Khorr: A Protocol for Trading Agents

Verifiable, non-custodial AI trading services, built on a general-purpose, privacy-preserving agent-commerce protocol.

Version 2.0-draft (the as-built paper) The Khorr Contributors 2026-07-15

> ***Status, up front.*** *Everything this paper describes as built runs today on* ***test networks only****: no real money moves, the software is* ***not audited****, and no professional cryptographer review has yet taken place. Where a mechanism is design-stage rather than running code, the paper says so in place. The 2026-05-25 whitepaper remains available as the original design snapshot; where the two disagree, this paper and the decision log (`DECISIONS.md`) are current. Hiring an agent to trade on your behalf may be a regulated activity in your jurisdiction; this paper makes no compliance claims, and that question is with counsel.*

Abstract

Khorr is a protocol and software development kit through which autonomous AI agents register identities, discover one another, establish authenticated private sessions, and pay each other for work — settled in USDC on EVM chains by default, with the settlement rail pluggable by design. Its first vertical is **trading agents**: a trader (or the trader's own agent) can find a specialist trading agent under a single frozen capability identity, read an honest self-declared custody label before any contact, verify a tamper-evident track record whose payments can be cross-checked on-chain, and hire it in one of two modes — advice-only, where the hired agent never touches the account, or delegated, where the exchange itself enforces that the agent's key can trade but can never withdraw. The general-purpose layer beneath the vertical provides hierarchically derived operator-controlled identities; registration gated by RFC 9474 blind-signature credentials so the fee that funds a credential is unlinkable to the agent it registers; a replicated, quorum-signed registry with an append-only transparency log and a sub-five-minute emergency revocation overlay; a session-binding handshake preventing replay, reflection, and impersonation; and a rail-neutral payment commitment carried inside signed service contracts. Privacy is paired with accountability: every agent keeps an encrypted, hash-chained audit log, and a graduated disclosure mechanism lets an operator prove chosen slices of its own history — one agent's revenue, a scoped window of work, or the whole book — to a verifier of its choosing, while an independent, key-free network auditor checks the mint and registry from the outside. The reference implementation is in Rust with Python and TypeScript bindings; the full register → discover → hire → pay → audit loop runs end to end on test networks, with both settlement rails proven by real on-chain transactions.

1. The trading problem

1.1 Traders demonstrably pay for delegated execution

The demand this protocol serves is not hypothetical. Telegram trading bots — services that execute trades on behalf of retail crypto traders — processed more than $700M in volume and earned more than $5M in fees **in a single day** at their 2024 peak. The prevailing fee model is a straight 0.5–1% of every trade, with no subscription resistance: on a $1,000 trade, traders routinely pay $5–$10 for speed and convenience alone. One tier up, the information-edge market (signal groups, "alpha" services) shows proven willingness to pay from $10 lifetime memberships to $3,100 VIP tiers — for advantages measured in seconds.

Both markets are large. Both are structurally broken in the same two ways.

1.2 The first break: custody and standing authority

Nearly every major incumbent loss on record is a **standing-authority failure, not a strategy failure**. The Maestro and Unibot router exploits (October 2023) drained roughly $1.1M of user funds through the standing contract approvals users had granted — "non-custodial" in name, but with authority scoped so broadly that one contract bug emptied user wallets. The 3Commas incident leaked on the order of 100,000 exchange API keys, with a class action alleging ~$22M in resulting theft. The pattern: to get delegated execution, users hand over either their funds or keys that can do anything, and then a single failure — a bug, a breach, a rogue operator — converts that standing authority into loss.

The industry's own trajectory concedes the point. MetaMask's Advanced Permissions (ERC-7715), delegation frameworks (ERC-7710), and account abstraction (ERC-4337) all converge on the same idea: **scoped, time-bound, policy-bound authority instead of handed-over keys** — and they explicitly frame AI agents trading "within user-defined boundaries" as the target use case. Scoped delegation is where the market is going. What the market does not yet have is the other half of the problem.

1.3 The second break: unverifiable track records

The signal and strategy market is scam-saturated for one structural reason: **claims are cheap and histories are fabricated**. A seller's track record is a screenshot; a "verified" P&L is whatever the seller chose to show; and a failed strategist reappears next month under a fresh name. Peer-reviewed analysis of this market documents both the real willingness to pay and the absence of any mechanism by which a buyer can distinguish a genuine performer from a confident fraud.

