Ghostdrive: Private Memory with Verifiable Records on a Purpose-Built Ledger

Roma Nebo

Technical White Paper · November 2026 · v3.0

Abstract. An organisation that keeps its documents and its AI assistants in a cloud service must trust the provider not to read the data, to still hold it, to leave records unchanged, and to keep what an assistant saw. We describe a system that replaces each of these with a check. All keys derive from one 32-byte root that never leaves the user's devices. Data is split by content-defined chunking, padded and sealed with AES-256-GCM into content-addressed blocks of at most 256 KiB, so a storage provider holds uniform ciphertext and no key. Providers prove possession every epoch by opening Merkle paths at positions fixed by a public randomness beacon; a provider missing a fraction f of a file passes an epoch with probability $(1 - f)^c$, and a contract pays only for proven epochs. Files are placed as Reed-Solomon 10 + 4 shards or as three copies. Records that third parties must check (the origin of a file, each step of a signing, the log of what an AI was shown) are reduced to fingerprints and written to the DRIVE 369 Network, a permissioned QBFT ledger with two-second blocks and immediate finality, so a verifier needs the record and one ledger read, and never the operator's word. AI answers served by the network carry a signed receipt, and agents act only inside a signed, time-limited mandate with four classes of action reserved to a person. We give the key hierarchy, the block and proof formats, the per-observer privacy analysis, closed-form detection and availability bounds, and measurements of the reference implementation: 454 MB/s chunk-and-seal throughput, a 50 KB storage proof verified in 13.6 ms, and +0.073% storage overhead. Each mechanism is marked live, built or planned, and each section states what it does not provide.

1. Introduction

An organisation that places its documents and its AI assistants with a cloud service takes four things on trust: that the provider cannot read the data, that the data is still there, that a record is what it was, and that an assistant kept what it saw to itself. Encryption in transit and at rest leaves the keys with the provider. A storage bill proves payment, not possession. A log is vouched for by the party that stores it. A prompt sent to a model leaves no durable trace of what was sent, where, or what came back.

This paper describes a system that replaces each of these with something that can be checked. Keys are created on the user's device and never leave it (Section 3). Data is sealed into uniform ciphertext blocks before it leaves (Section 4) and moves between devices and people only in sealed form (Section 5). Storage providers prove possession every epoch against a public randomness beacon, and a contract pays them only for proven epochs (Section 6). What each party on the path can and cannot learn is stated per observer (Section 7). Records that third parties must be able to check are fingerprinted on a ledger built for that purpose, the DRIVE 369 Network (Sections 8 and 9). An AI answer from the network arrives with a signed receipt, and an agent acts only inside a signed, time-limited mandate (Section 10).

Compared with version 2.3 of this paper, three things change. The ledger of record is the project's own permissioned chain and no longer a public one. Verifiable records (proof of origin, sealed signing steps,

sealed AI-use logs) are specified as mechanisms with their exact claims. Token economics is removed: the network's token appears only as the utility unit that pays for storage and compute, and its supply, allocation and emission are outside the scope of this paper.

Evidence convention. Every mechanism is marked with its state on 15 November 2026 (application version 0.11.13): *LIVE* means deployed and in use, *BUILT* means in the main branch with tests and no shown deployment, *PLANNED* means design only. Each section ends with its limitations. Measurements are in Section 12, closed-form calculations in Section 11, and the full status table in Section 16.

2. System Model

The system has one trusted place, the user's device, and four kinds of outside party (Figure 1). A *relay* forwards sealed frames and helps devices find each other. A *storage provider* holds ciphertext blocks under a funded deal. An *AI operator* runs a model for users who have none. A *validator* orders fingerprints on the ledger. The adversary may control any outside party, may observe the network, and may hold any ciphertext indefinitely. It does not control an unlocked device or the release build of the application, which together form the trusted computing base.

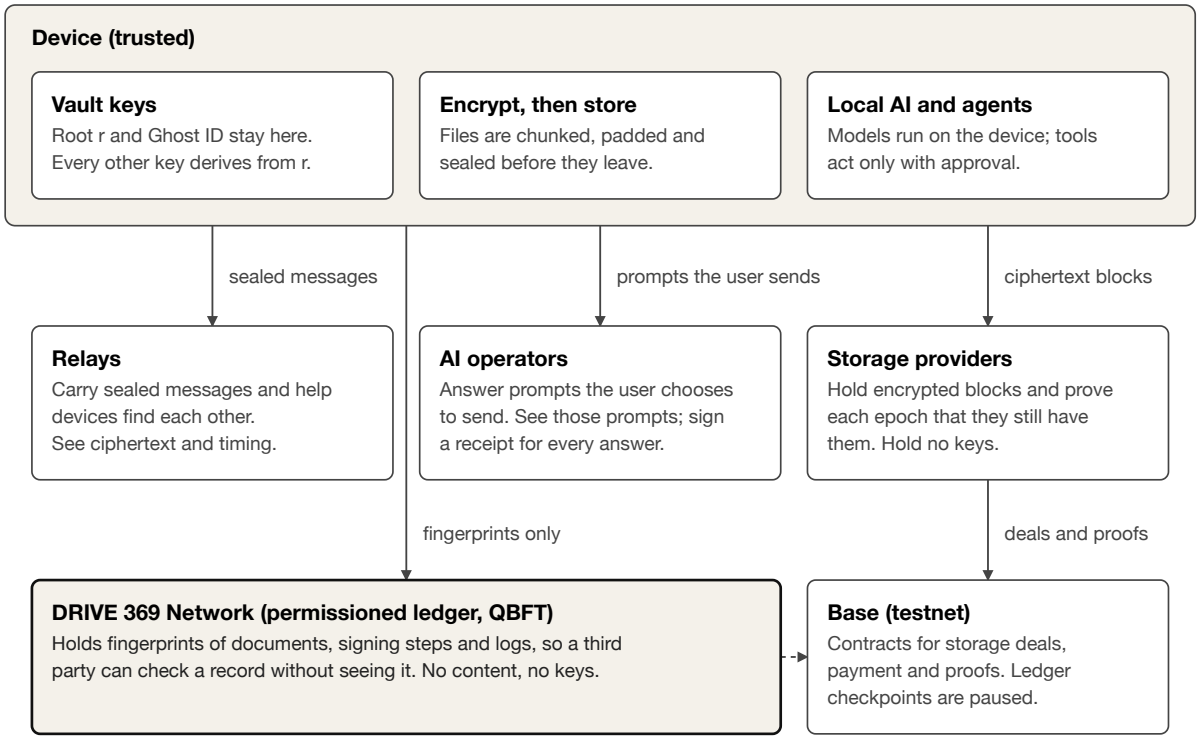


Figure 1. System model. Keys never leave the device. Each outside party receives the least it needs: ciphertext, a prompt the user chose to send, or a fingerprint.

The design goals follow from that model. **G1, confidentiality:** no outside party obtains a key or plaintext, with the single stated exception of a prompt the user sends to an AI operator. **G2, possession:** a provider that has lost a fraction of a file is detected with a probability that is computable in advance (Section 11). **G3, integrity of records:** a third party can check that a record existed in a given form without trusting the operator of any server, and without seeing the record. **G4, bounded agency:** an AI agent can do nothing the owner did not sign for, and four classes of action always require a person.

3. Identity and keys

This section establishes that one 32-byte secret on the user's devices is the origin of every key, and what follows for devices, wallets and recovery.

