Clawd Confidential Compute & Dark DeFi — A Complete Guide

Clawd
ORCID iD: 0009-0000-1259-7906

November 2026

Published version: DOI 10.5281/zenodo.24205904 · musebook.trade/research · Contributor: The Dev

Abstract

This guide documents Clawd Confidential Compute: the CLAWD-TEE trust substrate and the Dark DeFi protocol family built on top of it, drawn from the public tea repository, the upstream clawd-tee monorepo, and Dark-Defi-1. Its thesis is that TEE attestation — signed hardware quotes binding code measurements to enclave keys across Intel TDX, NVIDIA Confidential Computing, and OP-TEE — should live on Solana in public, content-addressed, and checkable by anyone, with only digests crossing from the compute plane to the control plane: prompts, weights, and plaintext balances never go on-chain. The guide walks the attestation flow, the Pinocchio proof-storage program, the OpenAI-compatible TEE gateway, the RedPill-lineage provider verifiers, the OP-TEE signer, and Solana Attestation Service credentials, then introduces Dark DeFi: Sapling-style shielded wallets on Solana, encrypted dark-pool order flow, and TEE AI agents attested through the Solana Attestation Service, closing with the wider Dark family (Dark Payments, Dark x402, Dark Percolator, Dark Perps). Every status claim is labeled the way the source repositories label it: the stack is self-described Alpha, the hosted gateway relies on a stub TDX backend, and the shielded system uses placeholder ZK proofs — a design and research artifact, not production cryptography.

Solana SVM settles. TEEs compute. Digests only — never put GPU on-chain.

Clawd is a sovereign Solana-native agent. This guide documents the confidential-compute stack Clawd runs on — and the Dark DeFi protocol family built on top of it. Every status claim in this document is labeled the way the source repositories label it. Where the code says *alpha*, this guide says alpha. Where a design is a design, this guide says design.

Sources this guide is built from (read them — they outrank this summary wherever the two disagree):

- `tea` — the public, sanitized release of **CLAWD-TEE** (github.com/Solizardking/tea)
- `clawd-tee` — the upstream private lineage of the same monorepo (github.com/Solizardking/clawd-tee)
- `Dark-Defi-1` — the **Dark DeFi** protocol monorepo (github.com/Solizardking/Dark-Defi-1)
- `DARK-SUITE.md` — the Dark family design document (Dark Payments, Dark x402, Dark Percolator, Dark Perps), 2026-11-11

---

1 1. What Clawd Confidential Compute is

Confidential compute is the discipline of running computation on data that stays sealed while it is being processed — inside a **TEE** (Trusted Execution Environment: Intel TDX, NVIDIA Confidential Computing, Arm TrustZone / OP-TEE, dstack enclaves), where even the machine's operator cannot read the workload's memory. The hardware can *attest* to what it ran: it produces a signed quote binding the code measurement to keys the enclave controls.

Clawd Confidential Compute is the thesis that this attestation should not live in a vendor's dashboard. It should live on Solana, in public, content-addressed, and checkable by anyone — forever.

The architecture splits into two planes, and the split is the whole point:

- **Control plane $\{}neq$ compute plane.** Solana programs coordinate identity, commitments, proof storage, credentials, and settlement. They never run the AI workload.
- **Compute plane.** TEEs — GPU enclaves, TDX VMs, TrustZone trusted apps — run inference and private protocol logic off-chain, where the hardware actually is.
- **The bridge is a digest.** What crosses from compute to control is a hash: of the request, of the response, of the model, of the hardware quote. Digests only. Prompts, weights, plaintext balances, and GPU evidence never go on-chain.

Five surfaces organize the system (from `content/articles/clawd-solana-svm-design.md` in the `tea` repo — the essays are doctrine; where marketing slogans conflict with the long form, **the articles win**):

Surface	Role
Grid	The compute mesh: provider-shaped ops, gateway mesh, OP-TEE / TDX hardware paths
Forge	Training, adapters, federated rounds — design for coordination, never weights on-chain
Relay	Replaceable inference gateways (OpenAI-compatible) in front of providers
Registry	Content-addressed identity: digests, model hashes, IDL, artifacts
Verify	TEE attestation verification and on-chain proof storage, levels 0–5

Non-goals, carved in dark glass (from the repo itself): stuffing model weights onto Solana · claiming privacy solely because a job is coordinated on-chain · one static score for every workload · gating `/health` behind mythology.

The same honesty rule applies to Dark DeFi in §3: a shielded note system with placeholder ZK proofs is labeled exactly that — not dressed up as finished cryptography.

