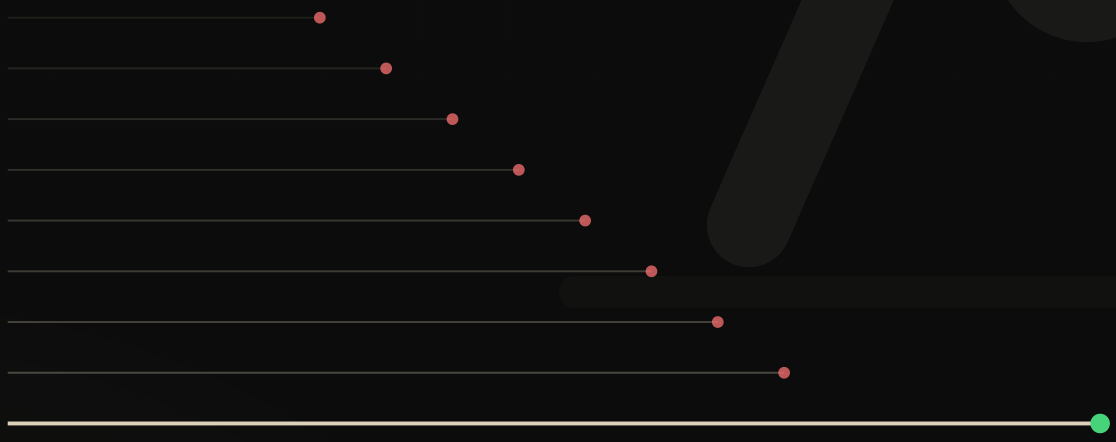
Ava

LITEPAPER

AVA LITEPAPER · VERSION 1.0 · NOVEMBER 2026

# A mandate-bounded execution layer for autonomous agents

The agent proposes. Ava decides, signs, settles and records, under authority a human granted in advance and the agent cannot widen.




AUTHORS

Kamal Nayan Singh · Aarav Raj


VERIFIED AGAINST

commit eff936c, 10 November 2026

READ MORE

getava.xyz · x.com/avatheaiagent · t.me/avatheaiagent

ABSTRACT

Language-model agents can now plan financial actions but cannot be trusted to hold the keys that carry them out. Ava is an execution layer that separates the two: an agent proposes, and a deterministic service decides, signs, settles and records under authority that a human granted in advance and that the agent cannot widen.

This paper formalizes Ava as a labelled state-transition system over principals, mandates, capability rows, execution claims and receipts, and derives the conditions under which a signature may exist.

i **Mandates.** A mandate binds capital, constraints and exit conditions under a terms hash that is invariant to lifecycle changes.

ii **Capability gate.** A three-predicate gate separates code existence, reachability and operator consent from historical proof.

iii **Preview-hash confirmation.** Human approval is a property of a recomputed digest rather than of a request flag, so an agent cannot confirm on a human's behalf.

iv **Single-flight claims.** An idempotency key is bound to the canonical digest of an economic action, with cumulative budget and a wallet nonce reserved atomically.

v **Receipt chain.** Every outcome is hashed under RFC 8785, linked to its predecessor, optionally signed with Ed25519 and periodically anchored on a public ledger, with integrity, on-chain proof and settlement agreement kept as separate verdicts.

We state the threat model explicitly. Ava is a custodial, single-operator service, not a consensus protocol. Its guarantees are conditional on the honesty of its operator, its key custodian and its RPC providers, and we identify which properties survive the failure of each. We prove safety propositions for the gate ordering and budget conservation, characterize the residual risks the implementation leaves open, and propose a reproducible evaluation methodology instead of reporting benchmarks that do not exist.

| | | |
|---|---|---|
| **7**<br>**mainnet settlements**<br>on Base, Avalanche, BNB Chain and Monad | **5**<br>**ordered steps**<br>before a signature can exist | **14**<br>**execution states**<br>4 terminal; none re-signs |
| **29**<br>**MCP tools**<br>18 advertised by default | **26**<br>**durable tables**<br>behind one API process | **~1,000**<br>**tests**<br>and scripted gate checks |

KEYWORDS autonomous agents · delegated authority · intent execution · idempotency · transparency logs · EIP-712 · RFC 8785 · custodial key management · settlement verification · Model Context Protocol

CONTENTS


| | | |
|---|---|---|
| 01 | Introduction and problem statement | 04 |
| 02 | Design goals, non-goals and claims Ava does not make | 06 |
| 03 | Background and related work | 07 |
| 04 | System model | 08 |
| 05 | Protocol architecture | 11 |
| 06 | Core mechanisms | 12 |
| 07 | Protocol algorithms | 21 |
| 08 | State transitions and execution lifecycle | 24 |
| 09 | Cryptographic foundations | 26 |
| 10 | Security model and threat analysis | 28 |
| 11 | Correctness and theoretical properties | 32 |
| 12 | Economic analysis | 34 |
| 13 | Complexity, performance and evaluation methodology | 35 |
| 14 | Implementation architecture | 37 |
| 15 | Limitations, trade-offs and implementation discrepancies | 39 |
| 16 | Future research directions | 41 |
| 17 | Conclusion | 42 |
| | References | 42 |
| | Appendix A. Notation | 43 |
| | Appendix B. Refusal codes and constants | 44 |


Figures are new in this edition: 15 diagrams of the architecture and mechanisms. Every claim about the implementation was checked against commit eff936c.

01

# Introduction and problem statement

A language model asked to supply 300 USDC to a lending market can describe the transaction, encode its calldata, and even explain the risks. It cannot execute it safely, because execution requires a signing key, and a key held by a model is a key held by whatever text the model last read. Prompt injection, hallucinated addresses and silent misreadings of an amount are not edge cases for such a holder; they are its normal operating environment.

The obvious remedies each fail in a specific way. Asking the human to sign every transaction removes the agent's usefulness for anything recurring. Giving the agent a hot wallet with a spending cap bounds the loss but not its direction: the cap says how much, not into what. Smart-contract session keys [ERC-4337, ERC-7579] constrain targets and selectors on chain, but cannot express an off-chain notion such as "the amount the bridge actually delivered" or "only if the preview a human saw is the one being signed."

Ava takes a different position. The agent never holds a key and never decides. It submits a request to a deterministic service; the service holds the key in a custodial enclave (Turnkey), evaluates the request against authority recorded earlier by a human, and either refuses with a typed reason or signs exactly what was authorised. Every execution that reaches a chain is written as a receipt to an append-only, hash-linked ledger that a third party can check without an Ava credential; mandate refusals are recorded separately and counted on the agent's track record.

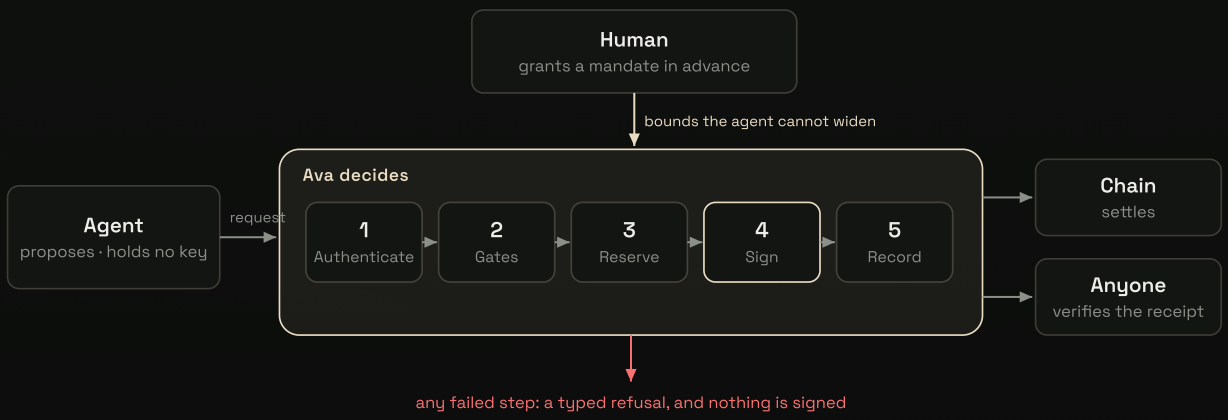



**FIGURE 1** The shape of the system. The agent proposes; Ava decides under bounds a human set in advance; the chain settles; anyone can verify the receipt.

## 1.1 The problem, stated precisely

Let a principal $u$ hold capital and let an agent $g$ act for $u$. The agent emits a stream of requests $r_1, r_2, \ldots$ over an untrusted channel. Each request names an action $a = (k, c, v, x, q)$: an action kind $k$ (lend, withdraw, swap, bridge), a chain $c$, a venue $v$, an asset $x$ and an amount $q$. Let $\mathrm{Auth}(u)$ be the set of actions $u$ has authorised, given as a mandate, a signed plan or a signed standing envelope. Let $\mathrm{Exec}(c, v)$ be true when the operator can actually route kind $k$ to venue $v$ on chain $c$.

The execution problem is to build a function that maps each request to either a refusal or a signed transaction, such that four properties hold.

**Authority.** A transaction is signed for action $a$ only if $a \in \mathrm{Auth}(u)$ and $\mathrm{Exec}$ holds for $a$'s route.

**Confirmation.** On mainnet, a transaction is signed only if a human approved an artifact whose digest equals the digest of what is signed.

**At-most-once.** Two requests that describe the same economic action under the same idempotency key produce at most one signature.

**Faithful record.** Every reported fill corresponds to a finalized on-chain effect whose amount, asset and recipient equal the authorised ones, and the record of fills cannot be silently rewritten.

Section 4 makes these definitions formal. Sections 6 and 11 show how the implementation meets them and where it does not.

## 1.2 Contributions

- A formal model of an agent execution layer as a gated state machine whose only path to a signature passes through an ordered conjunction of predicates (Sections 4, 6, 8).
- A confirmation primitive, preview-hash binding, that removes the agent from the set of parties able to assert human consent (Section 6.3).
- A claim identity that defines "the same economic action" as a canonical digest of nine fields, with atomic budget and nonce reservation (Section 6.5).
- A receipt model that keeps integrity, proof and settlement as orthogonal verdicts and anchors the log on a public chain (Sections 6.7, 9).
- A threat analysis that states which guarantees depend on the operator, the custodian and the RPC providers (Section 10), and an honest list of where the implementation diverges from its own design (Section 15).

## 1.3 Scope of claims

Everything described as implemented here was verified against the repository at commit `eff936c` (10 November 2026). Mainnet settlement is evidenced by seven transactions listed in the repository and re-read from public nodes at build time: Morpho on Base (two), Aave v3 on Avalanche, BNB Chain and Monad (one, one and two), and a LI.FI-routed swap on BNB Chain. A historical settlement is evidence that a path worked once; it is not a claim that the path is enabled now.




**FIGURE 2** Mainnet settlements behind the claims in this paper: seven transactions on four chains.

02

# Design goals, non-goals and claims Ava does not make

The design follows from one asymmetry.

> A false refusal costs the user a retry; a false fill costs the user money and teaches the agent to trust a lie.

Every goal below is that asymmetry applied to a different surface.

| GOAL | STATEMENT | MECHANISM (SECTION) |
|---|---|---|
| **G1** Fail closed | Any absent input on the path to a signature (credential, mandate, route, adapter, operator switch, idempotency store) produces a typed refusal, never a default. | Gate conjunction (6.2, 7) |
| **G2** Separation of proposal and authority | The party that proposes an action is never the party that authorises it. | Spend audience, preview binding, controlled-signer exclusion (6.1, 6.3) |
| **G3** Kind primacy | An action is routed by kind to its protocol family and never falls through to a different venue. | Registry keyed by kind × chain × venue (6.2) |
| **G4** At-most-once signing | A retried or concurrent duplicate cannot produce a second signature. | Claim store and state machine (6.5, 8) |
| **G5** Claimed versus proven | A fill is reported only after finality and an effect comparison against the authorisation. | Attestation, settlement standing (6.6) |
| **G6** Third-party verifiability | Anyone holding a receipt hash can check integrity, chain linkage, mandate binding and on-chain agreement. | Receipt chain, anchors (6.7, 9) |
| **G7** One implementation, two transports | MCP and REST reach the same functions, so no gate exists on one surface only. | Shared executors (14) |

## 2.1 Non-goals

- **No consensus.** Ava is operated by one party. It does not replicate state across mutually distrusting nodes and makes no Byzantine fault tolerance claim.
- **No trustless custody.** Ava-provisioned wallets are custodial. Keys live in Turnkey, in a sub-organization per user, and the Ava API is authorised to request signatures from them. A user who needs non-custodial operation links an external wallet, which Ava observes but never signs for.
- **No token or incentive layer.** There is no Ava token, staking, fee market or reward. Section 12 analyses only operating costs and adversarial cost.
- **No zero-knowledge proofs.** Nothing in the system proves a statement in zero knowledge. Verification is by recomputation over public or disclosed data.
- **No price oracle security.** USD notionals use public price APIs. Policy limits are therefore as reliable as those feeds (Section 10).

### 2.2 What this paper means by ”verifiable”

A property is **verifiable** in this paper when a party outside Ava, holding only public chain data and a disclosed receipt, can recompute it. It is **operator-attested** when only Ava’s own database can confirm it. Throughout, each guarantee is labelled with which of the two it is.

03

# Background and related work

Ava composes known primitives. Its contribution is the arrangement: which predicate guards which transition, and which party is allowed to satisfy it.

**Protection principles.** Saltzer and Schroeder’s fail-safe defaults and least privilege [SS75] are the direct ancestors of G1 and G2. Ava’s deny-by-default route allowlist and its refusal to read identity from request bodies are fail-safe defaults applied to an HTTP surface.

**Programmable accounts and session keys.** Ethereum made authority programmable at the account level [Bu14, Wo14]. ERC-4337 [ERC4337] and modular accounts [ERC7579] let a session key act within on-chain rules on targets, selectors and value. These rules are enforced by consensus and are therefore stronger than anything an off-chain service offers. They are also limited to what the EVM can observe in one transaction. Ava’s predicates refer to off-chain facts (a human’s approval of a rendered preview, a bridge’s delivered amount, a registry switch) and to cross-chain sequences. The two are complementary: Section 16 proposes moving Ava’s per-move bounds into an on-chain validator.

**Intents and solvers.** Intent-based systems such as CoW Protocol, UniswapX and the ERC-7683 cross-chain intent standard [ERC7683] let a user sign what they want and let solvers compete on how. Ava’s intents differ in who signs: the user signs bounds (a mandate, a plan, a standing envelope) and Ava, not a solver market, executes within them. Ava routes swaps through solver markets where they exist and does not compete with them.

**Typed structured signatures.** EIP-712 [EIP712] gives wallets a human-readable, domain-separated digest to sign. Ava uses it for three authorisation objects (Section 9.3), each with its own domain name so that a signature for one can never verify as another.

**Tamper-evident logging.** Hash-chained logs date to Haber and Stornetta’s time-stamping [HS91] and Schneier and Kelsey’s secure audit logs [SK99]; Crosby and Wallach analysed their efficiency and auditing [CW09]; Certificate Transparency deployed them at Internet scale with Merkle trees and gossip [RFC6962]. Ava’s receipt chain is the simplest member of this family, a per-scope linear hash chain, and inherits its known weakness: without an external witness the log owner can rewrite history. Ava’s anchor transactions are that witness, published on a public chain in the manner of [HS91].

**Idempotency and exactly-once effects.** Helland [He12] argues that at-least-once delivery plus idempotent operations is the practical substitute for exactly-once execution. Ava applies this to blockchain transactions, where the operation is not naturally idempotent (each signature consumes a nonce and moves funds), so idempotency must be imposed by a reservation keyed on the economic content of the action.

**Agents and adversarial inputs.** Indirect prompt injection [Gr23] shows that an agent reading untrusted content can be steered by it. Ava’s design assumes the agent is compromised in this sense and places no security-relevant decision inside the model.

**Agent transport.** The Model Context Protocol [MCP25] standardises how a model calls tools. Ava exposes its surface as MCP tools over JSON-RPC on HTTP; MCP itself provides no authorisation semantics beyond transport, so Ava authenticates per tool call.

> WHAT IS NEW
>
> To our knowledge, no prior system combines (a) a human-consent primitive bound to a recomputed artifact digest rather than a flag, (b) at-most-once signing keyed on a canonical economic digest with atomic cumulative-budget and nonce reservation, and (c) receipts whose settlement agreement is evaluated separately from their existence on chain, in a service whose primary client is a language model.

04

# System model

We model Ava as a labelled transition system whose state is the contents of its durable stores and whose labels are authenticated requests. The model abstracts HTTP, MCP and storage engines; Appendix A collects the notation.