3.1 Root, Ghost ID and key hierarchy

A vault root is 32 random bytes, shown to the user as a 24-word BIP-39 English mnemonic (*LIVE*). The words encode the root directly, with no PBKDF2 stretching and no passphrase.

A Ghost ID (*LIVE*) is the vault's Ed25519 public key, written as 52 characters of lowercase base32 with no checksum or version byte. It is identical on every device that opens the vault.

Every derivation is HKDF-SHA256 with a fixed protocol salt and its own info label. Encryption is AES-256-GCM with a 12-byte IV and a 16-byte tag. The root yields eleven values (*LIVE*). The principal ones:

Table 1. Principal keys derived from the vault root.

Key	Derived how	Used for
Identity seed	info "drive369/v3/identity", 32 bytes	Ed25519 signing key behind the Ghost ID
Document root	info "drive369/v3/doc-root", HMAC-SHA256	Per-document keys
Sync key	info "drive369/v3/doc-sync", AES-256-GCM	Sync frames between devices
Rendezvous key	info "drive369/v3/rendezvous", HMAC-SHA256	Daily meeting point of the vault's devices
Mailbox seed	info "drive369/v3/box", 32 bytes	X25519 and X-Wing key pairs for sealed messages
Mesh root	info "drive369/v3/mesh", HMAC-SHA256	Per-epoch keys by which a vault's devices authenticate each other

By default, file content keys are random and not derived from the root.

$$k_\ell = \text{HKDF-SHA256}(\text{ikm} = r, \text{salt} = s_0, \text{info} = \ell), \quad \text{GhostID} = \text{base32}(\text{Ed25519.pub}(k_{\text{identity}})) \tag{1}$$

where r is the 32-byte vault root, s_0 the fixed protocol salt and ℓ a label such as `drive369/v3/identity`.

3.2 Wallets, passkeys and unlock

A wallet never becomes the key (*LIVE*): signing in with a wallet creates a key on the device, kept by a passkey, and links the wallet with one signature. Legacy vaults derive their keys from a wallet signature over a fixed message. That path still opens existing vaults. It was retired because any site requesting the same text can phish the signature.

A passkey never holds the root (*LIVE*). The WebAuthn PRF extension returns 32 bytes, from which HKDF-SHA256 derives an AES-256-GCM key that wraps the root. On macOS the desktop app wraps the root with a random 32-byte secret sealed by the OS keychain and prompts for Touch ID (*LIVE*).

3.3 Devices, revocation and re-key

Every linked device holds the root. Linking by QR code (*BUILT*) carries a one-time 32-byte secret and an ephemeral X25519 key in a URL fragment valid for 10 minutes. Both screens show a six-digit code derived from the key-exchange transcript. After the user confirms on both, the root is sealed to the new device's key with XChaCha20-Poly1305.

Each device has a record signed by the Ghost ID (*LIVE*). Per-device Ed25519 and X25519 keys, and a signed device list valid for at most 30 days, are *BUILT*.

Revoking a device (*LIVE*) removes its record and advances the mesh epoch. The user's other devices and backup nodes then refuse that peer in document sync and block exchange.

A move to new words (*LIVE*) is the response to a compromise. It creates a new root, new keys and a new Ghost ID. Index records and documents are re-sealed under the new keys. File blocks are not touched, because read capabilities carry their own content keys. The old and the new Ghost ID both sign a succession statement, so receipts signed by the earlier identity still verify.

Limitations

- Guardian or social recovery. It is *PLANNED*. Losing both the device key or passkey and the 24 words is unrecoverable.
- Lock-out by revocation alone. A revoked device still holds the root: it can derive any mesh epoch key, see that documents changed and reappear under a new network identity. Only a move to new words locks it out.
- Protection of the past. Everything sealed under the old words stays readable to whoever saw it. Contacts and groups are not notified of a move automatically, and a holder of the old words can sign a rival succession.
- A hardware-bound desktop key. The Touch ID prompt is a presence check by the app.

4. Encrypted storage format

This section establishes what a file becomes before it leaves the device, and what a party holding the resulting blocks can learn from them.

4.1 Chunks, keys and padding

Plaintext is split by content-defined chunking (*LIVE*): a gear rolling hash with FastCDC-style normalisation over a table generated from a fixed seed, so any implementation reproduces the same boundaries. Defaults are a 32 KiB minimum, a 128 KiB average and a 262,116-byte maximum.

Each chunk is padded and sealed with AES-256-GCM (*LIVE*) into one block of at most 256 KiB: a 12-byte IV, the ciphertext and a 16-byte tag. The IV is deterministic, the first 12 bytes of an HMAC-SHA256 over the AAD and the hash of the padded chunk, so a key and IV pair is never used with two different plaintexts.

The default key policy is random (*LIVE*): one fresh 32-byte key per file. Equal plaintext yields unrelated blocks, so a block holder cannot tell that two files or two versions share content.

The convergent policy is opt-in per upload (*LIVE*). The file key is HMAC(convergent secret, SHA-256(plaintext)), and each chunk has its own key derived the same way from the chunk's hash. Identical chunks of one owner produce identical blocks across files and versions. This saves space and reveals to a block holder that the same chunk was stored again. The secret derives from the vault root, so nothing matches across owners.

File chunks are always padded (*LIVE*) to a Padme size class, at a cost of at most about 11%. Only a chunk already at the maximum size is sealed unpadded. Other sealed data uses power-of-two buckets from 1 KiB (256 bytes for document blocks) to 256 KiB. Sync frames between devices are padded by default (*LIVE*). Messages are written padded by default in the web app (*LIVE*). Padded writes of document blocks and presence records are *BUILT* and off by default.

$$(k_{\text{enc}}, k_{\text{iv}}) = \text{HKDF-SHA256}(k_{\text{file}}), \quad \text{IV} = \text{HMAC}(k_{\text{iv}}, \text{AAD} \,\|\, \text{SHA-256}(m))_{[0..12)} \tag{2}$$

$$\text{block} = \text{IV} \,\|\, \text{AES-256-GCM}(k_{\text{enc}}, \text{IV}, m, \text{AAD}), \quad \text{cid} = \text{CIDv1}(\text{raw}, \text{SHA-256}(\text{block})) \tag{3}$$

where m is the padded chunk. The IV is a function of key and plaintext, so one (key, IV) pair never seals two different plaintexts.

$$k_{\text{file}} \leftarrow \text{random 32 bytes (default)}; \quad k_{\text{file}} = \text{HMAC}(k_{\text{conv}}, \text{SHA-256(plaintext)}) \text{ (convergent, opt-in)} \tag{4}$$

4.2 Manifests, capabilities and addressing

A manifest (*LIVE*) is canonical JSON with no timestamps: name, MIME type, size, the SHA-256 of the whole plaintext, the chunk references, the chunking parameters and the key policy. It is stored only encrypted under the file key. A file with more than 2000 chunks keeps its references in encrypted list blocks, two levels at most: 4,000,000 chunks, about 500 GB at the default average. Directory manifests are *PLANNED*.

A read capability (*LIVE*) is the string `drive369:v3:<manifest CID>:<key>`: the address of the encrypted manifest and the 32-byte file key. Whoever holds it can read the file from any source of blocks.

Every block is addressed by a CIDv1 with the raw codec and SHA-256, computed over the exact ciphertext (*LIVE*). Stores recompute the hash on every write and read; a block that no longer matches is quarantined and fetched again. After decryption the reader checks the total size and the whole-plaintext hash against the manifest.