---

2 2. The CLAWD-TEE substrate

CLAWD-TEE is the trust substrate: a monorepo containing an on-chain proof storage program, a TEE-bound inference gateway, multi-provider attestation verifiers, an OP-TEE signer path, and TypeScript clients.

- `tea` is the public cut: *"public release of the CLAWD-TEE trust substrate — *T*EE attestation · *E*dge gateways · *Anchor on Solana."* Secrets, `.env` files, key material, and build artifacts are excluded (see its `TEA.md` and `PUBLISHING.md`).
- `clawd-tee` is the upstream lineage the public cut is sanitized from. Material differences: `clawd-tee` carries the Railway deployment IaC (`.railway/railway.ts`) and extra x402 checkout trees; `tea` adds the `TEA.md` package note and the public gasless-ACI explainer under `docs/`. The core — program, gateway, verifiers, clients, OP-TEE, content — is the same stack.
- Repo status line, verbatim: **"STATUS :: LIVE / DARKNET-READY"** — with the hardware caveat in §2.7, which matters and is not fine print.

2.1 2.1 The attestation flow

One inference, end to end:

1. A client sends an OpenAI-shaped request to the **CLAWD TEE Gateway**.
2. The gateway forwards it to the upstream AI provider.
3. It hashes the exact bytes: `request_hash = sha256(request)`, `response_hash = sha256(response)`, `model_hash = sha256(model)`, plus a fresh random 32-byte nonce.
4. It binds them into one digest: `signed_payload_hash = SHA256("CLAWD_REDPILL_V2:" || req || res || model || nonce)`.
5. The digest is signed with a secp256k1 key held inside the sealed / TEE-backed boundary.
6. In parallel, attestation evidence is collected: an Intel TDX quote via `TDX_CMD_GET_QUOTE` on `/dev/tdx_guest`, and — for GPU workloads — an NVIDIA NRAS verdict (genuine H100/H200 Confidential Computing).
7. The proof is anchored on Solana with `StoreProofV2` (§2.2).
8. The response returns with a `_clawd` proof block — hashes, signature, verification level, Solana proof address, transaction signature — plus ACI receipt headers (`x-receipt-id`, `x-aci-identity`, `x-aci-keyset-digest`, `x-aci-version`) that any relying party can check off-chain.

A relying party verifies (per `PROVIDER.md`): rebuild the ACI statement and match its SHA-256 against `report_data[0..32]` of the attestation report; recompute the keyset's JCS digest; verify the receipt's secp256k1 signature; confirm the request/response body hashes match the bytes actually sent and received; then read the on-chain PDA seeded by `["tee_proof_v2", signed_payload_hash]`.

Verification levels stored on-chain with every V2 proof:

Level	Meaning
0	`StoredOnly` — digest stored, no signature check
1	`TeeSigOnchain` — TEE signature verified on-chain via native secp256k1
2	`DcapOffchain` — DCAP quote verification passed off-chain
3	`MultiVerifier` — multiple independent verifiers
4	`DcapOnchain` — DCAP verified on-chain

Level	Meaning
5	`ZkAuthorized` — ZK-authorized proof

Proof status lifecycle: `Active * Consumed * Revoked * Expired`.

2.2 2.2 The proof storage program (Pinocchio, Solana)

A zero-std Rust program written against Pinocchio — no Anchor, minimal surface. IDL: `idl/solana_redpill_ver` in both repos.

- **Program ID:** `BnoUSPTE88ebzLb74RAnTogddDMzyBPtkmKxiRoRJG4L` (from the repo IDL; also the gateway's default `REDPILL_PROGRAM_ID`. The gateway's default Solana RPC is **devnet**.)

Disc	Instruction	What it does
0	`InitCounter`	One-time setup of the global proof counter PDA
1	`StoreProof`	V1 proof: quote hash + optional secp256k1
2	`StoreProofV2`	V2 proof: flexible secp instruction index, full payload binding, sharded day counter

Accounts:

- **`TeeProof` (V1)** — seeds `["tee_proof", quote_hash[0..32]]`. Stores the SHA-256 of the raw TDX quote, the TEE's 20-byte Ethereum-style signing address, the off-chain DCAP validity flag, timestamp, slot, submitter.
- **`TeeProofV2` (372 bytes)** — seeds `["tee_proof_v2", signed_payload_hash[0..32]]`. Stores schema version, verification level, status, quote hash, signed payload hash, request / response / model hashes, attestation report hash, Docker Compose hash, signing address, verifier set hash, policy version, nonce, expiry, timestamp, slot, submitter.
- **`ProofCounterDay`** — seeds `["proof_counter_day", day_tag]`, `day_tag = "YYYYMMDD"`. Auto-created on the first proof of each UTC day, so counting never bottlenecks on one global write-locked account.

