

Unseen · Whitepaper

Sealed copy trading on Solana perps

The protocol in full: how a leader's signal becomes one pooled order from an unattributed account, how each day's NAV is verified before money moves, how trades are revealed and checked, and the programs and services that run it.

Version 1.0 · October 2026

Contents

1. Introduction
2. System overview
3. Strategies, shares and the pool
4. Signals
5. The Sealed Executor
6. Exec accounts and commitments
7. Venue: Jupiter Perps
8. Epochs and settlement
9. Fees
10. Reveal
11. Attested statistics
12. On-chain programs
13. Security design
14. Infrastructure
15. Interfaces
16. Parameters and encodings

1. Introduction

Unseen is a copy-trading protocol for perpetual futures on Solana. A trader, the *leader*, runs a strategy; *followers* deposit USDC into it and receive shares in the strategy's ledger. Every signal the leader sends becomes one pooled order for the whole strategy, so the leader and every follower receive the same fill and pay the same costs per share. Each new position opens from a fresh execution account that carries a 32-byte commitment instead of a strategy identifier. An observer sees that a trade happened, with its market, side and size, but cannot tell which strategy made it until the strategy's reveal delay has passed and the execution is published.

This chapter states the problem Unseen addresses, what the protocol treats as its product, why it is built on oracle-priced perpetuals, and the design goals the remaining chapters specify.

1.1 What copy trading sells

Copy trading on Solana is a large paid market whose revenue goes to the tools rather than to the traders being copied. On DefiLlama (fetched 2026-10-15), all-time revenue was \$466.5M for Axiom, \$442.0M for Photon, \$288.2M for GMGN and \$203.0M for BullX; the Axiom and GMGN figures also count other chains. GMGN charges 1% per trade and Axiom 0.75–0.95% net. The trader whose wallet sits on a PnL leaderboard is the input to these products and earns nothing from them.

What a copier pays for is *attribution*: a wallet address together with its public history. A single trade is cheap to reproduce once visible on-chain; the information with value is which trades come from a source worth following, and on a public chain anyone who watches has it. Real-time monitoring over WebSockets and Yellowstone gRPC is sold as a standard RPC product, so any profitable address is found and followed.

1.2 The parties and what they bear

Party	What happens today	Cost to them
Tracked trader	Tools rank wallets by PnL, win rate and ROI and offer auto-copy with per-trade fees	Copied without payment. In thin markets, copiers bid up the trader's entries and crowd the exits. No revenue.
Follower	Copy bots enter after the lead wallet; in low-liquidity tokens the lead's buy has already moved the price, and some "smart money" wallets are bots or wash traders	Worse fills, records that cannot be verified, exposure to becoming exit liquidity
AI-agent builder	A public wallet or vault shows every position as it happens; Hyperliquid vaults have dedicated public pages	The strategy can be learned from its trade log
Privacy-tool user	Vanish routes private spot swaps through shielded execution; Umbra offers private transfers and swaps	Privacy, with no mechanism to earn from followers

Table 1.1 — Parties to copy trading today.

Hyperliquid's vaults show that the payment side works when built into a venue: vault leaders take a 10% profit share and must keep at least 5% of the vault. Every vault position is public, and Hyperliquid is not on Solana. Profit shares paid to leaders and private execution for traders exist today in separate products. Unseen combines them.

1.3 Attribution as the product

Unseen sells attribution as the product. The trade is executed in public, on a public venue, at the venue's oracle price. What the protocol withholds, for a bounded and declared time, is the link between the trade and the strategy that made it. Four mechanisms produce this separation.

- 1. Pooled execution.** A strategy's equity is the leader's stake plus follower deposits, tracked as shares. A signal produces one order for the whole strategy. There is no separate leader order, so there is no ordering between leader and followers, and every share carries the same fill and fees (§3 Strategies, shares and the pool).
- 2. Unlinkable execution accounts.** Each new position opens from a fresh program-derived account with a random 64-bit index, funded from one pool vault shared by every strategy. The account stores its market, side, notional, collateral and a 32-byte commitment, never a strategy identifier or signal sequence. Each execution extends an on-chain hash chain of commitments (§6 Exec accounts and commitments).

3. **Sealed signals.** The leader's client checks the executor's Intel TDX attestation, signs the signal with a registered key and encrypts it to a key held only by the attested confidential VM. The signal does not appear in public before the trade lands (§4 Signals, §5 The Sealed Executor).
4. **Delayed, collective reveal.** Each strategy declares an immutable reveal delay D between 24 hours and 30 days, 7 days by default. The *opening* of an execution, the data that hashes to its commitment, is posted on-chain once the execution is closed, D has passed since closure, and every strategy active in that UTC day has passed its own delay. Anyone can then rebuild every strategy's trade log and NAV from chain data (§10 Reveal).

While the record is withheld, two mechanisms keep it checked. An independent verifier recomputes each daily epoch's NAV from on-chain fills and the confidential openings and co-signs it before any deposit or withdrawal settles (§8 Epochs and settlement). Hourly statistics of return, drawdown and exposure band are quantized, delayed and jointly signed by the executor and the verifier (§11 Attested statistics).

A paying follower receives the leader's exact fill and a record verified at every epoch, while the leader keeps the strategy's positions private for D . An observer sees aggregate flow without attribution until publication.