## 4.1 Universes

$$\begin{aligned}
&\mathcal{U} : \text{principals (users)} && \mathcal{G} : \text{agent instances},\ \text{own} : \mathcal{G} \to \mathcal{U} \\
&\mathcal{C} : \text{chains} && \mathcal{V} : \text{venues} \\
&\mathcal{X} : \text{assets},\ d : \mathcal{X} \to \mathbb{N} && \mathcal{K} = \{\textsf{lend}, \textsf{withdraw}, \textsf{swap}, \textsf{bridge}, \textsf{transfer}, \textsf{stake}, \textsf{hold}\} \\
&\mathcal{W} : \text{wallets},\ \phi : \mathcal{W} \to \{\textsf{evm}, \textsf{solana}, \textsf{sui}\} && \mathbb{B} = \{0,1\}^{256}\ (\text{digests})
\end{aligned}$$

Amounts are exact. An amount of asset $x$ is a pair $(n, d(x))$ with $n$ a non-negative integer in base units; arithmetic is defined only between amounts of equal scale (the implementation refuses mixed scales with `EXECUTION_CLAIM_BUDGET_MISMATCH`). We write $q$ for an amount in base units.

An **action** is a tuple

$$a = (k, c, v, x, q, \omega, \rho) \in \mathcal{K} \times \mathcal{C} \times \mathcal{V} \times \mathcal{X} \times \mathbb{N} \times \mathcal{W} \times \mathcal{T}$$

where $\omega$ is the signing wallet and $\rho$ the target contract actually called.

## 4.2 Credentials and principals

A bearer secret $\tau$ is resolved by the authentication map

$$\mathrm{Res}(\tau) = \begin{cases} (u, \textsf{api_key}, \textsf{spend}) & \text{if } (u,\tau) \in \textsf{Keys} \\ (u, \textsf{session}, \alpha) & \text{if } (H(\tau), u, \alpha) \in \textsf{Tok} \ \wedge\ \neg\text{revoked} \qquad \alpha \in \{\textsf{spend}, \textsf{dashboard}\} \\ \bot & \text{otherwise} \end{cases}$$

with $H = \text{SHA-256}$. Only $H(\tau)$ is stored. A request carrying a $\tau$ with $\mathrm{Res}(\tau) = \bot$ is refused (401) and never downgraded to anonymous. Any identity asserted in a header, query or body is a *claim* $\iota$; the request is refused (403) whenever $\iota \neq u$. The predicate that admits live spend is

$$\mathrm{CanSpend}(\pi) \iff \pi.\text{kind} = \textsf{api_key} \ \vee\ (\pi.\text{kind} = \textsf{session} \wedge \pi.\alpha = \textsf{spend})$$

(a developer-only header principal also satisfies it, but is unreachable when authentication is enforced, which is the default).

## 4.3 State

The global state is the tuple of durable stores

$$S = (\mathsf{Ten},\ \mathsf{Tok},\ \mathsf{M},\ \mathsf{Reg},\ \Pi,\ \mathsf{Cl},\ \mathsf{Bud},\ \mathsf{Non},\ \mathsf{PR},\ \mathsf{Env},\ \mathsf{Rev},\ \mathsf{L},\ \mathsf{Anc})$$

| SYMBOL | STORE | TABLE(S) |
|---|---|---|
| Ten | users, agent instances, wallets, Turnkey sub-organization per user | `ava_tenancy_*` |
| Tok | token digests with audience | `ava_auth_tokens` |
| M | mandates | `ava_mandates` |
| Reg | capability registry (process constant) | in code |
| Π | policy envelope and limits (process constant per mode) | in code and environment |
| Cl | execution claims, keyed by claim key | `ava_execution_claims` |
| Bud | cumulative budget per mandate hash | `ava_execution_budgets` |
| Non | nonce lease per (chain, wallet) | `ava_execution_nonce_leases` |
| PR | policy reservations (notional, exposure) | `ava_policy_reservations` |
| Env, Rev | standing envelopes, revocations | `ava_standing_*` |
| L | receipt ledger and chain | `ava_receipts`, `ava_receipt_chain` |
| Anc | confirmed anchors | `ava_receipt_anchors` |

The initial state $S_0$ has every store empty, Reg and Π set from code and environment, and no tokens.

## 4.4 Transitions

A request is a labelled input $\ell = (\tau, \mathrm{op}, \mathrm{args}, \kappa)$ where $\kappa$ is an optional caller-supplied idempotency key. The system is

$$\mathcal{S} = (\mathbf{S},\ \mathcal{I},\ \delta,\ S_0), \qquad \delta :\ \mathbf{S} \times \mathcal{I} \times \Omega \to \mathbf{S} \times \mathcal{O}$$

where $\Omega$ is the environment (chain state, RPC responses, custodian responses, prices, wall-clock time) and the output space is

$$\mathcal{O} = \{\mathsf{refuse}(\mathrm{code}, \mathrm{detail})\} \cup \{\mathsf{preview}(P, h_P)\} \cup \{\mathsf{filled}(\rho_R)\} \cup \{\mathsf{unconfirmed}(\rho_R)\} \cup \{\mathsf{view}(\cdot)\}$$

Every refusal carries `filled = false` and `liveSubmit = false`. Only the filled arm may carry a transaction hash that the MCP layer marks `mayClaimFill = true`.

## 4.5 Parties and trust

| PARTY | ASSUMED | CONSEQUENCE IF THE ASSUMPTION FAILS |
|---|---|---|
| Agent $g$ | Adversarial. Arbitrary inputs, replays, concurrency; may read injected content. | None by design: no security decision depends on $g$. |
| Network | TLS terminates at the edge; the adversary sees ciphertext only. | Bearer tokens leak; see 10.2. |
| Human $u$ | Honest about their own intent; may be phished. | Approves a harmful but correctly displayed preview. |
| Operator O (Ava process) | Honest and correctly configured. | Total: O holds the authority to request signatures. Receipts and anchors limit undetected rewriting (10.6). |
| Custodian K (Turnkey) | Signs only when O's API credential asks, in the right sub-organization. | Keys misused outside Ava; out of Ava's control. |
| RPC providers | Return canonical chain data. | False confirmation or false balance; mitigated by multiple Base URLs and pinned reads (10.5). |
| Price source | Approximately correct USD prices. | Notional limits mis-evaluated (10.7). |

## 4.6 Adversary

The adversary $\mathcal{A}$ controls every agent, may obtain any dashboard-audience token (for example by phishing a device link), may submit any number of concurrent and replayed requests, and may observe all public chain data. $\mathcal{A}$ cannot invert SHA-256 or Keccak-256, cannot forge secp256k1 ECDSA or Ed25519 signatures, and does not control O, K or a majority of the RPC endpoints O consults. Section 10 relaxes each of these.

## 4.7 Invariants

The implementation is designed to maintain, for every reachable state:

**I1 (no signature without authority).** If a claim is in a state where a signature may exist, then its subject was admitted by every gate in Section 6.2.

**I2 (budget conservation).** For every mandate hash $m$: $\mathrm{consumed}(m) + \mathrm{reserved}(m) \le L(m)$, where $L(m)$ is the cumulative limit bound at first reservation.

**I3 (one nonce per reservation).** Distinct live reservations on the same (chain, wallet) hold distinct nonces.

**I4 (key binding).** An idempotency key, once bound to a request digest, is never rebound to a different digest.

**I5 (chain integrity).** For consecutive receipts in one chain scope, $\mathrm{prev}(R_{i+1}) = h(R_i)$ and $\mathrm{seq}(R_{i+1}) = \mathrm{seq}(R_i) + 1$.

**I6 (proof monotonicity).** A receipt reported as filled has proof standing chain-confirmed.

Section 11 states which of these are proved from the code, which hold only under assumptions, and which have known exceptions.

05

# Protocol architecture

Ava has three trust zones. The agent and the human sit outside it and make no security decision. The Ava process is the only party that can ask the custodian for a signature, and it asks only after five ordered steps. The external services are trusted only for their stated jobs: storage, key custody, chain reads and prices.

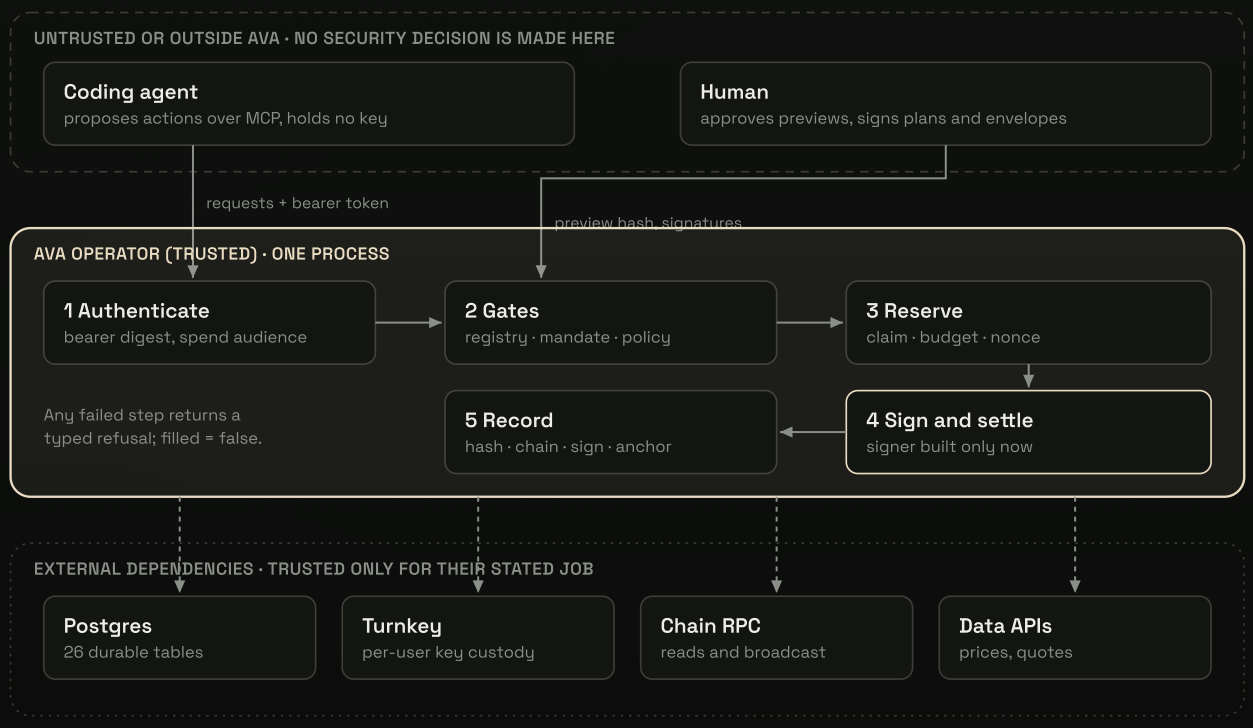



**FIGURE 3** Protocol architecture: three trust zones, five ordered steps. Only the Ava process can ask for a signature, and only after every gate.

Read top to bottom. A request enters at step 1 carrying a bearer token, and the human's preview hash and signatures enter at step 2. The signer exists only at step 4, after steps 1 to 3 have passed, and every outcome is recorded at step 5. A failure at any step returns a typed refusal and nothing is signed.

The zones map to the trust table of Section 4.5. The dashed boundary marks parties the design treats as adversarial or merely untrusted. The solid boundary marks the single trusted operator, whose compromise Section 10.6 treats separately. The dotted boundary marks dependencies whose failure modes Sections 10.5 and 10.7 bound.

06

# Core mechanisms

Each mechanism below is stated as a definition, then as the predicate or update it contributes to the transition function $\delta$, then with its failure codes. Section 7 composes them into algorithms.

## 6.1 Mandates and the terms hash

**DEFINITION 6.1 (MANDATE)**

A mandate is a record

$$M = (\mathrm{id},\ u,\ \mathrm{portal},\ \mathrm{obj},\ \underbrace{(x, Q, c)}_{\text{capital}},\ \Gamma,\ \Phi,\ \mathrm{act},\ \mathrm{st},\ t_0,\ \sigma_M)$$

with objective $\mathrm{obj} \in \{\mathbf{earn}, \mathbf{hold}, \mathbf{trade}, \mathbf{rebalance}, \mathbf{protect}\}$, capital $Q$ of asset $x$ on chain $c$, constraints $\Gamma$ = (maxDrawdownPct, maxSlippageBps, maxNotionalUsd, allowedKinds, preferredVenue), exit conditions $\Phi$, a derived action act, a status $\mathrm{st} \in \{\mathbf{active}, \mathbf{paused}, \mathbf{triggered}, \mathbf{completed}, \mathbf{cancelled}\}$, creation time $t_0$ and an optional EIP-712 signature $\sigma_M$.

**DEFINITION 6.2 (TERMS AND TERMS HASH)**

The terms of $M$ are the projection $T(M) = (\mathrm{id}, u, \mathrm{portal}, \mathrm{obj}, \mathrm{capital}, \Gamma, t_0)$, and

$$h_M \ = \ \mathrm{SHA256}\big(\,\mathrm{canon}(T(M))\big).$$

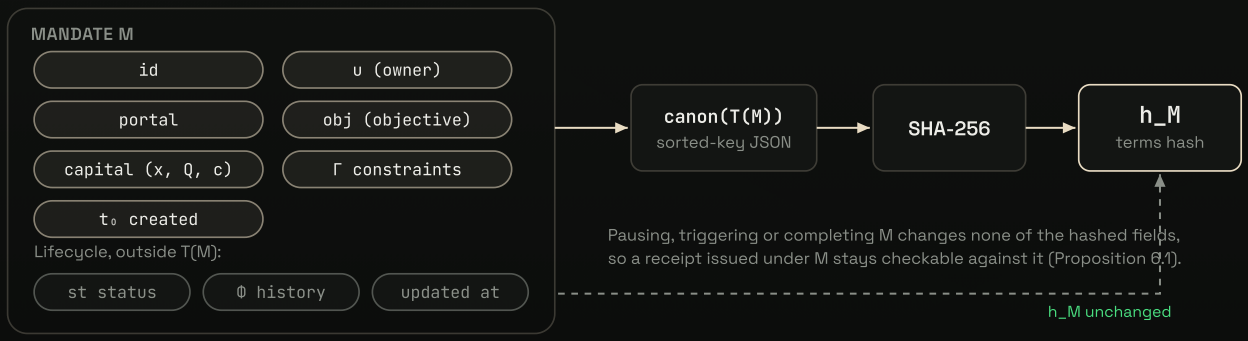



**FIGURE 4** What the terms hash covers. Lifecycle fields sit outside the projection, so a mandate's hash survives pausing, triggering and completion.

**PROPOSITION 6.1 (LIFECYCLE INVARIANCE)**

Let $M'$ differ from $M$ only in st, $\Phi$-evaluation history, or update time. Then $h_{M'} = h_M$.

*Proof.* $T$ omits exactly those fields, and canon is a function of its input. ∎

The consequence is that a receipt issued under $M$ remains checkable against $M$ after $M$ is paused, triggered or completed. An amendment that changes $\Gamma$ or capital produces a new terms hash; the verifier reports a receipt whose stored hash matches an earlier version as *terms-amended* rather than *mismatch*.

**Mandate gate.** For a request by $u$ to perform an action of kind $k$ with USD notional $\nu$:

$$\begin{aligned}\mathrm{MGate}(M,u,k,\nu) \iff\ & M.u = u \ \wedge\ M.\mathrm{st} = \textsf{active} \\ & \wedge\ \big(\Gamma.\mathrm{allowedKinds} = \varnothing \vee k \in \Gamma.\mathrm{allowedKinds}\big) \\ & \wedge\ \big(\Gamma.\mathrm{maxNotionalUsd} = \varnothing \vee \nu \le \Gamma.\mathrm{maxNotionalUsd}\big)\end{aligned}$$

A failed ownership test returns 404 rather than 403, so the existence of another user's mandate id is not disclosed. Refusals are persisted to the agent's record. For withdrawals the notional term is omitted, because returning capital does not commit more of it.

**Condition evaluation.** Each condition $\varphi \in \Phi$ is a tuple $(x, \mathrm{metric}, \triangleright, \theta)$ with metric $\in$ {price, price_change_pct} and $\triangleright \in \{<, \le, =, \ge, >\}$. Given a snapshot $s$ (a live CoinGecko price when none is supplied), evaluation returns the set of matched indices $\{i : \mathrm{metric}_i(s) \triangleright_i \theta_i\}$ and moves st to triggered when it is non-empty. Evaluation never dispatches an action.