4.3 What a storage node sees

A storage node holds opaque blocks of at most 256 KiB, their addresses and their size classes. It observes who requests which block and when. Manifest, list and chunk blocks look alike, and nothing visible to the node links a manifest to its chunks. Readers request chunk blocks in random order, so a source does not see sequential access.

Limitations

- Revocation of a read capability. It carries no owner, expiry or signature, and once handed out it cannot be withdrawn. There is no write, directory or attenuated capability.
- Deduplication under the default policy, or equality hiding under the convergent one.
- Exact-size or access-pattern hiding. The size class, the addresses requested and the timing remain visible.
- Default padding of document blocks and presence records. Until it is switched on, their ciphertext length tracks the plaintext length.
- Complete test vectors. The known-answer vectors cover the unpadded chunk form; the padded form written today, manifests and list blocks have none.

5. Sync and collaboration

This section establishes how a person's devices, a group and outside parties exchange data while every server on the path holds only ciphertext.

5.1 Documents and device sync

Structured data lives in 32 Automerge CRDT documents (*LIVE*). Each local change is one block sealed with AES-256-GCM under the document's own key. A change is written locally first and announced afterwards, so the app works offline and converges by merge when devices meet.

Device sync (*LIVE*) exchanges announcements of document heads in frames sealed under the vault's sync key. Every received block is checked against its content address and then opened with the document key. The sender's peer identity is sealed inside the frame, so a frame replayed by another peer does not count.

Devices meet at a daily rendezvous value, HMAC-SHA256(rendezvous key, UTC day number), published in a Kademlia DHT and registered with a sealed presence record on up to three nodes. It cannot be linked across days without the vault.

A backup node (*LIVE*) holds the vault root and syncs as one more device. A relay mailbox (*LIVE*) keeps sealed document blocks for up to 30 days while all of a person's devices are off. A live editing channel for up to 8 peers is *BUILT*.

$$\rho_{\mathrm{d}} = \mathrm{HMAC\text{-}SHA256}(k_{\mathrm{rendezvous}}, d), \qquad d = \lfloor t_{\mathrm{UTC}} \,/\, 86\,400\ \mathrm{s} \rfloor \tag{5}$$

Two days yield unlinkable values to anyone without the vault.

5.2 Groups

A group (*LIVE*) is a fresh random 32-byte root. Members derive the same key hierarchy from it, and the group syncs as one more key ring. The root travels in a sealed message, and the group is added only after the recipient chooses to join.

Admins are the creator, the inviter and admins the inviter vouched for, at most 16, pinned in each member's private record at join time. The shared member list is writable by every holder of the root and is never used for authorisation.

Groups have no numbered key epochs. To remove a member, an admin draws a new root, copies the notes, files, rooms and library under the new keys, and sends each remaining member an invite naming the group it replaces. A recipient switches only if the sender is an admin that recipient pinned. An admin can sign a member's proof of membership, valid for at most 30 days, that a party without the group root can check (*BUILT*).

5.3 Messages, links and calls

Sealed messages (*LIVE*) use the X-Wing KEM (X25519 + ML-KEM-768), HKDF-SHA256 and AES-256-GCM, signed with the sender's Ed25519 Ghost ID key. A file is shared with a person by placing its read capability in the message body. An X25519-only envelope is still sent to contacts whose saved mailbox key is the older 32-byte form. Delivery counts only on a receipt signed by the addressee. A sealed-sender envelope (*BUILT*) names only the addressee on the outside. The web app opens it; sending it stays off until desktop builds that open it are released.

A shared link (*LIVE*) seals a note snapshot, or a file's read capability, under a fresh random AES-256-GCM key. A relay stores the sealed blob; the key sits in the URL fragment and never reaches a server. One-time secret links with a single view are *BUILT*.

Calls (*LIVE*) are a full WebRTC mesh of up to 12 people with no media server. Signalling passes through the host, sealed with AES-GCM under a key derived from the call key in the link fragment, and the host admits each joiner. On top of WebRTC transport encryption, every encoded audio and video frame is sealed with AES-256-GCM under a per-sender key derived from an epoch secret. The host draws a new epoch secret on every join or leave.

Limitations

- Recall of data. A removed group member keeps everything already received, and a read capability taken from a link or message cannot be revoked.
- Protection from a call's host, who issues every epoch secret. If the host leaves, the call ends; there is no reconnect and no host hand-over.
- Tested frame encryption outside Chromium. The Safari and Firefox path is not exercised in testing.

- Metadata privacy against relays. A relay sees IP addresses, sizes and timing of sealed frames, opaque daily rendezvous values, block addresses, and the sender and addressee Ghost IDs of each mail envelope. The share store sees the owner's Ghost ID on each post.
- A blind backup node. It holds the root and can read the vault, so it belongs on a machine the owner controls.

6. Storage network

This section establishes how ciphertext is placed with storage providers, how possession is checked on chain, how lost shards are replaced, and which parts run on a public testnet versus on the project's own machines.

All storage contracts are on Base Sepolia only. The providers serving deals today are the project's own machines, as are the relays and the storage broker.

6.1 Roles and joining

A storage-only node (*LIVE*) holds blocks and mail as ciphertext and never holds a vault. A backup node (*BUILT*) holds the owner's vault root and keeps a copy of every block of every listed file. It can read the file list, so it belongs on the owner's own machine. A relay (*LIVE*) is a circuit relay, bootstrap peer and block server that holds no key.

A provider joins by registering in the ProviderRegistry contract (*LIVE*): an endpoint, a price per GiB per epoch, a capacity, a region and a bond of at least 100 test USDC. The contract has no owner, admin or upgrade path. Before any transaction the owner's node has the provider sign a fresh nonce with its listed wallet (*BUILT*). Exit unlists at once. The bond returns after the unbonding period (86,400 s on testnet) and only when no deal is open.

6.2 Deals and proofs

Each shard is one deal in the StorageEscrow contract (*LIVE*) with a distinct provider: distinct wallet, peer, and IPv4 /24 or IPv6 /48. The owner funds the whole term up front; the provider accepts only after it has fetched every block and recomputed the committed root, a Merkle tree over 1,024-byte segments of the ciphertext: leaf = SHA-256(0x00 ‖ segment), node = SHA-256(0x01 ‖ left ‖ right). The provider needs no file key.

In every epoch the provider answers challenges derived from a drand quicknet round fixed more than one period after the epoch opens, so the provider cannot choose it. The contract verifies the BLS12-381 signature, derives the segment indexes from its hash, and checks one segment and one Merkle path per challenge. A provider missing a fraction f of the segments passes an epoch with probability $(1 - f)^c$ for c challenges. A verified proof releases that epoch's payment, less a protocol fee parameter of 369 bps. Two consecutive missed epochs fail the deal: the owner is credited the unpaid epochs, and anyone may slash 20% of the provider's bond.

Table 2. Storage tiers (testnet parameters).

