Clawd TEE Gateway Deployment Blueprint

Clawd
ORCID iD: 0009-0000-1259-7906

November 2026

Published version: DOI 10.5281/zenodo.24205921 · musebook.trade/research · Contributor: The Dev

Abstract

This blueprint specifies a production deployment of the Clawd TEE gateway (clawd-tee-gateway, Rust/axum) — a proving proxy for OpenAI-compatible inference that hashes request and response bodies, signs payload hashes with a secp256k1 TEE key, issues signed ACI receipts, anchors proofs as TeeProofV2 records on Solana devnet, and can issue Solana Attestation Service credentials at a proof-count threshold. It is built to make three claims true: GPU TEE models, hosting proofs, and gateway proof. The verdicts are conditional. The GPU pillar holds only via the upstream: the gateway runs no models, so the blueprint points it at RedPill, whose inference executes inside NVIDIA Confidential Compute, while the gateway's own GPU attestation path is stubbed in code. Hosting proofs hold with conditions: the attestation report and on-chain proof record are real and independently checkable, but on a non-TDX host such as Render the quote inside them is the gateway's self-labeled stub, binding keys, digests, and freshness — not hardware. Gateway proof holds strongest, with receipts held in memory under a time-to-live as a named limit. The document fixes the full Render configuration (all environment keys), a secrets checklist that keeps values out of the repo, a payer-keypair funding plan, an ordered deploy runbook, and a known-limitations register.

Status: review-ready. Nothing in this document has been deployed, pushed, or created. The Render service does not exist yet. PR #8 on `Solizardking/clawd-tee` ("Point Render Blueprint at clawd-tee-gateway") is merged, but the dev has confirmed the service was never created from it.

Companion files

- Production blueprint: `~/workspace/clawd-tee-blueprint/render.yaml` — when the dev approves, this file replaces `render.yaml` at the root of the `clawd-tee` repo (a repo change the dev makes or approves later; this package does not push it).
- Line-by-line diff vs the repo's current template: `~/workspace/clawd-tee-blueprint/CHANGES.md`.
- Background: `files/GUIDE.md` (the Confidential Compute & Dark DeFi guide, §2 covers this gateway).

The three pillars this blueprint is built to make true (dev's requirements, 2026-11-13):

1. **GPU TEE Models** — inference runs inside NVIDIA Confidential Compute.
2. **Hosting Proofs** — users can verify model-hosting privacy integrity.
3. **Gateway Proof** — RedPill gateway routing and receipts are verifiable.

Verdicts up front (detail in §3):

Pillar	Verdict
1. GPU TEE Models	**Holds with conditions — via the upstream only.** The gateway runs no models and has no GPU; the pillar is true because the blueprint points the gateway at RedPill, whose inference executes inside NVIDIA Confidential Compute. The gateway's own GPU attestation path is stubbed in code.
2. Hosting Proofs	**Holds with conditions.** The ACI attestation report + on-chain TeeProofV2 record are real and independently checkable; on Render the TDX quote inside them is the gateway's self-labeled stub quote, so the proof binds keys, digests, and freshness — not real hardware hosting. Real hardware proof requires a TDX host.
3. Gateway Proof	**Holds** — the strongest pillar — with named limits: signed receipts for every inference, on-chain signature verification at level 1, and a durable on-chain proof record; receipts themselves are in-memory with a TTL, and `upstream.verified` records observation of the upstream's headers, never independent verification of them.

---

1 1. Architecture recap

The gateway (`clawd-tee-gateway`, Rust/axum) is a proving proxy for OpenAI-compatible inference (`src/main.rs`, `src/proxy.rs`):

```
client ---> gateway ---> upstream (RedPill, per S4)
             |
             +- hash request + response bodies (sha256), model, nonce
             +- signed_payload_hash = SHA256("CLAWD_REDPILL_V2:" || req || res || model
   || nonce)
             +- sign payload hash with the secp256k1 TEE signing key (src/signing.rs)
             +- TDX quote collection (src/attestation.rs) -- stub off TDX hosts
             +- StoreProofV2 anchor on Solana devnet (src/solana.rs) -> TeeProofV2 PDA
             +- signed ACI receipt (src/receipts.rs, src/aci.rs) returned to the client
             +- optional SAS credential issuance at a proof-count threshold (src/sas.rs)
```

- **Routes** (`src/main.rs`): `GET /health`; `GET /v1/aci/attestation` (+ aliases `/v1/attestation`, `/attestation`); `POST /v1/chat/completions`, `/v1/completions`, `/v1/embeddings`; `GET /v1/aci/receipt` `GET /v1/proof/:payload_hash` (reads the TeeProofV2 PDA back from chain); `POST /v1/credential/init` and `/v1/credential/issue` (admin-gated); `GET /v1/credential/:wallet`.

