aphanite

V1 ARCHITECTURE WHITEPAPER

Money and messages that are yours alone.

Encrypted messaging where your wallet is your login. The server routes ciphertext it can never read — direct messages use the Signal Double Ratchet, groups use X25519 sealed boxes, and handle→key bindings are written to a public Merkle transparency log.

DOCUMENT	REFLECTS	DATE	STATUS
Architecture & Trust Model	Shipped v1 codebase	July 2026	Technical Preview

V1 ARCHITECTURE WHITEPAPER

Money and messages that are yours alone.

Encrypted messaging where your **wallet is your login**. The server routes ciphertext it can never read — direct messages use the **Signal Double Ratchet**, groups use **X25519 sealed boxes**, and handle→key bindings are written to a public **Merkle transparency log**.

DOCUMENT	REFLECTS	DATE	STATUS
Architecture & Trust Model	Shipped v1 codebase	July 2026	Technical Preview

CONTENTS

SECTION 01

01 · Executive Summary

Aphanite is wallet-native encrypted messaging that is actually built and running: your address is your identity, your device seals every message before it leaves, and the directory that maps names to keys is publicly auditable. This document describes the system as implemented, not as aspiration.

Mainstream messengers tie identity to a phone number and route messages through infrastructure that can read metadata and often content. Aphanite removes both dependencies. Identity is a self-custodied wallet address, so onboarding needs no carrier and no personal data. Message bodies are end-to-end encrypted on the sender's device, so the relay only ever forwards ciphertext it structurally cannot read. And every monetized feature is confined to a perimeter — premium handles, verified

accounts, larger groups, encrypted backup, and a developer API — none of which requires decrypting a message.

The engineering rests on a clean separation of two keys. The **wallet key** proves identity: it signs the Sign-In With Ethereum login and a one-line device-key authorization, and it never performs encryption and never spends. A per-device **X25519 messaging key**, generated in the browser and authorized by that wallet signature, does the actual encryption. Direct messages run the Signal protocols — X3DH then the Double Ratchet — for forward secrecy and post-compromise security; groups and the API use a per-recipient-device sealed box (ephemeral X25519 → HKDF-SHA-256 → AES-256-GCM), so multi-device delivery just works.

The component that could betray users is the directory that maps a handle to a key. Aphanite writes every handle→key and device authorization into an **append-only SHA-256 Merkle transparency log**. A client can request an inclusion proof and check a binding against the published root, and any key change appears as a new, visible leaf. We are candid in §12 about what this log does not yet do — roots are not signed, there is no gossip or consistency-proof protocol, and while established direct conversations are now routed by unlinkable mailbox tags, first contact and groups remain identity-addressed — because a privacy audience will check, and overclaiming is how privacy products lose trust.

THE ARCHITECTURE IN ONE SENTENCE

A wallet address is the root identity; a wallet-authorized X25519 device key seals each message with AES-256-GCM; a Merkle transparency log makes handle→key bindings inclusion-provable; a consent layer with per-sender rate limits gates first contact; and a blind relay stores and forwards ciphertext — addressed, for established direct conversations, by rotating unlinkable mailbox tags — with every revenue stream living outside the content path.

SECTION 02

02 · Design Principles

Four rules constrain every decision in the codebase. Where a principle and a convenient shortcut conflict, the principle wins.

#	PRINCIPLE	WHAT IT MEANS
01	**The server never touches message content.**	Bodies are sealed on the device before upload; the relay forwards opaque ciphertext. The server even runs a structural “proof-of-sealing” check that rejects anything that looks like unencrypted plaintext — without ever being able to read what is sealed.
02	**Money never comes from messages or metadata.**	Revenue is the perimeter — handles, verified accounts, larger groups, backup storage, and the developer API. Every paid capability is realizable over ciphertext and public identity data alone, so the business never has a reason to read content.
03	**Identity trust is made auditable.**	The server maps handles to keys, so in principle it could lie. Every binding is written to an append-only Merkle log with published roots and inclusion proofs, so a lookup can be checked and key changes are visible rather than silent.
04	**The wallet signs; it never spends.**	Logging in and authorizing a device are off-chain signatures — gasless, moving nothing. Aphanite never asks the wallet to send a transaction to use the app, which is both a security property and a trust signal.

SECTION 03

03 · System Overview

Aphanite ships as a single Next.js application that serves the marketing site, the chat app, the developer docs, and the /api/v1 endpoints. The client holds all keys and does all encryption; the server is a blind directory-and-relay.