	Cold tier	Hot tier
Layout	Reed-Solomon 10 + 4 over GF(2^8); smaller codes for files under 10 blocks	3 full copies
Readable from	Any 10 of 14 shards	Any 1 of 3 copies
Proof epoch	12 h	6 h
Challenges per epoch	16	16
Eligible providers	Any listed provider	The owner's pinned hot list, at least 3 addresses
Limit per operator	4 shards (needs 4 operators)	2 copies (needs 2 operators)
Distinct copies	Optional	Default: each copy XORed with its own ChaCha20 keystream (*BUILT*)
Repair grace, owner away	6 h	2 h

As testnet parameters, cold storage is priced at 3.69 and hot at 11.07 USD per TiB-month.

$$\text{leaf}_i = \text{SHA-256}(0\text{x}00 \parallel s_i), \quad \text{node} = \text{SHA-256}(0\text{x}01 \parallel \text{left} \parallel \text{right}), \quad R = \text{root over all 1 024-byte segments } s_i \tag{6}$$

$$i_j = \text{H}(\sigma_{\text{drand}}(e) \parallel \text{deal} \parallel j) \bmod n, \quad j = 1 \ldots c \tag{7}$$

where $\sigma_{\text{drand}}(e)$ is the beacon signature for epoch e, verified on chain, n the segment count and c the challenges per epoch.

$$\Pr[\text{pass one epoch} \mid \text{fraction } f \text{ missing}] = (1 - f)^c \tag{8}$$

6.3 Repair and retrieval

The owner's node (*LIVE* on testnet) slashes a failed deal, picks a new provider, rebuilds the shard from any k surviving shards and opens a replacement deal on the same terms. When the owner is offline, the RepairVault contract (*LIVE*) holds an owner-funded budget. Up to 16 named delegates holding a signed repair ticket (*BUILT*) may open a replacement deal for a committed shard, within the owner's price and term caps. A delegate handles ciphertext only and holds no wallet, vault root or file key of the owner.

Reads are ordinary block requests, and every block is checked against its content id. Retrieval probes (*BUILT*) test this path: the owner's node fetches short random runs of placed shards, at most 48 probes and 64 MiB a day. Three failed probes of one shard over at least 2 h hand it to repair. Probe scores gate nothing yet.

For people without a node, a storage broker (*LIVE* on testnet) keeps a hot cache and two copies on the Filecoin Calibration testnet. Space is accounted with blind RSA signatures (2048-bit), so the broker cannot tie stored blocks to the Ghost ID whose allowance covers them.

Limitations

- A mainnet deployment or an external audit of these contracts. The vendored BLS verification library is not audited upstream.
- Operator independence. With one operator on testnet the per-operator limits cannot be met, and pins report "Single-operator risk". Operator labels are declarations, not on-chain facts.
- On-chain proof of distinct copies. The contract proves that someone holds the data each epoch. A non-owner cannot yet read a whitened copy (*PLANNED*).

- Fast detection. A vanished hot provider is detected 12 to 18 h after its last proof.
- Beacon independence. A drand outage longer than about two proof windows fails every deal at once. This is a recorded open issue.
- Early exit from an active deal, automatic renewal, or on-chain retrieval proofs.

7. Network privacy

This section establishes what each party that touches the system can and cannot learn, and where metadata is still exposed.

Three things are protected: the content a person stores and says, the link between a person and what they store or pay for, and the person's keys, which never leave their devices. Content protection is complete at every observer below except the AI operator. Linkage protection is partial, and the table says where.

7.1 Who can learn what

Table 3. What each observer can and cannot learn.

Observer	Can learn	Cannot learn
Storage node	The content ids it holds, their padded size class, the deal's on-chain terms, a one-time deal wallet, the uploader's IP address and peer id during upload	Content, file names, exact sizes, which other nodes hold the rest of the file, the per-deal whitening key
Relay	Sender and recipient Ghost IDs on mail envelopes, mail length to its padding bucket, when a Ghost ID's mailbox is reachable, the signed address list of a mail domain, the full text of a letter sent on to ordinary email, client network address. The CDN tunnel in front of it sees client IP addresses	Content. Who filed a retrieval-probe report: reports are signed with a day key
Network observer or ISP	IP addresses, timing, message length up to its padding bucket	Content, exact length of padded traffic
Recipient of a shared link	Everything the read capability covers, and can pass it on	What else a link holder learns is not documented. Access cannot yet be withdrawn by re-encryption
Group member	Everything under the group root: notes, files, member list, rooms, library. After removal, everything already held stays readable	Content written to the successor group after a re-key under a fresh 32-byte root
Ledger reader	DRIVE 369 Network: hashes and the writing address. Base: provider and root per deal, padded segment counts, the funding graph of deal wallets, the Ghost ID behind a registered name	Documents, messages, names, log entries, identity numbers. Content: proofs publish segments of whitened ciphertext
AI operator (network AI)	The prompt and the answer in clear text. Its log records the first 8 characters of the asker's Ghost ID, the model and the input length	Keys and anything not sent in the request. Its stated policy is no retention; receipts do not claim deletion
The project itself	What a relay, the storage broker and the ledger validators learn, because it operates them today. It can link a seal's writing address to a customer. Aggregate page counts of the public site	Keys and content. There is no admin access. Nothing is counted from the app

The storage broker is a further observer for people without a node. It sees the Ghost ID when storage tokens are issued, and storage accounts, content ids and padded sizes at redemption. Blind signatures (*BUILT*) prevent it from joining the two, except through timing and network address.

7.2 Padding and metadata

File chunks are padded to Padme size classes inside the AEAD (*LIVE*), with overhead of at most about 11%. Sync frames between devices are padded by default once both ends of a link have announced support (*LIVE*); the first frame of a new link is not. Mail is padded by default in the web app (*LIVE*); desktop 0.11.13 still writes it unpadded. Padding of document blocks and presence records is *BUILT* and off by default. It is enabled by a node setting. These channels use power-of-two buckets up to 256 KiB, then Padme, at a measured cost of +52% bytes across four channels.

On chain, new pins use a fresh wallet per deal (*BUILT*, on by default), segment counts padded to buckets, terms rounded up to 1, 3, 6 or 12 months, and deals of one pin opened 10 to 120 minutes apart.

7.3 Transport

Nodes connect over libp2p on TCP and WebSockets, with the Noise handshake authenticating peer ids, Yamux multiplexing, circuit relay v2 and hole punching. Public relays are reached over TLS on port 443. Uploads go directly to the provider. Uploads over a relay circuit and an onion transport are *PLANNED*.

Limitations

- Anonymity toward a provider: it sees the uploader's IP address and peer id.
- A hidden funding graph: deal-wallet top-ups come from the main wallet and are public, as is their timing. The fix (deposit notes, a paymaster) is *PLANNED*.
- Hidden envelope addresses at the relay: it sees who writes to whom until sealed-sender mail is the default.
- Private network AI: the operator reads the prompt.
- Protection of an unlocked vault from malware, or from seizure of an unlocked device.
- Protection from a compromised release build: the app is the trusted computing base.
- Re-protection of the past after a move to new recovery words, or after a group member or device is removed.

8. DRIVE 369 Network

This section establishes what the project's own ledger is, who runs it, what is written to it, and how a third party checks a record without trusting the project's servers.

Seals and anchors are small, frequent and carry no value. The DRIVE 369 Network (*LIVE*, pilot) is a permissioned EVM chain for that load: writing costs nothing, a block is final once committed, and an organisation can hold and query its own copy. The explorer labels it "Testnet: a pilot network, not yet production".

8.1 Consensus and operators

Table 4. Ledger parameters.