1.4 Why oracle-priced perpetuals

A capable trader usually refuses followers because followers move the price on entry and exit and, in thin markets, take the leader's liquidity. On liquid oracle-priced perpetuals this cost is small and computable. Jupiter Perps fills requests at the oracle price, so follower flow does not move the price the leader trades at, and its price-impact fee is deterministic, about 1 bp per started ~\$375k of SOL notional. Its other fees, 0.06% to open and to close plus an additive fee on large net flow per minute, follow published formulas and are capped at 0.50% of trade size.

Followers therefore impose little cost on the leader, which removes the usual objection to follower capital. Free copying is equally cheap, so on these markets attribution is what separates a trader from a follower business. Unseen targets Jupiter Perps' SOL, ETH and wBTC markets, long and short, at up to 3× leverage. Memecoins are not served: capacity in seconds-old tokens is tiny and speed decides the edge. Oracle pricing and published fees also make capacity computable in advance: §3.7 derives a strategy's capacity from the per-minute flow slice, and §7 Venue: Jupiter Perps gives the fee arithmetic.

1.5 Design goals

Unseen is specified against nine goals, each enforced by a named mechanism (§13 Security design collects them).

Goal	Enforcing mechanism	Chapter
G1 Same fill for every share of a strategy	One pooled order per signal; batch price-impact fees split pro rata by notional, remainder by receipt order	§3, §5, §9
G2 No attribution before reveal	Fresh exec PDA with random u64 index; 32-byte commitment; one common collateral ratio and wire slippage for all strategies	§6, §12
G3 Signals readable only in the attested executor	Deployment-bound Ed25519 signal encrypted to a P-256 key bound into the TDX report data	§4, §5
G4 Correctness without trusting the enclave	Verifier signature on every execute, fill record, reduction, close and stats post; verifier co-signature and challenge window before finalization	§5, §8
G5 Custody bounded by the program	Transfers only pool → exec → venue and fixed returns to the pool; the enclave key cannot withdraw; payouts are pull claims at finalized values	§12
G6 Exit at the last final NAV	Protected 20%-per-epoch withdrawal reserve; owner-signed emergency withdrawal available while paused or while a proposal is stuck	§8
G7 Public verifiability after reveal	Openings hash to commitments; revealed evidence includes the signed signal, executor receipt and policy evidence; public NAV replay	§10
G8 Leader capital at risk on the same terms	At least 5% of strategy equity, checked at deposit, withdrawal request and settlement	§3
G9 Fees return to the token	The protocol's whole fee share buys \$UNSEEN and burns it; deposits, fees and payouts are USDC; shares are a non-transferable u128 ledger entry	§3, §9

Table 1.2 — Design goals and the mechanism that enforces each.

G4 shapes the architecture most. An order is in plaintext when it reaches a public venue, so the hot path runs in a trusted execution environment, which builds a transaction in milliseconds. Accounting correctness does not rest on that environment: settlement waits for an independent recomputation, and every risk-increasing transaction carries a second signature from a party outside the enclave.

1.6 Version 0

Version 0 covers Jupiter Perps SOL, ETH and wBTC, long and short, up to 3×; up to 10 invited strategies run by human leaders or AI agents through the same interface; USDC deposits; caps of \$250,000 per strategy and \$2,000,000 for the platform; allowlisted, screened followers outside the US and the EU; daily epochs, delayed reveal, attested hourly statistics and an open-source verifier; a leader console, a signal API and SDK, a page per strategy and a revealed-trade explorer.

1.7 Conventions

- **Amounts.** USDC amounts are integers in base units with six decimals (1,000,000 base units = 1 USDC). Shares are integers with nine decimals (10^9 share units = one whole share). NAV per share is in USDC base units per whole share.
- **Basis points.** 10,000 bps = 100%. A target weight of 15,000 bps is a position of 150% of strategy equity; leverage is already included in the weight.
- **Integers.** Money, shares, sequences, slots and indices are canonical decimal strings in JSON and fixed-width little-endian integers in byte encodings. Arithmetic is exact, with explicit floor or ceiling; no floating-point value enters a money path.
- **Notation.** H is SHA-256, $\|$ is byte concatenation and $\text{u64le}(x)$ is the 8-byte little-endian encoding of x . Public keys are 32 bytes, base58 in JSON; strategy identifiers, salts and commitments are 32 bytes, 64 lowercase hex characters.
- **Time.** Protocol time is the program's Unix clock. UTC day k is the interval $[86400k, 86400(k+1))$. Chain events also carry slots.
- **Parameters.** Values fixed by the protocol are stated as numbers. Values left to deployment configuration, such as the challenge window or the slice retry window, are described as configured parameters; a local development value is quoted only when labelled as one. §16 Parameters and encodings collects them all.

1.8 Organization of this paper

§2 gives the system overview and the lifecycle of one trade. §3 to §4 specify strategies, shares and signals; §5 to §7 the Sealed Executor, execution accounts and commitments, and the venue; §8 to §11 epochs and settlement, fees, reveal and attested statistics. §12 documents the on-chain programs, §13 the security design, §14 infrastructure, §15 interfaces, and §16 every parameter and byte encoding with a glossary.

