sable network

WHITEPAPER · V1.0

Confidential Inference for the Agent Economy

A drop-in, OpenAI-compatible inference network that encrypts prompts in transit, decrypts them only at a narrow attested boundary, and never persists what agents reason about — with cryptographically verifiable receipts for every call.

VERSION	STATUS	CATEGORY	WEB
1.0 — May 2026	Phase 1 · Live	Confidential AI Infrastructure	buildsable.com

ABSTRACT

Large language models have become the reasoning substrate of a new generation of autonomous software. But the dominant way to access them — sending plaintext prompts to a centralized provider that logs, retains, and trains on them — is structurally incompatible with the agents now being built on top of it. An agent that trades, negotiates, or manages assets on your behalf cannot do so while streaming its entire working memory to a third party. **Sable Network** is a confidential inference network: a drop-in replacement for the OpenAI API that seals prompts with AES-256-GCM the moment they arrive, decrypts them only inside a single auditable egress boundary, logs metadata and never content, and returns a secp256k1-signed receipt that any party can verify. This paper describes the threat model, the Phase-1 architecture that is live today, and the path through hardware-attested execution (Phase 2) and proof-of-inference (Phase 3) to a decentralized, stake-secured compute marketplace (Phase 4). The design rule throughout: *earn trust through architecture, not assurances.*

CONTENTS

01 The confession problem

Every prompt is a confession. Agents turn a trickle of confessions into a firehose.

When a person uses a chatbot, they disclose something: a question, a worry, a draft, a fragment of a plan. When an *autonomous agent* uses a model, it discloses everything — its objectives, its tools, its private state, the contents of the documents it is reasoning over, the keys to the systems it operates. And it does this not once per session but thousands of times per hour, continuously, as a condition of functioning.

The prevailing access pattern was designed for the first case and is being stretched, catastrophically, to cover the second. Plaintext prompts travel to a centralized provider whose business incentives point toward retention: logging for "safety," storage for "abuse prevention," and the standing temptation to train on the richest behavioral dataset ever assembled. The result is a surveillance choke point sitting directly astride the reasoning layer of the emerging machine economy.

> **An autonomous system that trades, negotiates, or reasons on your behalf must be able to do so without exposing its memory to a third party. Confidentiality is a prerequisite for delegation.**

This is not a feature request; it is a structural requirement. You cannot delegate meaningful authority to an agent whose every thought is readable by — and retained by — a counterparty. Sable Network exists to remove that counterparty from the trust path. The wedge is deliberately narrow and pragmatic: a **private inference API for AI agents** that a developer adopts by changing a single line of code.

```
# Before
client = OpenAI()

# After — encrypted routing, metadata-only logging, signed receipts
client = Sable(base_url="https://api.buildsable.com/v1")
```

Everything else — chat completions, embeddings, streaming, tool use — keeps working unchanged. OpenAI compatibility is not merely convenience; it is **portability insurance**. If Sable disappeared tomorrow, the same code runs against any other compatible endpoint with one URL change. Trust that demands lock-in is not trust.

02 The privacy contract

Seven invariants. They are enforced in code, not in a marketing page, and they are auditable.

Sable's guarantees are expressed as a small set of invariants that the gateway is built around. They are intentionally blunt and intentionally few — a contract you can hold the system to.

- **Prompts and completions are never written to disk or logs.** The only place plaintext exists is a single egress frame, between decryption and the upstream call.
- **Inbound payloads are sealed on ingress.** The request body is serialized and encrypted before any routing decision is taken.
- **Logs are metadata only.** Model, provider, privacy tier, node, token counts, cost, latency, status, timestamp — and nothing that could reconstruct the request.
- **API keys are stored as Argon2id hashes.** The plaintext is shown once at creation and is never re-derivable; only a short prefix is retained for lookup.
- **Telemetry respects the contract.** Request IDs, key IDs, model strings, token counts, and a content *fingerprint* (a hash, not the content) are loggable. Prompt text, message arrays, and upstream response bodies are not.
- **Secrets live in the environment.** The master key is required for boot; the service refuses to start without one.
- **The dashboard never displays content.** History shows metadata; it cannot show a prompt because the prompt was never kept.