Parameter	Value
Client and consensus	Hyperledger Besu, QBFT
Chain id	6290369
Validators	Five; 4 of 5 must sign to commit; tolerates 1 away
Block time and finality	2 seconds; final at quorum commit, no forks
Gas price	0; there is no native coin
Admission	Node allowlist and account allowlist

All five validators are run by the project's own team. One team holds every validator key, so the ledger is not independent of the project today. In fault tests on 13 November 2026 the chain kept producing blocks with one validator stopped, paused without forking when quorum was lost, and accepted a queued transaction after recovery.

The RPC node sits on a private network and answers only requests carrying an RS256-signed token. Public access is read-only, through explorer.ghostdrive.com, with a fixed list of read methods and per-visitor rate limits.

$$f = \lfloor (n-1)/3 \rfloor, \quad q = \lceil 2n/3 \rceil; \quad n = 5 \Rightarrow f = 1, q = 4 \tag{9}$$

A block is final once q validators have signed its commit; fewer than q live validators halt the chain instead of forking it.

8.2 What is written

Three record kinds are plain transactions to inert addresses, carrying fingerprints only:

- Vault seal (*LIVE*): 37 bytes. A tag, a kind byte and a 32-byte root over the heads of every device's signed, hash-chained AI-use log in a vault. A node writes at most one per 60 seconds. Whether customer vaults seal in production today is not established.
- Proof of Origin anchor (*LIVE*): 60 bytes. A tag, the log length and the 32-byte log head, written by the relay at most once per 60 seconds.
- Signing seal (*BUILT*): a Merkle root over up to 4,096 document-signing heads per 60-second batch.

No document, message, name, log entry or identity number is written. Seals come from a node's ledger address, which the operator can link to a customer: they are pseudonymous, not anonymous.

"Fingerprints only" describes these records, not the whole chain. The chain also hosts a service-credits contract for a pilot (*LIVE*, test accounts) and the project's DAO contracts (*LIVE*).

Public checkpoints to Base are paused. A checkpoint service (*BUILT*) wrote the hash of the newest block to Base Sepolia every 10 minutes, as calldata to an inert address. It ran, and it was paused on 2026-11-14. The network currently writes nothing to any public chain. Earlier checkpoints remain listed as history and fix the history up to their block only. Contracts for 10-minute epoch anchors on Base are *BUILT* with tests and not deployed; the design awaits approval.

8.3 Verification and the path to outside validators

A third party has three ways to check a record:

- On the explorer (*LIVE*): for a transaction or block, "Verify a record" shows whether it is on the ledger, how many validator commit signatures its block carries against the quorum, whether it is final, and whether a past checkpoint covers it.
- For a file (*LIVE*): the Notary page hashes the file in the browser, shows the signed origin claims for that SHA-256, recomputes the log to the anchored head, and shows the anchor.
- On your own node (*BUILT*, tested against a validator's own copy): a command-line check reads the signed claim, recomputes the log and reads the anchor from the organisation's own copy of the ledger.

Production is *PLANNED* with one validator per participating institution, each holding its own key, with certificate-based node identity and TLS between validators. An external audit of the validators, their key management and the RPC gateway is being scoped. No audit is claimed until one reports.

Limitations

- Decentralisation. All validators, and the checkpoint key, belong to one team. Nothing stops those validators from rewriting earlier blocks, and no public chain covers blocks written after the last checkpoint.
- Independent time. Block timestamps come from the validators' clocks and are not proven by anything outside the network.
- Truth of content. A record shows that a fingerprint matches what was written. It does not show that the content is true, or who authored or owns it.
- Anonymity. Writing addresses are linkable by the operator.
- An audit or a compliance claim. Neither exists today.

9. Verifiable records

This section establishes what each kind of record proves, what it leaves unproven, and what a verifier must hold to check it without trusting the operator.

Every record is a signed object (*LIVE*): an Ed25519 signature by a Ghost ID over a fixed domain string, the record's kind, and the canonical JSON of its body. One rule applies throughout: a signed record proves who said something and that it was not changed. It does not prove the thing said.

9.1 Proof of Origin

Proof of Origin (*LIVE*) is a public append-only log recording that a Ghost ID signed a file's SHA-256 at a time. Only the hash, size, kind and an optional public title leave the device. Each entry carries the hash of the entry before it, and the relay signs a checkpoint of the count and head after every append.

The head is anchored on the DRIVE 369 Network (*LIVE*) as 60 bytes of transaction data, at most once per 60 seconds when the log has grown. The app shows "anchored" only after it has read the transaction back from the chain and recomputed the log from the entry to the anchored head. A command-line check against an organisation's own ledger node is *BUILT*.

It proves that this Ghost ID signed this hash no later than the log accepted it. It does not prove authorship, ownership or copyright: anyone can register a file they did not make.

$$h_0 = 0, \quad h_i = \text{SHA-256}(h_{i-1} \,\|\, \text{entry}_i), \quad \text{anchor} = \text{tag} \,\|\, n \,\|\, h_n \tag{10}$$

A verifier holding entry i recomputes $h_i \ldots h_n$ and compares h_n with the 60-byte anchor read from the ledger.

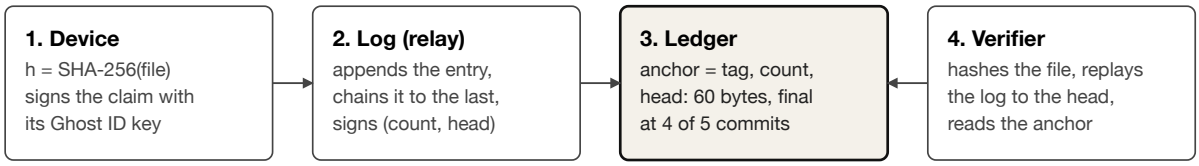


Figure 2. Path of a Proof of Origin record. The file never leaves the device; the ledger carries 60 bytes.

9.2 Document signing

Sign (*BUILT*, not yet released) sends a PDF to up to 50 signers. The sender signs an envelope that fixes the document's SHA-256, the signers, their fields and a deadline. Each step is a signed event carrying the hash of the event before it, so anyone holding the chain can replay it. Step hashes are batched every 60 seconds into a SHA-256 Merkle tree whose root is written to the ledger (*BUILT*).

The signed copy is the original PDF with fields stamped, a certificate page appended, and the full record attached as an Ed25519-signed JSON file. The public checker replays the record on the device, rebuilds the copy deterministically from the original, compares it byte for byte, and reads each seal's transaction from the ledger.

It proves that the named keys signed the original document's hash and their field values. This is not a PAdES or PKCS#7 signature: there is no X.509 certificate and no timestamp token, and a standard PDF reader shows no signature panel. The certificate says so: "This is an electronic signature record, not a certified electronic signature." A guest signature proves only that whoever held the invitation link signed.

9.3 Notary and AI-use logs

In sovereign builds, Proof of Origin appears as a record notary (*LIVE*, pilot) on the private ledger. The AI-use log (*BUILT*) is a per-device hash chain of signed receipts, one each time data leaves its private boundary under a consent grant. A receipt commits to hashes, never to content. The app folds the heads of all device logs into one root and asks the person's node to write it to the ledger every 15 minutes when it changed.

It proves that the sealed part of a log has not been cut or rewritten. It says nothing about the content sent.

Table 5. What each record puts on the ledger and what a verifier needs.

