agent action. Hard stops bypass rules and can be approved only in the application; every executed action yields a signed receipt naming the approval it used.

Every run and every executed action produces a receipt signed with the owner's Ghost ID key over a canonical encoding of what ran, what it read, what it did, and the approval it used (for instance "Hard stop (pay) approved by you in Ghostdrive" or "Always allowed: 3 of 10 today"). The application verifies each receipt against the Ghost ID and marks any line that no longer matches its signature. A week of an agent's work can thus be audited in a minute, and a forged or edited log line is visible. Local MCP servers are spawned only after the owner allows them, with their command hash pinned; a changed script is refused until allowed again, and tool calls with side effects pass the same approvals.

10. Endpoint Security

The weakest point of an end-to-end encrypted system is the end. The desktop application is built so that the window is treated as hostile: the renderer is sandboxed with context isolation and no Node integration, loads code only from its own `app://` scheme inside the signed archive, and enforces a strict content security policy; a typed bridge checks the calling frame's origin on every call; Electron fuses disable running as Node, `NODE_OPTIONS` and inspector flags and enforce archive integrity; and a release build refuses to start with remote-debugging switches, which fuses alone do not cover. The node runs in a separate utility process with its own lifecycle, so the window never needs network or disk access of its own.

The next step moves the vault root out of the window entirely: a key holder in the node process keeps keys in locked, zeroised memory and seals the root at rest to a Secure Enclave key that cannot leave the chip and opens only after Touch ID. The window then receives documents and signatures, never the root. Because the code is open source, Ghostdrive protects integrity rather than secrecy: the defence against a modified binary is code signing, archive integrity and reproducible builds anyone can check, not obfuscation. A *rekey* moves a vault to a new root: implemented today for vaults rooted in a wallet signature, and planned for phrase-rooted vaults whose words may have been exposed. Rekeying protects what is written next; whoever saw the old root can still read what was sealed under it, and no construction can undo that.

11. Security Analysis

Figure 9 shows the trust zones and Table 1 the adversaries considered with the mechanism that stops each. The two quantitative questions a reader should ask are how quickly a provider that discards data is caught, and how durable an erasure-coded deal is.

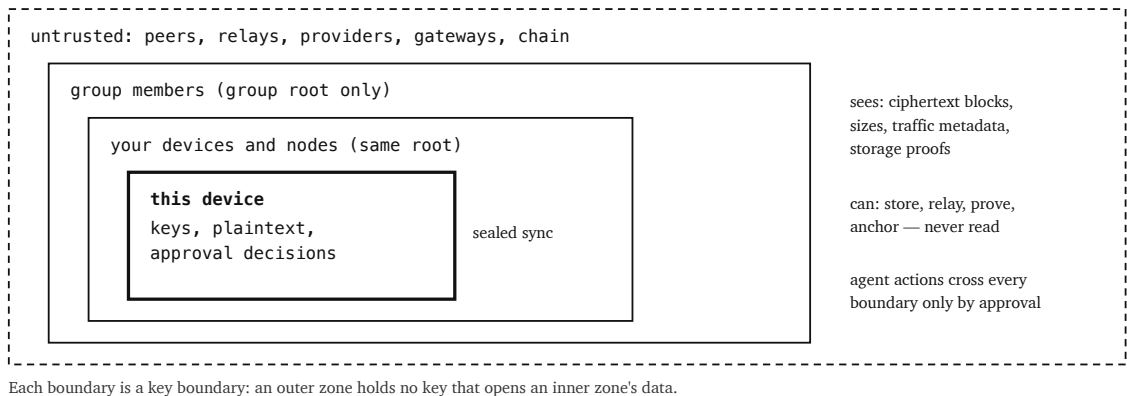


Figure 9: Trust zones. Confidentiality never depends on an outer zone behaving well; integrity depends on it only for availability, which proofs and erasure coding make measurable.

Table 1: Adversaries, mechanisms and residual risks.