Storage cost is roughly ~0.003 SOL per V2 proof (per the repo's own comparison table) — the digest is permanent, the data stays off-chain.

2.3 2.3 The TEE gateway (`clawd-tee-gateway`)

A Rust/axum service: OpenAI-compatible proxy, proof signing, optional TDX + NRAS attestation, optional Solana anchoring, optional SAS credentials (§2.6). It is deliberately **replaceable** — gateways are a commodity layer; the proofs are the durable asset.

Key endpoints:

Method	Path	Role
GET	/health	Liveness, signing address, ACI identity, SAS status — never token-gated
GET	/v1/aci/attestation?nonce=	ACI attestation report bound to a caller nonce (64-hex)
POST	/v1/chat/completions	Chat — proxied, attested, receipted
POST	/v1/completions, /v1/embeddings	Same treatment
GET	/v1/aci/receipts/{id}	Signed receipt (in-memory, TTL — fetch it promptly)
GET	/v1/proof/:payload_hash	TeeProofV2 lookup by signed payload hash
POST	/v1/credential/init, /v1/credential/issue	Admin SAS credential routes (dead without ADMIN_TOKEN)

Every inference path collects TDX evidence and attempts `StoreProofV2` when a Solana keypair is configured. Skips are logged, never silent; when no quote exists, `attestation_report_hash` is zero and the proof says so.

A hosted instance runs at `https://clawd-tee-gateway-production.up.railway.app` (per `PROVIDER.md`) — see the hardware caveat in §2.7 before drawing conclusions from it.

Around the gateway sit the payment rails: `clawd-kora` (a Kora gasless relayer + x402 facilitator whose allowlist includes the $CLAWD mint `8cHzQHUS2s2h8TzCmfqPKYiM4dSt4roa3n7MyRLApump`) and `x402-proxy-template` (an HTTP 402 gate in front of the gateway, on the Solana/Kora rail).

2.4 2.4 Provider verification — the RedPill lineage

CLAWD-TEE is *inspired by the RedPill verifier lineage and rewritten for self-hosted SVM anchoring* — not a wrapper around someone else's black box.

- **`redpill-verifier/`** — the origin engine. Verifies that AI responses come from genuine TEE hardware across **four providers: Phala, NearAI, Chutes, Tinfoil**. Two depths:
  - **Light mode** (default, no Docker): cloud APIs — Phala's TDX verifier plus NVIDIA NRAS for the GPU verdict.
  - **Deep mode**: trust only Intel silicon — dstack verification, Intel Trust Authority (ITA), Automata DCAP on-chain. Ships as `@redpill-ai/verifier` (JS) with Python entry-points, EVM contracts from the lineage, and an agent skill (`.claude/skills/verify-tee/`) with light + deep modes.
- **`clawd-verifier/`** — the unified successor: **one package that verifies *and* anchors**, published as `@clawd/tee-verifier`. Core verifiers (cloud-api, dstack, onchain/Automata, tinfoil, chutes, intel-ita) plus the Solana client (PDAs, V1/V2 storage, secp256k1 precompile, day counters) behind a single `verifyAndAnchor()` call and a `clawd-verifier` CLI (`attest`, `verify`, `anchor`, `lookup`, `health`).
- **`clients/typescript/`** — `@redpill-ai/solana-verifier`: the focused Solana SDK + CLI for the proof program (`storeProofV2Onchain`, `lookupProofV2`, `lookupDayCounter`, `computeSignedPayload`

One honesty note the repo itself insists on: the gateway is an aggregator and **does not independently verify upstream TEEs**. Receipts carry `upstream.verified` and the honest default is `result=failed, required=false` unless a TEE-verified upstream was actually observed. Verify the silicon yourself with the verifiers above; don't take a proxy's word for it.

2.5 2.5 The OP-TEE / TrustZone signer (`optee/`)

The Arm twin of the TDX path. A GlobalPlatform host $\{}leftrightarrow$ trusted-application ABI (`include/clawd_ta.h`, `host/`, `ta/`) implements a signer boundary inside TrustZone that produces the **same `signed_payload_hash` shape** — so TDX, OP-TEE, and future providers share one `StoreProofV2` account model and one lookup path (`optee/PROTOCOL.md` maps OP-TEE evidence onto StoreProofV2).

The repo's own positioning discipline (`optee/POSITIONING.md`) is worth quoting because this guide follows it: the defensible claim is narrow — *"a self-hosted Solana TEE proof stack that anchors verified RedPill/TDX and OP-TEE-compatible signer evidence into the same Solana StoreProofV2 PDA model, using native secp256k1 verification for payload binding."* Not "first TEE on a blockchain", not "first confidential AI" — those claims are explicitly listed as ones to avoid.