Record	Written to the ledger	Stays on the device	A verifier needs
Proof of Origin	Log count and head, 60 bytes	The file	File or hash, the public log, a ledger read
Signing step	Merkle root of a batch of step hashes	Document, envelope, events (the relay stores sealed bytes only)	Signed PDF, the original, a ledger read
AI-use log	One root over device log heads, 37 bytes	Receipts, in the vault	The logs, the seal record, a ledger read

Limitations

- No time is independent of the operator. Every signer's time is its own clock, and block times come from validators that one team runs.
- Before an anchor covers an entry, its time rests on the relay's signature alone.
- Sign is not a certified electronic signature, and its deployment is not established.
- No currently admitted writer of AI-use seals is recorded, so production sealing of that log is not established.

10. AI and agents

This section establishes where a prompt can go, who can read it there, and what an agent can do without the person.

10.1 Where a model runs

Every model call passes through one router with four providers (*LIVE*): the person's own node, the network's operator nodes, a model in the browser, and the person's own API key. Network AI is off until enabled, and one switch removes both remote providers.

The browser model runs on WebLLM over WebGPU: stock weights from 0.8B to 4B parameters. Nothing is downloaded until the person agrees, and every cached file is checked against a pinned SHA-256 before the model answers. The node serves a model through Ollama on the same machine or an OpenAI-compatible endpoint. Before any provider outside the device sees a prompt, the router requires a consent grant per data source and per provider. Revoking a grant stops future use and cannot recall what was already sent.

10.2 Network AI

An operator node can serve its model to people without a node (*LIVE*, opt-in). The request is signed by the asker's Ghost ID and bound to one operator, one model and a hash of the exact messages and parameters. It is valid for 120 seconds, and a repeat is refused.

The operator returns a signed receipt naming the request, the model, the token counts and the SHA-256 of the answer. The app refuses an answer whose receipt is missing or does not match the text received.

This path is not end-to-end private. The operator's machine reads the prompt in clear text to run the model. The receipt states `not_claimed: deletion, model_identity, accuracy`: it does not prove that nothing was kept, that the named model produced the answer, or that the answer is right. It is evidence of what the operator claims.

10.3 Tools and approvals

The node exposes 13 tools over the Model Context Protocol (*LIVE*): memory, files, search, identity, messages and Office tasks. An agent holds a derived token that opens the tool endpoint and no other route. It never holds the full API token or the vault keys.

Actions reach the person as approval cards (*LIVE*). Four hard stops are computed from what an action does: paying, sending to a new recipient, signing in or typing a secret, and deleting outside the vault. No rule removes them. An "always allow" rule is scoped to one kind, channel and recipient, and capped per day. Sharing a file capability asks every time. A hard-stop card is carried out only on an answer signed by a device key on the person's current signed device list (*BUILT*). Stop Ghost (*LIVE*) is one flag that revokes every rule and denies every waiting card.

10.4 Identity and delegation

An agent has no identity of its own. It acts for the owner's Ghost ID under a mandate (*LIVE*): an object signed by the owner that lists scopes and lasts at most 90 days. On the node, the Ghost Keeper steps down after 14 days without the person, and an Office run ends after 2 hours of work (*BUILT*).Hosting an agent on someone else's node under a separate agent key is *PLANNED*. The agent's model is the node's own Ollama only. An agent-to-agent gateway (*LIVE*, off by default) accepts text into a quiet inbox and never answers alone. Agent payments follow the x402 pattern on testnet (*BUILT*) and are not wired into the app.

Limitations

- Network AI gives no confidentiality toward the operator, and no proof of deletion, model identity or accuracy.
- On the person's own node the mandate is policy enforced by code: that node already holds the vault root.
- Callers of the agent-to-agent gateway name themselves and are not cryptographically authenticated.
- Screening rules protect a person from their own agents. A modified client can skip them.
- Whoever holds a connector link can read memory through the tools.

11. Calculations

Detection of lost data. A provider that no longer holds a fraction f of a deal's segments passes one epoch only if all c challenged segments fall in the part it kept, Eq. (8). Challenges are independent across epochs, so the probability of passing E consecutive epochs is $(1 - f)^{cE}$. With $c = 16$ and a 12-hour cold epoch, Table 6 gives the probability that loss goes unnoticed. A deal fails after two consecutive missed epochs, so a provider missing 10% of a file is caught within one day with probability 0.966, and a provider missing 1% within thirty days with probability above 0.9999.

Table 6. Probability that a provider missing a fraction f of a deal passes every proof.

Fraction missing f	One epoch, $c = 16$	One day cold (2 epochs)	30 days cold (60 epochs)
0.1%	0.984	0.968	0.383
1%	0.851	0.725	6.5×10^{-5}
5%	0.440	0.194	4.1×10^{-22}
10%	0.185	0.034	1.2×10^{-44}
25%	0.010	1.0×10^{-4}	1.1×10^{-120}
50%	1.5×10^{-5}	2.3×10^{-10}	1.0×10^{-289}

Proof size. A proof carries one 1 024-byte segment and one Merkle path per challenge, about $c \cdot (1\,024 + 32 \cdot \lceil \log_2 n \rceil)$ bytes. For a 100 MB deal ($n = 102\,848$ segments, depth 17) and $c = 32$ this bound is 50 176 bytes; the measured proof is 50 048 bytes (Section 12). Proof size grows with the logarithm of the deal size and linearly in c.

Availability of a pinned file. Let each provider be unavailable independently with probability q. A cold file is unreadable when more than 4 of its 14 shards are unavailable; a hot file when all 3 copies are:

$$P_{\text{cold}} = \Sigma_{i=5}{}^{14}\, \mathrm{C}(14, i)\, q^i\, (1 - q)^{14-i}, \qquad P_{\text{hot}} = q^3 \tag{11}$$

Table 7 evaluates both. The cold layout stores 1.4 bytes per byte of data and the hot layout 3; at $q = 0.01$ the cold layout is also the more available of the two, and above about $q = 0.02$ three copies are. The independence assumption is the weak point: it holds only when shards sit with different operators. The per-operator limits of Section 6 exist to enforce it, and on the present single-operator testnet it does not hold.

Table 7. Probability that a file is unreadable, for independent provider unavailability q.

Provider unavailability q	Cold: Reed-Solomon 10 + 4	Hot: 3 copies	Single copy
0.01	1.9×10^{-7}	1.0×10^{-6}	0.010
0.05	4.3×10^{-4}	1.3×10^{-4}	0.050
0.1	0.009	0.001	0.100
0.2	0.130	0.008	0.200

Finality. With $n = 5$ validators the ledger tolerates one faulty validator and commits with four signatures, Eq. (9). Two conflicting blocks at one height would each need four signatures, so at least $2q - n = 3$ validators would have to sign both. With one operator holding every key this is an organisational guarantee and not yet a cryptographic one (Section 8).

12. Evaluation

Table 8 lists measurements of the reference implementation. Client and proof figures are from the project's benchmark suite run on 16 October 2026 on one laptop (Apple M5 Pro, 18 cores, 64 GB); on-chain figures are from the Base Sepolia testnet and a local EVM; model speeds are medians of five runs on the same laptop. They are single-machine results and are reported as such: the swarm figures are loopback and measure protocol overhead, not network bandwidth, and no multi-operator network measurement exists yet.

Table 8. Measurements of the reference implementation.

