JACK: Just-in-time Autonomous Cross-chain Kernel

A Formal Architecture for Intent-Based, Privacy-Aware, and Policy-Enforced DeFi Execution

v1.0.2

Blockchain Foundation LatAm
research@lukas.lat

February 2026

Abstract

Liquidity, execution venues, and state have fragmented across heterogeneous blockchain ecosystems, creating a usability and safety bottleneck for decentralized finance. While bridges, aggregators, and routers enable cross-domain value movement, they do not provide a unified *execution abstraction* with explicit policy enforcement and a rigorous failure model.

This paper introduces JACK, a protocol-level execution kernel that transforms high-level user intents into verifiable, policy-constrained execution plans and settles them on programmable venues (e.g., Uniswap v4 hooks). JACKseparates (i) intent representation, (ii) solver coordination, (iii) private constraint handling, (iv) routing, and (v) settlement adapters.

Scope note (v1). JACK v1.0.2remains a practical, implementation-aligned specification. Private constraints are handled via a pluggable *Confidential Constraint Module (CCM)* interface, while stronger FHE+ZK enforcement remains a forward-looking research direction.

Contents

1 Versioning, Scope, and Non-Goals

1.1 Change Log

- **v1.0.2** (Feb 2026): Adds implementation-aligned details for LI.FI quote/route/status integration, Yellow Network provider notifications/auth persistence flow, deterministic `/api/quote` semantics with explicit fallback mode, and production process diagrams.
- **v1.0.1** (Feb 2026): Clarified v1 scope; replaced hard FHE claims with CCM interface; added explicit non-goals for cross-chain atomicity; added minimal solver economic model; expanded threat model and hook security considerations.
- **v1.0.0** (Feb 2026): Initial formal architecture draft.

1.2 What is in-scope for v1

- Concrete intent format with public envelope and private constraints.
- Solver competition with minimal economic security primitive (bonded execution).
- Routing via existing infrastructure (LI.FI) with explicit failure handling and deterministic API contracts.
- Settlement on a single destination chain using programmable venues (Uniswap v4 hooks).
- Pluggable privacy interface (CCM) to reduce information leakage.
- Provider event ingestion and persistence for execution observability (Yellow Network integration).

1.3 Non-goals for v1

- **Atomic cross-chain settlement** across heterogeneous finality domains.
- **Trustless bridge security** (bridges are treated as external dependencies with allowlists and circuit breakers).
- **Production-grade FHE+ZK** constraint proofs.
- **Global solver decentralization guarantees** (v1 prioritizes correctness and observability).

1.4 v1 vs. vNext

Component	**v1 (this document)**	**vNext (research / roadmap)**
Private constraints	CCM interface; confidential execution/coprocessor possible	FHE+ZK proof-carrying constraints; threshold key mgmt
Cross-chain execution	Best-effort routing + explicit failure states	Stronger atomicity/dispute games/optimistic safety
Solver economics	Minimal bonded execution + fees	Auctions, staking, slashing, anti-collusion
Venue policy	Uniswap v4 hooks enforce policy	Formal verification and hook attestation registries
Provider telemetry	Yellow notifications and persistence	Multi-provider reconciliation and cryptographic attestations

2 Introduction

DeFi execution has moved from single-chain composability to a multi-domain environment where users must still manually choose routes, bridges, and venues. This transaction-centric model does not scale and is vulnerable to adversarial observation and manipulation [3].

JACKintroduces an intent-first kernel: users specify *what* they want, solvers compete to provide an execution plan, and settlement-time policies are enforced on-chain (e.g., via Uniswap v4 hooks) [4]. The design objective is verifiable settlement with explicit failure semantics and pluggable privacy.

3 Notation and Preliminaries

Let $\mathbb{B} = \{0, 1\}$ be the Boolean domain. For probabilistic polynomial-time algorithm A, write $y \leftarrow A(x)$. Let λ denote the security parameter.