LAYER	ROLE	GUARANTEE
Connect Wallet	ENTRY	Sign-In With Ethereum proves address ownership with a signature — no gas, no password, no email. Session is an HttpOnly JWT.
Wallet Key	IDENTITY	The address is who you are. The wallet signs a one-line authorization for a device key; it never encrypts and never spends.
Device Key	X25519	Generated in-browser, stored locally. One key per device; the sender seals a copy for each of a recipient's devices.
Handle Registry	+ MERKLE LOG	Human-readable names → keys. Every binding is an append-only SHA-256 Merkle leaf with an inclusion proof.
Consent + Rate Limits	ANTI-SPAM	First contact is a request, not an inbox delivery — gated by recipient approval, spam marks, and per-sender rate limits.
Relay	BLIND STORE-FORWARD	Holds sealed envelopes for offline recipients; client acks decrypted messages; delivered mail is pruned.
Encryption	DR-V1 · BOX-V1	Direct messages: X3DH + Double Ratchet (dr-v1). Groups & API: X25519 sealed boxes (box-v1). All AES-256-GCM.

Figure 1 — The Aphanite v1 stack as implemented. All key generation and encryption happen client-side; the server sees only ciphertext and routing data.

The reference server persists state through a single append-structured store and is deliberately simple; it is the directory, the consent ledger, and the relay queue, but it is never a decryption oracle. The cryptography is built entirely on audited primitives from the `@noble` libraries (X25519, SHA-256, HKDF, AES-GCM, Argon2id) with `siwe` for login and `jose` for sessions — Aphanite composes standard building blocks rather than inventing new ones.

SECTION 04

04 · Identity & Device Keys

A wallet address is the root identity. The wallet key is kept out of the encryption path; a per-device X25519 key does the work, authorized by a single wallet

signature.

Two keys, two jobs. Every user is identified by a self-custodied wallet address over secp256k1. But wallet keys are built for signing transactions — they rotate poorly and live in hardware you don't want in the message hot path — so Aphanite never encrypts with them. Instead, each device generates its own **X25519 messaging keypair** in the browser and stores it locally. The wallet's only cryptographic role in the protocol is to authorize that device key.

Device authorization. To bind a device key to the address, the wallet signs a short, human-readable authorization message — the user sees exactly what they approve (legacy string retained for signature continuity):

THE SIGNED AUTHORIZATION

```
sealed Device Authorization
Address: 0x…
Device messaging key: <x25519-pubkey>
This key may encrypt and decrypt your sealed messages on this device.
It does not authorize spending.
```

The signature is stored with the device record on the server and the binding is written to the transparency log (§6). Because each device is an independently authorized key, Aphanite supports multiple devices per account and per-device revocation without ever moving the wallet key: a sender simply seals a copy of each message to every one of a recipient's active device keys.

Verification & revocation. Users can compare a **safety number**— derived by hashing both parties' device keys — exactly as in Signal, and can run an in-chat **live verification** in which a contact signs a random challenge with their wallet to prove the address behind the conversation. Revoking a device marks it inactive and appends a `device-revoke` entry to the transparency log, so the change is publicly visible to anyone auditing that address.

HONEST NOTE — SENDER AUTHENTICATION

A direct-message session (dr-v1) mutually authenticates both device identities through the X3DH key agreement. A group/API sealed box (box-v1) authenticates the message's integrity but not the sender on its own, so Aphanite asserts the sender at the application layer (the authenticated session and the envelope's sender address) and lets either party prove identity on demand via live wallet-signature verification. See §12.

SECTION 05

05 · Sign-In With Ethereum

Login has no password and no email. Proving control of the wallet is the login, using the Sign-In With Ethereum standard (EIP-4361) — and it never costs gas.

The flow is a standard challenge–response. The client requests a one-time **nonce** (a random 128-bit value that expires in ten minutes and can be used once). The wallet signs a human-readable SIWE message embedding that nonce and the site's domain. The server verifies the signature, confirms the nonce is unused and unexpired, marks it burned, and issues a session as a signed, HttpOnly cookie valid for two weeks. Sessions are individually revocable, and users can see and end active sessions from settings.

> CRITICAL RULE — THE WALLET SIGNS, IT NEVER SPENDS
>
> Connecting and signing SIWE messages costs nothing and moves nothing on-chain. Aphanite never asks the wallet to send a transaction to log in. The nonce's single-use, time-boxed lifecycle is what stops a captured signature from being replayed, and the domain binding in the SIWE message is what stops a phishing clone from reusing a signature on another origin.