Adversary	Mechanism	Residual risk
Network observer, relay, ISP	Noise per connection; content sealed by L1 beforehand	Traffic metadata
Malicious peer serving data	CID re-hash on receipt (I2)	Wasted bandwidth
Storage provider dropping data	Unpredictable segment challenges; stake; repair; erasure coding	Correlated provider loss
Forged or rewritten registry	Pinned verification key; anchor accepts only extensions	Circuit bug; publisher key may append
Low-quality registry entries	2-of-3 evaluator signatures verified in-circuit	Evaluator collusion
Script in the application window	Sandbox, origin-checked bridge, CSP	Reads open documents until E5
Malware with the owner's privileges	Hardened runtime, Keychain bound to signature (with Developer ID)	Ad-hoc signed builds today
Agent acting against the owner	Approvals, four hard stops, kill switch, signed receipts	Owner approves a harmful card
Future quantum computer	X-Wing for messages; symmetric sealing for data	Noise sessions are classical
Exposure of the 24 words	Rekey to a new root, device revocation	Data sealed before the rekey

11.1 Detection of a provider that discards data

Suppose a provider silently discards a fraction f of a deal's segments. Because challenge indices are uniform and unpredictable (Eq. 8), each of the c challenges independently lands on a missing segment with probability f , and the provider survives an epoch with probability

$$P_{\text{pass}} = (1 - f)^c, \quad P_{\text{detect}} = 1 - (1 - f)^c \quad (11)$$

Table 2 evaluates Eq. 11. At the default of 32 challenges, a provider holding only 90% of the data is caught in a single epoch with probability 0.966, and two consecutive misses forfeit its stake. Because the probability compounds across epochs, a provider discarding even 5% of a deal fails within a handful of epochs with near certainty.

Table 2: Probability of detecting, in one epoch, a provider missing a fraction f of segments, for c challenges.

f	$c = 2$	$c = 8$	$c = 32$	$c = 64$
0.01	0.0199	0.0773	0.2750	0.4744
0.05	0.0975	0.3366	0.8063	0.9625
0.10	0.1900	0.5695	0.9657	0.9988
0.25	0.4375	0.8999	0.9999	≈ 1.0000

11.2 Durability of an erasure-coded deal

Let each of $n = k + m$ providers be lost independently within a repair interval with probability p . A stripe is lost only if more than m providers are lost before repair, where $C(n, i)$ is the binomial coefficient:

$$P_{\text{loss}}(k, m, p) = 1 - \sum_{i=0}^m C(n, i) p^i (1 - p)^{n-i} \quad (12)$$

Table 3 compares codes. The comparison carries a lesson the v1.1 paper understated. RS(10,4) stores $1.4\times$ the data, less than half of three-way replication, but at equal loss probability per provider it is *less* durable per stripe than three replicas, because it spreads a stripe over fourteen providers. Its value is cost, and it depends on prompt repair, which the provider service performs every epoch. At equal storage overhead ($3\times$), erasure coding is better than replication by several orders of magnitude: RS(4,8) loses a stripe with probability 1.7×10^{-7} where replication loses 10^{-3} . Deals should choose (k, m) from the owner's durability target, not by default.

Table 3: Probability that a stripe is lost before repair, per Eq. 12.

Scheme	Overhead	Tolerates	$p = 0.05$	$p = 0.10$	$p = 0.20$
3 replicas	$3.0\times$	2 losses	1.3×10^{-4}	1.0×10^{-3}	8.0×10^{-3}
RS(10,4)	$1.4\times$	4 losses	4.3×10^{-4}	9.2×10^{-3}	1.3×10^{-1}
RS(4,8)	$3.0\times$	8 losses	3.7×10^{-10}	1.7×10^{-7}	6.2×10^{-5}
RS(10,20)	$3.0\times$	20 losses	1.3×10^{-15}	4.9×10^{-15}	4.5×10^{-9}

11.3 Keys

The root carries 256 bits of entropy; guessing it is out of reach of any adversary, classical or quantum (Grover's algorithm leaves 128 bits). Document and chunk keys are 256-bit AES keys derived by HKDF and HMAC; the deterministic IV of Eq. 3 is a function of the key and the plaintext, so a repeated IV implies a repeated plaintext under the same key and leaks only that two blocks of the same owner are equal, which is the intended deduplication. The convergent policy does not leak equality across owners, because their keys differ.