We denote public-key encryption as $\mathsf{PKE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, fully homomorphic encryption as $\mathsf{FHE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$, and zero-knowledge proof systems as $\mathsf{ZK} = (\mathsf{Prove}, \mathsf{Verify})$.

4 System Architecture

JACKis decomposed into five orthogonal layers:

1. **Intent Layer**
2. **Solver and Coordination Layer**
3. **Privacy / Constraint Layer (CCM)**
4. **Execution Routing Layer**
5. **Settlement Adapter Layer**

4.1 Kernel Model

$$\mathcal{K} = \langle \mathcal{I}, \mathcal{S}, \mathcal{C}, \mathcal{R}, \mathcal{V} \rangle$$

where $\mathcal{I}$ denotes intent representation, $\mathcal{S}$ solver coordination, $\mathcal{C}$ privacy/constraint enforcement, $\mathcal{R}$ routing, and $\mathcal{V}$ settlement venues.

5 Reference Implementation Mapping (v1.0.2)

5.1 Control Plane and APIs

The current implementation maps architecture layers into operational API surfaces:

Layer	Interface	Behavior in v1.0.2
Intent Layer	`/api/intents`	Create/list intents, persist lifecycle metadata, normalize execution context.
Routing Layer	`/api/quote` + LI.FI SDK	Deterministic quote schema, explicit fallback mode when upstream quote/route/status calls fail.
Provider Telemetry	Yellow provider notifications	Authenticated notification ingestion, persistence, and replay-safe lifecycle updates.
Settlement Layer	Adapter + hook contracts	Submit settlement transaction and verify policy constraints during execution.
Observability	Intent state records	Canonical state machine trace for created/quoted/executing/settled/aborted/expired.

5.2 Deterministic Quote Contract

For a request q, the quote endpoint returns:

$$\mathsf{QuoteResult}(q) = \langle \texttt{mode}, \texttt{route}, \texttt{reasonCode}, \texttt{expiresAt} \rangle$$

where `mode` is either `provider` or `fallback`. This creates explicit downstream semantics for UI and solver behavior without silent partial failures.

6 Intent Model

6.1 Formal Intent Definition

An intent is:

$$I = \langle U, A, T, \Phi, \Omega \rangle$$

where U is user identifier, A target asset set, T destination execution environment, Φ private constraint payload, and Ω public execution envelope.

6.2 Public and Private Components

$$I = (I_{pub}, \mathsf{Enc}(I_{priv}))$$

In v1, encryption may be omitted depending on CCM implementation, but sensitive constraints should not be publicly broadcast in cleartext.

7 Solver-Based Execution

7.1 Competition Model

For candidate plan π, the kernel verifies:

1. compatibility with I_{pub},
2. satisfaction of constraints via CCM evidence,
3. verifiability of final settlement on venue v.

7.2 Minimal Economic Security (v1)

- User signs intent with max fee and deadline.
- Solver registers execution with bond b.
- Invalid or expired execution can trigger slash path.
- Winner receives fee after verified settlement.

8 Privacy / Constraint Layer: CCM

8.1 CCM Interface

$$\mathsf{CCM.Verify}(I_{pub}, \Phi, x) \to \mathbb{B}$$

where x are solver execution parameters (route, expected output, timing).

8.2 Implementation Paths

- Confidential execution/coprocessor with signed attestations (v1 path).
- FHE evaluation of constraint circuits (research track) [1, 2].
- ZK proofs over committed constraints (research track).

9 Routing and Provider Integration

9.1 Routing Graph

$$G = (V_{chains}, E_{bridges})$$

Edges encode cost, latency, and risk. In v1, route planning is delegated to LI.FI infrastructure with protocol-side allowlists and caps [5].

9.2 Yellow Notification Integration

Yellow provider callbacks are authenticated and persisted before state transitions. This creates an append-only event trace used to reconcile provider state and API state [6].