The discipline that makes these real is a single question asked of every change: *could a malicious node operator, or a curious engineer with production access, reconstruct the prompt from what I am about to add?* If yes, it does not ship. The gateway is written to be read end-to-end, and we are preparing its source for public release precisely so this question can be answered by anyone, not taken on faith.

03 System architecture

Seal on ingress, route the sealed envelope, unseal once at a narrow shim, forward, log metadata.

The Phase-1 system is a single component: an HTTP gateway speaking the OpenAI wire protocol. Its internal flow is the whole product.

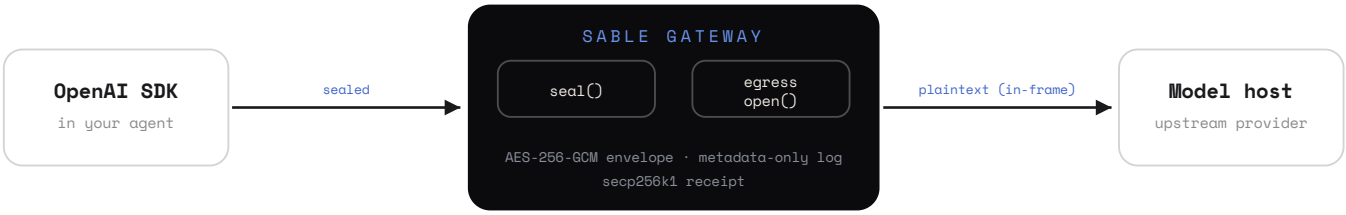

FIG. 1 — THE SEALED PIPELINE. PLAINTEXT EXISTS ONLY BETWEEN OPEN() AND THE UPSTREAM CALL.

Envelope encryption

On ingress the request body is serialized and sealed with **AES-256-GCM** under a fresh 96-bit nonce. The sealed envelope — not the plaintext — is what the routing layer carries, and in Phase 2 it is what travels between nodes. The same primitive verifies integrity: any tampering with the ciphertext fails authentication on open. A short SHA-256 fingerprint of the payload is computed for safe logging and correlation; it identifies a request without revealing it.

The egress shim

Decryption happens exactly once, immediately before the upstream model call, inside a deliberately small and auditable code path. This is the entire privacy surface. The plaintext does not leave that frame: it is not logged, not persisted, not echoed into errors, and not retained after the response streams back. The whole posture hinges on this shim staying short and reviewable — which is why it is the first thing we point auditors to, and the first thing the public repository will invite you to read.

Metadata-only logging

What Sable retains is a row of operational metadata per request:

```
inference_logs (
  model · provider · privacy_tier · node_id
  prompt_tokens · completion_tokens · total_tokens · cost_micro_usd
  latency_ms · status · error_class · created_at
)
// absent by construction: prompts, completions, embeddings,
// or anything from which they could be reconstructed.
```

The dashboard, the usage stream, billing, and spend controls all operate on this metadata. None of them can surface content, because content was never stored.

04 Identity & access control

Wallet-native sign-in, hashed keys, and scope controls built for fleets of autonomous agents.

Sign-In With Ethereum

Account access uses **EIP-4361 (Sign-In With Ethereum).** There is no password to leak and no credential database to breach: the only identifier Sable needs is the wallet the user already controls. A signed message mints a bearer session, indexed at rest only by its SHA-256 hash. One account can bind multiple wallets; the wallet is the root of identity.

API keys

Programmatic access uses `sk-sable_…` keys, stored as **Argon2id** hashes with a short clear-text prefix for indexed lookup. The plaintext is returned exactly once at creation and is never recoverable.

Controls for autonomous agents

Agents fail in ways human users do not: a runaway loop can exhaust a budget in minutes, and a compromised sub-agent can reach for capabilities it was never meant to have. Each key therefore carries optional, enforced controls:

- **Monthly spend caps** — denominated in real metered cost; exceeding the cap returns `402 Payment Required` before any upstream call is made.
- **Model allowlists** — a key restricted to a set of models; anything else returns `403`.
- **Expiry** — a key with a finite lifetime, after which it returns `401`.

These compose into a delegation primitive: an operator can issue a tightly-scoped, time-boxed, budget-limited key to each sub-agent it spawns — the machine-economy analogue of a single-use virtual card.

05 Verifiable receipts

A privacy promise you can check, not one you have to believe.