2. System overview

This chapter describes Unseen as a whole: the parties and what each can do, the components, how funds move, the path of one trade from signal to reveal, and what an observer can see at each stage. Later chapters specify each part in detail.

2.1 Actors

Table 2.1 lists what each party holds and what it can and cannot do. Each refusal in the last column is enforced by a program check, a required second signature or the absence of the necessary key.

Actor	Holds	Can	Cannot
Leader (human)	Owner wallet of the strategy; a separate signal key; at least 5% of strategy equity	Create a strategy within protocol bounds; sign target-weight signals; rotate the signal key; read its own positions through a one-use signed proof; claim its performance fees	Move pool funds; trade outside its declared markets and leverage; change D after creation; read another strategy's executions before reveal
Agent	Same as a leader; model and code stay with the agent operator	Same interface as a human leader	Same as a human leader
Follower	Wallet; a share balance in the strategy's ledger	Request a deposit or withdrawal; cancel a pending request; claim payouts and refunds; withdraw at the last final NAV in an emergency; challenge an epoch with a bond	Transfer shares; deposit during the shadow period, beyond a cap, or without eligibility
Sealed Executor	Enclave Ed25519 key registered on-chain; a P-256 encryption key bound into its TDX quote	Decrypt signals; size, slice and order trades; build and sign execution transactions; propose each epoch's NAV; post stats and reveals	Withdraw funds; execute, record a fill, reduce, close or post stats without the verifier's signature; settle deposits or withdrawals at a NAV the verifier has not co-signed
Verifier	Registered verifier key; authorized access to openings before reveal	Admit receipts in order; co-sign each execution and stats post; recompute and co-sign each epoch's NAV, liquid cash and evidence hash	Sign an execution alone, since the enclave signature is also required; move funds
Guardian	Multisig key	Pause execution and finalization; revoke the enclave key immediately; resolve challenges and guardian reviews; void epochs; unwind positions to	Withdraw funds; send rescued funds anywhere but the pool vault

Actor	Holds	Can	Cannot
		the pool after enclave expiry or revocation	
Admin	Squads multisig; program upgrade authority	Register an enclave key and rotate the verifier or guardian, each after a 48-hour timelock; issue leader invitations; record follower eligibility	Activate an enclave key or a role change before its timelock expires
Venue	Jupiter Perps program; Jupiter-run keepers	Fill position requests at the oracle price	Learn which strategy owns an execution account

Table 2.1 — Actors, what they hold, and what they can and cannot do.

Leaders and agents use one interface. The protocol records the kind (human or agent) on the strategy and applies identical rules to both. A follower's eligibility is a program account written by the admin after screening; a leader's admission is a single-use invitation account (§3.10 Application and admission).

2.2 Components

Figure 2.1 shows the components and the channels between them.

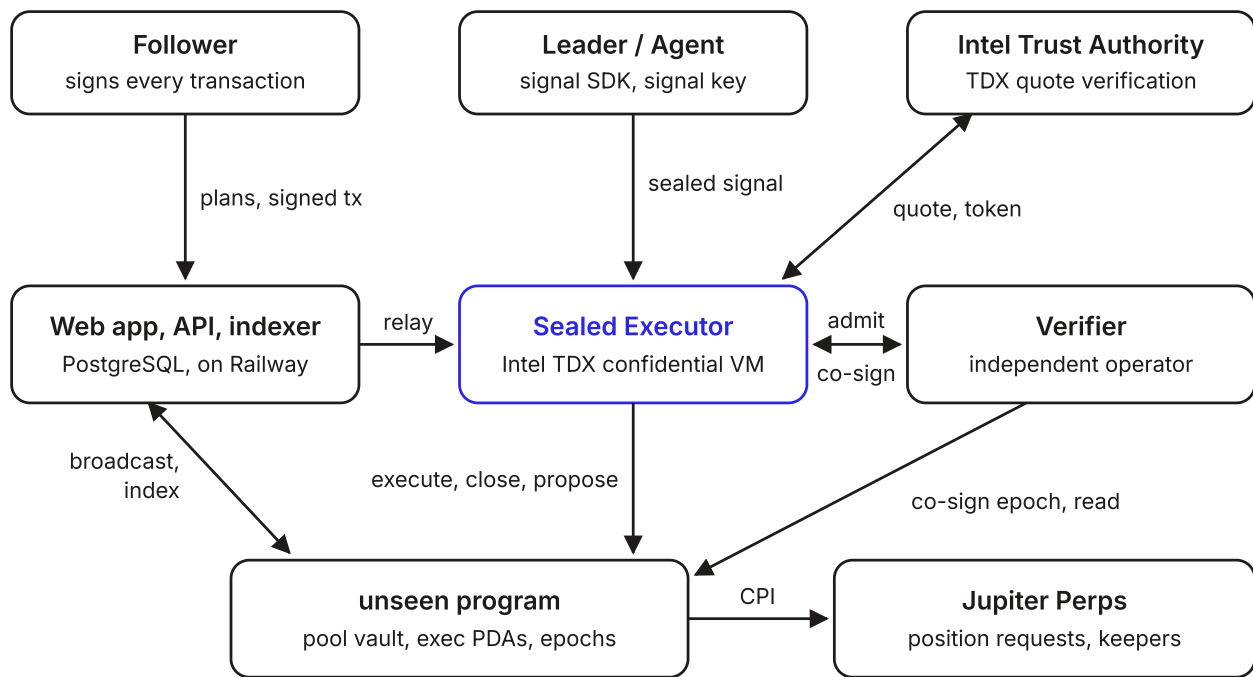


Figure 2.1 — Component map. Signals reach the Sealed Executor only as ciphertext, directly or through the API relay. Every execution transaction carries the signatures of both the executor and the independent verifier.