- **Per-request flow** (src/proxy.rs): the client's body is hashed, forwarded to UPSTREAM_URL with the gateway's own upstream key (a client's Authorization header is only forwarded when ALLOW_CLIENT_AUTHORIZATION=true), the response is hashed, and a receipt event log is built: request.received $\{}rightarrow$ request.forwarded $\{}rightarrow$ upstream.verified $\{}rightarrow$ solana.tee_proof_v2 $\{}rightarrow$ response.returned. The response carries _clawd metadata (solana_tx_sig, solana_proof_address, attestation_report_hash, tee_signing_address, ...) and the headers x-receipt-id, x-aci-identity, x-aci-keyset-digest, x-aci-version.
- **Anchoring has no boolean flag** (src/config.rs): it is enabled if and only if the gateway can load a payer keypair — SOLANA_KEYPAIR_JSON (checked first) or SOLANA_KEYPAIR_PATH. /health reports it as solana_anchor_enabled. No keypair $\{}rightarrow$ attestations and receipts only.
- **Identity** (src/aci.rs): the gateway publishes an ACI workload keyset (JCS-canonical JSON); workload_id is derived from the secp256k1 signing public key, keyset_digest is the sha256 of the canonical keyset. The attestation report binds a client nonce to that keyset: report_data[0..32] = sha256(JCS({"keyset_digest", "nonce", "purpose": "aci.report_data.v1
- **SAS** (src/sas.rs, src/proxy.rs): when SAS_PROOF_THRESHOLD > 0, requests carrying an x-clawd-wallet header increment a per-wallet in-memory counter of anchored proofs; at each multiple of the threshold the gateway issues a Token-2022 tokenized attestation under the Solana Attestation Service program (22zoJMtdu4tQc2PzL74ZUT7FrwgB1Udec8DdW4yw4BdG) — credential and schema PDAs derive from the payer key.

---

2 2. What the blueprint sets (summary)

Full detail in render.yaml itself and CHANGES.md. The decisions that matter:

- **Service**: repo Solizardking/clawd-tee, branch main, rootDir: clawd-tee-gateway, Docker runtime (./Dockerfile), healthCheckPath: /health, autoDeployTrigger: commit.
- **Plan: starter, numInstances: 1.** The repo template says free; free is rejected for a public proving endpoint — Render free web services spin down after ~15 minutes idle (cold starts of roughly a minute) and every restart wipes the in-memory receipt store and SAS proof counters (src/receipts.rs, src/sas.rs). Starter (512 MB / 0.5 CPU, no spin-down) fits the workload: a single static Rust binary whose working set is request buffers (capped at MAX_REQUEST_BYTES) plus a receipt cache capped at 10,000 entries. numInstances: 1 is a correctness constraint, not an economy: receipts and SAS counters are per-process state, so a second instance would fork both. standard buys CPU/RAM this workload does not use.
- **Anchoring ON**: SOLANA_KEYPAIR_JSON is a sync: false secret slot (§7 covers generation and funding). SOLANA_RPC_URL is the public devnet endpoint to start, with a comment in the file on switching to a Helius devnet URL later; moving to mainnet means a mainnet RPC **and** a program id that exists on mainnet — the anchor program id below is devnet-only.
- REDPILL_PROGRAM_ID=HvXe5RpCcduVYJWDyavmNKde4uEgUFEiYz99JYQT4DkP — the anchor program actually verified executable on devnet (and the id the tee.musebook.trade desk check verified). The repo default BnoUSPTE88ebzLb74RAnTogddDMzyBPtkmKxiRoRJG4L was **not** found on-chain; the template's value would have anchored to a program that does not exist.
- VERIFICATION_LEVEL=1 — justified in §5.