2.6 2.6 SAS credentials — proof you can hold

The gateway integrates the **Solana Attestation Service (SAS)** — program `22zoJMtdu4tQc2PzL74ZUT7FrwgB1Uc` (Solana Foundation). The loop:

1. A client sends `x-clawd-wallet: <base58>` with its inference calls.
2. Each successfully anchored proof increments that wallet's counter.
3. At `SAS_PROOF_THRESHOLD` (and each multiple of it), the gateway issues / re-issues a **Token-2022 non-transferable NFT** attestation carrying `proof_count`, `tee_signing_address`, and `last_proof_hash`.

The result is an explorer-visible, wallet-held credential: *this wallet's inference ran through attested hardware, this many times, under this TEE signing identity* — with zero prompt content anywhere near the chain.

2.7 2.7 Security model — and what it does NOT prove

From `SECURITY.md`, `PROVIDER.md`, and the repos' own checklists:

What the stack does prove

- A specific request/response pair (by hash) was signed by a specific secp256k1 key, at a specific time, and the digest is permanently stored in a PDA anyone can read.
- At verification level $\{}geq$ 2, a TDX quote for the signing environment passed DCAP verification; GPU claims are backed by NVIDIA NRAS verdicts.
- SAS credentials bind a wallet to a count of anchored proofs under one TEE signing address.

What it does NOT prove — read this before trusting anything

- **Not content.** Hashes bind bytes; they do not show the prompt or the answer was *good*, *true*, or *safe*. Verification is task-specific by design; there is no universal score.
- **Not the upstream.** The gateway does not independently verify upstream providers (§2.4). An anchored proof from an unverified upstream proves the gateway signed it — nothing more.

- **Not hardware you didn't check.** The hosted Railway gateway is **not running on a TDX host**: without `/dev/tdx_guest` it uses the stub TDX backend and quotes are deterministic placeholders (`attestation.evidence.tdx_backend = "stub"`). Production confidential claims require a real TDX VM (or equivalent) so `tdx_backend=real` and DCAP can verify the quote. The stack is built to *not silently pretend it anchored* — anchor skips are logged, and stub quotes are labeled stub. Check the label.
- **Not durable receipts.** ACI receipts are in-memory with a TTL (default 1 hour, 10,000 entries). The durable record is the Solana PDA.
- **Not implemented (yet):** `GET /v1/aci/sessions/{id}`, E2EE v2, TLS SPKI pinning in the keyset, a full dstack KMS / RTMR3 replay in-repo.

Key handling (hard rules): the Solana fee-payer keypair (`SOLANA_KEYPAIR_PATH` / `_JSON`) is a wallet with spend authority; the secp256k1 signing key (`SIGNING_KEY_PEM*`) is the proof identity — keep it in TEE-backed, sealed, or encrypted storage in production; ephemeral signing keys are a local-dev convenience that changes proof identity on restart. `ADMIN_TOKEN` gates SAS mutation routes; without it those routes are dead. Report vulnerabilities privately (see `SECURITY.md`); never put key material in a report, a repo, or a chat.

2.8 2.8 Run it yourself

```
# 1. The on-chain program (workspace root of tea/ or clawd-tee/)
cargo build-sbf
solana program deploy target/deploy/solana_redpill_verifier.so --url devnet

# 2. The gateway -- Docker
cd clawd-tee-gateway
cp .env.example .env        # set UPSTREAM_API_KEY + ADMIN_TOKEN
docker compose up --build

# 2b. ...or local
UPSTREAM_URL=https://api.openai.com \
UPSTREAM_API_KEY=<provider-api-key> \
ADMIN_TOKEN=<long-random-admin-token> \
SOLANA_KEYPAIR_PATH=~/.config/solana/id.json \
REDPILL_PROGRAM_ID=BnoUSPTE88ebzLb74RAnTogddDMzyBPtkmKxiRoRJG4L \
SAS_PROOF_THRESHOLD=5 \
cargo run

# 3. The TypeScript client
cd clients/typescript && npm install && npm run build
node dist/cli.js store-model --model deepseek/deepseek-v4-flash
node dist/cli.js lookup-v2 --payload-hash <64-hex-chars>
node dist/cli.js day-counter --day 20260619

# 4. Verify + anchor in one package
cd clawd-verifier/js && npm install && npm run build
npx clawd-verifier health --gateway-url https://clawd-tee-gateway-production.up.railway.
    app
npx clawd-verifier attest --model phala/gpt-oss-120b
npx clawd-verifier anchor --keypair ~/payer.json \
  --model phala/gpt-oss-120b --rpc-url https://api.devnet.solana.com
```

```
# 5. Before publishing anything derived from the repos
sh scripts/security-scan.sh
sh scripts/check-publish-history.sh
sh scripts/test-security-guards.sh
sh scripts/verify-imported-trees.sh
```

---

3 3. Introducing Dark DeFi

Dark DeFi is the protocol family that takes the confidential-compute substrate of §2 and points it at finance: **shielded wallets, dark pools, and TEE-resident AI agents on Solana**, with Zcash's Sapling design ported to TypeScript and Solana programs.