The **unseen program** is an Anchor program on Solana. It holds the protocol configuration with the complete registry of strategies, the shared pool and its USDC vault, one account per strategy, per share holder, per epoch and per execution, the commitment receipts and the daily reveal-period certificates. It moves funds only along the paths described in §2.3 and checks every transaction against the limits in §3 Strategies, shares and the pool and §12 On-chain programs.

The **Sealed Executor** is a Rust service running inside an Intel TDX confidential virtual machine. It generates its Ed25519 signing key inside the VM and binds that key and a derived P-256 encryption key into a TDX quote obtained through configfs-tsm and verified by Intel Trust Authority; the attestation is published for clients. The executor accepts sealed signals, signs receipts, sizes, slices and schedules orders, sends transactions through a configured sender (RPC, Jito bundles or Helius Sender), proposes epoch NAVs, posts statistics with the verifier and publishes reveals. Its state is an authenticated-encrypted journal whose every save is anchored to an independent monotonic counter service, so an older copy of the journal cannot be restored silently (§5 The Sealed Executor).

The **Verifier** is an independent Rust service run by a separate operator with its own RPC endpoint and its own signing key. It admits each executor receipt into its own journal in strict sequence, independently checks and co-signs every execution, fill record, reduction, close and statistics post, recomputes each epoch's NAV and liquid cash from chain data and the authorized openings, and co-signs the proposal on-chain. It also answers public replay of revealed history without any private input (§8 Epochs and settlement).

Public services run as separate Railway services: the API (wallet sign-in, transaction plans, public reads, the server-sent event stream and the sealed-signal relay), the indexer and jobs worker, which reconciles chain accounts into PostgreSQL and writes an unattributed event outbox, and the database. The web app serves strategy pages, the leader console and the revealed-trade explorer. None of these services holds an enclave, verifier or user key, and the public database contains no table that links an execution to a strategy before reveal (§14 Infrastructure).

External services are Jupiter Perps for execution, Intel Trust Authority for attestation tokens, Range for deposit screening when its key is configured, and Squads for the admin multisig.

2.3 Custody: the pool loop

All strategies share one pool account and one USDC vault, a token account whose authority is the pool PDA. Funds enter and leave it only along fixed paths.

- **In.** A leader's initial stake at strategy creation; follower deposits, held in escrow in the vault but outside the pool's executable cash until settled at an epoch; challenge bonds.
- **Trading loop.** `execute` moves a position's collateral from the pool vault to a fresh execution vault and from there into the venue request. Partial and full closes return collateral and realized PnL from the venue to the execution vault and on to the pool vault; a never-filled request is refunded the same way. Execution fees go from the pool vault to the protocol fee account, a token account of a program address no key signs for, which only `buyback_and_burn` spends; venue fees are paid out of collateral.
- **Out.** Withdrawal payouts, refunds, leader performance fees, the protocol share and returned bonds are credited as claims in program accounts and pulled by their owners into token accounts of the configured USDC mint that belong to them.

The enclave key has no instruction that moves funds to an arbitrary destination. Its transfers are pool → exec → approved venue and fixed-destination returns to the pool. Guardian rescue after enclave expiry or revocation also returns funds only to the pool. Because payouts are pull claims, a frozen or missing recipient token account cannot block another owner's settlement.

2.4 Lifecycle of a trade

Figure 2.2 follows one trade from signal to reveal. The numbered steps below match the arrows.