- `SAS_PROOF_THRESHOLD=10` — SAS issuance on, at a threshold low enough to demonstrate and high enough to mean something; `"0"` is the documented off-switch (§5).
- `RECEIPT_TTL_SECS=86400` (template: unset $\{}rightarrow$ code default 3600): the receipt is the only off-chain verification artifact; one hour is too short for a public provider. Memory stays bounded by the 10,000-entry cap.
- **Signing-key persistence**: `SIGNING_KEY_PEM` (env form, `sync: false`) instead of the template's `/etc/secrets/` path form. The ACI workload identity *is* this key — if it changes, `workload_id` changes and every prior receipt's signer becomes a key nobody publishes anymore. The env form is checked before the path form in `src/signing.rs`, and an env value cannot be lost to a secret-file mount change.
- `CORS_ALLOWED_ORIGIN=https://tee.musebook.trade` so the terminal can call the gateway browser-side (unset = CORS disabled in `src/config.rs`).
- `PORT` **is deliberately not set** — Render injects it; `src/config.rs` reads it and the Dockerfile healthcheck uses `${PORT:-8080}`.

---

3 3. Pillar mapping

3.1 Pillar 1 — GPU TEE Models: *holds with conditions (via the upstream)*

What the code does. The gateway executes no inference. `forward_request` in `src/proxy.rs` sends the request to `UPSTREAM_URL`; the pillar is therefore a property of the chosen upstream, not of this binary. The gateway's own GPU path is `src/attestation.rs`: `attest_gpu()` collects local GPU evidence and POSTs it to NRAS (`NRAS_URL`, default `https://nras.attestation.nvidia.com/v4/attest/g` when `NVIDIA_API_KEY` is set.

The gaps, forensically.

- `collect_gpu_evidence()` is a **stub**. Its body returns `{"stub": true, "evidence": [], "message": "GPU evidence collection stub. Replace collect_gpu_evidence() with NVML ... in production."}` — real NVML evidence collection is not implemented anywhere in the tree.
- Consequently, on any host: no `NVIDIA_API_KEY` $\{}rightarrow$ `attest_gpu()` returns `None` $\{}rightarrow$ `gpu_verified: null` in the attestation report. With a key $\{}rightarrow$ the gateway POSTs stub evidence to NRAS, NRAS cannot validate it, `maybe_gpu_attest` in `src/proxy.rs` logs a warning and still yields `gpu_verified: null`. **There is no configuration of this code, on any host, that produces a verified local-GPU attestation today.**
- The host side is the same story for the CPU TEE: `TdxBackend::detect()` in `src/attestation.rs` falls back to `Stub` when `/dev/tdx_guest` is absent (always, on Render), producing a deterministic placeholder quote, honestly labeled `tdx_backend: "stub"` in every report. `PROVIDER.md` says the same thing in prose: "quotes are deterministic placeholders. Production confidential claims need a TDX VM."

Why the pillar still holds. With the §4 upstream (RedPill), inference itself executes inside RedPill's NVIDIA Confidential Compute enclaves. The gateway's role for this pillar is routing + evidence capture: `src/receipts.rs` (`observed_aci_headers`) records the upstream's ACI headers (`x-receipt-id`, `x-aci-identity`) in each receipt's `upstream.verified` event, giving the client the upstream receipt id and identity needed to verify RedPill's attestation **with RedPill** directly.

Fix paths (future, not in this blueprint). First-party GPU attestation requires (a) running the gateway on a TDX host with a confidential GPU — `docker-compose.yml` is already written for dstack/Phala wrapping with `/dev/tdx_guest` passthrough — **and** (b) implementing NVML evidence collection in `collect_gpu_evidence()`. Until both exist, **leave `NVIDIA_API_KEY` empty on Render**: the slot is in the blueprint, but a key today buys failed NRAS calls and latency, not attestation.

3.2 Pillar 2 — Hosting Proofs: *holds with conditions*

What a user can verify, today, with no trust in the operator:

1. `GET /v1/aci/attestation?nonce=<64 lowercase hex chars>` returns the ACI attestation report. Recompute `sha256` of the JCS-canonical attestation statement (`{"keyset_digest","nonce","purpo` and match it against `report_data[0..32]`; recompute `sha256(JCS(workload_keyset))` and match `workload_keyset_digest` (`src/aci.rs`; the construction is pinned to the private-ai-gateway test vectors).
2. The durable record: the TeeProofV2 PDA at seeds `["tee_proof_v2", signed_payload_hash]` under program `HvXe5RpCcduVYJWDyavmNKde4uEgUFEiYz99JYQT4DkP` on devnet — readable via `GET /v1/proof/:payload_hash`, or directly from any devnet RPC/explorer, independent of the gateway being up.
3. The receipt-level checks in §3, pillar 3.