12. Evaluation

All figures in Table 4 are produced by `npm run bench` in the open-source repository, at commit `b57fb56f` on an Apple M5 Pro (18 cores, 64 GB), Node 26. They are protocol-level measurements; swarm figures use TCP loopback on one machine and therefore measure protocol overhead, not network bandwidth.

Encryption adds 78.7% to the time of the plaintext pipeline under the convergent policy and 66.1% under the random-key policy, while still running above 420 MB/s, far faster than any network link it feeds. On loopback, three peers are slower than one because the CPU, not the link, is the bottleneck; on real links striping should scale with peers, which remains to be measured. The asymmetry of the registry is deliberate: folding is expensive and done by few provers; verification is constant and done by everyone.

Table 4: Measured performance (docs/BENCHMARKS.md).

Operation	Input	Result
Chunk + hash + encrypt, convergent	100 MB, median of 3	422.4 MB/s
Chunk + hash + encrypt, random key	100 MB, median of 3	454.2 MB/s
Same pipeline without encryption	100 MB	754.7 MB/s
Storage overhead (IV + tag + manifest)	679 chunks	+0.073%
Deduplication, one owner	10 versions of a 4 MB document	4.37×
Swarm retrieval, 1 / 3 peers	50 MB, loopback	162.8 / 126.7 MB/s
Storage tree build	100 MB → 102,848 segments	2.11 s
Epoch storage proof, 32 challenges	102,848 segments	50,048 B; 13.6 ms
On-chain proof verification	340-segment deal, EVM	1,701,410 gas
Registry circuit compile	once per prover	4.3 s
Registry fold, 2-of-3 signatures	per entry	13.7 s
Registry proof size	any history length	25,940 B
Registry light-client verify	any history length	348 ms
Open vault index	10,000 files, sealed snapshot	8 ms
Search vault index by name	10,000 files, trigram index	11.3 ms

13. Comparison

Table 5: Properties of deployed systems. ● full · ◐ partial or opt-in ○ absent.

Property	Ghostdrive	IPFS	Filecoin	Arweave	Storj	Cloud drive
End-to-end encryption by default	●	○	○	○	●	○
Blocks interoperable with IPFS	●	●	◐	○	○	○
Storage proofs anyone can verify	●	○	●	◐	◐	○
Works offline, device authoritative	●	◐	○	○	○	◐
Post-quantum messages between people	●	○	○	○	○	○
O(1) registry verification	●	○	○	○	○	○
Agent with hard stops and signed receipts	●	○	○	○	○	○
Erasure (crypto-shredding)	◐	◐	◐	○	●	●
Provider can read data	no	yes	yes	yes	no	yes

Erasure is partial for Ghostdrive because only the random-key policy can be shredded (§4.2). No prices are published: storage will be priced in a stable asset once real provider costs are measured.

14. Implementation Status and Future Work

The implementation is open source (Apache-2.0 or MIT) in TypeScript, with Solidity contracts and an o1js circuit. Table 6 separates what runs and is tested from what is designed or in progress. Every claim in Sections 3–9 marked as running is backed by tests in the repository.

Table 6: What is implemented and what is planned, October 2026.