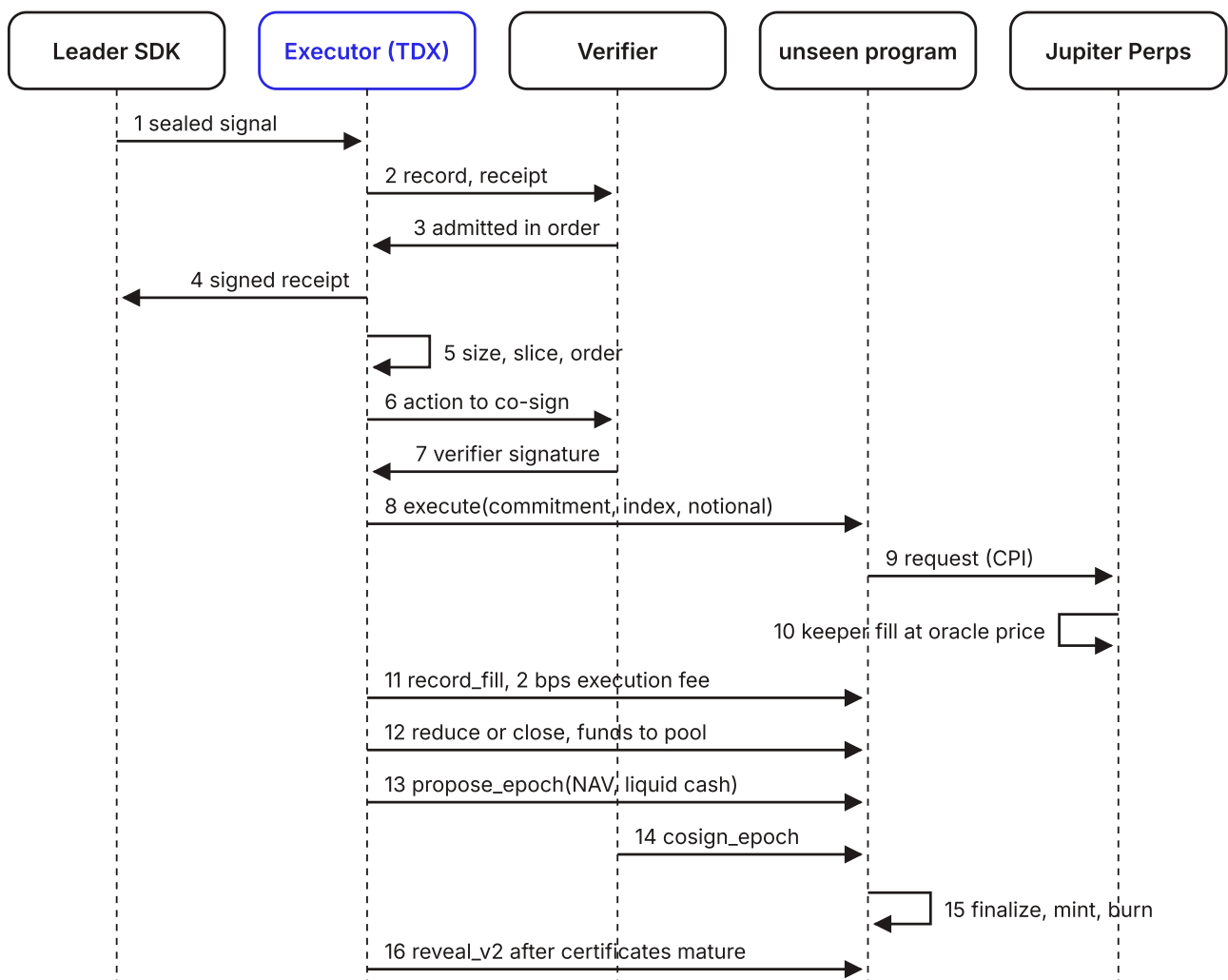


Figure 2.2 — Lifecycle of one trade. Steps 8, 11 and 12 are transactions signed by both the executor and the verifier. Step 10 is Jupiter's own keeper transaction. Steps 13 to 15 recur daily; step 16 follows the strategy's reveal delay and the daily period certificates.

Signal and admission (steps 1–4). The leader's client verifies the executor's TDX attestation, signs a deployment-bound signal of target weights with the registered signal key and encrypts it to the attested key; the ciphertext goes to the executor directly or through the API relay. The executor decrypts and checks it, assigns the next global receipt sequence and a strictly increasing microsecond timestamp, signs a receipt and persists the record. The verifier admits the record only if its receipt sequence is exactly one greater than the last it admitted. The client checks the returned receipt against the signal it sent (§4 Signals).

Sizing and ordering (step 5). The executor recomputes the strategy's equity and turns each target weight into a signed notional difference from the current position: reductions first, then new exposure, split into slices no larger than the configured slice size. A later receipt cannot overtake an unfinished action in the same market from an earlier receipt (§5 The Sealed Executor).

Execution (steps 6–9). For each slice that opens exposure, the executor draws a random 64-bit index and a 32-byte salt, computes the policy evidence hash and the commitment, and asks the verifier to authorize the action; the verifier checks the signed target, sizing, the strategy's own liquid cash and the slice policy, and adds its signature. The `execute` transaction checks the

enclave key, the verifier signature, the market, the common collateral ratio, the anonymous pool envelope over the complete strategy registry, the pool-wide flow limit per market and minute, and the signal's validity slot. It then extends the commitment hash chain, moves collateral pool → exec vault → venue and creates the position request with a retry deadline of the current slot plus the configured retry window (§6 Exec accounts and commitments, §7 Venue: Jupiter Perps).

Fill (steps 10–11). A Jupiter keeper fills the request at the oracle price in its own transaction. The executor and the verifier then record the fill on-chain, which charges the 2 bps execution fee on the filled notional. An unfilled request is retried until its deadline and then dropped with its collateral refunded to the pool.

Adjustments and closure (step 12). A later signal that reduces exposure in the same direction partially closes the same execution account, keeping its entry price and charging fees on the reduced notional only. A signal that flips direction closes the old account and opens a fresh one. When a position is flat, all balances return to the pool vault.

As an example with the protocol's parameters: a strategy with 30,000 USDC of equity and a signal of SOL long at 15,000 bps targets 45,000 USDC of SOL notional. At a common collateral ratio of 3,334 bps (the local development value; the permitted range is 3,334–10,000 bps) the request carries $\lceil 45,000 \times 3334 / 10000 \rceil = 15,003$ USDC of collateral. The fill is charged 9 USDC of execution fee (2 bps) and Jupiter's 0.06% opening fee of 27 USDC plus its impact fee (§7), all borne by the strategy as a whole.