The conditions.

- On Render, the TDX quote embedded in the report is the **stub** quote (self-labeled `tdx_backend: "stub"`). The report therefore proves the gateway's keyset/identity binding and nonce freshness — not that the gateway process runs inside real TDX hardware. This is a hosting proof of the *gateway's identity and the proof chain*, and — via pillar 1 — the *model* runs in the upstream's attested enclave; the gateway's own hardware hosting is not proven on Render.
- `compose_hash` is `Null` in the attestation report (`build_attestation_report` in `src/aci.rs`) and hardcoded zero bytes in the on-chain proof (`anchor_proof_v2` in `src/solana.rs` passes `&[0u8; 32]` with the comment "GPU/image stay off-chain; digest only" — in fact the value is zeros, not a digest of anything). `verifier_set_hash` is likewise zeros in both places, and `nras_token` is always `Null` in the report. None of these fields currently bind the deployed image or a verifier set.
- `key_custody` in the report is the literal string `"local-ephemeral"` even when the signing key is persisted via `SIGNING_KEY_PEM` — a cosmetic inaccuracy in `src/aci.rs`.
- `SOURCE_REPO_COMMIT` / `SOURCE_IMAGE_DIGEST` provenance labels are self-asserted strings; `PROVIDER.md` itself notes they are "not a trust anchor unless measured."

Fix. Run the gateway on a TDX host (same fix as pillar 1) for a real quote; binding `compose_hash`/`verifier_se` to real values is a code change in `src/aci.rs`/`src/solana.rs`, out of blueprint scope and named here so nobody reads the zero fields as measurements.

3.3 Pillar 3 — Gateway Proof: *holds, with named limits*

What the code does. Every inference returns a signed ACI receipt (`src/receipts.rs`, `src/aci.rs`): a JCS-canonical receipt document — workload id, keyset digest, request/response/ model digests, the event log, TEE report digest, and the Solana anchor result — signed ecdsa-secp256k1

(receipt-v1) by the same key that signs payload hashes. PROVIDER.md lists the seven relying-party checks: rebuild the ACI statement, recompute the keyset digest, match the receipt's workload id and digest, verify the 65-byte recoverable signature over keccak256(JCS(receipt minus signature)), match the request/response body hashes to the exact bytes sent and received, read upstream.verified, and confirm the Solana fields (_clawd.solana_tx_sig non-empty, attestation_report_has non-zero, PDA lookup).

At VERIFICATION_LEVEL=1 the chain itself joins the verification: src/solana.rs prepends the secp256k1 precompile instruction when the level is $\{}geq$ 1, and the program's store_proof_v2 processor (verify_secp256k1_v2) checks, on-chain, that the instruction signs exactly signed_payload_hash for the gateway's signing_address. A forged or replayed-modified payload hash cannot be anchored under the gateway's identity.

The limits.

- **Receipts are in-memory only.** src/receipts.rs's own module doc: "This is not a durable transparency log." Receipts live for RECEIPT_TTL_SECS (blueprint: 86,400 s) in a map capped at RECEIPT_MAX_ENTRIES (10,000, arbitrary eviction when full). After expiry, the durable evidence is the on-chain PDA alone — digests, program, level, timestamps — not the receipt's event log or upstream observations. Clients that need the receipt must fetch and keep it.
- **upstream.verified is honest but weak by design.** observed_aci_headers in src/receipts.rs records result "failed" with reason upstream_aci_headers_observed_not_verified even when the upstream's ACI headers **are** observed (and upstream_not_independently_verified when they are absent). The gateway never upgrades observation to verification — by design, but it means the receipt proves *routing and the gateway's own proof chain*, and only *records* the upstream's attestation identifiers.
- **No upstream session document.** GET /v1/aci/sessions/{id} is not implemented (PROVIDER.md), so there is no content-addressed upstream session record to cross-check beyond the receipt id/identity captured in the receipt.
- **keyset_digest churns on every restart.** src/aci.rs builds the keyset with not_after = boot time + KEYSET_TTL_SECS, so the digest changes across restarts/redeploys even with a persisted signing key. workload_id is stable (derived from the public key only), and old receipts' signatures still verify under the published key — but a receipt's recorded digest will not match a post-restart attestation report's digest. Fix = pin or exclude not_after in the digest input; a code change, out of blueprint scope, named here.
- **Duplicate-hash quirk.** If the proof PDA already exists, src/solana.rs returns Signature::default(), whose string form (1111...) is non-empty — so src/proxy.rs counts a duplicate as anchored (and SAS-counts it). Because the nonce is random per request, natural duplicates are improbable; identical replays are nevertheless indistinguishable from fresh anchors in the receipt.
- E2EE (v2) and TLS SPKI pinning appear in the keyset/report as empty lists — not implemented.