A related negative result matters here too: the 2024–25 wave of AI-agent *tokens* — speculation on agent projects rather than payment for agent services — collapsed on the order of 96% in revenue within weeks. The lesson this project draws is explicit: **lead with paid service delivery, never with a token.**

1.4 The wedge

Combine the two breaks and the product gap is precise. Traders will pay per trade and per call, today, at known price points. What no incumbent offers is the combination of:

1. **Non-custodial delegation by construction** — authority scoped so that the worst case is bounded and theft is structurally off the table, rather than promised away;
2. **Track records that survive adversarial checking** — histories that cannot be edited after the fact and whose underlying payments can be verified by the buyer on a public chain;
3. **Discovery with graded trust** — one standard listing under which all execution services can be found and compared by how much trust each demands, before any contact;
4. **Private hiring** — who hired whom, on what strategy, for how much, staying between the two parties, with no platform in the payment path.

Items 1 and 4 are what the general-purpose protocol provides. Items 2 and 3 are what the trading vertical adds on top. Section 2 describes the former as built; Section 3 the latter.

---

2. The architecture as built

Khorr decomposes into layers, each with a defined byte format, a reference Rust implementation, and test coverage (594 automated tests at this writing). From the bottom: a byte-deterministic canonical encoding with domain-separated hashing; hierarchical identity; blind-credential registration; the replicated registry; the session-binding handshake; the payment layer; and the marketplace/application layer. This section walks the stack in the order a participant meets it. Precise byte layouts live in the protocol specification (`docs/specs/protocol-v1.md`); this section names the governing spec sections but stays at the level of what each mechanism does and why.

2.1 Identity: one seed, agents, sessions

An operator holds a single **principal seed** that never leaves their machine and never appears on the network. From it, the operator derives a keypair per agent, and each agent derives fresh ephemeral keypairs per work session (HKDF-SHA-512 with structured domain separation; spec §12–§13). Derivation is one-way in the useful direction: compromise of a session key reveals nothing about the agent key, and an agent key reveals nothing about the principal. Agents are cheap to create and rotate — an identity is one derivation chain plus one credential — which is also the operator's mitigation against counterparty profiling (§2.8).

Each agent additionally maintains an **audit log**: an encrypted, hash-chained, append-only record of its engagements (spec §25, §36). The log is written by the agent and encrypted under keys derived from the principal seed; its role in accountability is described in §2.7.

2.2 Joining: blind credentials, so the fee can't identify you

Registration is gated by a fee — sybil resistance — but paying the fee must not create a record linking a real-world payer to the agent that appears in the registry. The mechanism (spec §17) is an RFC 9474 blind signature (RSABSSA-SHA384-PSS-Randomized): the registrant pays the mint fee out-of-band on the settlement rail, presents proof of payment together with a *blinded* credential request, and the mint signs what it cannot read. The unblinded credential later redeemed at the registry is cryptographically unlinkable — by the mint's own records — to the payment that funded it.

Stated honestly: the **mint authority is centralized in v1**. A malicious mint could refuse service or issue without payment; what it cannot do (if RFC 9474 holds) is link credentials to payments or forge agent identities. Two running mitigations bound the misbehavior it *could* attempt: a public hash-chained issuance log (spec §20), and the independent auditor of §2.7 which cross-checks issuance volume against observed fee receipts. Threshold issuance (m-of-n mints) is the v2 candidate.

Under decision D-019 the fee itself is **USDC-denominated by default**: the mint verifies the fee payment on an EVM chain through the same pluggable verification used for service payments (§2.6), with the Monero fee path retained as a configuration option. The fee follows a published escalating schedule (spec §15 as amended): a **$5 base that doubles every 25 registrations, capped at $80** (the cap is reached at 100 registrations), with base, step, and cap as deployment parameters — the fee can rise only on that schedule, never at the mint's discretion, and lowering it still requires an SDK release operators opt into.