Epoch (steps 13–15). Once a day per strategy, the executor proposes the strategy's NAV and liquid cash; the verifier recomputes both at the proposal slot and prices and co-signs them with an evidence hash; after the configured challenge window, anyone can finalize the epoch, which charges the performance fee on high-water-mark gains, protects 20% of net NAV as the next withdrawal reserve, and lets pending deposits and withdrawals settle at the final NAV (§2.5).

Reveal (step 16). After the execution is closed, its strategy's delay has passed and the reveal-period certificates for the days it was opened and closed have matured, the executor posts the opening on-chain together with the signed signal, receipt and policy evidence. The program checks that the opening hashes to the stored commitment. Rent is then reclaimed, leaving a permanent archive of the execution's public fields and a tombstone that prevents reuse of its index (§10 Reveal).

2.5 The daily cycle

Deposits and withdrawals never settle at an unverified price. Each strategy has its own schedule: a next boundary timestamp one day (86,400 seconds) after its last finalization, and a next boundary slot at the configured number of slots per epoch. The executor may propose once the timestamp is due and no later than the boundary slot plus a configured proposal deadline. A proposal whose NAV per share differs from the last final value by more than 15% requires guardian review. The verifier's co-signature must match the immutable proposal exactly and arrive within the deadline; it opens the challenge window, during which anyone can post a bond to freeze finalization until the guardian resolves the challenge. Finalization is permissionless once the window has passed with

no open challenge and any required guardian review resolved. Settlement of individual share accounts is also permissionless and follows a FIFO queue for withdrawals within the epoch's 20% budget. If a proposal cannot finalize, it is voided after a configured timeout, and owners keep the ability to exit at the last final NAV throughout (§8 Epochs and settlement).

2.6 What is public when

Table 2.2 lists each class of information and who can read it at each stage.

Information	Before execution	After execution, before reveal	After reveal
Signal: markets, sides, weights, validity, deviation bound	Executor only; the relay sees ciphertext	Not published; the resulting trade is visible on an unattributed exec account	Public, as the signed signal in the revealed evidence
Which strategy made a trade	Executor only	Behind the commitment; known to the executor, the verifier and that strategy's leader	Public
Exec account: market, side, notional, collateral, commitment, chain position, policy hash, operation sequence	Not applicable	Public	Public, kept in the archive after rent reclamation
Receipt sequence and receipt time	Executor and verifier	Bound inside the V2 commitment	Public and auditable
Strategy parameters, AUM, cap, deposits and withdrawals, epochs and co-signatures	Public	Public	Public
Return, maximum drawdown, exposure band	Not applicable	Hourly, quantized to 100 bps, delayed at least one hour, signed by executor and verifier	Recomputable from chain data
Aggregate pool state: open notional per market, locked collateral, flow, total commitment count	Public	Public	Public

Table 2.2 — Visibility of information by stage.

Before reveal, three parties can read private state, each through a distinct channel. The executor holds everything in its sealed journal. The verifier receives openings through an endpoint that requires a fresh Ed25519 proof from the registered verifier key, valid for 60 seconds, with a nonce that cannot be replayed. A leader reads only its own strategy through a one-use proof signed by

the leader wallet, also valid for at most 60 seconds. The public API, the indexer and the database never receive the mapping.

The time at which an execution becomes public is the latest of three clocks. Let t_c be the execution's closing time, D_s its strategy's delay, d_o and d_c the UTC days of its opening and closing, f_d the first time at or after the end of day d at which the whole pool held no open notional (pending requests count as open), and $D_{\max}$ the longest delay of any strategy ever registered:

$$t_{\text{reveal}} \geq \max(t_c + D_s, f_{d_o} + D_{\max}, f_{d_c} + D_{\max})$$

The second and third terms are the maturity times of the immutable reveal-period certificates the program issues for those days. They implement the rule that no opening of a day is published until every strategy active in that day has passed its own delay, without publishing which strategies were active (§10 Reveal).

2.7 Clusters and deployment

The same program and services run on two clusters. On mainnet, the venue is Jupiter Perps and the mint is USDC. Devnet has no Jupiter Perps, so the devnet profile deploys the program with the fake venue adapter and a devnet test USDC mint, which exercises the whole flow end to end. Production configuration refuses fixture mode, development routes, the local fake attestation, the in-process test chain and the accelerated clock: the program accepts a clock scale other than 1 only in its separate local-testing build. Deployment consists of deploying the program, initializing Config, Pool and their accounts, starting the executor in its TDX VM so it can produce its attestation, registering the enclave key through the admin, and starting the Railway services (§14 Infrastructure).

3. Strategies, shares and the pool

A strategy is a pooled account of equity owned by its leader and its followers in proportion to their shares. This chapter specifies the strategy and share accounts, how equity and NAV per share are computed, how shares are minted and burned, how the performance fee and leader compensation change the share supply at finalization, the caps and the leader stake, the strategy's states including the shadow period, how leaders and followers are admitted, and the shared pool that holds every strategy's cash.

3.1 The strategy account

Each strategy is a program account at the PDA with seeds `["strategy", id]`, where `id` is a 32-byte identifier. The API's plan builder derives it as $H(\text{wallet} || ":" || \text{idempotency key})$ over the leader's base58 wallet address, so a retried creation plan produces the same account. Table 3.1 lists the fields that matter for accounting and admission; §12 On-chain programs gives the complete layout.