## 6.2 The capability predicate

The registry Reg is a finite set of rows indexed by $(k, c, v, \mathrm{adapter})$. Each row carries five independent booleans and an execution kind:

$$\rho = (k, c, v, \mathrm{ad};\ \mathrm{impl},\ \mathrm{api},\ \mathrm{mcp},\ \mathrm{op},\ \mathrm{proven};\ \mathrm{ek}), \qquad \mathrm{ek} \in \{\textsf{live}, \textsf{dry-run}, \textsf{descriptor-only}\}$$

**DEFINITION 6.3 (EXECUTABILITY ON SURFACE S)**

$$\mathrm{Exec}_s(\rho) \iff \mathrm{impl} \wedge (\mathrm{ek} = \textsf{live}) \wedge \mathrm{reach}_s \wedge \mathrm{op}, \qquad \mathrm{reach}_{\mathsf{api}} = \mathrm{api},\ \mathrm{reach}_{\mathsf{mcp}} = \mathrm{mcp}.$$

The flag **proven** (a transaction has settled on mainnet through this row) is deliberately absent from Exec. History is evidence, not permission: a row that settled once and was then switched off is not executable, and a row that is executable but has never settled (Aave v3 on Arbitrum at the time of writing) is reported as such. Every row defaults all five flags to false, so a row added without review is inert.

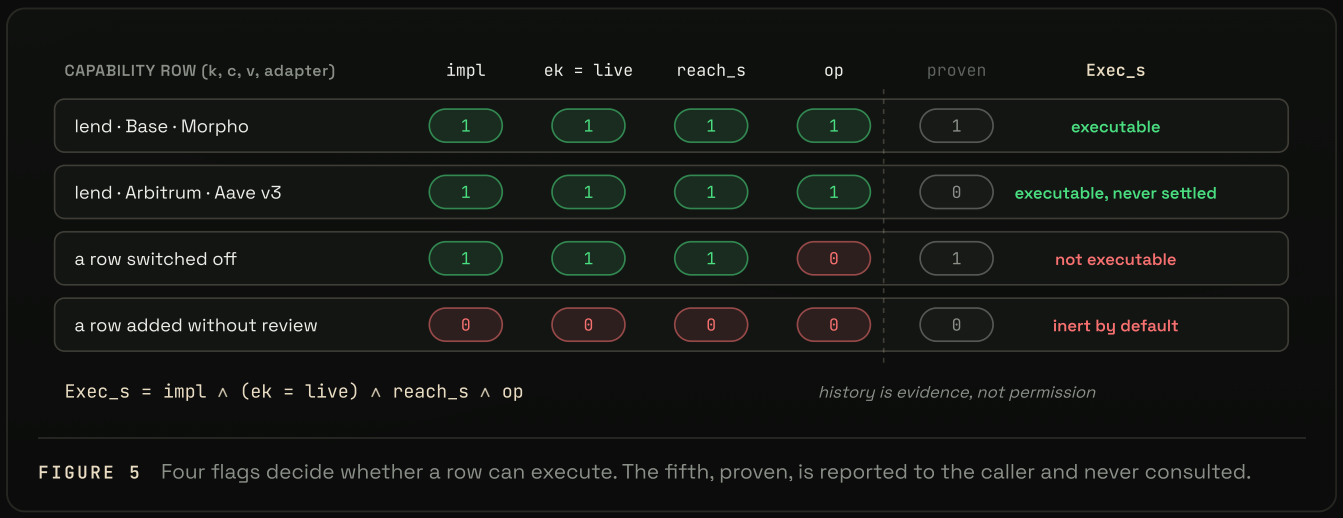



**FIGURE 5** Four flags decide whether a row can execute. The fifth, proven, is reported to the caller and never consulted.

The four failure modes are reported in the order a fix must happen (not implemented, descriptor only, dry run only, unreachable, operator disabled), because each is fixed by a different actor: a build, a wiring change, or a switch.

**Kind primacy.** Routing is a partial function $\mathrm{route}(k, c, v) \to \rho$. If the request names a venue $v$ not wired for $(k, c)$, the result is `LEND_VENUE_MISMATCH`; there is no fallback $v' \neq v$. Formally, for all requests, route is either the named row or undefined.

## 6.3 Preview-hash binding

The confirmation mechanism separates *what* a human approves from *who* transmits the approval.

> **DEFINITION 6.4 (PREVIEW PRE-IMAGE AND DIGEST)**
>
> For a lend action $a$ under mandate $M$, the pre-image is
>
> $$P(a, M) = (v,\ \mathrm{chainId},\ \mathrm{marketId},\ \omega,\ \omega_{\mathrm{recv}},\ k,\ x,\ q,\ M.\mathrm{id},\ h_M), \qquad h_P = \mathrm{SHA256}\big(\mathrm{canon}(P)\big).$$

The server recomputes $h_P$ from its own state on every request. The request's supplied value $\hat{h}$ is compared with it:

$$\mathrm{humanConfirmed} \iff \hat{h} \neq \bot \ \wedge\ \hat{h} = h_P.$$

The policy engine requires humanConfirmed whenever the order notional exceeds a threshold $\theta$. On mainnet $\theta = 0$ USD, so every mainnet order requires it. The first call therefore always returns `refuse(POLICY_HUMAN_CONFIRMATION_REQUIRED)` carrying $P$ and $h_P$; the second, carrying $\hat{h}$, may proceed.




**FIGURE 6** Preview-hash binding. The second call succeeds only if it carries the digest of exactly what was previewed; the server never learns whether a human looked.

PROPOSITION 6.2 (CONTENT BINDING)

If a signature is produced for action $a$ under $M$ in a mainnet request with supplied digest $\hat{h}$, then $\hat{h} = h_P(a, M)$. Consequently any change to venue, chain, market, owner, receiver, operation, asset, amount or mandate terms between approval and signing causes refusal, except with probability bounded by SHA-256 second-preimage resistance.

*Proof sketch.* The signer is constructed only after the policy gate returns allow, which on mainnet requires humanConfirmed, which requires $\hat{h} = h_P$ computed from the same $a$ and $M$ that the executor subsequently signs. A different $a'$ with $h_P(a') = \hat{h}$ is a second preimage. ∎

**What the mechanism does not prove.** $h_P$ is returned to the caller in the refusal. An agent can therefore resend it without showing anything to a human. Preview binding guarantees that *what is signed equals what was previewed*; it does not guarantee that *a human looked*. Human presence is an obligation of the client (the agent's UI must display $P$ and wait), not a property the server can verify. Section 10.3 analyses this gap and Section 16 proposes closing it with a user-held key.

Note also that $P$ contains no chain state. Two calls with identical parameters yield the same $h_P$ even if the position or balance has changed between them; balance sufficiency is checked separately by the executor before signing.

## 6.4 Policy evaluation

Let $\Pi_{\text{mode}} = (\nu_{\text{order}}, \nu_{\text{day}}, E_{\max}, s_{\max}, \theta, \text{kill})$ be the limits for mode ∈ {testnet, mainnet}. The implemented defaults are

| LIMIT | SYMBOL | MAINNET | TESTNET |
|---|---|---|---|
| Maximum order notional | $\nu_{\text{order}}$ | 2,500 USD | 1,000,000 USD |
| Maximum rolling 24 h notional | $\nu_{\text{day}}$ | 10,000 USD | 10,000,000 USD |
| Maximum open exposure | $E_{\max}$ | 25,000 USD | 10,000,000 USD |
| Maximum slippage | $s_{\max}$ | 100 bps | 500 bps |
| Human confirmation above | $\theta$ | 0 USD | 1,000,000 USD |

The USD notional of an action uses a spot price $p_x$ from a public price API:

$$\nu(a) = \frac{q}{10^{d(x)}} \cdot p_x.$$

The signing gate returns allow iff no violation is found and confirmation is satisfied:

$$\text{PGate}(a) = \begin{cases} \text{reject} & \text{if some } v \in \mathcal{V}(a) \text{ holds} \\ \text{confirm} & \text{if no violation} \wedge \nu(a) > \theta \wedge \neg\text{humanConfirmed} \\ \text{allow} & \text{otherwise} \end{cases}$$

$$\mathcal{V}(a) = \{\neg\text{allowlist}(v, c, x),\ \nu(a) > \nu_{\text{order}},\ s(a) > s_{\max},\ D + \nu(a) > \nu_{\text{day}},\ E + \Delta E(a) > E_{\max},\ \text{expired},\ \text{kill}\}$$

where $D$ is the rolling daily notional and $E$ the open exposure read at evaluation time. An ERC-4626 deposit or withdrawal has no price path, so $s(a) = 0$ for those actions; inventing a non-zero slippage would describe a risk the transaction does not carry. A price that cannot be fetched yields `POLICY_NOTIONAL_UNPRICEABLE`, never a zero notional.

The read of $D$ and $E$ here is advisory. It is repeated durably and atomically by the reservation in Section 6.6, which is the binding check.

## 6.5 Single-flight execution claims

A blockchain transaction is not idempotent: each signature consumes a nonce and moves funds. Idempotency must therefore be imposed before the signature exists, and it must be keyed on what the transaction *does*, not on how the request was phrased.

**DEFINITION 6.5 (CLAIM SUBJECT AND REQUEST DIGEST)**

The subject of an action $a$ by principal $u$ acting through credential $\gamma$ under mandate $M$ is the nine-tuple

$$\sigma(a) = (u,\ \gamma,\ h_M,\ h_I,\ c,\ \omega,\ \rho,\ q,\ x), \qquad h_\sigma = \mathrm{SHA256}\big(\mathrm{canon}(\sigma)\big)$$

where $h_I$ is the canonical intent digest (the preview digest $h_P$ for lend; keccak-256 of the deposit call-data for an Across bridge leg). The claim key for a caller-supplied idempotency key $\kappa$ is $K(u,\kappa) =$ `"ava.execution.v1:"` $\|\ u\ \|$ `":"` $\|\ \kappa$. Placing $u$ inside the key makes cross-principal collision impossible under a unique constraint on $K$ alone.

**DEFINITION 6.6 (CLAIM OPERATION)**

$\mathrm{Claim}(\sigma, \kappa, L, n_0)$ executes as one serializable database transaction:

$$\textbf{if } K \in \mathrm{dom}(\mathsf{Cl}) : \ \textbf{return} \begin{cases} \mathsf{duplicate}(\mathsf{Cl}[K]) & h_\sigma = \mathsf{Cl}[K].h \\ \mathsf{refuse}(\texttt{CONTENT_MISMATCH}) & \text{otherwise} \end{cases}$$

$$\mathsf{Cl}[K] \leftarrow (h_\sigma,\ \mathsf{authorized})$$

$$B \leftarrow \mathsf{Bud}[h_M] \ \text{(created with limit } L \text{ if absent; row-locked)}$$

$$\textbf{if } B.\mathrm{consumed} + B.\mathrm{reserved} + q > B.L : \ \textbf{return } \mathsf{refuse}(\texttt{BUDGET_EXHAUSTED})$$

$$n \leftarrow \max\big(\mathsf{Non}[c,\omega],\ n_0\big); \quad \mathsf{Non}[c,\omega] \leftarrow n+1$$

$$B.\mathrm{reserved} \leftarrow B.\mathrm{reserved} + q; \quad \mathsf{Cl}[K] \leftarrow (h_\sigma,\ \mathsf{reserved},\ n)$$

An existing budget row whose asset, scale or limit differs from the request is refused ( `BUDGET_MISMATCH` ): the cumulative limit is fixed at first reservation and cannot be raised by a later request.

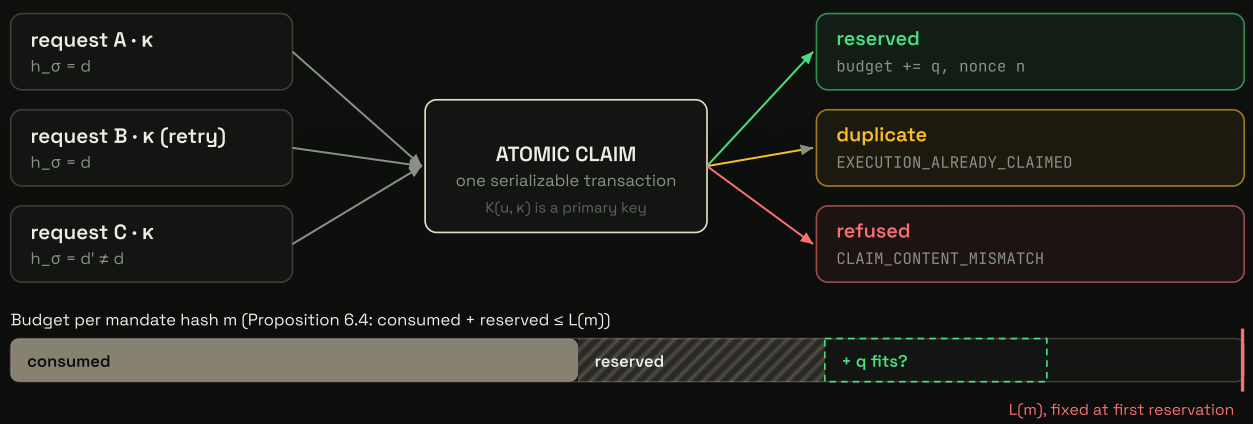



**FIGURE 7** Single-flight claims. One request per key reaches reserved; a retry with the same content is a duplicate; the same key with different content is refused. Budget can only be reserved within the limit fixed at first reservation.

**PROPOSITION 6.3 (AT-MOST-ONCE RESERVATION)**

For a fixed $(u, \kappa)$, at most one transaction ever moves $\mathsf{Cl}[K]$ from absent to reserved, and every later Claim with the same key returns either duplicate (same $h_\sigma$) or a refusal.

*Proof sketch.* The insert is conditional on $K$ being absent and $K$ is a primary key, so concurrent inserts serialize and exactly one succeeds. Every other caller reads the committed row and compares digests. ∎

**PROPOSITION 6.4 (BUDGET CONSERVATION, I2)**

If every transition that changes $\mathsf{Bud}[m]$ is either Claim or the consumption/release paths that move an amount from reserved to consumed or back to zero, then $\mathrm{consumed}(m) + \mathrm{reserved}(m) \le L(m)$ holds in every reachable state.

*Proof sketch.* Induction on transitions. Claim adds $q$ to reserved only after checking the sum with $q$ added against $L$ under a row lock. Consumption moves an amount from reserved to consumed and leaves the sum unchanged. Release decreases it. ∎

**The key is never invented.** If the caller supplies no $\kappa$, no claim is taken: the request runs unreserved. A generated key would make every retry look new and silently defeat the mechanism. The cost is that protection depends on the client; Section 15 lists the surfaces where this is the case today.

## 6.6 Durable policy reservations

The policy gate's daily and exposure reads (6.4) are advisory. The binding check is a reservation keyed by the claim key, executed under a Postgres advisory lock on $(\mathrm{mode}, u)$:

$$D_u(t) = \sum_{r \in \mathsf{PR}_u,\ t-\Delta \le t_r \le t} \nu_r, \qquad E_u = \max\Big(0,\ \sum_{r \in \mathsf{PR}_u} \Delta E_r\Big), \qquad \Delta = 24\,\mathrm{h}$$

The reservation for an action with notional $\nu$ and exposure change $\Delta E$ ($+\nu$ for supply, $-\nu$ for withdraw) succeeds iff

$$D_u(t) + \nu \le \nu_{\mathrm{day}} \quad \wedge \quad E_u + \Delta E \le E_{\max},$$

and is idempotent on its key. It is taken before the signer is built, so a broadcast that never resolves still counts against the day. A process-local version of this check previously returned the whole daily allowance after every deploy, and two replicas each granted it in full; the durable reservation is the fix.

## 6.7 Finality and settlement attestation

**Finality.** For chain $c$ and risk class $r \in \{\text{low, standard, high}\}$, a transaction included at depth $\delta$ is final iff the chain's finalized tag covers it or $\delta \ge F(c, r)$:

| CHAIN | LOW | STANDARD | HIGH | BLOCK TIME |
|---|---|---|---|---|
| Ethereum | 12 | 32 | 64 | 12 s |
| Base | 12 | 30 | 120 | 2 s |
| BNB Chain | 15 | 30 | 60 | 3 s |
| Avalanche | 1 | 3 | 6 | 2 s |

| CHAIN | LOW | STANDARD | HIGH | BLOCK TIME |
|---|---|---|---|---|
| Monad | 3 | 10 | 20 | 1 s |
| Unknown chain | 32 | 64 | 128 | 12 s |

The risk class is a step function of notional: low for $\nu \leq$ 1,000 USD, standard for $\nu \leq$ 100,000 USD, high above. A reorg is detected by a changed canonical block hash at the recorded height, which distinguishes a reorganization from a lagging provider.