One wallet can also **link** additional wallets to a single identity: a linked wallet signs a binding statement, and thereafter signing in with it resolves to the same primary account — useful for users who hold multiple addresses but want one inbox.

SECTION 06

06 · Handle Registry & Transparency Log

The registry maps a handle to an address and its device keys. Because it is a service Aphanite runs, every binding is written to an append-only Merkle log so lookups can be checked and key changes are visible.

A handle is a short human-readable name (three to twenty characters, `a–z 0–9 _`), unique and one per address. Claiming a handle, and every device authorization or revocation, appends a leaf to a single **append-only SHA-256 Merkle tree**. After each append the tree's root is recomputed and published with its size and timestamp; the public transparency endpoint returns the current root, recent entries, and — for any handle or address — an **inclusion proof** that the binding is present under that root.

- **Auditable bindings.** A client can fetch and verify an inclusion proof before trusting a key, checking that the handle→key binding it was served is actually in the published tree.
- **Visible key changes.** A new device or a revocation is a new leaf, so a contact's key rotation is observable — the basis for a "safety number changed" warning.
- **Free users included.** The device-key registry is independent of handles, so even users who never claim a handle have an authorized, logged key that others can encrypt to.

WHAT THE LOG DOES AND DOES NOT YET GUARANTEE

The current log proves inclusion under a published root and makes key changes visible. It does not yet sign its roots, publish consistency proofs between roots, gossip roots between clients, or blind handle indices — so detection of a server that shows different histories to different people is not yet cryptographically enforced. Hardening the log (signed roots, consistency proofs, gossip, optional on-chain anchoring) is on the roadmap (§13). We state this plainly rather than imply a stronger property than the code provides.

SECTION 07

07 · Consent, Rate Limits & Anti-Spam

Open address-to-address messaging is a spammer's paradise. Aphanite gates first contact behind recipient consent, backs it with recipient-controlled spam marks and hard blocks, and throttles abuse with per-sender rate limits — no payment gates anywhere in the messaging path.

A sender's first message to someone new does not land in their inbox; it becomes a **request**carrying an encrypted intro. When the recipient accepts, both parties are added to each other's allowlist and messages flow normally thereafter. All of this consent state gates delivery, never content, so it is fine for the server to hold — knowing that A may reach B reveals nothing about what they say.

Spam marks and blocks. On top of consent, recipients hold two unilateral controls. Marking a sender as **spam** silently stops delivery from that sender — the sender is not notified, there is no path to buy or work around the mark, and only the recipient can lift it. A **hard block** stops delivery entirely. Both are delivery-layer state: they gate routing, never content.

RATE LIMITS INSTEAD OF PAYMENT GATES

Per-sender rate limits cap how many requests any one sender can originate, so mass outreach is throttled at the relay before it reaches anyone's Requests queue. Combined with consent and recipient-controlled spam marks, this keeps the messaging path free of payment gates entirely: no one can pay to reach you, and no one has to pay to be heard.

SECTION 08

08 · Message Encryption & Transport

Messages are encrypted on your device with audited primitives and routed by a relay that stores and forwards ciphertext only until it is delivered. Direct messages and groups use two purpose-built encryption suites.

Direct messages — the Signal Double Ratchet (dr-v1). One-to-one conversations use the same protocols behind Signal and WhatsApp. The first message performs an **X3DH**key agreement against the recipient's published prekeys, and the conversation then runs the **Double Ratchet**: a fresh key is derived for every single message and thrown away after use, and new key material is continuously mixed in as the two sides exchange. This gives two strong properties — forward secrecy (stealing today's keys can't unlock yesterday's messages) and post-compromise security (a conversation automatically heals itself after a compromise). Sessions live encrypted on your device.

Groups & the API — sealed boxes (box-v1). Group messages and the developer API use a per-message **sealed box**: the client generates a throwaway ephemeral X25519 keypair, agrees a shared secret with each recipient device key, derives a 256-bit key with HKDF-SHA-256, and encrypts with AES-256-GCM. A group message is sealed once per member device, so the content guarantee is identical to a 1:1 chat. Discarding the ephemeral key gives sender-side forward secrecy; because there is no ratchet here, groups don't yet self-heal after a device-key compromise the way direct messages do (see §12).