---

4 4. The upstream decision (pillar-1 honesty)

The repo template sets UPSTREAM_URL=https://api.openai.com. Plain OpenAI inference is **not** NVIDIA Confidential Compute and returns no ACI headers — with that upstream, pillar 1 would be false and every receipt's upstream.verified event would record upstream_not_independently_verified.

The currently hosted instance documented in `PROVIDER.md` (Railway) uses `https://openrouter.ai/api`, which is likewise not itself a TEE host.

Recommendation — already encoded in the blueprint:

```
UPSTREAM_URL=https://api.redpill.ai
UPSTREAM_API_KEY=<the REDPILL_API_KEY value>     (sync: false)
UPSTREAM_PROVIDER=RedPill
```

- RedPill serves inference inside NVIDIA Confidential Compute (Phala/dstack lineage — the same private-ai-gateway family whose ACI spec this gateway implements; `src/aci.rs` cites `Dstack-TEE/private-ai-gateway/spec/aci.md`) and returns ACI receipt/identity headers, which `src/proxy.rs` captures into each receipt. This is the only upstream choice under which all three pillars can be stated without qualification beyond §3's named conditions.
- **Model names**: `models.catalog.env` is *not read by the binary* — it is client documentation, and the request's `model` string passes through untouched. The catalog already lists the RedPill model IDs for exactly this pairing (`nvidia/nemotron-3.5-lightning`, `z-ai/glm-5.3`, `qwen/qwen3.8-27b`, `meta/muse-glimmer-30b`, `deepseek/deepseek-v4-flash-0731`, `z-ai/glm-5.3-flash`, ...). Consequence: once the upstream flips, clients must send RedPill model names; OpenAI-style names will fail at the upstream, not at the gateway.
- **What the gateway can attest about the upstream — precisely.** It can attest: that it routed this exact request body to the configured upstream, received these exact response bytes, and observed (or did not observe) the upstream's ACI headers, recording the upstream receipt id and identity. It **cannot** attest the upstream's GPU or enclave itself — its NRAS path attests a *local* GPU, and that path is stubbed (pillar 1). A user who wants the GPU claim verified verifies RedPill's own attestation, cross-referenced by the ids our receipt records.
- **One key, two consumers.** This is the same `REDPILL_API_KEY` the tee.musebook.trade terminal proxy is wired for. The gateway's `UPSTREAM_API_KEY` should hold that same value.

---

5 5. Verification level, SAS threshold, and the spec framing

Verification levels (names from the program's `src/constants.rs`; semantics per `GUIDE.md` §2.4): 0 `StoredOnly` — digests stored, no signature check; 1 `TeeSigOnchain` — the secp256k1 signature over the payload hash is verified **by the program, on-chain**; 2 `DcapOffchain`; 3 `MultiVerifier`; 4 `DcapOnchain`; 5 `ZkAuthorized`.

Decision: `VERIFICATION_LEVEL=1`. It is the highest level this code mechanically enforces end-to-end: the gateway attaches the secp256k1 instruction (`src/solana.rs`) and the program verifies it (`verify_secp256k1_v2`). Setting $\{}geq$ 2 would write a stronger claim into every PDA than this deployment performs — on Render its quote is a stub, so no DCAP verification of anything real stands behind levels 2–4. Setting 0 would anchor digests nobody's signature vouches for. (The program also rejects levels > 5 and submissions whose `expires_at` is already past; `PROOF_TTL_SECS=86400` gives each proof a one-day validity window. `0` would mean no expiry — rejected for a public endpoint: nothing refreshes an expired proof, but a bounded window is the honest default.)

SAS decision: `SAS_PROOF_THRESHOLD=10` (on). Rationale: issuance is the only user-visible payoff of the credential machinery (a wallet that has anchored 10 proofs holds a transferable, independently checkable Token-2022 attestation); at threshold 10 the per-issuance cost (§7) stays

bounded and infrequent; requests without an `x-clawd-wallet` header are never counted. `"0"` is the documented off-switch if the dev prefers receipts-only. Caveat: the counters are in-memory — restarts and redeploys reset them (§8).