Its own status badge, verbatim from the repo: **Alpha**. And its own warning, also verbatim in spirit: *do not use on mainnet with real funds until audited.* This section reports the repo's claims with the repo's labels attached — §3.5 lists exactly what is and is not production-grade, in the repo's own accounting.

3.1 3.1 Shielded wallets — a Sapling port on Solana

The wallet layer ports the **Zcash Sapling** design:

- **ZIP-32 HD keys.** One BIP-39 seed derives a full key hierarchy: spending key $\{}rightarrow$ expanded spending key (`ask` + `nsk` + `ovk`) $\{}rightarrow$ full viewing key (`ak` + `nk` + `ovk`) $\{}rightarrow$ incoming viewing key $\{}rightarrow$ payment addresses. Spending power and viewing power are separate keys, by construction.
- **43-byte diversified addresses.** Each payment address is an 11-byte diversifier + 32-byte diversified transmission key — the same format as Zcash Sapling. One seed yields unlimited diversified addresses that are unlinkable to outside observers.
- **Notes, commitments, nullifiers.** Value lives as encrypted *notes*. A note's commitment enters an incremental SHA-256 Merkle commitment tree; spending a note publishes its *nullifier* — proof that *some* note was spent, without revealing which. That is the double-spend defense.
- **ChaCha20-Poly1305 note encryption.** Note contents are sealed with AEAD encryption to the recipient's key; only the holder of the incoming viewing key can read them.
- **Wallet family in the SDK:** `ShieldedWallet` (full lifecycle — deposit, transfer, balance, payment proofs), `ViewOnlyWallet` (built from an exported viewing key: auditors can decrypt notes, never spend), `MultisigShieldedWallet` (M-of-N threshold approval over shielded transfers), plus `SaplingHDWallet`, `NoteEncryptionUtils`, `CommitmentTree`, and `NullifierSet`.

The on-chain side is an Anchor program implementing a **shielded note pool**: deposited SOL becomes an on-chain encrypted note (a `ShieldedNote` PDA holding the commitment and ciphertexts) custodied by a pool vault PDA; `shielded_transfer` spends a note and emits two new notes (payment + change) with no SOL movement; `withdraw` reveals the nullifier and SOL exits the vault.

3.2 3.2 Dark pools — encrypted order flow

Layer 2 of Dark DeFi's seven-layer model is the dark pool: **encrypted constant-product AMMs with order matching before settlement**, using public venues (Jupiter) for price discovery. Amounts stay encrypted end-to-end until settlement — the MEV-resistance claim is precisely that: a searcher cannot front-run an order whose size it cannot read.

Honest labeling, per the repo's own status table: the swap plumbing that exists today is `PrivateSwapManager` (Jupiter Ultra routing with oracle price validation via a multi-source Birdeye + Jupiter price oracle). The **FHE encrypted AMM is roadmap** (§3.5), not shipped. What exists is a private swap path and the pool program below — not a live FHE order book.

3.3 3.3 TEE AI agents, attested via SAS

The agent layer (`@openclawdsol/dark-tee-agents`) runs the full pipeline:

```
spawn enclave -> verify SGX/TDX quote -> seal prompt
  -> pay via x402 (shielded) -> infer -> SAS on-chain receipt
```

- **`ConfidentialAgent`** — lifecycle: spawn, seal, infer. Providers are pluggable (`LocalEnclaveProvider` for dev/test, `HttpInferenceProvider` for a remote TEE endpoint).
- **`DarkTEEAttestation`** — SGX/TDX quote parsing and verification (provider, measurement, freshness, signature).
- **`DarkAttestationService`** — SAS credential / schema / attestation management under the credential name **"Dark Protocol Agents"**, with two schemas: `DarkAgentIdentity` and `DarkInferenceReceipt`.
- **x402 `dark-shielded` payments** — confidential inference is paid per call through the x402 scheme where the published artifact is a commitment, not a value.
- **`ClawdTeeAgent`** — the family's resident agent: it converses privately (prompt sealed into the enclave, reply sealed back to the caller's one-time x25519 key) and trades privately (policy-checked decisions — allowed mints, max atomic amount, max slippage — executed through a private swap router against the shielded wallet). What touches the chain is only the SAS identity attestation and a per-call inference receipt of hashes and commitments.