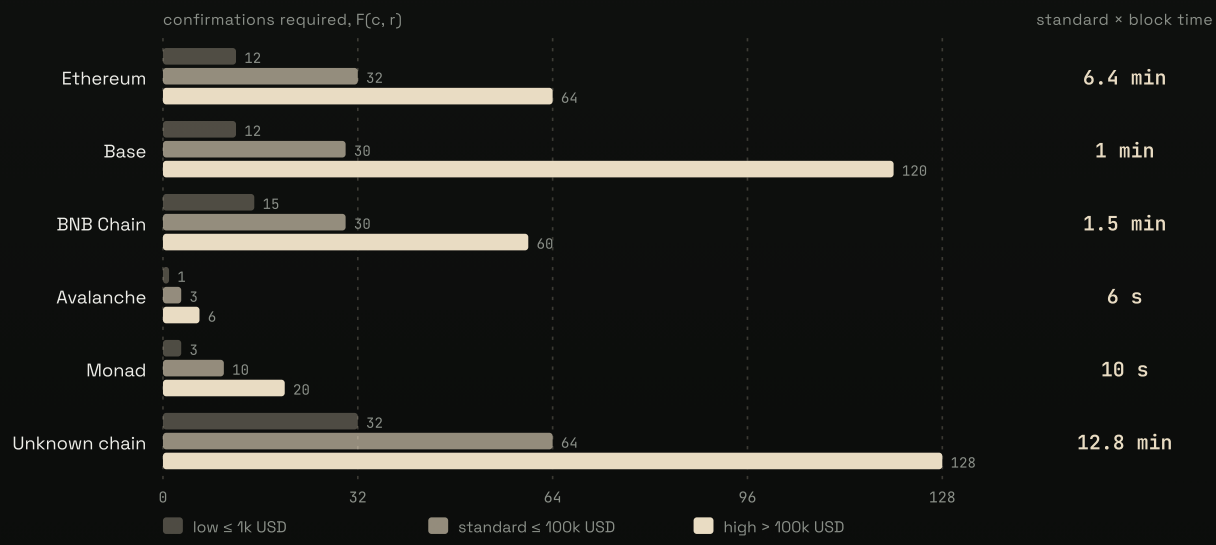



**FIGURE 8** The finality table drawn to scale, with the wall-clock wait it implies at the standard risk class.

**Attestation.** After inclusion, Ava re-fetches the transaction and its logs and compares a claimed effect $\varepsilon$ = (status, txFrom, txTo, token, amount, transferFrom, transferTo, market) with the observed one $\hat{\varepsilon}$. Let $F \subseteq$ fields be those derivable for the transaction type. Then

$$\text{proof}(R) = \begin{cases} \textsf{none} & \text{no tx hash} \\ \textsf{unconfirmed} & \text{tx hash, not yet compared} \\ \textsf{chain-confirmed} & \forall f \in F:\ \varepsilon_f = \hat{\varepsilon}_f \\ \textsf{chain-contradicted} & \exists f \in F:\ \varepsilon_f \neq \hat{\varepsilon}_f \end{cases}$$

The four values are not a total order. chain-contradicted is strictly worse than none: a receipt whose hash names a real transaction that does something else is a misdescription, not missing evidence.

**Settlement standing** is a second, independent verdict. It compares the authorised amount, asset and recipient (from the chain claim the receipt was issued under) with the settled ones (from the attestation's observation), yielding not-evaluated, unconfirmed, confirmed-unverified, mismatched or matched. A receipt can be chain-confirmed and settlement-mismatched; collapsing the two would hide exactly the failure that matters.

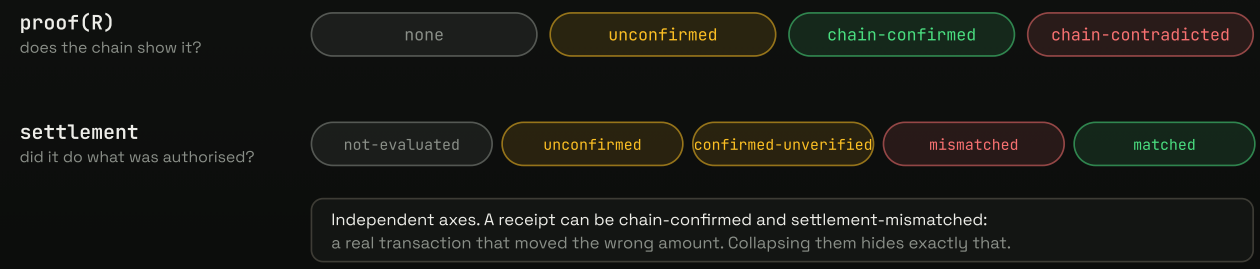



**FIGURE 9** Proof and settlement are separate axes, and each receipt carries both.

Venue-specific verification adds to this: the Aave executor re-reads the Pool's Supply or Withdraw event, the Compound executor reads `Comet.balanceOf` in the settling block, and the Morpho executor reads the ERC-4626 position pinned to the confirmed block so that a lagging endpoint cannot produce a false negative.

## 6.8 Receipts

A receipt $R$ is a structured record of an execution: identifiers, mandate hash, plan and quote hashes, the policy envelope and gate digest, the chain claim, the attestation and status. Let $\bar{R}$ be $R$ with its integrity block and signatures removed. Then

$$h(R) = \mathrm{SHA256}\big(\mathrm{JCS}(\bar{R})\big)$$

where JCS is the RFC 8785 canonicalization. Every receipt is written through one storage wrapper that assigns a chain scope $\zeta$ (the agent instance holding its mandate, else its user), a sequence number and a predecessor:

$$\mathrm{seq}_\zeta(R_{i+1}) = \mathrm{seq}_\zeta(R_i) + 1, \qquad \mathrm{prev}(R_{i+1}) = h(R_i), \qquad \mathrm{sig}(R_i) = \mathrm{Ed25519}_{sk}\big(\mathrm{hex}(h(R_i))\big)$$

The signature is present only when a receipt key is configured; otherwise the receipt records why it is unsigned. Applying chaining at the storage boundary rather than in each executor means a new execution path cannot forget it.

**Anchors.** Periodically, for every scope whose head moved, the scheduler publishes the heads $A = \{(\zeta, \mathrm{seq}, h)\}$ as UTF-8 JSON in the calldata of a self-send transaction on Base, signed by a Turnkey key, and records the anchor only after the transaction is mined. A receipt is anchored iff its sequence is at or below an anchored sequence for its scope.

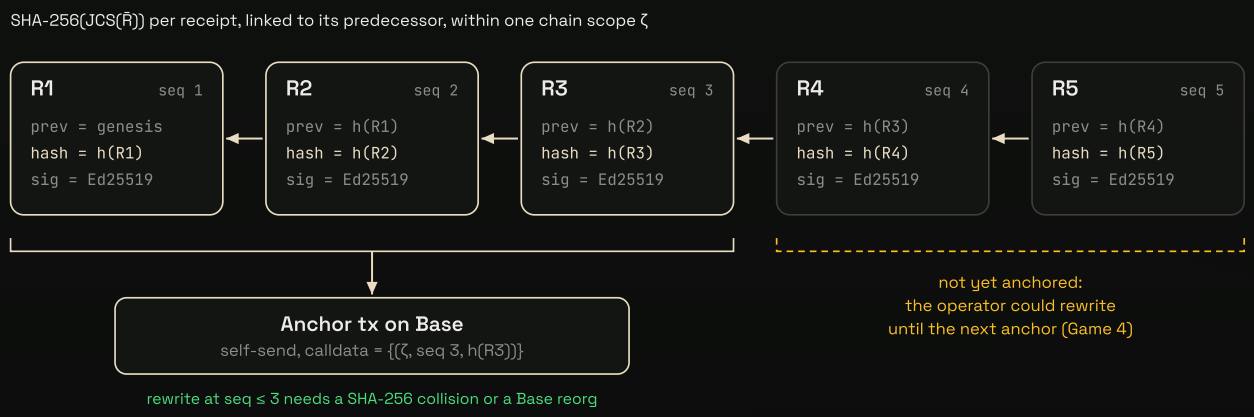



**FIGURE 10** The receipt chain. Each receipt commits to its predecessor; an anchor on Base fixes a head, so everything at or below it can no longer be rewritten undetectably.

## 6.9 Standing envelopes

For instructions the user will not be present to approve, the user signs bounds once.

**DEFINITION 6.7 (STANDING ENVELOPE)**

$$\mathcal{E} = (\mathrm{id},\ u,\ \mathrm{obj},\ x,\ d,\ \mathrm{trigger},\ O_{\mathcal{E}},\ V_{\mathcal{E}},\ C_{\mathcal{E}},\ \lambda,\ t_0,\ t_{\exp}), \qquad \lambda = (q_{\mathrm{move}},\ q_{\mathrm{cum}},\ g_{\min},\ m_{\mathrm{day}})$$

with a cron trigger, allowed operations, venues and chains, a per-move cap, a cumulative cap, a minimum gain in basis points and an optional daily move count. Its hash $h_{\mathcal{E}}$ is signed with EIP-712 by a key outside Ava (Section 9.3).

**Move predicate.** A proposed move $\mu = (\mathrm{op}, v, c, x', q, g)$ at time $t$, with durable consumption $Q_{\mathrm{cons}}$ and moves today $m$, is permitted iff

$$\begin{aligned}\mathrm{Perm}(\mathcal{E}, \mu) \iff\ & \neg\,\mathrm{revoked}(\mathcal{E}) \wedge t < t_{\exp} \wedge \mathrm{op} \in O_{\mathcal{E}} \wedge v \in V_{\mathcal{E}} \wedge c \in C_{\mathcal{E}} \wedge x' = x \\ & \wedge\ 0 < q \le q_{\mathrm{move}}\ \wedge\ Q_{\mathrm{cons}} + q \le q_{\mathrm{cum}}\ \wedge\ g \ge g_{\min}\ \wedge\ (m_{\mathrm{day}} = \bot \vee m < m_{\mathrm{day}})\end{aligned}$$

$Q_{\mathrm{cons}}$ and $m$ are read from durable rows, not from the request. After a broadcast the debit $Q_{\mathrm{cons}} \leftarrow Q_{\mathrm{cons}} + q$ is written; a failed debit is logged rather than raised, because a 5xx would invite a retry that passes Perm again and spends twice.

Perm by itself only bounds what may be done. Whether anything is done unattended depends on an executor bound to the scheduler, and none is bound in production (Section 15).

## 6.10 Multi-leg workflows

**Decomposition.** A natural-language request $m$ is decomposed deterministically into legs $\ell_1, \ldots, \ell_n$, each an action with an optional dependency set $\mathrm{dep}(\ell_i) \subseteq \{\ell_j : j < i\}$ and an amount that is either fixed or the amount delivered by a dependency. The legs form a DAG ordered by index. Clauses that cannot be read are returned as unsupported with reasons, never dropped.

**Plan identity.** With $L = (\mathrm{planLeg}(\ell_1), \ldots, \mathrm{planLeg}(\ell_n))$, total budget $B$ and asset $x$:

$$h_{\mathrm{plan}} = \mathrm{SHA256}\big(\,\mathrm{canon}(L,\ B,\ x,\ u)\big), \qquad \kappa_i = \texttt{wf_}\,h_{\mathrm{plan}}[0{:}16]\,\texttt{_}\,\mathrm{id}(\ell_i)$$

Including $u$ prevents a signed plan from being replayed by another user; including each leg's recipient prevents two transfers to different addresses from sharing a hash (an earlier projection omitted it).

**Execution.** The execute route re-decomposes $m$ rather than accepting legs from the request, verifies the user's EIP-712 signature over $h_{\text{plan}}$, admits the plan only if every leg's kind has a bound executor and an executable registry row, and then runs legs in index order. A leg runs only after every dependency reached the verified state, and a leg whose amount depends on a bridge receives the bridge's observed delivered amount $r_j$, not the requested amount:

$$q_i = \begin{cases} q_i^{\text{fixed}} & \text{if fixed} \\ r_j & \text{if } q_i \text{ is received from } \ell_j \end{cases}, \qquad r_j \le q_j \ (\text{fees})$$

The workflow succeeds iff every leg is verified. A failed leg makes every transitive dependent skipped, and the aggregate is never reported as success.

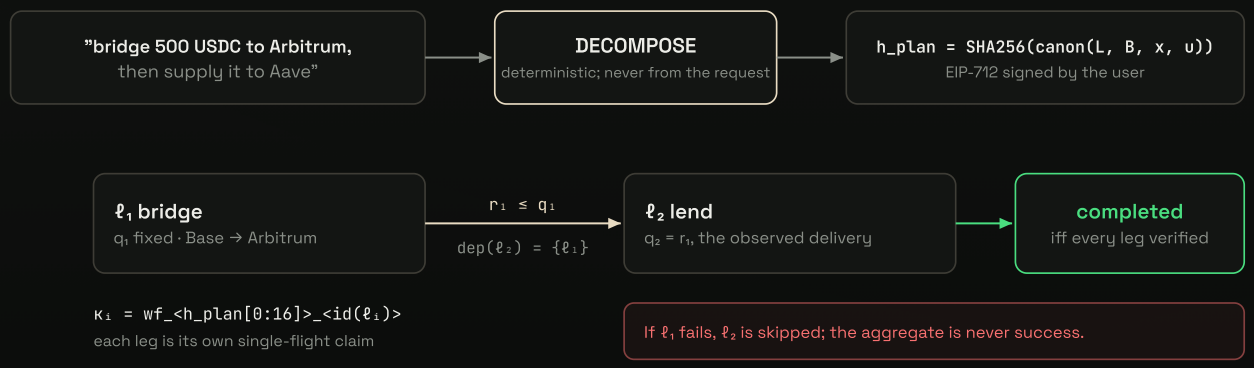



**FIGURE 11** A two-leg workflow. The plan is re-derived and signed; the lend leg receives the amount the bridge actually delivered, and a failed leg skips its dependents.

07

# Protocol algorithms

The pseudocode abstracts transport and storage. `REFUSE(code)` aborts the request with no state change except where noted; `ATOMIC` marks a serializable database transaction. Costs assume a constant number of indexed lookups per store access, $R$ for one RPC round trip and $K$ for one custodian round trip.

**ALGORITHM 1 · AUTHENTICATE**

```
AUTHENTICATE(req) → principal | anonymous
  tau ← bearer(req) or api_key(req)
  if tau present:
    if (u, tau) in Keys:                pi ← (u, api_key, spend)
    elif row ← Tok[SHA256(tau)] live:   pi ← (row.u, session, row.audience)
    else:                               REFUSE(UNAUTHORIZED)      # never downgrade
  claimed ← header x-ava-user-id
  if pi and claimed and claimed ≠ pi.u:   REFUSE(USER_ID_ASSERTION_MISMATCH)
  if not pi and enforced and not PUBLIC(method, path):   REFUSE(AUTH_REQUIRED)
  if LIVE_SPEND(method, path) and not CanSpend(pi):     REFUSE(SPEND_CREDENTIAL_REQUIRED)
  return pi or anonymous
```

*Cost:* $O(1)$ store reads, one hash. PUBLIC is a fixed allowlist of ten patterns, so the check is $O(1)$.

**ALGORITHM 2 · LIVE LEND (MORPHO SHAPE; AAVE ADDS STEP 7B)**

```
LEND(pi, mandateId, op, q_dec, h_hat, kappa) → filled | unconfirmed | REFUSE
 1  M ← Mandates[mandateId];  if M = nil or M.u ≠ pi.u: REFUSE(MANDATE_NOT_FOUND)
 2  if Gamma.allowedKinds excludes lend:                 REFUSE(MANDATE_KIND_NOT_ALLOWED)
 3  rho ← route(lend, M.c, venue);  if not Exec_api(rho): REFUSE(registry code)
 4  q ← toBase(q_dec, d(x))                 # exact; refuses non-representable amounts
 5  w ← active wallet of family(M.c) for pi.u
 6  pos0 ← READ_POSITION(w, rho)            # R; refuses on read failure
 7  P ← (v, chainId, market, w, w, op, x, q, M.id, h_M);  h_P ← SHA256(canon(P))
 8  MGATE(M, pi.u, lend, nu(q))             # persists refusal on failure
 9  dec ← PGATE(a, humanConfirmed := (h_hat = h_P))
10  if dec = reject:  REFUSE(POLICY_REJECTED)
11  if dec = confirm: REFUSE(POLICY_HUMAN_CONFIRMATION_REQUIRED, P, h_P)
12  if kappa and Cl wired:  r ← ATOMIC CLAIM(sigma(a), kappa, L(M), n0)
13      if r = duplicate:   REFUSE(EXECUTION_ALREADY_CLAIMED)
14  ATOMIC RESERVE_POLICY(pi.u, claimKey, nu, dE)
    # ----------- no signature can exist above this line -----------
15  signer ← TURNKEY_SIGNER(w, suborg(pi.u))
16  if balance(w, x) < q:   REFUSE(LEND_INSUFFICIENT_BALANCE)
17  if allowance(w, vault) < q: tx_a ← sign(approve(vault, q)); BROADCAST(tx_a); WAIT(tx_a)
18  tx ← sign(deposit or withdraw(q, w));  require tx.from = w and tx.to, data unchanged
19  BROADCAST(tx)
20  loop until RECEIPT(tx) or timeout         # R per poll
21  if timeout: APPEND_RECEIPT(status = submitted, proof = unconfirmed); return unconfirmed
22  att ← ATTEST(tx, claim(P))              # re-read tx and logs, compare fields
23  pos1 ← READ_POSITION(w, rho, minBlock = tx.block)
24  R ← APPEND_RECEIPT(status = filled, att, pos0, pos1, gate digest, h_M)
25  return filled(R, verifyUrl)
```

*Cost:* a constant number of store transactions and RPC reads, plus 1 to 2 custodian signatures and $P$ polling rounds, where $P \approx t_{\text{confirm}}/\text{poll interval}$. Wall-clock latency is dominated by inclusion time (Section 13).