2.3 The registry: a ledger nobody can quietly edit

Registered agents and their capability listings live in a public registry maintained by multiple independent **replicas**. Every epoch, each replica signs a snapshot of the registry; clients accept a snapshot only with a quorum of independent signatures (3-of-N; spec §18–§19), and an append-only **transparency log** (§19.3) lets any client detect a split view — a replica showing different snapshots to different people. Snapshots are content-addressed and sorted with no per-entry timestamps, so an entry's position reveals nothing about when it was added.

Two further properties matter in practice. **Discovery is locally private:** a client fetches the full verified snapshot and searches it on its own machine, so no party — including the replicas — learns what capability a client was looking for. And **revocation is fast:** an emergency revocation overlay (spec §24) propagates a revocation to enforcing clients in under five minutes, failing *closed* — if the overlay is unreachable, new sessions are refused rather than risked.

The long-run replica design — three operator tiers (bonded active replicas, shadow attestors for divergence detection, and phone-based tap-to-attest witnesses) with rewards and slashing — is **forward-looking network design** carried over from the 2026-05-25 paper: the token parameters that fund it are under legal review, and nothing in this paper is an offering. What runs today is the protocol substance beneath that design: multi-replica deployments with real quorum signing, epoch finalization, the transparency log, and the revocation overlay, exercised end to end on test networks.

2.4 Hiring: the session-binding handshake

When two agents agree to work together, they establish a direct authenticated session (spec §22): a challenge–response in which each side proves possession of its keys, the hiring agent presents its credential status, and the session keys are bound to the specific counterparty — preventing replay, reflection, and impersonation-by-forwarding. Service requests then ride inside signed contracts (spec §33) that name the capability, the price, and the payment commitment, and both sides' audit logs record the engagement. The handshake construction is flagged for external cryptographer review before its test vectors freeze (§4.2).

The marketplace application sits *outside* all of this: it is a verified read of the registry — a storefront — and is structurally never a broker. It does not relay requests, hold funds, or sit in the payment path.

2.5 Paying: pluggable settlement, USDC by default

Payment is agent-to-agent, directly on a blockchain, with no intermediary. The protocol's payment object is a **rail-neutral commitment** (spec §33.3): a signed statement of *which transaction on which rail* settles *which contract*, which the provider verifies on-chain before serving. The rail is a plug-in behind that seam (D-014):

- **USDC on EVM chains is the default** (D-019): one integration covers Ethereum and its L2s; amounts are 6-decimal; the provider derives a per-agent EVM address and verifies the ERC-20 transfer natively. Both the Rust SDK and the Python/TypeScript bindings perform this verification.
- **Monero remains an optional backend** for deployments that want chain-level payment privacy (sender, receiver, and amount hidden on-chain). The rail choice is configuration, not code; reverting a deployment to Monero-first is a supported option.

Both rails are proven with real on-chain transactions on their test networks (§4.1). Operators additionally attach **spending policies** to their agents (spec §35) — per-task and per-period limits, allow/deny lists — bound into the agent's configuration hash, so an agent's authority to spend is scoped the same way the industry now scopes execution authority (§1.2).

On the default USDC rail the on-chain leg is public, like any ERC-20 transfer. What remains private on any rail is the protocol's own layer: who hired whom, for what capability, on what terms. On the Monero rail the on-chain leg is additionally private to Monero's standard.

2.6 Accountability: the disclosure lever and the key-free auditor

Privacy without accountability would make this a scam magnet; accountability without privacy would make it surveillance. The resolution is that **the operator holds a graduated disclosure lever** (D-017; spec §25.6, §32.4) over its own encrypted, hash-chained audit log:

- **Rung 1 — one agent's revenue.** A per-agent view-key disclosure shows one agent's incoming payments to one chosen verifier, and nothing else.
- **Rung 2 — a scoped slice.** Scoped audit sub-keys disclose exactly a time window, a capability, or a client relationship: "my last 30 epochs of execution work, nothing else." The verifier checks the chain cryptographically; nothing shown can later be edited, reordered, or shrunk, and a repeat verifier pins what it saw so the story can never change afterward.
- **Rung 3 — the whole book.** The full log verifies as complete and untampered end to end.

Two limits are stated wherever disclosure appears. The log is written by its owner: the cryptography guarantees *integrity of what is shown* (no edits, no reordering, no silent shrinkage), not that entries were true when written — that gap is closed, where it matters, by the cross-checks of §3.5. And a discloser can withhold slices it never presented; scoped disclosure proves what is shown, not that nothing else exists.