This is where the two stacks interlock: Dark DeFi's agents stand on the same pattern as CLAWD-TEE's gateway (§2) — sealed compute, attested hardware, digests and receipts on Solana — applied to trading instead of raw inference.

3.4 3.4 Program and package inventory — exactly as the repo states it

On-chain programs

Program	ID	Network / status (per repo)	Source
Solana Attestation Service (SAS)	`22zoJMtdu4tQc2PzL74ZUT7FrwgB1Udec8DdW4yw4BdG`	Solana Foundation, devnet + mainnet	README; `packages/sas-lib/src/dark/progra`

Program	ID	Network / status (per repo)	Source
Dark Protocol (shielded balances)	3KWLFYco7T2rUZkzjSSthJ[illegible]9HAsRmvupDTomGC	**DEPLOYED** (2025-11-11) · Mainnet: not deployed	docs/reference/PROGRAM_INFO.md; packages/tee-agents/src/config.t packages/sas-lib/src/dark/progra
Shielded Wallet	4753b1cCrPzwr7taWWD8y[illegible]fTR7wCFdv1TsAb	**DEPLOYED** (2025-11-11) · Mainnet: not deployed	docs/reference/PROGRAM_INFO.md
Dark Protocol program (`dark-protocol-program/`, newer Anchor tree)	E8zL7h9qHjC7sMf2WCYhd[illegible]yAhTfevo74ga	"Devnet + mainnet pending" — the repo documents a build, not a deployment, for this ID	dark-protocol-program/programs/d (`declare_id!`); README package map
CLAWD-TEE proof storage (§2.2)	BnoUSPTE88ebzLb74RAnT[illegible]KxiRoRJG4La	cluster: devnet	4La/idl/solana_redpill_verifier.
Dark Percolator (Dark family, §4)	Cf8dRz9xhvdwU825LBu6S[illegible]MiJwVh8LciJK1hPz	**DEPLOYED** — Market 0 64kBzfr4wFLLiNBAh5iZ1xQLsEpFvAmqETFArkUKSccN, 50,000,000 insurance locked	On-chain verified (project records)

Two warnings that ship with the devnet Dark Protocol programs, from `PROGRAM_INFO.md` itself, and which any honest guide must repeat:

- The program-ID keypairs for `3KWLFY...` and `4753b1...` were **leaked in an earlier revision of the repo's own docs and must be considered permanently compromised**.
- The upgrade authority (`5m8qxm4j75VQCv34oUThE5QNbKJG7KHfd85NdK4dVagS`) was not in the leak, but **its keypair is currently unaccounted-for** — so the deployed devnet programs are effectively immutable until that key is recovered or fresh programs are deployed under new IDs.

Treat those two devnet programs as frozen alpha artifacts, not as infrastructure to build production on.

Packages (npm, `@openclawdsol` scope)

Package	Version (per repo)	What it is	Status (per repo)
`@openclawdsol/dark-protocol-sdk`	0.3.1	Shielded wallet, Sapling HD, note encryption, private swaps, price oracle, React `DarkSwap` widget	Published

Package	Version (per repo)	What it is	Status (per repo)
`@openclawdsol/dark-tee-agents`	v0.1.0	TEE agents (incl. `ClawdTeeAgent`), SAS attestation, shielded x402, `dark-tee` CLI	Published
`@openclawdsol/dark-protocol`		Low-level protocol primitives: client, Sapling, note encryption, swap routing, AI agent manager	Published
`dark-defi`	—	Umbrella meta-package re-exporting the stack	Published
`sas-lib`	—	SAS TypeScript client + Dark schema extensions (`DarkAgentIdentity`, `DarkInferenceReceipt`, `DarkAgentRouter`, `AgentRegistry`)	Local workspace only
`dark-x402-terminal`	—	Interactive terminal CLI — wallet, swap, agents, dashboards (the "lobster shell")	Published
`darkswap`	—	Next.js front-end for the shielded swap	Dev (Vercel-ready)
`agentwallet/`	—	Agent wallet CLI/server with vault + deploy helpers (Cloudflare / E2B)	In-repo

Also in the repo, for completeness: `programs/` (the Anchor workspace holding the deployed devnet `dark_protocol` + `shielded_wallet` programs) and `percolator-launch/` (a launch/staking/oracle program collection the Dark DeFi repo vendors — distinct from the Dark Percolator program in §4).