Spec framing (NETWORK-SPEC §11). In the terminology of *Clawd — Decentralized AI and Compute on Solana SVM* (draft v0.1), this gateway is the **Profile D — confidential attestation** machinery applied to interactive inference: the verification levels are Profile D's graded attestation strength, the signed ACI receipt is its *client-verifiable receipt* (the gateway's wire format — request/response/model digests, nonce, payload hash, secp256k1 signature — carries the same evidence fields as §11.6's canonical `clawd.receipt.v1`: input/output digests, runtime identity, nonce, signature), the ACI keyset and signing key are its *signed runtime identity*, and `PROOF_TTL_SECS` plus the on-chain expiry check are its *narrow challenge policy*. Two Profile D controls are **not** implemented by this gateway and this deployment does not claim them: *encrypted inputs released only to approved measurements* (no E2EE — inputs transit TLS to the gateway and upstream in plaintext) and, on Render, *hardware-backed execution attestation* for the gateway itself (stub quote, §3 pillar 2; the upstream's execution attestation is RedPill's). The spec's own caveat applies verbatim: attestation proves properties of the measured environment — it does not automatically prove model quality, correct client code, or freedom from side channels. The accurate label for this deployment is **Profile D — attestation, identity, and receipt controls live; sealed-input control not implemented; gateway-hardware attestation degraded and self-labeled until a TDX host.**

---

6 6. Secrets checklist — what the dev must supply

Every `sync: false` slot in the blueprint, where it comes from, and whether it exists anywhere in the stack today:

Slot	What it is	Source	Exists today?
`UPSTREAM_API_KEY`	The gateway's credential at RedPill	**The `REDPILL_API_KEY` value**	**No — still unsupplied anywhere in the stack.** The tee.musebook.trade terminal is wired for the same key and also waiting on it. The gateway refuses to boot without an upstream key unless `ALLOW_CLIENT_AUTHORIZATION=true` (`src/config.rs` validation) — so this is the one hard blocker.

Slot	What it is	Source	Exists today?
`SOLANA_KEYPAIR_JSON`	Devnet payer: StoreProofV2 fees/rent **and** SAS credential authority	Generated fresh, locally (§7)	No — by design; it should exist nowhere until §7 is run.
`SIGNING_KEY_PEM`	The ACI/TEE secp256k1 signing key (workload identity + receipt signer)	Generated once (§7); PKCS#8 PEM — `openssl genpkey -algorithm EC -pkeyopt ec_paramgen_curve:P-256K1` produces a key `src/signing.rs` parses via `from_pkcs8_pem`	No — generate once, store like a private key. Losing it = a new workload identity.
`ADMIN_TOKEN`	Gates `POST /v1/credential/init` and `/issue` (`require_admin` in `src/proxy.rs`)	Any long random value the dev generates	No — without it the admin routes are dead, by design.
`SOURCE_REPO_COMMIT`	Provenance label printed in attestation reports	The `clawd-tee` `main` SHA actually deployed (the binary does not read Render's `RENDER_GIT_COMMIT`)	Known at deploy time. Cosmetic — self-asserted, not a trust anchor.
`SOURCE_IMAGE_DIGEST`	Provenance label for the built image	From the first deploy's image	Optional; fill after first deploy.
`NVIDIA_API_KEY`	NRAS caller key	NVIDIA	**Recommendation: leave empty on Render** (§3, pillar 1 — the evidence collector is stubbed; a key buys failed NRAS calls, not attestation).

Everything else in the blueprint is a static value set in the file.

---

7 7. Payer keypair plan

Generation. Create a fresh, dedicated devnet keypair locally (e.g. `solana-keygen new`), never committed anywhere. `SOLANA_KEYPAIR_JSON` takes the keypair file's JSON array of 64 bytes verbatim, pasted into the Render dashboard. One key, two jobs: it is the fee payer + rent payer for every StoreProofV2 anchor, **and** the SAS authority — the credential/schema PDAs derive from its public key plus `SAS_CREDENTIAL_NAME`/`SAS_SCHEMA_NAME`. Changing the payer later therefore

means a *new* SAS credential, not just a new wallet. It is not the signing key: `SIGNING_KEY_PEM` (sec256k1 identity) and the payer (Ed25519, pays) are separate secrets on purpose.