9.3 Failure Model

Cross-domain execution is best-effort:

$$\texttt{CREATED} \to \texttt{QUOTED} \to \texttt{EXECUTING} \to (\texttt{SETTLED} \mid \texttt{ABORTED} \mid \texttt{EXPIRED})$$

Provider or bridge faults transition to `ABORTED` with explicit reason codes. Partial completion is not treated as atomic.

10 Settlement Adapter Layer

Each venue v implements:

$$Execute(v, \pi) \to tx \qquad \text{and} \qquad Verify(v, tx) \to \mathbb{B}$$

Uniswap v4 pools with hooks act as policy-enforced settlement venues [4].

11 Process Diagrams

11.1 Intent Execution Lifecycle

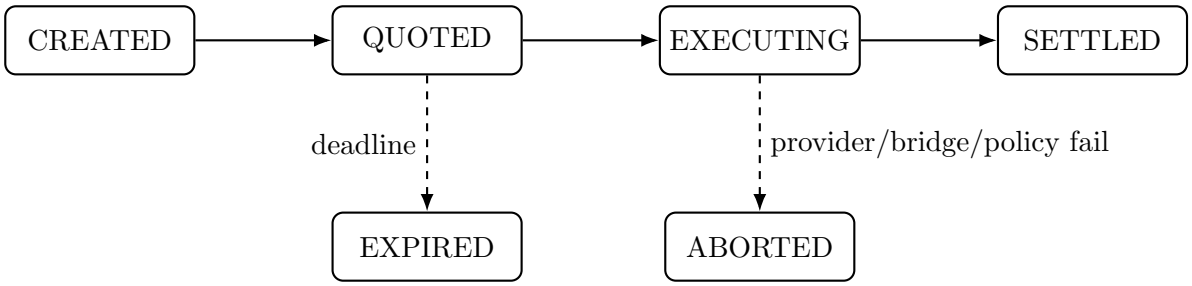

Figure 1: Canonical lifecycle enforced by JACK state transitions.

11.2 Quote and Fallback Flow

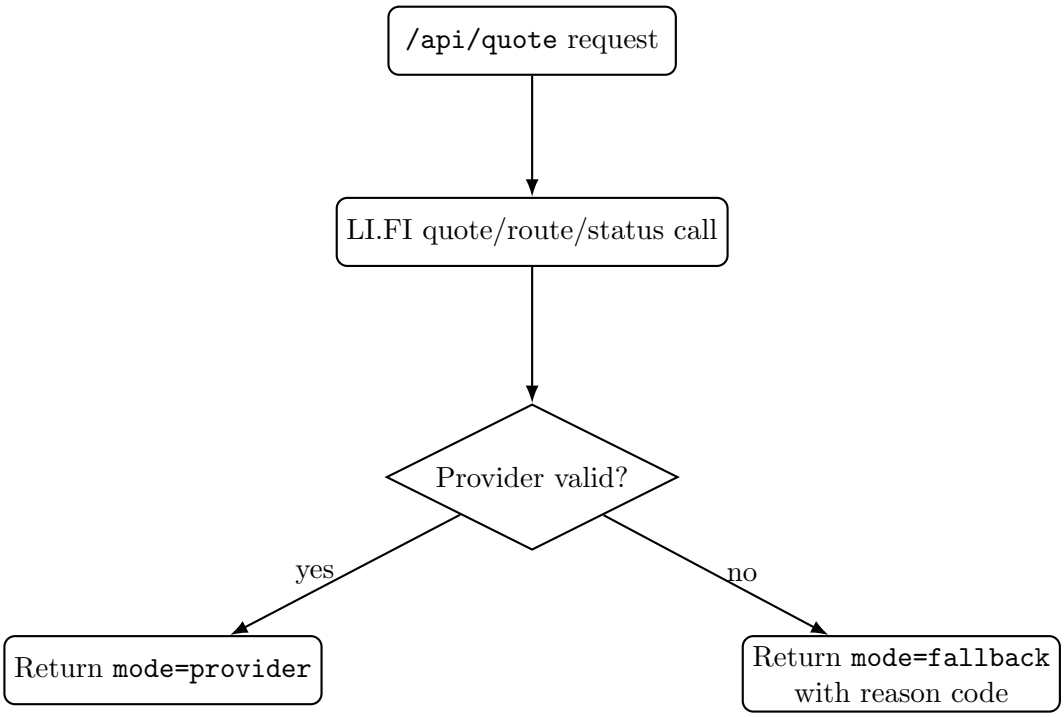

Figure 2: Deterministic quote response behavior with explicit fallback mode.