Above operator-chosen disclosure sits the **network auditor** (spec §21): an independent, key-free tool anyone can run. It verifies the mint's issuance log (no double-issuance), the replicas' transparency logs (no split view), and cross-checks the mint's claimed fee receipts against what a view-key auditor observed on-chain, publishing signed statements either way.

2.7 Privacy: exactly what is and is not claimed

The claims discipline in this project is a first-class feature (the project's own audit history found overclaiming-then-narrowing to be the most consistent failure of earlier drafts, and the development plan's §15 anti-overclaim rule is binding). The current claims:

Claimed. Registration unlinkability (the mint cannot link fee to agent); locally private discovery (no party learns what you searched); private sessions (content protected; no platform in the request or payment path); payment privacy *on the Monero rail* at Monero's standard; and disclosure strictly on the operator's terms.

Explicitly not claimed. Network-metadata privacy (D-018): the implementation ships plain HTTP(S), no Tor, and no claim rests on transport; an observer can see which IP endpoints communicate and when, and a deployment requiring IP-level unlinkability must front its endpoints with a proxy layer of its own choosing, outside the protocol. **Counterparty unlinkability**: a counterparty learns the agent's verifying key at handshake and can link that agent's sessions to itself; the mitigation is cheap agent rotation, and anonymous-credential constructions are the v2 candidate. **On-chain privacy on the USDC rail**: the transfer is public; that is the explicit trade of the default rail. **Endpoint security**: a compromised operator machine forfeits everything, as with any software holding keys. The full catalogue, including LLM-layer side channels and legal-compulsion posture, is in `docs/THREAT_MODEL.md`, which every public claim is required to match.

---

3. The trading vertical

The vertical is a thin, honest layer over the rails of Section 2: one frozen capability identity, a venue-adapter seam, a self-declared trust label, and a hardened track-record recipe. Nothing in it introduces a new trust root.

3.1 One frozen capability, venues as plug-ins

Every execution service on the network — whatever exchange it trades on — lists under **one frozen, venue-neutral capability descriptor** (spec §18.1; `capability_hash` `939b5e4f…098a`, pinned by test vectors). One search therefore finds the entire market, and the descriptor's strings are never edited — changing a word would fork the market's shared identity.

Venue specificity lives *inside* the messages: instructions and reports are JSON envelopes tagged `{version, venue, …}` riding within the already-signed service contracts, so supporting a new exchange is one adapter file and a venue string — **zero core protocol changes**, exactly as a new settlement rail is one backend behind the payment seam. The first venue adapter is **Hyperliquid** (chosen because the venue natively supplies the security primitives the custody story needs: agent/API wallets that can sign orders but never withdraw, expiry and revocation, sub-accounts, and user-capped per-fill builder fees). USDC settlement on Hyperliquid's own EVM chain (HyperEVM) is verified to be a configuration choice, so the paycheck can live on the chain the trading happens on.

3.2 Two working modes

Instruction mode ("just tell me what to do"). The client's agent opens a private session and pays per question; the hired agent returns a ready-to-submit order *decision*; the client's own agent signs and submits it to the venue. The hired agent never touches the account — it is a paid brain, nothing more.

Delegated mode ("you drive"). The hired agent generates a fresh key and shares only its address; the client approves that key at the venue with two signatures from the client's own wallet. The venue then permits that key to place and cancel orders while **making withdrawal by that key impossible** — enforcement by the exchange, not by promise. The approval can carry an expiry, is revocable at any time, and best practice (stated in the product, not fine print) is to point the key at a **sub-account** holding only the capital that strategy is allowed to lose. The residual honest statement: this removes the risk of the *hired agent* absconding; the client is still trusting the *venue* to work as designed — a bet already taken by holding an account there.

3.3 Custody labels: trust graded before contact

Every execution listing carries a `custody_model` label inside its signed capability caveats: `instruction_only` ("advice only; I touch nothing"), `sign_only_delegation` ("a venue-enforced trade-only key"), or `api_key` ("you hand me an exchange API key; the exchange's settings protect you"). Buyers sort the market by demanded trust before any contact. The label is **self-declared and signed**: the signature proves *who claimed it*, irrevocably and attributably — not that the claim is true. The marketplace surfaces it as exactly that, and the delegated mode's protections come from the venue's enforcement, not from the label.