Quantity	Result	Conditions
Chunk + hash + encrypt, random-key policy	454.2 MB/s	default policy
Chunk + hash + encrypt, convergent policy	422.4 MB/s	opt-in policy
Chunk + hash, no encryption (baseline)	754.7 MB/s	
Storage overhead of sealing	+0.073%	679 chunks, unpadded form
Padding overhead, file chunks	≤ 11%; about 3% at 128 KiB	Padme size classes
Padding overhead, sealed updates	+52% bytes	four channels, 638.5 → 968.5 KiB
Same-owner deduplication (convergent)	4.37×	10 versions of a 4 MB document
Storage Merkle tree build	2.11 s	100 MB, 102 848 segments
Epoch storage proof, 32 challenges	50 048 bytes; verify 13.62 ms	off chain
Proof submission on chain	680k–820k gas; 38–43 KB calldata	Base Sepolia, 32 challenges
Vault index open; name search	8 ms; 11.3 ms	10 000 files, sealed SQLite snapshot
Swarm transfer, 1 peer; 3 peers	162.8; 126.7 MB/s	loopback, one machine
Ledger block time; finality	2 s; at commit	5 validators, QBFT
Browser model, 0.8B / 2B / 4B parameters	63.2 / 12.4 / 3.4 tokens/s	WebGPU, 256 output tokens
Node model, 4B / 9B parameters	26.6 / 16.0 tokens/s	native runtime, same laptop

Sealing costs about 40% of raw chunk-and-hash throughput and still runs an order of magnitude faster than a typical uplink, so encryption is not the bottleneck of an upload. The cost of privacy is in padding, not in cryptography: file chunks pay a few per cent, while small sealed updates pay about half again their size, which is why that padding is still opt-in (Section 4). A storage proof verifies in milliseconds off chain; its on-chain cost is dominated by calldata and bounds how small an epoch can usefully be.

13. Deployment models

This section establishes who runs which part of the system in each way of deploying it, and what state each is in.

The products built on the protocol are Drive, Notes, Mail (business mail on the customer's own domain), Calls, Pass (passwords and secrets), Terminal, Office (agent tasks with approvals), Rooms (research), Radar, Proof of Origin, Sign (*BUILT*), the node with its MCP server, and an SDK (*BUILT*). Unless marked they are *LIVE*; the storage market is live on testnet only.

Table 9. Deployment models.

Model	Who runs the nodes	Who runs the ledger	State
Public network	The project runs the public relays and the first storage providers	Contracts on Base Sepolia; DRIVE 369 Network with five validators, all run by the project team	*LIVE*, testnet
Own node	The person or organisation	As above; an organisation can read the ledger through its own node	*LIVE*
Sovereign or enterprise build	The customer's own node or the operator's network.	DRIVE 369 Network only. One validator per institution is *PLANNED*	*BUILT*, one pilot
Open-source core	Whoever builds it	Not applicable	*LIVE*

Own node. One program takes several roles by configuration. It is installed from source and needs Node.js 22.12.0 or later; the desktop app bundles the same node. A storage-only node refuses to restore a person's vault and serves blocks and mail as ciphertext only. It is the recommended form for servers and relays.

Sovereign build. A brand is chosen at build time. It changes names, languages, theme and hidden pages. No brand field sets relays, bootstrap peers or protocol parameters, so every brand speaks to the same relays and nodes. The one sovereign build that exists is chain-free: it uses the private ledger only, and the app refuses requests to public-chain, wallet and explorer hosts. Its terms call it a pilot with no service-level guarantee.

Open-source core. The public repository contains the core only: the protocol, swarm, SDK and node packages, the contracts and selected documents, under Apache-2.0 or MIT. The applications are not in it. It was first published in November 2026.

13.1 Clients and hosting

The web app (*LIVE*) is a static build served from Cloudflare Workers static assets, with a strict Content-Security-Policy and no third-party scripts. It installs as a PWA. Server code does exist: the Worker that serves the app shell, one edge function that forwards a brand site's sealed forms as ciphertext, a worker that mints one-hour TURN credentials for signed requests, and the relay's services.

The desktop app (*LIVE*, beta) is Electron 44.10.1: the web app in a sandboxed window, plus the node in a utility process. macOS builds are ad-hoc signed and not notarised. Windows builds are an unsigned zip. There is no auto-updater: the app reads a release manifest and offers a link, and the person downloads the build and compares its SHA-256 by hand. The manifest carries no signature.

Limitations

- No deployment today has independent operators. The relays, the first storage providers and all five validators are run by the project.
- A sovereign build does not yet give a separate network. Data-residency statements are capability claims.

- A relay sees connection IP addresses, the sizes and timing of sealed frames, and block identifiers.
- Desktop integrity rests on a manual hash check against an unsigned manifest.
- Everything is testnet. No contract is on a main network.

14. Security assurance

This section establishes what has been checked, how, and what has not.

Internal runs. Three internal security runs were done on 8 and 12 November 2026 with an automated multi-agent review. All three read source only. No target code was executed, so no lead could be confirmed by a run itself. They produced 47 leads needing validation and rejected 8. All 47 are recorded as fixed or mitigated. The second run, on the vault cryptography, group rotation and mailbox sealing, found no candidate that breaks a cryptographic primitive; the weak spots were binding and lifecycle gaps one layer above. The third covered the authority agents get on the person's computer. These runs are not an external audit.

Tests and vectors. The threat model lists 27 invariants, each mapped to the test that states it, and says where no test exists (*BUILT*). Known-answer vectors for protocol v3 (*BUILT*) cover key derivation from a vault root, the content cipher, content identifiers, recovery phrases, storage-proof challenges and the mailbox. A test fails when the implementation drifts from them.

Open source and disclosure. The core is dual-licensed Apache-2.0 or MIT. Vulnerabilities are reported privately through the public repository's security reporting (*LIVE*, as policy), with a target of an answer within 3 days and a fix for critical issues within 14 days.

External audits. An audit of the contracts on Base Sepolia is pending (*PLANNED*). An audit of the ledger's validators, key management and RPC gateway is being scoped (*PLANNED*), with a penetration test to follow.

Limitations

- No external audit of any current contract, the ledger or the applications. None is claimed.
- No run executed code: the internal findings are source review only.
- The vendored BLS library used for on-chain challenge verification is not audited upstream.
- Signed desktop updates and a published build hash are not done.
- No certification is claimed. ISO/IEC 27001 alignment is planned.

15. Settlement

Storage and compute are metered and paid in the network's utility unit. Storage is paid per deal and per proven epoch through an escrow contract (*LIVE*, testnet): the owner funds the term, and each verified proof releases one epoch's payment to the provider. A person without a node buys space from a storage broker with one signature and no gas (*LIVE*, testnet). Network AI answers carry token counts and no price.

Agent and broker payments follow the x402 pattern (*BUILT*). The seller answers HTTP 402 with a requirement: amount, asset, recipient, resource and a validity window. The buyer retries with a signed authorisation in a header. A facilitator checks that recipient, resource and amount match, settles on chain and pays the gas; funds never rest with it, and a replayed authorisation fails on its nonce. Paid plans in a stablecoin on a public main network through a hosted facilitator are *PLANNED*. Everything described here is on testnet and its units carry no value.

16. Status

Table 10 collects the state of every mechanism named in this paper. Every state rests on the project's own records: release notes, deployment records, configuration and code.

Table 10. State of each mechanism on 15 November 2026.