Component	Status
Data layer v3.1: root, key ring, sealing, CIDv1 blocks, capabilities	IMPLEMENTED
24-word recovery, passkey wrap, Ghost ID, device records	IMPLEMENTED
Automerge documents as sealed blocks; device sync; groups with rotation	IMPLEMENTED
libp2p swarm in node and browser; WebRTC via relay; mailboxes; X-Wing messages	IMPLEMENTED
Memory index on device and node; consent engine; MCP server	IMPLEMENTED
Agent: tasks, reminders, brief, connectors, approvals, hard stops, kill switch, receipts, MCP client	IMPLEMENTED
Storage escrow, segment proofs, drand beacon, provider registry, Reed–Solomon	IMPLEMENTED (testnet)
Registry v2 circuit, Base anchor contract, light client	IMPLEMENTED (testnet)
Desktop: sandboxed window, origin-checked bridge, fuses, node in its own process	IMPLEMENTED
Contract deployment on Base mainnet; alias registry live	PLANNED
Private mesh for the owner's devices; NAT hole punching; device revocation	IN PROGRESS
Rekey of phrase-rooted vaults to new words	IN PROGRESS
Key holder with Secure Enclave seal; Developer ID signing	PLANNED
Post-quantum device pairing; size padding of blocks; external audit	PLANNED
Proof of useful contribution and its reward scheme	SPECIFIED (not built)

Three limits deserve emphasis. First, metadata: the network sees who talks to whom; padding, mesh-only routing and cover traffic reduce but do not remove this. Second, erasure under the convergent policy is impossible by construction; owners must choose the random-key policy for data they may need to destroy. Third, a contribution-reward scheme has been specified with hidden rotating evaluation sets, evaluator quorums, bonds, delayed vesting and clawback, but no part of it is implemented, and this paper makes no claim about it.

15. Conclusion

We have described a system for personal memory in which privacy is the default rather than a setting. Keys descend from a root only the owner holds; every byte is sealed before it leaves the device; untrusted peers store, relay and prove possession of ciphertext they cannot read; the owner's devices remain authoritative and work offline; and a personal agent acts only through approvals whose every use is signed. The network contributes durability and reach, never trust. What remains — a private mesh among the owner's devices, keys held outside the window in hardware, deployment of the settlement contracts, and an external audit — extends the same principle: the owner's memory should be readable by no one but its owner, and verifiable by anyone.

References

1. S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System,” 2008.
2. J. Benet, “IPFS — Content Addressed, Versioned, P2P File System,” 2014.
3. Protocol Labs, “Filecoin: A Decentralized Storage Network,” 2017.
4. D. Vorick, L. Champine, “Sia: Simple Decentralized Storage,” 2014.
5. W. Xia et al., “FastCDC: A Fast and Efficient Content-Defined Chunking Approach,” USENIX ATC, 2016.

6. P. Maymounkov, D. Mazières, “Kademlia: A Peer-to-Peer Information System Based on the XOR Metric,” IPTPS, 2002.
7. M. Kleppmann, A. Wiggins, P. van Hardenberg, M. McGranaghan, “Local-first software: you own your data, in spite of the cloud,” Onward!, 2019.
8. M. Shapiro, N. Preguiça, C. Baquero, M. Zawirski, “Conflict-free Replicated Data Types,” SSS, 2011.
9. T. Perrin, “The Noise Protocol Framework,” revision 34, 2018.
10. D. Connolly, P. Schwabe, B. Westerbaan, “X-Wing: general-purpose hybrid post-quantum KEM,” IETF CFRG draft, 2024.
11. NIST, “FIPS 203: Module-Lattice-Based Key-Encapsulation Mechanism Standard,” 2024.
12. H. Krawczyk, P. Eronen, “HMAC-based Extract-and-Expand Key Derivation Function (HKDF),” RFC 5869, 2010.
13. M. Palatinus, P. Rusnak, A. Voisine, S. Bowe, “Mnemonic code for generating deterministic keys,” BIP-39, 2013.
14. I. S. Reed, G. Solomon, “Polynomial Codes over Certain Finite Fields,” J. SIAM, 1960.
15. I. Meckler, E. Shapiro, “Coda: Decentralized Cryptocurrency at Scale,” 2018.
16. o1Labs, “Kimchi: A Plonk-based proof system over the Pasta curves,” 2022.
17. drand, “League of Entropy: quicknet unchained randomness beacon,” 2023.
18. L. Wang et al., “Multilingual E5 Text Embeddings,” 2024.
19. W3C, “Web Authentication: An API for accessing Public Key Credentials, Level 3 (PRF extension),” 2025.