Try it (offline demos first — that ordering is deliberate):

```
npm install @openclawdsol/dark-protocol-sdk
npm install @openclawdsol/dark-tee-agents
npm install @openclawdsol/dark-protocol

git clone <dark-defi repo> && cd Dark-Defi-1
npm install && npm run build
npm run tee:demo        # offline TEE agent walkthrough
npm run clawd:demo      # Clawd: private converse + shielded trade, offline
npm run terminal        # the dark-x402-terminal CLI
```

3.5 3.5 What is protected, what is not — the repo's own accounting

Dark DeFi's README carries a security model box. This guide preserves it rather than improving on it:

Protected today (TypeScript SDK layer)

- Note contents — ChaCha20-Poly1305 AEAD
- Address linkability — diversified + ephemeral addresses
- Viewing-key separation — spend $\{}neq$ view
- Double-spend — nullifier tracking

Not yet production-grade (repo's own list)

- **Jubjub curve operations are a hash-based approximation**, not production curve arithmetic
- **ZK proofs are placeholders** — no Groth16 circuits yet (roadmap)
- **FHE encrypted AMM** — roadmap
- **Threshold ElGamal key custody** — roadmap (BPF stack constraints)
- **No formal security audit** — alpha only

In plain language: Dark DeFi today is a faithful-in-structure Sapling port with simplified cryptography under the hood, a real devnet note-pool program, published SDK packages, and demo-grade agent flows. It is a working alpha to build and learn on — **devnet and demos, not mainnet funds**. Any source that tells you otherwise is contradicting the repo.

---

4 4. The Dark family map

Dark DeFi does not stand alone. It is one member of a family designed in `DARK-SUITE.md` (2026-11-11) around **two privacy rails**:

- **Helius Solana Privacy Rings (SPR)** — *shielded value*: encrypted UTXOs in a state tree, view tags, nullifiers. The payments rail. The SPR API is read-only (seven methods: encrypted-UTXO reads, shielded-tx lookups by tag / signature / nullifier, Merkle proofs, non-inclusion proofs, user records); deposits, transfers, withdrawals, and decryption happen user-side through the `@heliuslabs/spr` SDK with the owner's own keys. Documented on both devnet and mainnet hosts.
- **MagicBlock Private Ephemeral Rollups (PER)** — *shielded state*: accounts delegated into a TEE-backed rollup behind permission lists. The trading rail (positions, intents, strategy). Mainnet PER endpoints are team-gated — a real prerequisite, still standing.

The family:

Name	What it is	Rail
Dark Payments	The umbrella: every product below. Value and state move on Solana without publishing the link between who paid, who traded, and how much — while settlement, solvency, and aggregate flow stay publicly verifiable	Both
Dark x402	An x402 machine payment where the payer pays *shielded* inside an SPR ring and the facilitator verifies settlement from shielded-transaction reads + a Merkle inclusion proof instead of a public transaction link. Reference implementation tree is public: **github.com/themusebook/dark402**	SPR
Dark Percolator	The private-perps *engine*: our own Percolator program — public market account, per-trader portfolios private inside a MagicBlock PER, sealed intents crossed in batches at the oracle mark	PER
Dark Perps	The trader-facing product on that engine: hidden positions, hidden intents, hidden liquidation prices; public mark, public aggregate open interest, public insurance. (The Phoenix shadow pool is its market-making / hedge side in the design.)	PER
Dark DeFi (§3)	The Sapling shielded-wallet + dark-pool + TEE-agent protocol family: private balances and private agent trading, attested via SAS	Sapling pool + TEE/SAS

How they compose. Three primitives recur everywhere: *shielded transactions* (deposit $\{}rightarrow$ shielded transfer $\{}rightarrow$ withdraw inside a ring or pool), *shielded swaps* (funding leg and proceeds leg shielded around a swap leg — v0 of the suite design is explicit that the venue still sees the swap leg itself), and *shielded messaging* (x402m agent messages whose payment proposals settle as shielded transfers). Dark DeFi's Sapling pool and SPR are two implementations of the shielded-value primitive; PER is the shielded-state primitive for perps; CLAWD-TEE (§2)

is the attestation substrate the agents in every branch stand on.

Honest status table — the whole family, nothing upgraded:

Component	Status
CLAWD-TEE proof program + gateway + verifiers	Code complete, self-hostable; default cluster devnet; hosted gateway runs a **stub TDX backend** (not TDX hardware)
SAS credentials via CLAWD-TEE gateway	Implemented; issues when configured with `ADMIN_TOKEN` + threshold
Dark Payments site	Live — https://percolator.musebook.trade
Dark x402	Design (suite doc) + public reference tree (themusebook/dark402)
Dark Percolator program	**Devnet — deployed** (`Cf8dRz9x...K1hPz`), Market 0 created, insurance locked; PER privacy layer per design
Dark Perps (product)	Design — trader-facing surface not shipped
Dark DeFi SDK packages	Published on npm — **Alpha**
Dark DeFi note-pool programs (`3KWLFY...`, `4753b1...`)	**Devnet — deployed**, frozen (compromised program keypairs; upgrade-authority key unaccounted-for)
Dark DeFi `dark-protocol-program` (`E8zL7h...`)	Built; mainnet pending; deployment not documented in-repo
Dark DeFi FHE AMM / Groth16 ZK / threshold ElGamal	Roadmap
tee.musebook.trade (§5)	In build — see §5 for exactly what that means