Area	Mechanism	State	Note
Identity	Ghost ID: Ed25519 key, base32, 52 characters	live	One key per vault, derived with HKDF-SHA256
Identity	Vault root as 24 BIP-39 recovery words	live	The operator cannot recover a vault
Identity	Passkey unlock (web), Touch ID (macOS)	live	A passkey wraps the root and never holds it
Identity	Per-device keys and signed device list	built	Deployment not shown
Identity	Social or guardian recovery	planned	No code exists
Storage format	Content-defined chunks, AES-256-GCM blocks, CIDv1	live	Blocks of at most 256 KiB
Storage format	Padme padding of file chunks	live	A holder sees a size class only
Storage format	Random file key by default	live	Convergent keys are opt-in
Storage format	Padding of mail	live	Web app pads mail; desktop 0.11.13 writes it unpadded
Storage format	Padding of document blocks, presence	built	Off by default; readers accept padded data
Sync	Device sync of sealed CRDT documents	live	Sync frames are padded
Sync	Groups with key rotation	live	Only an admin the recipient pinned can rotate
Sync	Group membership proofs a third party can check	built	Signed by an admin, 30 days at most
Sync	Post-quantum message envelopes (X-Wing)	live	X25519 with ML-KEM-768
Sync	Mail between Ghost IDs, addresses on the customer's own domain	live	Web app; desktop 0.11.13 predates it
Sync	Sealed-sender mail envelopes	built	Web app opens them; sending is off until nodes update
Sync	Mail sent on to ordinary email	built	Not end-to-end encrypted; the relay reads those letters
Sync	Calls with end-to-end frame encryption	live	Up to 12 people; exercised on Chromium
Sync	Relay document mailbox	live	Sealed blocks kept 30 days; the web app uses it
Storage network	Escrow deals and per-epoch storage proofs	live	Base Sepolia testnet only
Storage network	Hot tier 3 copies, cold Reed-Solomon 10 + 4	live	Testnet; operator diversity cannot be met yet
Storage network	Repair while the owner is away	live	Testnet
Storage network	Storage broker with blind-signed credits	live	Testnet
Privacy	One-time wallet per storage deal	live	Funding source still visible on chain
Privacy	Storage-only node	live	Holds ciphertext; sees sizes, timing, addresses
Privacy	Onion transport, uploads over a relay circuit	planned	Providers see the uploader's address today

Area	Mechanism	State	Note
Ledger	DRIVE 369 Network: Besu, QBFT, chain id 6290369	live	Five validators, all run by the project team
Ledger	Public checkpoints on Base	built	Paused on 14 November 2026
Ledger	Epoch anchors on Base	built	Contracts and tests; not deployed
Ledger	One validator per institution	planned	Design stage
Records	Proof of Origin log and ledger anchors	live	Proves a signed hash by a time, not authorship
Records	Verification against an organisation's own ledger node	built	Command-line check
Records	Document signing, signed PDF, checker, seals	built	Not PAdES; not yet released
Records	AI-use log seals	built	Not shown sealing in production
AI	Local models in the browser and on the node	live	Weights pinned by SHA-256
AI	Network AI with signed receipts	live	Opt-in; operator reads prompts in clear text
AI	MCP tools, approvals, four hard stops	live	Device-signed answers for hard stops are built
AI	Agent mandate	live	Hosting on a third-party node is planned
AI	Node agent limits: away 14 days, Office run 2 hours	built	Node side; not in desktop 0.11.13
AI	Agent payments (x402 pattern)	built	Testnet; not wired into the app
Deployment	Web app; desktop app (Electron, beta)	live	Desktop unsigned, no auto-updater
Deployment	Open-source core	live	Core packages and contracts, Apache-2.0 or MIT
Deployment	Sovereign build	built	One pilot
Assurance	Internal security runs	live	Source review only
Assurance	External audits of contracts and ledger	planned	None completed; none claimed

17. Conclusion

We have described a system in which an organisation's data, messages and AI memory are sealed on its own devices, stored by parties that cannot read them and are paid only for proven possession, and accompanied by records that a third party can check against a ledger without seeing their content. The parts that carry the confidentiality claim (key derivation, the block format, sealed sync and messages) are live and covered by known-answer vectors. The parts that carry the availability and integrity claims are live on a testnet and on a five-validator ledger that one team operates.

The limits are stated with the same weight as the mechanisms. An AI operator reads the prompts it is sent. A relay sees addresses and timing. A record proves that a fingerprint existed, not that its content is true. The ledger is not yet independent of the project, its public checkpoints are paused, and no external audit has reported. The next steps follow directly: independent validators with one key per institution, external audits of the contracts and the ledger, default padding of document and presence updates, sealed-sender mail by default, and storage providers run by parties other than the project. Until those are done, this paper should be read as a specification with a working reference implementation, and each claim checked against its state in Table 10.

References

1. S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. 2008.

2. H. Krawczyk, P. Eronen. HMAC-based Extract-and-Expand Key Derivation Function (HKDF). RFC 5869, 2010.
3. S. Josefsson, I. Liusvaara. Edwards-Curve Digital Signature Algorithm (EdDSA). RFC 8032, 2017.
4. S. Josefsson. The Base16, Base32, and Base64 Data Encodings. RFC 4648, 2006.
5. M. Dworkin. Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC. NIST SP 800-38D, 2007.
6. NIST. Secure Hash Standard. FIPS 180-4, 2015.
7. NIST. Module-Lattice-Based Key-Encapsulation Mechanism Standard. FIPS 203, 2024.
8. D. Connolly, P. Schwabe, B. Westerbaan. X-Wing: general-purpose hybrid post-quantum KEM. IETF draft-connolly-cfrg-xwing-kem.
9. M. Palatinus, P. Rusnak, A. Voisine, S. Bowe. Mnemonic code for generating deterministic keys. BIP-39, 2013.
10. W3C. Web Authentication: An API for accessing Public Key Credentials, with the PRF extension.
11. ISO/IEC 7816-4. Identification cards: organization, security and commands for interchange.
12. K. Nikitin, L. Barman, W. Lueks, M. Underwood, J.-P. Hubaux, B. Ford. Reducing Metadata Leakage from Encrypted Files and Communication with PURBs. PoPETs, 2019.
13. W. Xia et al. FastCDC: a Fast and Efficient Content-Defined Chunking Approach for Data Deduplication. USENIX ATC, 2016.
14. I. S. Reed, G. Solomon. Polynomial Codes over Certain Finite Fields. J. SIAM 8(2), 1960.
15. R. C. Merkle. A Digital Signature Based on a Conventional Encryption Function. CRYPTO, 1987.
16. P. Maymounkov, D. Mazières. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. IPTPS, 2002.
17. T. Perrin. The Noise Protocol Framework. 2018.
18. D. Chaum. Blind Signatures for Untraceable Payments. CRYPTO, 1982.
19. M. Shapiro, N. Preguiça, C. Baquero, M. Zawirski. Conflict-free Replicated Data Types. SSS, 2011.
20. M. Castro, B. Liskov. Practical Byzantine Fault Tolerance. OSDI, 1999.
21. H. Moniz. The Istanbul BFT Consensus Algorithm. 2020. QBFT as implemented in Hyperledger Besu.
22. drand. A distributed randomness beacon (quicknet, BLS12-381).
23. EIP-712: Typed structured data hashing and signing. EIP-2612: Permit extension for EIP-20 signed approvals. EIP-3009: Transfer with authorization. EIP-2537: Precompile for BLS12-381 curve operations.
24. x402: an open payment protocol over HTTP status 402.
25. Model Context Protocol specification.
26. Ghostdrive: A Privacy-First Chain for Sovereign AI Memory and Verifiable Learning. Technical white paper v2.3, November 2026 (superseded by this paper).