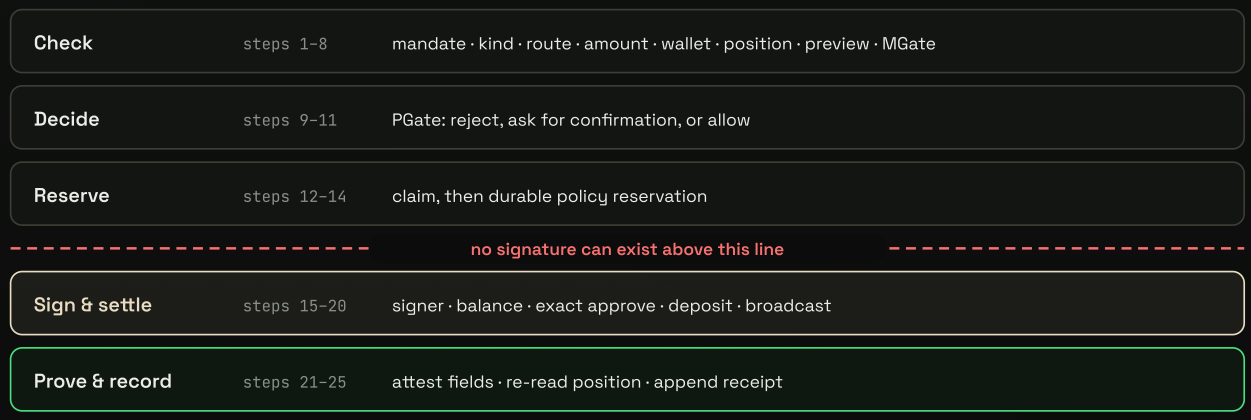



**FIGURE 12** Algorithm 2 in five bands. Steps 1 to 14 can only refuse; the signer first exists at step 15.

**ALGORITHM 3 · WORKFLOW EXECUTION**

```
EXECUTE_WORKFLOW(pi, m, sig, previewHashes) → completed | failed | REFUSE
  require CanSpend(pi)
  (legs, B, x) ← DECOMPOSE(m)                 # deterministic; never from request
  if legs empty:            REFUSE(WORKFLOW_NO_EXECUTABLE_ACTIONS)
  if sum fixed(legs) > B:   REFUSE(WORKFLOW_OVER_BUDGET)
  h_plan ← SHA256(canon(planLegs(legs), B, x, pi.u))
  s ← ECRECOVER(EIP712("Ava Workflow Plan", h_plan, legs), sig)
  if s ≠ sig.signer or s in AvaControlled:  REFUSE(PLAN_SIGNATURE_*)
  for l in legs: if kind(l) unbound or not Exec_api(route(l)): collect refusal
  if refusals: REFUSE(409, refusals)          # all-or-nothing admission
  n0 ← PENDING_NONCE(wallet)
  for i in 1..n in index order:
    if some j in dep(l_i) not verified: mark l_i skipped; continue
    q_i ← fixed(l_i) or received(l_j)
    CLAIM(sigma(l_i), kappa_i = wf_<h_plan[0:16]>_<id_i>, L = B, n0)
    outcome_i ← EXECUTOR[kind(l_i)](l_i, q_i, previewHashes[id_i])
  return completed iff every outcome_i = verified
```

*Cost:* $O(n + |E|)$ over the leg DAG for scheduling, $n$ claims, and $\sum_i \text{cost}(\text{executor}_i)$. Because legs run sequentially, latency is the sum of per-leg latencies, not their maximum.

**ALGORITHM 4 · STANDING TICK (AS DEPLOYED)**

```
STANDING_TICK(now)                    # every 60 s when the scheduler is enabled
  for E in Envelopes.due(now):
    if REVOKED(E):          record(E, not run, STANDING_REVOKED);  continue
    if not E.tickEnabled:   record(E, not run, "not opted in");    continue
    if execute unbound:     record(E, not run, "driver disabled"); continue   # production
    ...                               # rotation path, unreachable in production
```

*Cost:* $O(|\text{due}|)$ per tick.

**ALGORITHM 5 · APPEND AND ANCHOR**

```
APPEND_RECEIPT(R)
  zeta ← agentInstance(M(R)) or user(R)
  ATOMIC:
    head ← Chain[zeta];  R.seq ← head.seq + 1;  R.prev ← head.hash
    R.hash ← SHA256(JCS(strip(R)));  R.sig ← Ed25519(sk, hex(R.hash)) if sk else reason
    Chain[zeta] ← (R.seq, R.hash);   Ledger.put(R)

ANCHOR_TICK()
  H ← { (zeta, seq, hash) : head moved since last anchor }
  if H empty: return
  tx ← TURNKEY_SIGN(selfsend(data = UTF8(JSON(H))));  BROADCAST(tx);  WAIT(tx)
  Anchors.put(tx.hash, H, block)             # only after inclusion
```

Append is $O(|R|)$ for canonicalization and one Ed25519 signature. An anchor costs $O(s)$ bytes of calldata for $s$ moved scopes and one transaction regardless of how many receipts each scope gained.

**ALGORITHM 6 · VERIFY**

```
VERIFY(ref) → verdicts
  R ← ref is 64-hex ? Ledger.byHash(ref) : Ledger.byExecution(ref)   # by-id needs owner or opt-in
  integrity  ← (SHA256(JCS(strip(R))) = R.hash) and (R.prev = hash(pred(R))) and SigOK(R)
  mandate    ← SHA256(canon(T(M(R)))) = R.mandateHash   # match | terms-amended | mismatch | missing
  proof      ← proof(R);  settlement ← settlement(R)
  anchor     ← exists A: A.zeta = R.zeta and A.seq ⩾ R.seq
  return verdicts + recipes for independent recomputation
```

*Cost:* $O(|R|)$ hashing, one signature verification, $O(1)$ lookups. A third party can repeat every step except the lookups, which it replaces with the disclosed receipt and public chain reads.

08

# State transitions and execution lifecycle

Every money path is meant to move a claim through one shared state machine, defined in `packages/execution-contract`. The machine exists because the dangerous states are the ambiguous ones: ”did we sign?” and ”is it final?” are different questions from ”did the request succeed?”, and a boolean filled flag answers neither.

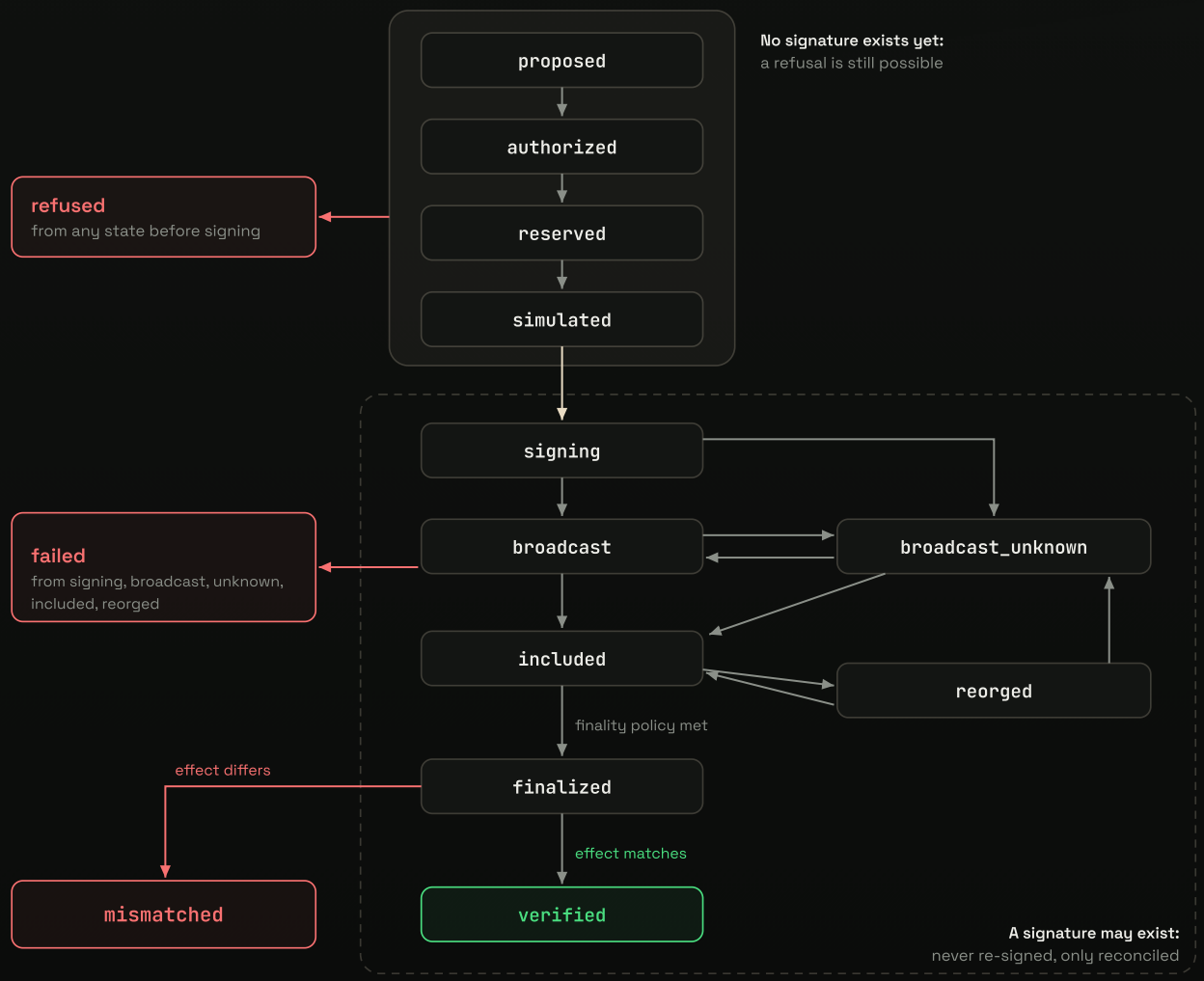



**FIGURE 13** Execution state machine: 14 states, 4 terminal. Once a signature may exist, an execution is never signed again.

The shaded column holds the four states before any signature. Only these can reach refused, so a refusal is never reported for an action that may already be on chain (Proposition 11.3). Everything inside the dashed boundary is a state where a signature may exist. No arrow leaves that boundary back to signing, so the only way out of broadcast_unknown or reorged is reconciliation to a known outcome (Proposition 11.2). verified is reachable only through finalized (Proposition 11.4).

Formally, with the state set $\Sigma$ and transition relation $\rightarrow$ as drawn, the safety-relevant subsets are

$$\mathrm{SME} = \{\text{signing}, \text{broadcast}, \text{broadcast_unknown}, \text{included}, \text{finalized}, \text{reorged}, \text{verified}, \text{mismatched}\}$$
$$\mathrm{Term} = \{\text{verified}, \text{mismatched}, \text{refused}, \text{failed}\}$$

and the structural property is that the preimage of signing under $\rightarrow$ is {simulated}, which is disjoint from SME. The transition function is enforced at runtime: `assertTransition` throws on any edge not in the table.

**Lifecycle in the implementation.** The claim store writes authorized and then reserved in one transaction (6.5). The live Morpho and Aave executors do not drive the remaining states through this machine. They record their own outcome as filled, on a successful inclusion receipt, or unconfirmed, on a confirmation timeout. That is discrepancy D2 of Section 15.

09

# Cryptographic foundations

Ava uses no novel cryptography. Its security rests on the standard properties of five primitives, each used for one job. The table names the job and the property the job needs.

| PRIMITIVE | USED FOR | PROPERTY RELIED ON |
|---|---|---|
| SHA-256 | token digests, terms hash $h_M$, preview digest $h_P$, request digest $h_\sigma$, plan hash, receipt hash | preimage resistance (tokens); second-preimage and collision resistance (bindings) |
| Keccak-256 | EIP-712 digests, Across deposit calldata digest | collision resistance |
| secp256k1 ECDSA with public-key recovery | user signatures over mandates, plans, standing envelopes; transaction signatures by Turnkey | existential unforgeability under chosen-message attack (EUF-CMA) |
| Ed25519 [RFC8032] | optional receipt signatures | EUF-CMA |
| CSPRNG, 32 bytes | session tokens (`ava_st_`, `ava_dt_`), device codes (`ava_dc_`) | unpredictability |

Custodian authentication (Ava to Turnkey) uses request stamping with a P-256 API key; that channel is outside Ava's model and we treat it as an authenticated channel between O and K.

## 9.1 Canonicalization

Every binding in Ava is a hash of structured data, so it is only as strong as the injectivity of the serialization. Let $\mathrm{enc} : \mathrm{Values} \to \{0,1\}^*$ be the canonical encoder. A binding $h = H(\mathrm{enc}(v))$ is sound when enc is injective on the domain of $v$; then a collision in $h$ implies a collision in $H$.

The implementation contains three encoders: a full RFC 8785 JCS implementation for receipts, and two sorted-key JSON encoders (one in the mandates package, one in the execution-contract package) for terms, previews, plans and claim subjects. All three sort object keys by code unit and emit ECMAScript number serialization, so they agree on the value domains Ava actually hashes (strings, integers below $2^{53}$, booleans, nested objects and arrays). They differ at the edges: handling of undefined members and of non-finite or very large numbers. Amounts are always carried as decimal or base-unit strings precisely so that no binding depends on floating-point formatting. We list the unification of encoders as a hardening item (Section 15).

> **LEMMA 9.1 (BINDING)**
>
> Let enc be injective on domain $D$ and $H$ collision-resistant with advantage $\varepsilon_H$. Then any adversary that outputs $v \neq v' \in D$ with $H(\mathrm{enc}(v)) = H(\mathrm{enc}(v'))$ yields a collision for $H$, so its success probability is at most $\varepsilon_H$.

Applied to $h_P$ this is Proposition 6.2; applied to $h_\sigma$ it is the content check in the claim (6.5); applied to $h_{\mathrm{plan}}$ it prevents a signature over one plan from authorising another.

## 9.2 Bearer tokens

A session token is $\tau = \text{prefix} \,\|\, \text{b64url}(r), r \leftarrow \{0,1\}^{256}$. The server stores $H(\tau)$ only. An attacker holding a database dump must find a preimage of a 256-bit-entropy input, which costs about $2^{256}$ hash evaluations; no salt is needed because the input already has full entropy. The prefix (`ava_st_` for spend, `ava_dt_` for dashboard) is a recognition aid only. Audience is a stored attribute of the digest's row, so relabelling a token's prefix grants nothing.

## 9.3 EIP-712 authorisation objects

All three user-signed objects share the domain type `EIP712Domain(string name, string version, uint256 chainId)` and differ in name and primary type:

| OBJECT | DOMAIN NAME | PRIMARY TYPE | MESSAGE |
|---|---|---|---|
| Mandate | Ava Mandate | `AvaMandate(string mandateId, string terms)` | terms = canon(T(M)) |
| Workflow plan | Ava Workflow Plan | `AvaWorkflowPlan(string planHash, string legs)` | legs = canon(planLegs) |
| Standing envelope | Ava Standing Authorization | `AvaStandingAuthorization(string envelopeId, string bounds)` | bounds = canon(envelope) |

The signed digest is the standard

$$d = \text{keccak}\big(\texttt{0x1901} \,\|\, \text{hashStruct}(\text{domain}) \,\|\, \text{hashStruct}(\text{message})\big)$$

Verification recovers the signer $s = \text{ecrecover}(d, \sigma)$, requires $s$ to equal the recorded signer, and then refuses if $s \in \text{AvaControlled}$, the configured set of addresses whose keys Ava holds. That last test is what makes the signature mean *the user*: without it, Ava could authorise a plan with a key it controls and the check would pass.