Privacy truth discipline (from the suite design, applied family-wide): state exactly who can see what, and never claim more than the machinery delivers. A ring/pool **deposit and withdrawal are public events** — an observer sees wallet W shield or unshield at time T, and the boundary amount. What is hidden is the interior: transfer amounts, the sender$\{}leftrightarrow$recipient link inside the pool, and private balances (computed locally by the wallet from its own viewing keys — no server sees them).

---

5 5. tee.musebook.trade — the TEE terminal

Every stack needs a front door. **tee.musebook.trade** is Clawd's: our own terminal over the confidential-compute stack — the place where the pieces in this guide become one surface a person can actually use.

What the terminal is, as designed and now in build:

- **A window onto the CLAWD-TEE substrate.** Attestation reports, proof lookups by payload hash, verification levels, day counters, SAS credential status — the §2 machinery, readable by humans instead of only by CLI.
- **Connected to the Dark DeFi programs.** The terminal reads the Dark family's on-chain footprint — the devnet note-pool programs, the Dark Percolator market, SAS agent identities

and inference receipts — and presents the family's real state (devnet labeled devnet, alpha labeled alpha) rather than a marketing mock of it.
- **A RedPill TEE trading terminal.** The trading surface of the same stack: TEE-attested inference and agent decisions (the RedPill lineage of §2.4) wired to the Dark DeFi agent flows of §3.3 — verify the quote first, then let the agent work. The terminal wires a server-side `REDPILL_API_KEY` slot for this; until that key is configured the RedPill features report *not configured* instead of pretending to run.

The terminal is being built in parallel with this guide. This section is the product intro, not a launch announcement: features land as they are verified, and the terminal follows the family rule — a surface that is not live yet is labeled as such, in the UI, not just in the docs.

---

6 6. Getting started

Read the doctrine first (the repos' rule, and this guide's):

1. `tea/content/soul/SOUL.md` — the character harness
2. `tea/content/articles/brave-new-world-blockchain-ai.md` — provenance, federated learning, Consensus Learning, the on-chain reinforcement loop
3. `tea/content/articles/clawd-solana-svm-design.md` — Grid / Forge / Relay / Registry / Verify, the full protocol draft

Repositories

Repo	What
github.com/Solizardking/tea	CLAWD-TEE, public sanitized cut
github.com/Solizardking/clawd-tee	CLAWD-TEE, upstream lineage
github.com/Solizardking/Dark-Defi-1	Dark DeFi protocol monorepo
github.com/themusebook/dark402	Dark x402 / Dark Payments reference tree (public)

Packages

```
# Confidential compute (S2)
npm install @clawd/tee-verifier            # verify + anchor, one package
npm install @redpill-ai/solana-verifier    # Solana proof-program client
npm install @redpill-ai/verifier           # multi-provider TEE verification

# Dark DeFi (S3)
npm install @openclawdsol/dark-protocol-sdk
npm install @openclawdsol/dark-tee-agents
npm install @openclawdsol/dark-protocol
npm install dark-defi                      # umbrella meta-package
```

Sites & services

Surface	URL
TEE terminal	https://tee.musebook.trade
Dark Payments	https://percolator.musebook.trade
Hosted TEE gateway (Railway; stub TDX — §2.7)	https://clawd-tee-gateway-production.up.railway.app
$CLAWD	mint `8cHzQHUS2s2h8TzCmfqPKYiM4dSt4roa3n7MyRLApump`

The one-paragraph version, for quoting:

> Clawd Confidential Compute splits the world in two: Solana settles, TEEs compute, and only digests cross between them. CLAWD-TEE anchors TEE attestation proofs into Solana PDAs anyone can verify; Dark DeFi builds shielded wallets, dark pools, and attested AI agents on top. Statuses are labeled honestly — live, devnet, alpha, design — because in this stack, proofs beat myths.

Document status: written 2026-11-13 from the repositories listed above. Claims about code, program IDs, and statuses are as the repos state them on that date; where the repos disagree with each other, the disagreement is noted in §3.4 rather than smoothed over. Nothing in this guide is financial advice, and nothing in it is a promise of returns — it is a technical guide to an alpha-stage stack.