12 Execution Algorithm

Algorithm 1 JACKKernel Execution (v1)

User constructs and signs intent $I = (I_{pub}, \Phi)$
API stores intent and requests quote from LI.FI provider path
If provider quote invalid, return deterministic fallback quote result
Solvers submit candidate plans π with parameters x
for all solver submissions **do**
    Verify public compatibility with I_{pub}
    Verify CCM evidence: $\mathsf{CCM.Verify}(I_{pub}, \Phi, x) = 1$
end for
Select winning solver $\pi^\star$ under fee and policy constraints
Execute routing (best-effort) and ingest Yellow provider notifications
Submit settlement to venue v and enforce hook policy
Persist terminal state and emit public events

13 Verification and Observability

13.1 Execution Correctness

$$\mathsf{CCM.Verify}(\cdot) = 1 \ \wedge \ \mathsf{Verify}(v, tx) = 1$$

13.2 Operational Evidence

Observers can verify:

- deterministic quote mode and reason codes,
- lifecycle transitions with persisted provider notifications,
- settlement correctness and hook-level policy decisions.

14 Adversarial Model

We consider:

- malicious solvers (invalid routes, griefing, censorship),
- adversarial observers (MEV, timing inference),
- routing/provider failures (outages, stale routes, auth failures),
- malicious venue/hook logic (access control or reentrancy defects),
- oracle manipulation affecting policy checks.

15 Security Properties (v1)

1. **Policy enforceability**: settlement cannot bypass on-chain hook policy.
2. **Execution integrity**: execution is bound to signed intent and verified evidence.
3. **Fail-closed behavior**: missing evidence or policy violations abort execution.
4. **Deterministic APIs**: quote response mode is explicit and machine-checkable.
5. **Risk containment**: allowlists, caps, and circuit breakers limit blast radius.

16 Evaluation Plan (v1.0.2)

Measure:

- intent-to-settlement latency,
- provider success vs fallback rate for `/api/quote`,
- yellow notification ingestion latency and replay rejection rate,
- hook gas overhead and policy rejection rate,
- route success under degraded bridge/provider conditions.

17 Limitations and Future Work

- stronger economic security (auctions, staking, slashing),
- decentralized solver participation and anti-collusion mechanisms,
- formal verification and attestation registries for hooks,
- cross-domain atomicity/dispute mechanisms,
- production-ready proof-carrying private constraints.

18 Conclusion

JACKtreats execution as a programmable primitive: users submit intents, solvers compete to satisfy them, routing is delegated with deterministic contracts, and on-chain venues enforce policy at settlement. Version v1.0.2aligns the specification with the implemented LI.FI and Yellow provider flows while preserving a clear path to stronger privacy and cross-domain guarantees.

References

[1] Craig Gentry. Fully Homomorphic Encryption Using Ideal Lattices. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing (STOC)*, 2009.

[2] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast Fully Homomorphic Encryption over the Torus. *Journal of Cryptology*, 33(1), 2020.

[3] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Ittay Eyal, and Emin Gün Sirer. Flash Boys 2.0: Frontrunning, Transaction Reordering, and Consensus Instability in Decentralized Exchanges. In *IEEE Symposium on Security and Privacy*, 2020.

[4] Uniswap Labs. Uniswap v4 Core Architecture. https://github.com/Uniswap/v4-core, 2024.

[5] LI.FI. LI.FI Integrator Documentation. https://docs.li.fi/, 2026.

[6] Yellow Network. Yellow Network Provider and Notification Documentation. https://docs.yellow.org/, 2026.