**Domain separation.** Because the domain names differ, a signature over a plan has a different $d$ than a signature over a standing envelope with the same strings, so cross-object replay requires a Keccak collision. Including chainId in the domain prevents a signature produced for one chain's domain from verifying under another.

**A limitation.** The domain has no verifyingContract and no nonce. Replay within the same object type is prevented by content ($h_{\text{plan}}$ includes $u$; the envelope includes its id and is stored once), not by the signature scheme. A revoked standing envelope is unusable because revocation is checked on every move, not because its signature stops verifying.

## 9.4 Receipt signatures and anchors

With key pair $(sk, pk)$ identified by issuerKeyId, each receipt carries $\text{Ed25519}_{sk}(\text{hex}(h(R)))$. Signing the hex string rather than the raw digest is a deliberate interoperability choice: any verifier can reproduce the exact bytes from the published hash.

A hash chain gives tamper evidence only relative to a previously observed head [CW09]. If the log owner alone holds every head, it can rebuild the chain from genesis with altered content and every link still verifies. Anchoring fixes this: once $(\zeta, s, h)$ is in a mined Base transaction, rewriting any receipt with sequence $\leq s$ in scope $\zeta$ requires either a Keccak or SHA-256 collision or a reorganization of Base past the anchoring block.

**PROPOSITION 9.2 (ANCHORED IMMUTABILITY)**

Let $A = (\zeta, s, h_s)$ be anchored at a final Base block. Any ledger state in which the receipt at $(\zeta, s)$ hashes to $h' \neq h_s$, or in which some receipt at $(\zeta, i \leq s)$ differs from the anchored history while the chain from $i$ to $s$ still verifies, implies a SHA-256 collision.

*Proof sketch.* The chain from $i$ to $s$ is a sequence of hashes each committing to its predecessor. Changing $R_i$ changes $h(R_i)$, so verification at $i+1$ requires $\mathrm{prev}(R_{i+1}) = h(R_i')$, which changes $R_{i+1}$ and therefore $h(R_{i+1})$, and so on up to $s$. Reaching the anchored $h_s$ with a changed input is a collision. ∎

Anchoring does not prove completeness. It says nothing about receipts written after the latest anchor, and it cannot show that every execution produced a receipt. An operator that signs a transaction and never writes the receipt is not detected by the log; it is detected only by comparing the wallet's on-chain history with the ledger (Section 10.6).

10

# Security model and threat analysis

We define four security games against the adversary of Section 4.6, bound the adversary's advantage where the code supports a bound, and say plainly where it does not. Advantages are over the coins of the adversary and of the hash functions modelled as collision-resistant with advantage $\varepsilon_H$, and signatures as EUF-CMA with advantage $\varepsilon_\Sigma$.

| | | | | |
|---|---|---|---|---|
| UA | **Unauthorised spend** | $\mathrm{Adv} \leq \frac{Q_\tau}{2^{256}} + \varepsilon_\Sigma + \varepsilon_H$ | NEGLIGIBLE | assumes an honest operator |
| DUP | **Duplicate execution** | $\mathrm{Adv} = 0$ | ZERO WHEN KEYED | unkeyed and MCP lend: bounded by limits only |
| SUB | **Content substitution** | $\mathrm{Adv} \leq \varepsilon_H$ | NEGLIGIBLE | Proposition 6.2 |
| PRES | **Forged presence** | $\mathrm{Adv} = 1$ | OPEN | the server binds content, not presence |
| REW | **Log rewriting** | $\mathrm{Adv} \leq \varepsilon_H + \Pr[\mathrm{reorg}]$ | ANCHORED ONLY | unanchored history: Adv = 1 |

## 10.1 Security games

**Game 1: Unauthorised spend (UA).** The challenger runs Ava with honest 0 and K. $\mathcal{A}$ receives, for a target user $u$, any number of dashboard tokens, every agent channel, and oracle access to every API route under any credential it holds. $\mathcal{A}$ wins if Ava produces a signature on behalf of $u$'s wallet for an action $a$ such that either (a) $\mathcal{A}$ held no spend-audience credential for $u$, or (b) $a$ has no mandate, signed plan or signed envelope of $u$ covering it.

$$\mathrm{Adv}^{\mathrm{UA}}_{\mathcal{A}} \leq \frac{Q_\tau}{2^{256}} + \varepsilon_\Sigma + \varepsilon_H$$

where $Q_\tau$ is the number of token guesses. *Argument.* Every live-spend route and spend tool requires CanSpend (Algorithm 1), which requires a spend token (a 256-bit preimage) or an API key; a lend requires a mandate owned by $u$; a workflow requires an EIP-712 signature by a non-Ava key over the recomputed plan ($\varepsilon_\Sigma$, $\varepsilon_H$). The bound is conditional: it assumes O is honest and correctly configured, because O can request signatures from K directly.

**Game 2: Duplicate execution (DUP).** $\mathcal{A}$ holds a spend credential and sends any schedule of concurrent and repeated requests carrying one idempotency key $\kappa$. It wins if two signatures are produced for claim key $K(u, \kappa)$.

$$\mathrm{Adv}^{\mathrm{DUP}}_{\mathcal{A}} = 0 \quad \text{when } \kappa \text{ is supplied and CI is wired (serializable insert on a primary key)}.$$

When $\kappa$ is absent, or the transport does not wire CI (the MCP lend tool today), the game is not defended by claims and $\mathcal{A}$ wins trivially. The duplicates are then bounded only by the durable policy reservation and the mandate's capital:

$$N_{\mathrm{dup}} \;\le\; \min\left(\left\lfloor \frac{\nu_{\mathrm{day}} - D_u(t)}{\nu(a)} \right\rfloor, \left\lfloor \frac{E_{\max} - E_u}{\nu(a)} \right\rfloor\right) \qquad \text{(supply actions)}$$

With mainnet defaults and an empty day, a 500 USD supply can be duplicated at most 20 times in 24 hours. Because each duplicate deposits into the user's own position (10.4), the harm is unwanted exposure and gas, not loss of principal.

**Game 3: Content substitution (SUB).** $\mathcal{A}$ observes an approved preview $(P, h_P)$ and wins if Ava signs an action $a' \neq a(P)$ under the human confirmation $h_P$.

$$\mathrm{Adv}^{\mathrm{SUB}}_{\mathcal{A}} \;\le\; \varepsilon_H \qquad \text{(Proposition 6.2 and Lemma 9.1)}.$$

**Game 3′: Forged presence (PRES).** $\mathcal{A}$ wins if a mainnet signature is produced with no human having seen $P$. Here

$$\mathrm{Adv}^{\mathrm{PRES}}_{\mathcal{A}} = 1,$$

because the refusal returns $h_P$ to the agent, which may resend it. We state this without softening: the server binds content, not presence. Presence is a client obligation. The residual protection is that the agent can only confirm the exact action the mandate already authorised, under the policy limits, into the user's own position.

**Game 4: Log rewriting (REW).** Here $\mathcal{A}$ is the operator. It wins if, after an anchor $A = (\zeta, s, h_s)$ is final on Base, it presents a ledger that verifies and differs from the anchored history at some sequence $\le s$.

$$\mathrm{Adv}^{\mathrm{REW}}_{\mathcal{A}} \;\le\; \varepsilon_H + \Pr[\text{Base reorganizes past the anchoring block}]$$

by Proposition 9.2. For sequences above the latest anchor, Adv = 1: the operator can rewrite unanchored history.

## 10.2 Threat catalogue

| THREAT | PREREQUISITE | IMPACT | MITIGATION IN CODE | RESIDUAL RISK |
|---|---|---|---|---|
| Prompt injection steering the agent | Agent reads hostile content | Agent requests a harmful action | Agent decides nothing; mandate, registry, policy and preview gates apply | Agent can still request any action inside the mandate and limits (PRES) |
| Spend token theft | Read of the agent's MCP config file | Spends within mandates and limits | Token stored 0600 by the CLI; digest-only storage; destinations restricted (10.4) | No token expiry or rotation API is exposed today |

| THREAT | PREREQUISITE | IMPACT | MITIGATION IN CODE | RESIDUAL RISK |
|---|---|---|---|---|
| Dashboard token escalation | Phished device link | View access | Audience stored server-side; spend routes and tools refuse dashboard tokens before parsing | None for spend; read exposure of the session |
| Identity spoofing via userId | Any request | Acting as another user | Claims compared with the principal; mismatch refused, never overridden | None found |
| Replay of a request | Captured body | Re-execution | Claim content check (6.5); single-use device codes | Unkeyed requests are not protected (DUP) |
| Concurrent duplicates | Two in-flight requests | Double spend | Serializable claim, nonce lease, durable reservation | MCP lend path has no claim (Section 15) |
| Malicious or lagging RPC | Control of an endpoint | False balance, false receipt | Block-pinned reads, receipt re-read and field comparison, multiple Base URLs | A single dishonest endpoint can still lie about a receipt on chains without fallbacks |
| Price manipulation | Control of the price feed | Limits evaluated on a wrong notional | Unpriceable is a refusal, never zero | One public source, no median; see 10.7 |
| Reorganization | Chain reorg | Reported fill disappears | Finality table, reorg detection by block hash (state machine) | The Morpho and Aave paths report a fill on inclusion with a successful status, not on the finality table; only the Compound path consults it (Section 15) |
| Sybil sessions | None (session creation is open) | Resource exhaustion, Turnkey sub-org quota | Per-user sub-orgs isolate wallet quotas; `WALLET_PROVIDER_LIMIT_REACHED` surfaces the ceiling | No rate limit on session creation in the API |
| Operator kill or pause abuse | Network access | Halting all users | Operator routes disabled unless `AVA_OPERATOR_TOKEN` is set | Token is a single shared secret |
| Receipt enumeration | Guessing execution ids | Reading others' positions | By-id verify is owner-only or explicit opt-in; by-hash mode reveals verdicts only | None found |
| Self-authorisation by Ava | Operator bug | Ava signs a plan or envelope for the user | AvaControlled exclusion; no process mints its own agent credential | Depends on `AVA_CONTROLLED_SIGNERS` being complete |
| Operator compromise | Access to O's credentials | Arbitrary signing from user wallets | None inside Ava's model | Detectable only after the fact (10.6) |
| Custodian compromise | Breach of K | Arbitrary signing | Out of scope | Total |

## 10.3 The presence gap

The preview binding answers "is this what the human approved?" and cannot answer "did a human approve?". Two designs close the gap. (i) Require $\hat{h}$ to arrive as a signature by a user-held key over $h_P$, verified like the plan signature; then PRES reduces to EUF-CMA. (ii) Deliver $P$ to the human out of band (a push to a device paired at session creation) and accept confirmation only on that channel. Option (i) reuses existing machinery and is the one we propose (Section 16).

## 10.4 Destination restriction

**PROPOSITION 10.1 (NO EXFILTRATION THROUGH BOUND EXECUTORS)**

On every executor bound in production, the recipient of funds is either the signing wallet $\omega$ or the address that signed the user's authorisation.

*Argument from the code.* Lend executors set receiver = owner = $\omega$ for ERC-4626, Aave and Comet. Swap venues request recipient = $\omega$ and the router target must be in the registry row's allowlist. The Across bridge leg sets the recipient to the plan's signer and its source executor refuses a FundsDeposited event whose recipient differs. The transfer kind, which could name an arbitrary recipient, is not bound. ∎

This turns the threat of a stolen spend token from theft into misallocation: an attacker can move a user's capital between permitted venues within limits, but cannot send it to itself through Ava. The property is not cryptographic. It holds because of what is bound, and binding a transfer executor would void it unless the recipient were also bound into the user's signature.

## 10.5 RPC trust

Every confirmation is a read from an RPC endpoint. Let $p$ be the probability that an endpoint returns a fabricated receipt. With $k$ independent endpoints queried and agreement required, the probability of accepting a fabrication falls to $p^k$. The implementation tries Base endpoints in order for availability; it does not require agreement. Agreement across endpoints is a proposed hardening, and a light client would remove the assumption.

## 10.6 Detecting operator misbehaviour

If O is malicious, Ava's guarantees reduce to what an outside party can audit. Two audits are possible today. **Anchored consistency:** any receipt at or below an anchor cannot be altered undetectably (Proposition 9.2). **Completeness by reconciliation:** an auditor lists every transaction sent from a user's wallet on chain and checks that each has a receipt; a signed transaction without a receipt is a provable omission. Neither prevents misbehaviour; both make it attributable.

## 10.7 Oracle sensitivity

Policy compares USD notionals computed from one price. If the reported price is $\hat{p} = (1-\eta)p$, an order of true notional $\nu$ is evaluated as $(1-\eta)\nu$. A daily limit $\nu_{\text{day}}$ then admits true notional up to $\nu_{\text{day}}/(1-\eta)$. A 50% underreport doubles the effective daily limit. For stablecoin capital, which is the only capital on the live lend paths, $\eta$ is small in practice; for volatile assets a median of independent sources, or a limit expressed in base units, would remove the dependence.

11

# Correctness and theoretical properties

We state each property, the assumptions it needs, the mechanism responsible and whether the implementation meets it. "Holds" means we derived it from the code paths cited; it does not mean it was mechanically verified. Ava has roughly a thousand unit and integration tests and scripted gate checks, but no formal verification.

## 11.1 Safety: no signature without the gates

**PROPOSITION 11.1 (GATE ORDERING, I1)**

On every live executor path, the custodian signer is constructed after, and only after, the conjunction

$$\mathcal{G}(a) \;=\; \mathrm{CanSpend}(\pi) \;\wedge\; \mathrm{Exec}_s(\rho) \;\wedge\; \mathrm{MGate}(M) \;\wedge\; \mathrm{PGate}(a) = \textsf{allow} \;\wedge\; \mathrm{Claim} \neq \textsf{dup} \;\wedge\; \mathrm{Reserve} = \textsf{ok}$$

has been evaluated to true, with the Claim conjunct vacuous when no key or store is present.

*Proof sketch.* In the Morpho and Aave executors the signer is built inside the step that follows the reservation (comment: "Nothing above this line can produce a signature"); every earlier step either returns or throws a typed refusal. In the Compound and Suilend dispatchers the registry gate runs first and the signer exists only after the live-execution gate. Workflow legs run only after admission and per-leg claims. The structure is a sequence of guarded early exits followed by one signing call, so the signing call is reachable only along the path where every guard passed. ∎

*Assumptions.* O executes its own code; K signs only on O's request. **Status: holds.**

## 11.2 State-machine safety

The shared state machine (Section 8) is enforced by an assert on every transition. Two of its properties are structural.

**PROPOSITION 11.2 (NO RE-SIGNING FROM AMBIGUITY)**

Let SME = {signing, broadcast, broadcast_unknown, included, finalized, reorged, verified, mismatched} be the states in which a signature may exist. No transition leads from a state in SME to signing.

*Proof.* By inspection of the transition relation: signing is reachable only from simulated, and simulated ∉ SME. ∎

**PROPOSITION 11.3 (REFUSAL PRECEDES SIGNATURE)**

refused is reachable only from {proposed, authorized, reserved, simulated}, none of which is in SME. A refusal can therefore never be reported for an action whose signature may exist.

**PROPOSITION 11.4 (FINALITY IS MANDATORY FOR VERIFIED)**

Every path to verified passes through finalized, and every path to finalized passes through included.

These hold for the claim-store lifecycle. The Morpho and Aave lend receipts are written by their executors directly and use inclusion, not this machine's finalized state, as the threshold for filled (Section 15).

## 11.3 Budget conservation

Proposition 6.4 gives $\mathrm{consumed}(m) + \mathrm{reserved}(m) \le L(m)$ for every mandate hash $m$ whose reservations go through the claim store. The policy reservation gives, for every user $u$ and every time $t$,

$$D_u(t) \le \nu_{\mathrm{day}} \quad \text{and} \quad E_u \le E_{\max}$$

under the advisory lock, with the caveat of 10.7 that $\nu$ is computed from a price. **Status:** holds for paths that reserve; the policy reservation is taken on the Morpho, Aave and MCP lend paths.

## 11.4 Replay resistance

| OBJECT | REPLAY PREVENTED BY | HOLDS |
|---|---|---|
| Execution under key $\kappa$ | Claim content check: same $\kappa$ with a different $h_\sigma$ is CONTENT_MISMATCH; same $h_\sigma$ is duplicate | Yes, when $\kappa$ is supplied |
| Device code | Single consume; 10-minute TTL | Yes |
| Workflow signature | $h_{\mathrm{plan}}$ includes $u$; legs re-derived from the message; per-leg keys derive from $h_{\mathrm{plan}}$ | Yes across users; re-submitting the same signed plan re-uses the same leg keys and returns duplicates |
| Standing signature | Stored once per ($u$, envelope id); revocation checked per move | Yes |
| Mandate signature | Bound to mandate id and terms | Yes |
| HTTP Idempotency-Key on `/v1/execute` | Process-local map | Only within one process lifetime |

## 11.5 Integrity and authenticity of the record

**Integrity (I5):** holds within a scope, given that every receipt is written through the chaining wrapper, which is the only ledger instance the app constructs. **Authenticity:** holds when a receipt key is configured; otherwise receipts declare themselves unsigned and are integrity-protected only. **Immutability:** holds for anchored sequences (Proposition 9.2), not for unanchored ones.