Field	Meaning	Bound or initial value
<code>leader</code> , <code>kind</code>	Owner wallet; 0 for a human, 1 for an agent	Holder of a valid invitation
<code>signal_key</code>	Registered Ed25519 key that signs signals	Set at creation; rotatable by the leader
<code>performance_fee_bps</code>	Fee on follower gains above the high-water mark	0–2,500; plan default 1,500
<code>reveal_delay</code>	D in seconds, immutable	86,400–2,592,000; plan default 604,800
<code>markets</code>	Bitmask: bit 0 SOL, bit 1 ETH, bit 2 wBTC	1–7
<code>max_leverage_bps</code>	Declared ceiling on the sum of target weights	1–30,000, explicit
<code>drawdown_kill_bps</code>	Declared drawdown at which the strategy stops taking risk and deposits	1 up to the configured protocol maximum
<code>aum</code>	Equity at the last final NAV	Initial stake
<code>total_shares</code> , <code>leader_shares</code>	Share supply and the leader's part, u128	Shares minted for the stake
<code>pending_deposits</code> , <code>pending_withdraw_shares</code>	Requests awaiting the next settlement	0
<code>last_nav</code> , <code>high_water_mark</code>	NAV per share at the last finalization, and its running maximum	Genesis NAV
<code>liquid_cash</code> , <code>withdrawal_reserve</code>	Verified unlocked cash; cash protected for withdrawals	Stake; 0
<code>shadow_ends_at</code> , <code>waiver_ends_at</code>	End of the shadow period; end of the	Creation + 30 days; creation + 90 days if

Field	Meaning	Bound or initial value
	founding waiver	founding, else 0
<code>next_epoch_at</code> , <code>next_epoch_slot</code>	Next accounting boundary	Creation + 86,400 s; current slot + configured epoch slots
<code>cumulative_deposits</code> , <code>cumulative_withdrawals</code> , <code>cumulative_performance_fees</code>	Lifetime cash flows	Stake; 0; 0

Table 3.1 — Strategy account fields used by the share arithmetic.

The strategy account holds no position mapping: nothing in it identifies the execution accounts the strategy owns. Its parameters, AUM, flows and latest attested statistics are public from creation.

3.2 The share account

Each owner's holding is a program account at `["share", strategy, owner]`. Shares are a u128 balance in a ledger; there is no token mint and no transfer instruction.

Field	Meaning
<code>shares</code>	Share balance in units of 10^{-9} share
<code>cost_basis</code>	USDC paid in for the current balance, reduced pro rata on withdrawal
<code>pending_deposit</code>	Escrowed USDC awaiting settlement
<code>pending_withdraw</code>	Shares awaiting burn
<code>pending_epoch</code>	Epoch number at which the request is settled
<code>request_sequence</code>	Owner nonce; each new request must carry the next value
<code>queue_sequence</code>	Position of a withdrawal in the strategy's FIFO queue
<code>claimable_withdraw</code>	Payouts and refunds credited and not yet claimed
<code>last_settled_epoch</code>	Replay guard for settlement

Table 3.2 — Share account fields.

An account carries at most one pending request: a deposit requires no pending deposit or withdrawal, and so does a withdrawal. A request made while a proposal is active targets the epoch after it; otherwise it targets the next epoch.

3.3 Equity

The equity of strategy s at a valuation point is computed from its cash flows and its execution accounts. Let D_s , W_s and P_s be cumulative deposits (including the leader's stake), cumulative withdrawals paid and cumulative performance fees, and let X_s be the set of execution accounts whose openings belong to s . For each execution x , R_x is its realized PnL and f_x^{exec} , f_x^{venue} the execution and venue fees charged to it. Then

$$E_s = D_s - W_s - P_s + \sum_{x \in X_s} (R_x - f_x^{\text{exec}} - f_x^{\text{venue}}) + \sum_{x \in X_s^{\text{filled}}} U_x$$

where the unrealized PnL of a filled position with remaining notional n_x , entry price p_x and oracle price p_m is

$$U_x = \sigma_x \cdot \text{trunc}\left(\frac{n_x (p_m - p_x)}{p_x}\right), \quad \sigma_x = \begin{cases} +1 & \text{long} \\ -1 & \text{short} \end{cases}$$

Prices are in USD micros, amounts in USDC base units, and division truncates toward zero. A pending request contributes neither PnL nor fees; its collateral is still the strategy's equity at face value. For an epoch, the valuation point is the proposal slot and the oracle prices the program captured with the proposal. A venue fill that has landed but is not yet recorded on-chain blocks a proposal until it is recorded.

The same equity decomposes by asset as cash plus venue collateral plus unrealized PnL. Revealed history is checked against both identities (§10 Reveal). The verifier computes E_s independently from its own RPC and the authenticated openings; the executor's proposal must equal it exactly (§8 Epochs and settlement).

3.4 NAV per share

NAV per share π is expressed in USDC base units per whole share, a whole share being 10^9 share units:

$$\pi = \left\lfloor \frac{E \cdot 10^9}{S} \right\rfloor$$

where S is the total share supply. The value of s share units at NAV per share π is $\lfloor s \pi / 10^9 \rfloor$. A strategy starts at the genesis NAV π_0 , a deployment parameter that must be at least 1,000,000 (one USDC per share) and has no default; the local development profile uses 1,000,000. The program stores the last finalized value as `last_nav` and the running maximum as `high_water_mark`.