The blind relay. Sealed envelopes are posted to a store-and-forward relay that holds them for offline recipients. Delivery is confirmed by the client: the relay hands over pending envelopes without deleting them, and only marks them delivered once the client acknowledges the specific messages it actually decrypted — so a session with a stale key can never destroy mail meant for another device. Delivered envelopes are pruned after seven days and everything is pruned after thirty, so the relay is a short-lived buffer, not an archive. The server also runs a structural **proof-of-sealing** check, rejecting any envelope whose fields are not shaped like real sealed-box output —

catching a client that tries to send plaintext by mistake, without the server ever reading what is sealed.

Rich messages over the same channel. Delete-for-everyone, disappearing (burn-on-timer) messages, group fan-out, and live identity verification all ride inside the same encrypted channel as structured payloads, so the server sees only ciphertext for these control actions too.

Unlinkable mailbox tags & sealed sender. Once a direct conversation is established, the two clients exchange a per-conversation secret via an encrypted control message inside the channel, and delivery moves off identity addressing. Each side derives a rotating mailbox tag as `HMAC-SHA256(per-conversation key, epoch day index)`, rotating daily; envelopes then carry only `{tag, ciphertext}`, submitted and fetched over unauthenticated, sessionless calls that carry no session identity. A recipient pulls and acknowledges mail by naming the tag; per-device routing uses HMAC-derived device hints rather than device keys. The relay therefore stores ciphertext under an opaque label — no sender, no recipient, no device identity — and cannot see who is talking to whom in an established conversation. Plaintext lengths are hidden by padding messages to fixed-size buckets, and clients poll on a jittered schedule to blunt timing correlation.

WHAT THE RELAY CAN STILL SEE

Message content is unreadable to the relay in every mode. For established direct conversations, sender, recipient, and device identity are hidden behind rotating tags as described above. The residuals: first contact (consent requests) and group messages remain identity-addressed in v1, and the relay still observes client IP addresses and coarse timing — running the connection over Tor or a VPN is the user-side mitigation for IP exposure. Tag-addressing for groups is on the roadmap. This is named as a residual in §12, not glossed over.

SECTION 09

09 · Groups

Group chats reuse exactly the same sealed-box machinery: a group message is fanned out as ordinary encrypted envelopes to each member's devices. Message bodies never touch the group record.

A group stores only its membership and metadata — name, visibility, creator, an optional avatar, and an optional password hash. To send to a group, the client seals the message pairwise to every member device, tagged with the group's identity so the recipient files

it correctly. This keeps the content guarantee identical to one-to-one chat, at the cost of sending one sealed copy per recipient device.

Groups can be public or invite-only, and public groups may be gated by a join password. Free accounts can host groups up to eight members; the larger-groups entitlement raises the creator's cap to sixty-four. Official, verified system groups (such as a curated news channel) carry a checkmark and float to the top of discovery. Group creators can also mint scoped **bot tokens** so automated services can post into a group through the developer API.

DESIGN TRADE-OFF

Pairwise fan-out keeps groups trivially consistent with the one-to-one security model and required no new cryptography to ship. The cost is that sending scales with the number of member devices, which is why group sizes are capped. A tree-based group scheme is a natural future optimization, noted in the roadmap.

SECTION 10

10 · Backup & Multi-Device

History lives on your devices. Optional backup is engineered so the server stores only ciphertext it cannot read — privacy and a sellable feature pointing the same direction.

A backup encrypts the device's full Aphanite state — messages, settings, contacts, and the device keypair itself — under a key derived from a user passphrase with **Argon2id**, the memory-hard password KDF, and then AES-256-GCM. Only the resulting opaque blob is uploaded; the server holds bytes it cannot decrypt, and without the passphrase the backup is unrecoverable. Restoring on a fresh browser re-creates the exact messaging identity and can immediately decrypt history. Backups can run automatically on a schedule or be exported as a file.

Multi-device follows from the same design: each new device generates its own wallet-authorized key, and either an existing device or an encrypted backup provisions its history. No plaintext ever transits the server.

THE ELEGANT PART OF THE MODEL

Encrypted backup is simultaneously the most-wanted convenience feature and a perfect fit for "money never touches content." The server charges for storage and sync while holding an opaque blob — privacy and revenue point the same way, so they never come into tension as the product grows.

SECTION 11

11 · The Paid Perimeter & Developer API

Aphanite makes money without reading a word. Every paid feature is an entitlement unlocked by an on-chain payment and metered on identity, capacity, or convenience — never on content.