## 11.6 Determinism

Given identical stores, request and environment $\Omega$, $\delta$ is deterministic except for generated identifiers (execution ids, token bytes). All bindings ($h_M, h_P, h_\sigma, h_{\mathrm{plan}}, h(R)$) are deterministic functions of their inputs, which is what allows a third party to recompute them. Decomposition and mandate parsing are deterministic; the copilot's optional language-model fallback is the only nondeterministic parser, and its output is an intent that then meets the same deterministic gates.

### 11.7 Liveness

Ava makes no strong liveness claim. A request terminates with an output or a timeout given that Postgres, the custodian and at least one RPC endpoint respond. A broadcast that does not confirm within the executor's timeout ends as unconfirmed rather than blocking; the receipt keeps the hash so a later read can resolve it. The single-machine deployment has no failover, so availability is that of one Fly.io machine and its database.

### 11.8 Soundness and completeness of reporting

**Soundness** (a reported fill corresponds to a matching on-chain effect): holds on the lend paths, because filled requires a successful receipt and a field-by-field attestation, and the MCP result type allows a transaction hash only in the filled arm. **Completeness** (every on-chain effect has a receipt): not guaranteed. A crash between broadcast and receipt write loses the receipt. The claim store's broadcast and broadcast_unknown states exist to make such gaps reconcilable, but a reconciler that scans them is not implemented.

12

# Economic analysis

Ava has no token, no fees charged in protocol, no staking and no rewards, so there is no incentive mechanism to analyse for equilibria. The economics that do exist are operating costs, the cost a rotation must beat to be worth doing, and the cost of attacking the service. All three are analytical; no measured cost data is reported here.

### 12.1 Cost of one execution

For a lend action on an EVM chain with base fee $b$, priority fee $t$ and gas usage $g_{\text{approve}}, g_{\text{op}}$:

$$C_{\text{exec}} = (b + t)\left(\mathbf{1}[\text{allowance} < q] \cdot g_{\text{approve}} + g_{\text{op}}\right) + C_K + C_{\text{RPC}}$$

where $C_K$ is the custodian's per-signature charge and $C_{\text{RPC}}$ the metered RPC cost. The approve term is present on almost every supply, because Ava approves the exact amount rather than an unlimited allowance and the deposit consumes it; this is the safer choice and the more expensive one. The trade-off is explicit: exact approvals cost one extra transaction per supply and remove the standing-allowance risk if a vault is compromised.

### 12.2 Amortized anchoring

An anchor publishes $s$ moved scope heads in one transaction with calldata length $\ell(s) = \Theta(s)$. With $n$ receipts written per anchor interval across those scopes, the per-receipt anchoring cost is

$$c_{\text{anchor}} = \frac{(b + t)\left(21000 + 16\,\ell(s)\right)}{n}$$

using the EVM intrinsic cost and the upper bound of 16 gas per non-zero calldata byte. Because one anchor covers every receipt at or below each published sequence, increasing the interval lowers $c_{\text{anchor}}$ in proportion to $n$ while lengthening the window in which unanchored receipts can be rewritten (Game 4). The interval is therefore a direct price on the rewrite window.

### 12.3 When a rotation is worth doing

A standing envelope carries a minimum gain $g_{\min}$ in basis points. Consider moving amount $q$ from a venue paying annual rate $r_0$ to one paying $r_1$ for a holding period $\tau$ years, with total move cost $C$ (withdraw plus supply plus any bridge fee). The move is profitable iff

$$(r_1 - r_0)\, q\, \tau \;>\; C \quad\Longleftrightarrow\quad g \;=\; 10^4\,(r_1 - r_0) \;>\; \frac{10^4\, C}{q\, \tau}.$$

The implemented check $g \geq g_{\min}$ is therefore rational only if the user sets

$$g_{\min} \;\geq\; \frac{10^4\, C_{\max}}{q_{\min}\, \tau_{\min}},$$

with $q_{\min}$ the smallest move the envelope allows and $\tau_{\min}$ the shortest interval between rotations implied by the trigger. A 2 USD round-trip cost on a 200 USD move rotated hourly ($\tau = 1/8760$) requires $g_{\min} \geq 876{,}000$ bps, which shows that hourly rotation of small amounts is never rational at that cost. The envelope does not compute this bound today; Section 16 proposes deriving a floor for $g_{\min}$ at signing time.

### 12.4 Cost to an attacker

| ATTACK | ATTACKER COST | DEFENDER COST | BOUND |
|---|---|---|---|
| Sybil sessions | One HTTP request each | One row each; a Turnkey sub-organization only if a wallet is provisioned | Unbounded without a rate limit |
| Duplicate supplies with a stolen spend token | Requests only | User pays gas and bears exposure | $N_{\text{dup}} \cdot C_{\text{exec}}$ with $N_{\text{dup}}$ from Game 2 |
| Price under-reporting | Must corrupt the price source | Limits admit up to $\nu_{\text{day}}/(1-\eta)$ | 10.7 |
| Rewriting unanchored receipts | Operator access | Lost auditability for that window | Bounded by anchor interval |

The only attack with near-zero cost and non-zero defender cost is Sybil session creation. It does not move user funds; it consumes storage and, if wallets are provisioned, custodian quota.

13

# Complexity, performance and evaluation methodology

No benchmark results exist in the repository, and none are reported here. This section gives the asymptotic costs, an analytical latency model, and a reproducible methodology for measuring both.

### 13.1 Asymptotic costs

Let $N$ be the total number of receipts, $N_u$ the reservations of user $u$ within the daily window, $|R|$ the serialized size of a receipt, $s$ the number of scopes with moved heads, and $n$ the number of legs in a workflow with dependency edges $E$.

| OPERATION | TIME | STORAGE WRITTEN | NOTES |
|---|---|---|---|
| Authenticate | $O(1)$ | 0 | one digest lookup |
| Mandate create | $O(\|M\|)$ | $O(\|M\|)$ | canonicalization of terms |
| Gate evaluation | $O(1)$ | 0 | fixed number of predicates and allowlist lookups |
| Claim | $O(\log N)$ | $O(1)$ rows | B-tree insert and row locks |
| Policy reservation | $O(N_u)$ | $O(1)$ | sum over the window; indexed by user, so not $O(N)$ |
| Receipt append | $O(\|R\| + \log N)$ | $O(\|R\|)$ | JCS, one SHA-256, optional Ed25519 |
| Verify | $O(\|R\|)$ | 0 | hashing, one signature check, $O(1)$ lookups |
| Anchor | $O(s)$ | $O(s)$ calldata | one transaction per interval |
| Workflow scheduling | $O(n + \|E\|)$ | $O(n)$ claims | legs run sequentially |
| Standing tick | $O(\|\text{due}\|)$ | $O(\|\text{due}\|)$ | evaluation records |

Storage grows linearly in executions. The claim, budget, reservation and receipt tables each gain $O(1)$ rows per execution; no table grows super-linearly.

## 13.2 Latency model

The end-to-end latency of a mainnet lend, from the confirming call to the filled response, decomposes as

$$T_{\text{lend}} = T_{\text{gates}} + T_{\text{db}} + k\,(T_K + T_{\text{bcast}} + T_{\text{incl}}) + T_{\text{attest}}, \qquad k \in \{1, 2\}$$

with $k = 2$ when an approve precedes the operation. For a chain with block interval $\beta$ and a transaction submitted at a uniformly random time that is included in the next block, $\mathbb{E}[T_{\text{incl}}] = \beta/2 + T_{\text{prop}}$. The polling loop adds a discretization error of at most one poll interval $\pi$, so

$$\mathbb{E}[T_{\text{lend}}] \approx T_{\text{gates}} + T_{\text{db}} + k\Big(T_K + T_{\text{bcast}} + \tfrac{\beta}{2} + T_{\text{prop}} + \tfrac{\pi}{2}\Big) + T_{\text{attest}}.$$

On Base ($\beta$ = 2 s) inclusion is a few seconds; the custodian and RPC round trips are of the same order. Waiting for the finality table instead of inclusion would add $F(c, r) \cdot \beta$: 30 × 2 s = 60 s on Base at the standard risk class. That is the latency cost of closing the gap noted in 11.2.

A workflow of $n$ sequential legs has latency $\sum_i T_i$, and a bridge leg adds the bridge's fill time on the destination chain, which dominates.

## 13.3 Throughput

Throughput per wallet is bounded by nonce serialization: two executions on the same (chain, wallet) receive distinct nonces from the lease, and the second cannot be included before the first. Throughput across wallets is bounded by the single API machine (Fly.io shared-cpu-1x, 1 GB, request soft limit 200 and hard limit 400 concurrent) and by the custodian's and RPC providers' rate limits. Policy reservations serialize per user under an advisory lock, so users do not contend with each other.

## 13.4 Proposed methodology

Measurements should be reproducible from the repository and separable into components.

| EXPERIMENT | SETUP | METRIC | PARAMETERS |
|---|---|---|---|
| E1 Gate overhead | In-process app, in-memory stores, stubbed signer and RPC | p50, p95, p99 of time to the signing call | $10^4$ requests, refusal and allow mixes |
| E2 Claim contention | PGlite or Postgres 16; $c$ concurrent requests sharing one $\kappa$ | duplicates produced (expected 0), p99 latency | $c \in \{2, 8, 32, 128\}$ |
| E3 Fork execution | Anvil mainnet fork of Base via testkit-fork; real Morpho vault | $T_{\text{lend}}$ breakdown per term of 13.2 | automine on and off; block time 2 s |
| E4 Testnet end to end | testbench against a staging API | success rate, latency distribution, gas per op | 100 supplies, 0.1 USDC each |
| E5 Storage growth | Synthetic receipts | bytes per receipt; verify time vs $N$ | $N$ up to $10^6$ |
| E6 Anchor cost | Base Sepolia | gas per anchor vs $s$; $c_{\text{anchor}}$ vs interval | $s \in \{1, 10, 100\}$ |

Report each as a distribution, not a mean, and keep fork, testnet and mainnet results in separate tables, since only mainnet reflects real RPC and inclusion behaviour.

14

# Implementation architecture

The reference implementation is a TypeScript monorepo (pnpm workspaces; 7 applications, 49 packages). One process, `apps/api-worker`, holds every rule in this paper. It runs on one always-on Fly.io machine in Singapore beside a managed Postgres 16 database. The public site (Next.js on Vercel) proxies `/v1`, `/mcp` and `/health` to it from the same origin.

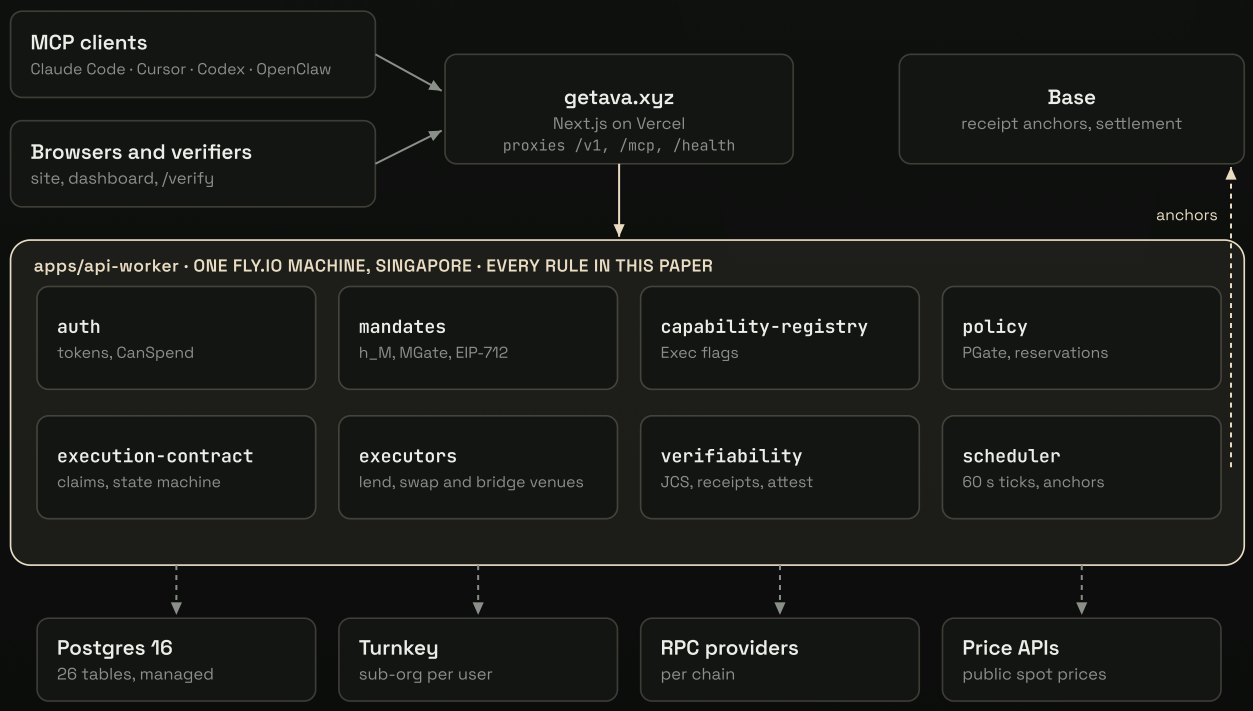



**FIGURE 14** What is deployed. One API process holds every rule; the site is a same-origin proxy; four external dependencies are trusted only for their stated jobs.

## 14.1 Code map

| CONCEPT IN THIS PAPER | LOCATION | NOTES |
|---|---|---|
| Authenticate, CanSpend | `apps/api-worker/src/middleware/auth.ts`, `spend-credential.ts`, `lib/principal.ts` | deny-by-default allowlist of public routes |
| Tokens, device codes | `lib/auth-tokens.ts`, `lib/device-codes.ts` | digest-only storage |
| Mandate, $h_M$, MGate, EIP-712 | `packages/mandates` (`hash.ts`, `enforce.ts`, `signature.ts`) | terms projection in `extractMandateTerms` |
| Reg, Exec | `packages/capability-registry` | every flag defaults to false |
| Preview $P$, $h_P$ | `lib/morpho-lend-execute.ts`, `lib/aave-lend-execute.ts` | `lendGatePreImage` |
| PGate, Π | `packages/policy`, `lib/policy-gate.ts`, `lib/policy-state.ts` | mainnet defaults in 6.4 |
| Claim, Bud, Non | `packages/execution-contract` (`claim.ts`, `state.ts`), `packages/ledger-postgres` (`postgres-execution-claim-store.ts`) | serializable transaction |
| PR | `packages/ledger-postgres` (`postgres-policy-reservation-store.ts`), `lib/policy-budget.ts` | advisory lock per (mode, user) |
| Finality table | `packages/execution-contract/src/finality.ts`, `lib/finality-gate.ts` | used by the Compound path |
| Attestation, proof, settlement | `lib/attest-onchain.ts`, `packages/verifiability` | five-field comparison for transfers |
| Receipt chain, signing | `lib/receipt-chain.ts`, `packages/verifiability` (`jcs.ts`, `signing.ts`) | applied at the storage boundary |
| Anchors | `lib/receipt-anchor.ts`, `lib/scheduler.ts` | scheduler-only gas spend |

| CONCEPT IN THIS PAPER | LOCATION | NOTES |
|---|---|---|
| Standing envelopes | `packages/mandates` (`standing*.ts`), `lib/standing-lifecycle.ts`, `apps/execution-worker` | Perm in `checkStandingMove` |
| Workflows | `packages/intent` (`decomposeWorkflow`), `routes/workflows.ts`, `lib/workflow-execute.ts`, `lib/workflow-runner.ts` | re-derivation on execute |
| MCP surface | `lib/mcp.ts`, `lib/mcp-lend-execute.ts` | 29 tools, 18 advertised by default |
| Custodian | `packages/wallet-turnkey`, `lib/morpho-turnkey-signer.ts` | sub-organization per user |

### 14.2 Persistence and boot

At boot in Postgres mode, every schema is applied idempotently (there is no separate migration runner) and the mandate cache is hydrated from the database before the first request is served. Requesting Postgres without a connection string is a boot error. Twenty-six tables back the stores of Section 4.3. Several stores remain process-local even in Postgres mode: the HTTP Idempotency-Key replay cache, automations, copilot pending plans, previews and paper balances, and the policy state that the advisory gate reads.