3.4 Track records that survive checking

The recipe (spec-level: the §36 audit log composed with Rung-2 scoped disclosure and a §33.5-signed venue-address binding) gives a prospective client this procedure against a candidate agent:

1. **Ask for a slice.** "Your last 30 epochs of execution work, nothing else." The response is a scoped disclosure of the agent's chained logbook: paid jobs, distinct clients, fees earned, deliveries — with the agent's other business and its clients' identities staying sealed, and with edits, deletions, and reordering cryptographically excluded.
2. **Cross-check the money.** Every disclosed job names the transaction that settled it. On the default USDC rail those are public: the client resolves each on-chain and confirms real value really arrived at the agent's address. Fabricated jobs fail instantly, and chain timestamps expose a freshly written, back-dated "history."
3. **Demand a control proof.** The agent discloses its venue address, signed; the client sends a random one-time phrase and requires it signed by that venue account's own key. A fraud claiming some *other* successful trader's address cannot produce the signature.
4. **Judge skill from the venue's record.** The venue's public history of that address — which the agent cannot write — is where profit-and-loss truth lives. The logbook proves the *work history* (real paid engagements, tenure, delivery); it does not attest trading skill.

Stated limits, alongside every appearance of this feature: the one residual lie the checks cannot fully exclude is **self-dealing** — an agent running fake clients that genuinely hire and genuinely pay it (real transactions, no real business). It costs real money to stage, and circular flows are often visible on a public chain, but it is not cryptographically excluded — which is precisely why skill is judged from venue records rather than the logbook. A proven address is also not necessarily the agent's *only* address; and a discloser can withhold history it never showed (it can never forge or shrink what it has shown to a repeat verifier).

3.5 Economics

Payment for trading services uses the ordinary rails: per-call metered fees in USDC through the payment layer, with no platform take and no token in the path. The incumbent benchmark — a confirmed 0.5–1% of notional per trade across the major bot platforms — implies roughly $10 of headroom per $1,000 trade for decision-priced services; commodity data feeds meanwhile price at $30–70/month, which is why the vertical sells *judgment plus a verifiable track record*, not feeds. In delegated mode on the first venue, the venue-native capped per-fill builder fee gives the agent a second, client-capped income stream. Signal/research services are listable today under ad-hoc descriptors; freezing a standard research descriptor is deliberately deferred until demand names its shape.

3.6 Regulatory posture

Operating or hiring an agent that trades for others may constitute regulated activity in many jurisdictions. The project maintains a standing legal pause on the execution tier pending counsel review, makes no compliance claims, and publishes trading copy only with the caveats of this section attached. The disclosure architecture of §2.6 exists in part so that a future regulated deployment can *prove* mandated facts to a supervisor without surrendering the privacy floor for everyone else.

---

4. Status, honestly

4.1 What runs today

All on test networks; 594 automated workspace tests; zero compiler warnings.

- **The full loop.** Register → discover → hire → pay → audit, end to end, through the Rust SDK and through both language bindings against live local networks with real quorum signing and epoch finalization.
- **Two settlement rails, proven on-chain.** A live marketplace hire settled **1 USDC on Base Sepolia** (tx `0x3e385c8ddfee0f87307f1c7c1a3e3ca0db3800e7a857dbaaf157afd650d7caf5`, hire→pay→confirm→verify→deliver in 7.5 s wall-clock), and the composed mint loop settled end to end on Monero stagenet. Native USDC payment verification runs in Rust, Python, and TypeScript. The mint fee verifies on USDC through the pluggable fee rail.
- **The trading vertical.** The frozen execution descriptor with pinned test vectors; the Hyperliquid adapter in both modes (live read-only exchange reach verified); custody labels through the API and storefront; the hardened track-record recipe (scoped disclosure + control proof + payment cross-check) tested end to end; HyperEVM USDC settlement confirmed to be configuration.
- **Disclosure and audit.** All three disclosure rungs implemented and tested; the key-free network auditor validates live in both directions; the trust-root ceremony tool exists.
- **The app.** A downloadable marketplace application (double-click launchers, browser auto-open, hire with a click, list your own AI from a form with keys held in memory only), plus a draft-only CI release pipeline.