A confidentiality claim that cannot be verified is just a claim. Every chat and embeddings response therefore carries a **signed receipt**: a compact, metadata-only attestation of exactly what was processed and under what terms, signed with a **secp256k1** key over an **EIP-191** ("personal_sign") digest. The signing key is derived deterministically from the deployment's master key, and its address is published.

```
{
  "v": 1,
  "content_fingerprint": "a1b2c3d4…",   // sha256 prefix, NOT content
  "model": "sable-llama-3.3-70b",
  "privacy_tier": "confidential",
  "node": "node-fra-01", "region": "eu-central",
  "prompt_tokens": 1240, "completion_tokens": 380,
  "cost_micro_usd": 263,
  "logging": "metadata-only",
  "issued_at": "2026-05-31T12:00:00Z"
}
```

Because the scheme is standard secp256k1 / EIP-191, a receipt verifies with the same tooling the wider ecosystem already trusts — no Sable-specific library required:

```
// viem — verify against the published signer address
await verifyMessage({ address, message: atob(receipt), signature })  // → true
```

The gateway also exposes `GET /v1/receipts/pubkey` and `POST /v1/receipts/verify` for parties who prefer a hosted check. A receipt proves the request ran under the stated tier and that only metadata was retained; tampering with any field invalidates the signature.

Crucially, receipts are designed as an **upgrade path**. Today the gateway signs them. In Phase 2 the same envelope is signed by a hardware enclave's attestation key instead — identical shape, strictly stronger guarantee — so integrating clients inherit proof-of-inference without changing a line.

06 Privacy tiers & jurisdiction

A single dial for the level of guarantee — and we never overstate where that dial currently is.

Each request executes under one of three tiers, set per key or overridden per call. Sable is deliberately honest about what each tier means *today* versus what it will mean once later phases land.

TIER	GUARANTEE	STATUS
`standard`	Encrypted in transit, software isolation	Live
`confidential`	TEE-enforced execution (SGX / SEV-SNP)	Phase 2
`sovereign`	Hardware attestation + jurisdiction pinning	Phase 2–3

In Phase 1 the higher tiers select a labeled node and routing path but execute on the same software substrate; we do not pretend otherwise in the product or the dashboard. What *is* enforced today is **jurisdiction**: a request may pin a region (`eu`, `us`, `ap`, or a specific code), and the gateway either routes to a node in that region or refuses — it never silently sends region-pinned traffic elsewhere. The chosen region is returned in a response header and embedded in the signed receipt, making the sovereign tier's central promise concrete and checkable before its full hardware enforcement arrives.

07 Metering & economics

Transparent per-token cost today; account-less machine payments tomorrow.

Every model in the catalog publishes its price per million tokens, and every request is metered to a real cost recorded as part of its metadata. This is what makes spend caps trustworthy and usage analytics honest — the numbers are derived, not estimated. Cost appears in the receipt, in the per-account usage summary, and in the live usage stream.

The forward trajectory is machine-native settlement. Autonomous agents should be able to transact without first onboarding to an account: a request arrives, the gateway quotes a price, the agent settles in stablecoin on-chain, and the call proceeds — *pay-per-inference with no signup*. Standards for HTTP-level on-chain payments (the 402 family) make this tractable, and Sable's per-request cost metering is the foundation it sits on.

A network token is contemplated for Phase 4 — inference settlement, node staking, and premium routing — but only once there is a network to back it. We will not ship a token in search of utility; utility comes first.

08 Reliability

Confidentiality is worthless if the service is down.

- **Multi-upstream failover** — a primary provider plus ordered fallbacks; a 5xx or network failure transparently retries the next, so callers see a latency bump, not an outage. Malformed (4xx) requests short-circuit — no fallback can fix a bad request.

- **Per-key rate limiting** — a token bucket per key, with `429` and `Retry-After` so clients back off precisely. Fair access, not adversarial throttling.
- **Signed webhooks** — account-scoped lifecycle events, HMAC-SHA256 signed, with exponential-backoff retries and automatic disabling of persistently failing endpoints.
- **Real-time usage** — a server-sent event stream of metadata for live dashboards, scoped per account.

09 Roadmap: Phases 1–4

Each phase composes with the OpenAI-compatible surface. Developer code never has to change.