### 14.3 Transports

REST routes and MCP tools dispatch to the same executor functions. MCP is JSON-RPC 2.0 over HTTP at `/mcp`. The endpoint is public so that a fresh client can call the session tool; every other tool authenticates per call, compares any userId argument with the credential, and refuses spend tools to dashboard credentials. `tools/list` advertises an 18-tool default, because some clients cap the total number of tools across servers; `tools/call` accepts all 29 by name.

### 14.4 Scheduler

When enabled, an in-process timer evaluates active automations and mandates against a live price every 60 seconds, evaluates due standing envelopes, and on a separate, slower timer publishes receipt anchors. It never submits an order. Anchoring is its only gas-spending duty.

The execution-worker and agent-service applications are libraries imported by the API, not separately deployed services.

15

# Limitations, trade-offs and implementation discrepancies

These are the places where the implementation at commit `eff936c` does not meet the model of Sections 4 to 11, or where a deliberate trade-off has a cost. Each is checkable in the code.

## 15.1 Discrepancies with the model

| # | MODEL SAYS | CODE DOES | EFFECT |
|---|---|---|---|
| **D1** | Claims guard every spend path (G4) | The MCP lend tool receives no claim store; REST lend reserves only when the caller sends an idempotencyKey | Duplicate MCP calls are bounded by policy reservations, not prevented (Game 2) |
| **D2** | verified requires finalized (Prop. 11.4) | Morpho and Aave lend executors report filled on successful inclusion | A reorg after inclusion could leave a filled receipt for a reverted transaction; low on the chains used, not zero |
| **D3** | Ambiguous broadcasts are reconciled | The claim store models broadcast_unknown, but no reconciler scans it | A crash between broadcast and receipt write loses the receipt (completeness, 11.8) |
| **D4** | Human confirmation means a human approved | The server can only check content binding (Game 3′) | An agent can self-confirm any action already inside the mandate and limits |
| **D5** | One canonical encoder | Three encoders, equivalent on the domains hashed today | Latent divergence if a binding ever includes undefined fields or non-safe integers |
| **D6** | Standing envelopes rotate unattended | No execute function is bound to the scheduler in production | Envelopes bound only calls that name them; nothing rotates on its own |
| **D7** | Durable state | Automations and the HTTP idempotency cache are in process memory | Automations vanish on deploy; replays of `/v1/execute` are only caught within one process |
| **D8** | Multiple RPC endpoints reduce trust (10.5) | Base endpoints are tried for availability, not compared | A single lying endpoint can affect confirmation |

The repository's README states that multi-leg execution is not wired end to end. The code now has `POST /v1/workflows/execute`, which admits, reserves and runs legs; most legs still refuse at the registry or the agent-credential gate. The README lags the code.

## 15.2 Deliberate trade-offs

- **Custody for usability.** Turnkey custody lets an agent act without a human signing each transaction. The cost is that Ava's operator is trusted with signing authority (Section 10.6). External-wallet linking exists for users unwilling to accept that.
- **Off-chain predicates.** Mandate, preview and standing checks are richer than what an on-chain validator can express, and weaker, because a compromised operator can skip them.
- **Exact approvals.** One extra transaction per supply in exchange for no standing allowance (12.1).
- **Fail closed.** Many legitimate requests are refused because a registry flag is off or a credential is absent. The design accepts false refusals to avoid false fills.
- **Single machine.** Simplicity and strong consistency from one database, with no failover and a single region.

## 15.3 Unverified assumptions

Whether production receipts are Ed25519-signed and whether anchoring is enabled depend on environment configuration that the repository cannot show; the verify endpoint reports both per receipt. Gas, custodian and RPC costs are not measured here. The root `vercel.json` and `railway.toml` can also boot the API; whether either is deployed could not be determined.

16

# Future research directions

Everything in this section is proposed, not implemented.

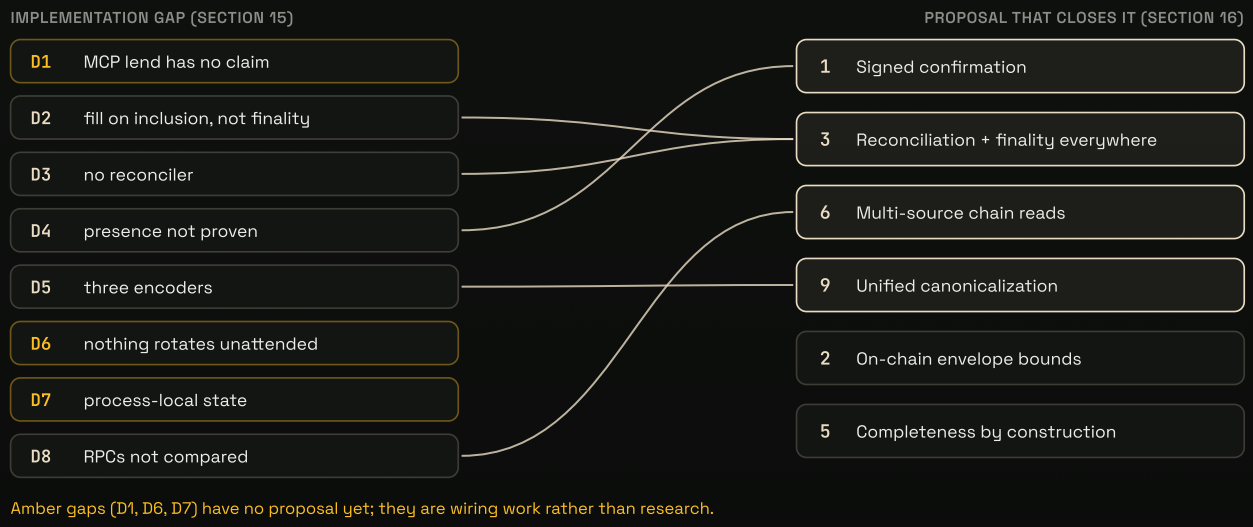



**FIGURE 15** Four of the ten proposals close a named discrepancy from Section 15.

1. **Signed confirmation (closes D4).** Replace the bare preview hash with a user-key signature over an EIP-712 struct `AvaPreview(bytes32 previewHash, uint64 expiry)`. Verification would reuse the plan-signature path, reducing Game 3′ to EUF-CMA and making human presence a cryptographic property rather than a client convention.
2. **On-chain enforcement of envelope bounds.** Express Perm (6.9) as an ERC-7579 validator module on a smart account, so per-move caps, cumulative caps, venue and chain allowlists and expiry are enforced by consensus. Ava's off-chain checks would then be a second line rather than the only line, and an operator compromise could no longer exceed the bounds.
3. **Reconciliation and finality for every path (closes D2, D3).** Drive all executors through the claim state machine, add a reconciler that resolves broadcast_unknown and included claims against the chain, and set filled only at finalized. The cost is the latency term $F(c, r) \cdot \beta$ of 13.2.
4. **Merkle receipt logs with inclusion proofs.** Replace per-scope linear chains with a Merkle log in the style of Certificate Transparency [RFC6962]. Anchors would then carry a tree root, and a third party could check inclusion of one receipt in $O(\log N)$ without the chain of its predecessors, and consistency between two anchors.
5. **Completeness by construction.** Bind the claim's nonce lease to the receipt sequence so that every nonce Ava consumes on a wallet maps to exactly one receipt or one recorded refusal. An auditor could then prove omissions from the on-chain nonce sequence alone.
6. **Multi-source chain reads.** Require agreement of $k$ independent RPC endpoints, or verify inclusion with a light client, reducing RPC trust from one endpoint to $k$ (10.5).
7. **Oracle-robust limits.** Express limits in base units for stable assets and use a median of independent price sources elsewhere, removing the $1/(1-\eta)$ amplification of 10.7.

8. **Derived gain floors.** Compute the lower bound on $g_{\min}$ of 12.3 from the envelope's trigger, minimum move and a gas estimate at signing time, and refuse envelopes whose bounds make every rotation unprofitable.
9. **Unified canonicalization.** Route every binding through the RFC 8785 encoder and add property tests that fuzz the three current encoders for divergence (closes D5).
10. **Formal verification of the gate ordering.** The safety argument of 11.1 is a control-flow property of a small number of functions. It is a candidate for model checking (for example, TLA+ for the claim state machine and its concurrent reservations).

17

# Conclusion

Ava treats a language-model agent as an untrusted proposer and builds everything security-relevant outside it. A spend happens only after a fixed conjunction of predicates: a spend-audience credential, a capability row that is implemented, reachable and switched on, a mandate owned by the caller, a policy decision that on mainnet requires a matching preview digest, a single-flight claim and a durable budget reservation. The signer is constructed after all of them. Outcomes are hashed, chained, optionally signed and anchored, and the verdicts that describe them keep integrity, chain proof and settlement agreement apart.

> The contribution is less any single primitive than the discipline of the arrangement, and an exact account of where it holds.

Content binding of confirmations, at-most-once reservation, budget conservation, destination restriction and anchored immutability follow from the code under stated assumptions. Human presence, completeness of the receipt log, finality on the lend paths and protection of unkeyed or MCP-originated duplicates do not yet, and Section 16 names the mechanism that would close each one. The system is custodial and single-operator, and it says so; its guarantees are those of an auditable service, not of a consensus protocol.

# References

**[Bu14]** V. Buterin. Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform. White paper, 2014.

**[CW09]** S. A. Crosby, D. S. Wallach. Efficient Data Structures for Tamper-Evident Logging. USENIX Security Symposium, 2009.

**[EIP712]** R. Bloemen, L. Logvinov, J. Evans. EIP-712: Typed structured data hashing and signing. Ethereum Improvement Proposals, 2017.

**[ERC4337]** V. Buterin, Y. Weiss, et al. ERC-4337: Account Abstraction Using Alt Mempool. Ethereum Improvement Proposals, 2021.

**[ERC4626]** J. Santoro, et al. ERC-4626: Tokenized Vaults. Ethereum Improvement Proposals, 2021.

**[ERC7579]** K. Kopp (zeroknots), et al. ERC-7579: Minimal Modular Smart Accounts. Ethereum Improvement Proposals, 2023.

**[ERC7683]** M. Toda, M. Rice, N. Pai. ERC-7683: Cross Chain Intents. Ethereum Improvement Proposals, 2024.

**[Gr23]** K. Greshake, S. Abdelnabi, S. Mishra, C. Endres, T. Holz, M. Fritz. Not What You've Signed Up For: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection. ACM Workshop on Artificial Intelligence and Security (AISec), 2023.

**[He12]** P. Helland. Idempotence Is Not a Medical Condition. ACM Queue 10(4), 2012.

**[HS91]** S. Haber, W. S. Stornetta. How to Time-Stamp a Digital Document. Journal of Cryptology 3(2), 1991.

**[MCP25]** Anthropic. Model Context Protocol Specification. modelcontextprotocol.io, 2025.

**[RFC6962]** B. Laurie, A. Langley, E. Kasper. Certificate Transparency. IETF RFC 6962, 2013.

**[RFC8032]** S. Josefsson, I. Liusvaara. Edwards-Curve Digital Signature Algorithm (EdDSA). IETF RFC 8032, 2017.

**[RFC8785]** A. Rundgren, B. Jordan, S. Erdtman. JSON Canonicalization Scheme (JCS). IETF RFC 8785, 2020.

**[SK99]** B. Schneier, J. Kelsey. Secure Audit Logs to Support Computer Forensics. ACM Transactions on Information and System Security 2(2), 1999.

**[SS75]** J. H. Saltzer, M. D. Schroeder. The Protection of Information in Computer Systems. Proceedings of the IEEE 63(9), 1975.

**[Wo14]** G. Wood. Ethereum: A Secure Decentralised Generalised Transaction Ledger (Yellow Paper), 2014.

Author lists for the Ethereum Improvement Proposals are abbreviated; the canonical text and full authorship are at eips.ethereum.org. Implementation references are to the Ava repository (github.com/kamalbuilds/ava-4.0) at commit `eff936c`.

# Appendix A. Notation

| SYMBOL | MEANING |
|---|---|
| $u, g, \omega$ | principal, agent instance, signing wallet |
| $a = (k, c, v, x, q, \omega, \rho)$ | action: kind, chain, venue, asset, base-unit amount, wallet, target contract |
| $d(x)$ | decimals of asset $x$ |
| $\tau, \alpha, \pi$ | bearer secret, audience, principal |
| $H$, canon, JCS | SHA-256, sorted-key canonical JSON, RFC 8785 |
| $M, T(M), h_M$ | mandate, its terms, terms hash |
| $\Gamma, \Phi$ | mandate constraints, exit conditions |
| $\rho = (\text{impl}, \text{api}, \text{mcp}, \text{op}, \text{proven}, \text{ek})$ | capability row and its flags |
| $\text{Exec}_s$ | executability on surface $s$ |
| $P, h_P, \hat{h}$ | preview pre-image, its digest, the caller-supplied digest |
| $\Pi = (\nu_{\text{order}}, \nu_{\text{day}}, E_{\max}, s_{\max}, \theta, \text{kill})$ | policy limits |
| $\nu(a), p_x$ | USD notional of $a$, spot price of $x$ |
| $D_u(t), E_u$ | rolling 24 h notional, open exposure |
| $\sigma(a), h_\sigma$ | claim subject, request digest |

| SYMBOL | MEANING |
|---|---|
| $\kappa$, $K(u, \kappa)$ | idempotency key, claim key |
| $L(m)$, consumed, reserved | cumulative limit and budget counters for mandate hash $m$ |
| $F(c, r)$, $\beta$ | required confirmations for chain $c$ and risk class $r$, block interval |
| $R$, $h(R)$, $\zeta$, seq, prev | receipt, its hash, chain scope, sequence, predecessor hash |
| $\mathcal{E}$, $\lambda$, Perm | standing envelope, its limits, move predicate |
| $Q_{\text{cons}}$, $m$ | durable consumption and moves today under an envelope |
| $h_{\text{plan}}$, $\kappa_i$ | plan hash, leg idempotency key |
| $\varepsilon_H$, $\varepsilon_\Sigma$ | collision advantage of the hash, forgery advantage of the signature |
| O, K | operator (Ava process), custodian (Turnkey) |

# Appendix B. Refusal codes and constants

| CODE | STAGE | MEANING |
|---|---|---|
| `UNAUTHORIZED` | authentication | presented credential does not resolve |
| `AUTH_REQUIRED` | authentication | no credential on a non-public route |
| `USER_ID_ASSERTION_MISMATCH` | authentication | claimed user differs from the principal |
| `SPEND_CREDENTIAL_REQUIRED` | authorization | principal cannot spend |
| `MANDATE_NOT_FOUND` | mandate | absent or not owned (404 by design) |
| `MANDATE_KIND_NOT_ALLOWED` | mandate | kind excluded by Γ.allowedKinds |
| `LEND_VENUE_MISMATCH` | routing | venue not wired for kind and chain |
| `POLICY_REJECTED` | policy | a limit or allowlist violated |
| `POLICY_HUMAN_CONFIRMATION_REQUIRED` | policy | no matching preview digest |
| `POLICY_NOTIONAL_UNPRICEABLE` | policy | price unavailable; never priced at zero |
| `POLICY_DAILY_NOTIONAL_EXCEEDED`, `POLICY_OPEN_EXPOSURE_EXCEEDED` | reservation | durable limit reached |
| `EXECUTION_ALREADY_CLAIMED` | claim | same key, same content, already reserved |
| `EXECUTION_CLAIM_CONTENT_MISMATCH` | claim | same key, different content |

| CODE | STAGE | MEANING |
|---|---|---|
| `EXECUTION_CLAIM_BUDGET_EXHAUSTED`, `_MISMATCH` | claim | cumulative limit or its denomination |
| `LEND_INSUFFICIENT_BALANCE` | executor | wallet balance below the amount |
| `PLAN_SIGNATURE_INVALID`, `_PLAN_MISMATCH` | workflow | signature does not cover the recomputed plan |
| `WORKFLOW_IS_STANDING` | workflow | repeated action must use a standing envelope |
| `STANDING_EXPIRED`, `_REVOKED`, `_OPERATION_NOT_ALLOWED`, `_VENUE_NOT_ALLOWED`, `_CHAIN_NOT_ALLOWED`, `_ASSET_NOT_ALLOWED`, `_MOVE_EXCEEDS_LIMIT`, `_CUMULATIVE_EXCEEDED`, `_GAIN_TOO_SMALL`, `_RATE_LIMITED` | standing | the corresponding conjunct of Perm failed |
| `RECEIPT_NOT_FOUND` | verify | absent, or by-id and neither owned nor public |

**Constants.** Device-code TTL 10 min. Scheduler interval 60 s. Daily policy window 24 h. Token entropy 256 bits. Finality risk thresholds 1,000 and 100,000 USD. Mainnet policy defaults as in 6.4.