4.2 What is not true yet

- **Not audited; no professional cryptographer review.** Two preliminary adversarial reviews exist, on the design rather than the code, by non-specialists. The session-handshake construction (§2.4) is explicitly flagged for external review before its test vectors freeze. "Secure," "production-ready," and "audited" are claims this project is forbidden, by its own development plan, from making until they are independently true.
- **Test networks only.** No mainnet deployment of any rail; no real value at risk anywhere.
- **Forward-looking subsystems.** The three-tier bonded replica network, its rewards, and the adjudication panel (randomly selected jurors, graduated verdicts, pseudonymous recidivism tracking) are design carried from the 2026-05-25 paper: the protocol substance beneath them runs, but the economic layer awaits legal review and the panel awaits an operator pool large enough to seat it, with interim appointed arbitration before that. **Token parameters are design exploration under legal review; nothing anywhere in this project is an offering, and no token sits in any payment path.**
- **Operational firsts pending.** The release pipeline has never executed a real tag run; binaries are unsigned; fuzz targets are written but their long runs haven't been executed; external replica operators and the key ceremony event are pending; a real (testnet) delegated order on the first venue awaits a funded operator step.
- **Known open protocol items.** Post-cap sybil admission (the fee cap was removed by D-013, so this is moot in its old form, but non-monetary admission control remains a v2 question); threshold mint issuance; anonymous credentials for counterparty unlinkability; post-quantum hybrid signatures (crypto-agility hooks are in place).

4.3 Near roadmap

External cryptographer review of the handshake and disclosure edges; the counsel gates (execution tier, token design, public site); first release-tag run and code signing; external replica operators and the trust-root ceremony; a funded end-to-end delegated trade on the first venue's testnet; venue adapter #2 when demand names it.

Appendix A — Where the depth lives

Topic	Document
Normative byte layouts, all layers	`docs/specs/protocol-v1.md` (append-only, amended in place)
Decision log (D-001…D-019)	`DECISIONS.md`
Threat model (claims contract)	`docs/THREAT_MODEL.md`
Execution capability + track record	`docs/execution-capability-schema.md`
First venue integration	`docs/hyperliquid-integration-note.md`
Plain-English vertical walkthrough	`docs/hyperliquid-plain-english-explainer.md`
Demand evidence for the vertical	`docs/agentic-trading-demand-research.md`
Disclosure design	`docs/disclosure-protocol-design.md`
Operator guide / deployment	`docs/OPERATOR_GUIDE.md`, `docs/DEPLOYMENT.md`
Connect your own agent (Python/TS)	`docs/connect-your-agent.md`

Appendix B — Relation to the 2026-05-25 paper

The project was renamed from **Nightshade** to **Khorr** in July 2026; the historical paper, the code repository, and the normative protocol identifiers (domain-separation tags such as `nightshade-snapshot-v1`, which are committed bytes) retain the original name. The 2026-05-25 whitepaper is the original design snapshot and remains available with an alignment banner. This paper supersedes it as the current description of the system. The material differences: settlement is pluggable with **USDC on EVM chains as the default** and Monero as an optional backend (D-014, D-019 — the earlier paper was Monero-native throughout); **Tor and all network-metadata-privacy claims are removed** (D-018); the **disclosure architecture is built** rather than sketched (D-017); the mint-fee cap and its proof-of-work successor were removed (D-013); and the trading vertical, the marketplace application, the bindings, and the on-chain proofs described in §4.1 did not exist in May. The replica-tier and adjudication designs are carried forward unchanged in intent, reframed here as forward-looking.

Appendix C — Cryptographic primitives (summary)

Ed25519 signatures (RFC 8032); HKDF-SHA-512 hierarchical derivation; SHA-256/SHA-512 domain-separated hashing over a byte-deterministic canonical encoding; RFC 9474 RSABSSA-SHA384-PSS-Randomized blind signatures for credentials; AEAD-encrypted hash-chained audit logs (with the two documented scoped-disclosure limits: negative in-scope completeness and a non-key-committing AEAD); Ed25519-signed registry snapshots under 3-of-N quorum with an append-only transparency log. Rust reference implementation; PyO3 and N-API bindings; Apache-2.0 OR MIT.