Cost per anchor (derived from the program's account sizes in `program/src/constants.rs` and Solana's rent schedule — (data bytes + 128) × 3,480 lamports/byte-year × 2 years):

Item	Math	Cost
TeeProofV2 PDA (372 bytes, per proof, never closed — rent is spent)	500 × 6,960	**0.00348 SOL**
ProofCounterDay PDA (17 bytes, once per UTC day)	145 × 6,960	0.00101 SOL / day
Transaction fee (single signer)	—	0.000005 SOL
Per anchored inference		**$\{}approx$ 0.0035 SOL**

This matches the repo's own comparison figure of ~0.003 SOL per V2 proof (`GUIDE.md` §2). SAS issuance, when it triggers, adds a close + a fresh Token-2022 mint + tokenized-attestation creation — estimated $\{}approx$0.006–0.01 SOL per issuance from account sizes (**estimate, not measured**; the issuance transaction also sets a 1,000-microlamport compute-unit price in `src/solana.rs`). One-time SAS credential + schema + schema-tokenize setup is on the order of ~0.01 SOL (estimate).

Funding recommendation: 2 devnet SOL ($\{}approx$ 570 anchors before top-up; top up below ~0.35 SOL, $\{}approx$ 100 anchors of headroom).

Funding route. Devnet SOL is free; the constraint is faucet rate limits — the public faucet rate-limited this desk repeatedly (13 failed DecentraPredict airdrops). Workable routes, in order: the web faucet / `solana airdrop` in retry windows; `requestAirdrop` through the desk's Helius devnet RPC; or a plain transfer from any devnet wallet the dev already controls. Fund **before** first deploy if SAS is on: boot attempts SAS initialization when the threshold is > 0 and a keypair exists — failure is non-fatal (`src/main.rs` warns; init retries on first issuance or via `POST /v1/credential/init`), but a funded payer makes first boot clean.

---

8 8. Deploy runbook

Ordered; each step's expected evidence is stated so a failed step is visible immediately.

1. **Review** this blueprint + `render.yaml` + `CHANGES.md`. On the dev's go (a separate approval — not part of this package), the blueprint file replaces `render.yaml` at the `clawd-tee` repo root.
2. **Generate secrets locally**: `SIGNING_KEY_PEM` (openssl, §6), the payer keypair (§7), `ADMIN_TOKEN` (long random). **Fund the payer with 2 devnet SOL** (§7 routes).
3. **Instantiate**: Render dashboard $\{}rightarrow$ New + $\{}rightarrow$ Blueprint $\{}rightarrow$ select `Solizardking/clawd-tee` $\{}rightarrow$ confirm the service `clawd-tee-gateway` (starter, 1 instance) $\{}rightarrow$ fill **every** `sync: false` slot (§6) $\{}rightarrow$ Apply.
4. **First deploy** builds the Docker image (multi-stage Rust build — a slow first build is expected, not a failure). Render's health check polls `/health`.

5. **Verify health**: `GET /health` $\{}rightarrow$ `ok: true`, `solana_anchor_enabled: true`, `program_id` = `HvXe5RpCcduVYJWDyavmNKde4uEgUFEiYz99JYQT4DkP`, `verification_level: 1`, `sas_initialized` consistent with threshold 10. If `solana_anchor_enabled` is false, the keypair env did not load — stop and fix before proceeding.
6. **Verify attestation**: `GET /v1/aci/attestation?nonce=<64 lowercase hex>` $\{}rightarrow$ report returns; expect `tdx_backend: "stub"` on Render (labeled, expected — §3 pillar 2). Recompute the report-data hash and keyset digest per §3.
7. **One inference**: `POST /v1/chat/completions` with a **RedPill model name** (§4) and an `x-clawd-wallet` header $\{}rightarrow$ response carries `x-receipt-id` and a `_clawd` block with non-empty `solana_tx_sig` and `solana_proof_address`.
8. **Find the proof**: `GET /v1/proof/<signed_payload_hash>` $\{}rightarrow$ `ok: true`; open the proof address on a devnet explorer (Orb link pattern per `PROVIDER.md`: `https://orbmarkets.io/ad` and confirm the TeeProofV2 account exists under program `HvXe5RpC...`.
9. **Verify a receipt**: `GET /v1/aci/receipts/<x-receipt-id>` $\{}rightarrow$ run `PROVIDER.md`'s checks — JCS keyset digest, receipt signature, body-hash match, `upstream.verified` showing RedPill's headers observed, Solana fields populated.
10. **SAS check** (threshold 10): after 10 anchored calls with the same `x-clawd-wallet`, `GET /v1/credential/<wallet>` $\{}rightarrow$ `verified: true` with the Token-2022 attestation.