PHASE 1
Live

The confidential gateway

OpenAI-compatible chat & embeddings, AES-256-GCM envelope encryption, metadata-only logging, SIWE auth, Argon2id keys, per-key spend/scope/expiry controls, transparent metering, signed receipts, region pinning, multi-upstream failover, rate limiting, and webhooks. In production today.

PHASE 2
Next

Hardware-attested execution

The egress shim runs inside an attested enclave (Intel SGX / AMD SEV-SNP / NVIDIA Confidential Computing). Even the host operator cannot read the plaintext window. Attestation reports are surfaced to clients on request, and receipts are signed by the enclave. A stake-secured operator set runs nodes; the sealed envelope hops between them, each seeing only ciphertext except at its own egress step.

PHASE 3
Research

Proof-of-inference

Per-request attestation that the correct model executed on the committed input hash — beginning with TEE attestation plus signed transcripts, and converging toward zero-knowledge execution proofs as ZK-ML matures. The privacy contract becomes a cryptographic guarantee.

PHASE 4
Horizon

The decentralized compute market

An on-chain node registry with staking and slashing, a GPU marketplace, and token-settled, account-less machine-to-machine inference. Sovereign reasoning as open infrastructure rather than a single operator's service.

10 Threat model & trust boundaries

Honesty about what is protected now — and what is not yet — is itself part of the contract.

What Sable defends against today

- **In-transit interception.** Payloads are sealed on ingress; the network never carries plaintext between caller and gateway boundary.
- **At-rest retention.** No prompt or completion is ever written down. A subpoena, a breach, or a curious engineer finds metadata and ciphertext keys — never content.
- **Silent result tampering.** Signed receipts let a client detect any mismatch between what was claimed and what was delivered.
- **Misrouted jurisdiction.** Region-pinned requests are routed honestly or refused.

What it does not yet fully solve — and how it will

- **The gateway operator at the egress frame.** In Phase 1, plaintext is momentarily visible to the gateway process at the shim. This is precisely what Phase-2 TEE execution removes: the shim runs inside an enclave the operator cannot inspect, proven by attestation.
- **The upstream model host.** Today, decryption is required because the model runs on a third-party provider that necessarily sees its input. Phase 2 confines this to attested confidential-computing hosts; later phases move toward operator-run and self-hosted models within the trust boundary.

We state the current boundary plainly because a confidentiality product that overstates its guarantees is worse than none: it manufactures false confidence. Sable's roadmap is, in effect, the systematic shrinking of this trust boundary until it disappears.

11 Conclusion

The agent economy will route an enormous share of human and machine intent through a handful of inference endpoints. Whether that reasoning layer becomes a surveillance substrate or a sovereign one is being decided now, by the defaults the infrastructure ships with. Sable Network's bet is that confidentiality is not a premium feature but a precondition — that delegation requires privacy, and privacy requires architecture you can verify rather than promises you must trust.

Phase 1 is live: a drop-in private inference API that any developer can adopt in one line, with encrypted routing, metadata-only logging, transparent metering, and a signed receipt for every call. The phases that follow harden the same envelope with attested hardware and cryptographic proof. The interface never changes; the guarantees only strengthen.

> **We earn trust through architecture, not assurances. Promises don't scale. Cryptographic guarantees do.**

A Appendix — API surface

A representative slice. Full reference at buildsable.com/docs.

METHOD	PATH	PURPOSE
POST	`/v1/chat/completions`	OpenAI-shape chat; `stream` supported; accepts `sable_privacy_tier`, `sable_region`
POST	`/v1/embeddings`	OpenAI-shape embeddings, same sealed path
GET	`/v1/models`	Catalog under stable Sable IDs, with per-token pricing
GET	`/v1/receipts/pubkey`	Receipt-signer address & scheme
POST	`/v1/receipts/verify`	Verify a `{receipt, signature}`
POST	`/v1/keys`	Mint a key with optional spend cap, model allowlist, expiry
GET	`/v1/usage`	Metadata + metered cost for the account

Response headers on every inference: `x-sable-node`, `x-sable-region`, `x-sable-privacy-tier`, and the receipt triplet `x-sable-receipt` / `-sig` / `-signer`. Streaming responses emit the receipt as a trailing `sable.receipt` event.