FEATURE	WHO PAYS	TOUCHES CONTENT?
Premium handles	Prosumers	**No** — public directory entry
Verified accounts	Businesses	**No** — checkmark / trust layer
Encrypted backup & sync	Prosumers	**No** — server holds opaque ciphertext
Larger groups (to 64)	Prosumers / teams	**No** — metered on capacity
Developer API keys	Developers	**No** — routes sealed envelopes only
Group bot tokens	Developers / teams	**No** — scoped posting into a group

Entitlements are recorded per address by an on-chain payment, then checked at feature boundaries. The **developer API** exposes a versioned `/api/v1` surface authenticated by `sk_live_` Bearer keys (stored only as salted hashes, shown once at creation, rate-limited, and revocable). It lets builders introspect their own identity, resolve public handles and device keys, and submit or pull already-sealed envelopes — the same guarantee sold to developers, since the API never sees plaintext. A published OpenAPI description documents the endpoints for directory lookups, key resolution, envelopes, and groups.

Consumer chat is free forever: it is the top of the funnel and the source of reputation and network effects. The business is the perimeter around it.

SECTION 12

12 · Threat Model & Honest Limitations

A security audience will check every claim, so we state exactly what Aphanite protects today, what it still trusts, and what is explicitly out of scope for v1.

Protected — message content

Bodies are end-to-end encrypted: direct messages with X3DH + the Double Ratchet (forward secrecy and post-compromise security), groups and the API with X25519 + AES-256-GCM sealed boxes. The relay, the directory, and the backup store all see only ciphertext.

Shrinking — the relay's view of the social graph

Established direct conversations use sealed sender with unlinkable mailbox tags: envelopes carry {tag, ciphertext} only, over sessionless calls, so the relay cannot see who is talking to whom, and bucket padding hides message lengths. The residuals: first contact (consent requests) and groups are still identity-addressed in v1, and the relay still sees client IPs and coarse timing — Tor or a VPN is the user-side mitigation for IP. Group tag-addressing is on the roadmap.

Still trusted — the directory, partially

The transparency log proves inclusion and makes key changes visible, but roots are not yet signed and there is no consistency-proof or gossip protocol, so a malicious server showing divergent histories is not yet cryptographically detectable. Hardening is on the roadmap.

Trade-off — post-compromise security is direct-message-only

Direct messages (dr-v1) ratchet and self-heal after a compromise. Group and API messages (box-v1) are independent sealed boxes with no ratchet, so a compromise of a device's long-term key exposes the group messages encrypted to it until the key is rotated. A ratcheted group suite is on the roadmap.

Cost of real E2EE — key loss

Lose every device and the backup passphrase, and history is gone. That is the price of encryption the server cannot break. Recovery UX makes this explicit so users opt in knowingly.

Operational — reference server

The v1 server is a straightforward reference implementation and requires standard production hardening (secret management, durable storage) before scale.

SECTION 13

13 · Roadmap

A disciplined v1 shipped the smallest system that is genuinely private and genuinely useful. The next increments harden the trust model and shrink the metadata surface.

Shipped in v1

- SIWE login with revocable sessions and linked wallets
- Wallet-authorized X25519 device keys, multi-device
- Handle registry + append-only Merkle transparency log
- Consent/requests + spam-mark, block, and rate-limit anti-spam
- Unlinkable mailbox tags + sealed sender for established direct conversations, with bucket padding and jittered polling
- Double Ratchet direct messages (`dr-v1`); sealed-box groups & API (`box-v1`)
- Blind store-and-forward relay; pairwise-fanout groups with verified system groups
- Argon2id-encrypted client-side backup & restore
- Paid entitlements, verified accounts, and a `/api/v1` developer API

Next

- Signed transparency roots, consistency proofs, and gossip
- Optional on-chain anchoring of log roots
- Tag-addressing for groups and first contact
- A ratcheted group suite for post-compromise security in groups
- Tree-based group encryption to lift member caps
- Independent security review of the crypto and directory

CLOSING POSITION

Aphanite's bet is that a messenger can be private by construction and profitable at the perimeter at the same time. The transparency log makes the private core auditable, the wallet-to-device hierarchy keeps the root key out of harm's way, and the paid perimeter proves privacy and revenue need not be in tension. What remains trusted is named and bounded — and the roadmap is the plan to shrink it.

About this document. Aphanite — v1 Architecture Whitepaper, July 2026. This edition describes the architecture as implemented in the current codebase. It is a technical and strategic overview, not a security audit; the cryptographic and directory constructions should be independently reviewed before high-risk use.