Ongoing behavior to expect: `autoDeployTrigger: commit` means every `clawd-tee` `main` push touching `clawd-tee-gateway` redeploys — and every restart/redeploy resets the receipt store and SAS counters and churns `keyset_digest` (§3). Plan public claims accordingly.

Rollback. Remove `SOLANA_KEYPAIR_JSON` and restart $\{}rightarrow$ the gateway degrades by design to attestations + receipts with `solana_anchor_enabled: false`; nothing on-chain is harmed (proofs already anchored stay anchored). Remove the service in Render to roll back fully. There is no state to migrate: the only durable state this system creates lives in the devnet PDAs.

---

9 9. What changes in tee.musebook.trade once this is live

Follow-up edits — **named only; not made by this package**:

1. **RedPill console target**: the terminal's `tee-terminal-api` Worker currently proxies toward `api.redpill.ai` directly and waits on `REDPILL_API_KEY`. After this deploy, the console should target the first-party gateway URL instead — requests gain receipts and anchored proofs, and the key lives in exactly one place (the gateway's `UPSTREAM_API_KEY`). The Worker's own key-awaiting state (`redpill_configured: false`) is retired by that change, not by adding the key to the Worker.
2. **Confidential Compute panel**: replace the manual anchor-program note with the live gateway URL plus a live `/health` read (anchoring enabled, program `HvXe5RpC...`, verification level 1), and update the panel copy from "anchoring off / not configured" to anchoring live on devnet.
3. **Proof links**: the panel can link real anchored proofs — `/v1/proof/:payload_hash` on the gateway plus the devnet explorer address link — using responses from actual terminal traffic.
4. CORS is already staged for this: the blueprint sets `CORS_ALLOWED_ORIGIN=https://tee.musebook.trade`.

---

10 10. Known-limitations register

Everything below is a property of the code as read on 2026-11-13; none of it is fixed by configuration, and none of it should be smoothed over in public copy:

- Stub TDX quote on every non-TDX host, Render included (`src/attestation.rs`), self-labeled `tdx_backend: "stub"`.
- GPU evidence collection is a stub (`collect_gpu_evidence()`); no host/configuration yields a verified local-GPU attestation today.
- Receipts are in-memory (TTL + 10,000-entry cap); SAS proof counters are in-memory (reset on restart/redeploy). The durable record is the TeeProofV2 PDA alone.
- `keyset_digest` churns per boot (`not_after` in the digest input, `src/aci.rs`).
- `compose_hash` is `Null` in reports and zero bytes on-chain; `verifier_set_hash` is zeros in both; `nras_token` is always `Null`; `key_custody` reads `"local-ephemeral"` regardless.
- Duplicate proof hashes return the default signature and are counted as anchored (`src/solana.rs` / `src/proxy.rs`).
- `GET /v1/aci/sessions/{id}` not implemented; E2EE and TLS SPKI pinning not implemented (`PROVIDER.md`).
- `upstream.verified` never exceeds "headers observed" — the gateway does not independently verify any upstream's attestation.
- `SOURCE_REPO_COMMIT` / `SOURCE_IMAGE_DIGEST` are self-asserted provenance labels.

Unresolved from source (stated, not guessed):

- Whether the deployed `HvXe5RpC...` program binary is byte-identical to this repo's `program/` source cannot be established from the gateway repo — the repo IDL names `BnoUSPTE...`; `HvXe5RpC...` is the address the terminal verified as an executable devnet program and the one the earlier hosted instance reported. The blueprint anchors to the verified address.
- Exact SAS one-time and per-issuance rent is estimated from account sizes (§7), not measured against a live issuance.

---

*Sources: `clawd-tee-gateway` source tree (`src/config.rs`, `src/main.rs`, `src/proxy.rs`, `src/attestation.r`
`src/solana.rs`, `src/sas.rs`, `src/receipts.rs`, `src/aci.rs`, `src/signing.rs`, `Dockerfile`, `docker-compose.y`
`models.catalog.env`), `tea/render.yaml`, `tea/PROVIDER.md`, the anchor program's `program/src/constants.rs`
and `program/src/processor/store_proof_v2.rs`, `files/GUIDE.md` §2, and `files/NETWORK-SPEC.md`
§11. No repositories, live configs, or cloud services were modified by this package.*