3.5 Minting and burning

Shares change only at strategy creation, at settlement after a finalized epoch, through leader fee compensation (§3.6) and through owner-signed emergency withdrawal at the last final NAV.

At creation the leader's stake k is transferred to the pool vault and minted at the genesis NAV:

$$S_0 = \left\lfloor \frac{k \cdot 10^9}{\pi_0} \right\rfloor, \quad S_0 > 0$$

A deposit a is escrowed in the pool vault at request time and minted at the final NAV per share π_f of the epoch it settles in:

$$\Delta S = \left\lfloor \frac{a \cdot 10^9}{\pi_f} \right\rfloor, \quad \Delta S > 0$$

Settlement adds a to the owner's cost basis, to the strategy's AUM and liquid cash, and to the pool's executable cash.

A withdrawal of s shares is paid from the epoch's withdrawal budget B (§3.6). With B_{used} already paid in that epoch, the amount available to the request at the head of the FIFO queue is

$$b = \min(B - B_{\text{used}}, \text{liquid cash}, \text{withdrawal reserve}, \text{pool cash})$$

and the shares burned and the payout are

$$s^* = \min\left(s, \left\lfloor \frac{b \cdot 10^9}{\pi_f} \right\rfloor\right), \quad \text{payout} = \left\lfloor \frac{s^* \pi_f}{10^9} \right\rfloor$$

The payout is credited to `claimable_withdraw` and the cost basis falls by $\lfloor c \cdot s^* / s_{\text{held}} \rfloor$. A remainder stays queued in the same position and carries to the next epoch. A request worth zero base units at the last final NAV is refused when made; if a later NAV makes a queued remainder worth zero, settlement releases only that reservation, keeps the owner's shares and cost basis, and advances the queue.

Every conversion rounds down. Rounding residue therefore stays in the strategy's equity and accrues to the remaining holders.

3.6 Finalization: performance fee, leader compensation and the high-water mark

Finalization applies the performance fee to followers only, and compensates the leader with shares so that the leader's own equity is unaffected by a fee the leader does not pay. Let E be the proposed NAV, S the share supply, S_L the leader's shares, φ the fee rate in bps and $\pi_g = \lfloor E \cdot 10^9 / S \rfloor$ the proposed NAV per share. The follower gain above the high-water mark and the fee are

$$g = \max(0, \pi_g - \text{HWM}), \quad V = \left\lfloor \frac{(S - S_L) g}{10^9} \right\rfloor, \quad F = \left\lfloor \frac{V \varphi}{10^4} \right\rfloor$$

The protocol share is $F_p = \lfloor F/5 \rfloor$ (20%), or zero while a founding waiver is active, and the leader receives $F_L = F - F_p$ as a pull claim. To keep the leader's pre-fee equity L , the program mints compensation shares to the leader:

$$L = \left\lfloor \frac{E S_L}{S} \right\rfloor, \quad N = E - F - L, \quad S'_L = \left\lceil \frac{L (S - S_L)}{N} \right\rceil, \quad \Delta S_L = S'_L - S_L$$

The rounding up assigns at most one share base unit to the leader. The finalized values are then

$$E' = E - F, \quad S' = S + \Delta S_L, \quad \pi_f = \left\lfloor \frac{E' \cdot 10^9}{S'} \right\rfloor, \quad \text{HWM}' = \max(\text{HWM}, \pi_f),$$

B , 20% of net NAV, is protected as the strategy's withdrawal reserve for the epoch. Finalization requires the verified liquid cash to cover the fee plus that reserve.

Table 3.3 works one finalization through with a genesis NAV of 1,000,000 (the local development value), a leader holding 10,000 shares, followers holding 20,000 shares, a proposed NAV of 33,000 USDC, the default fee of 1,500 bps and no waiver.

Quantity	Computation	Value
π_g	$\lfloor 33,000 \times 10^6 \times 10^9 / (30,000 \times 10^9) \rfloor$	1,100,000 (1.10 USDC per share)
g	$1,100,000 - 1,000,000$	100,000
V	$20,000 \text{ shares} \times 0.10 \text{ USDC}$	2,000 USDC
F, F_L, F_p	15% of V ; 80% / 20%	300 USDC; 240 USDC; 60 USDC
L, N	$33,000 \times 10,000 / 30,000$; $33,000 - 300 - 11,000$	11,000 USDC; 21,700 USDC
$S'_L, \Delta S_L$	$\lceil 11,000 \times 20,000 / 21,700 \rceil$ in base units	10,138.248847927 shares; 138.248847927 shares
E', S'	$33,000 - 300$; $30,000 +$ 138.248847927	32,700 USDC; 30,138.248847927 shares
π_f, HWM'	$\lfloor 32,700 \times 10^6 \times 10^9 / S' \rfloor$	1,084,999
Leader's claim on E'	$E' S'_L / S'$	11,000.000000 USDC
Followers' claim on E'	$E' (S - S_L) / S'$	21,699.999999 USDC
B	$\lfloor 32,700 / 5 \rfloor$	6,540 USDC

Table 3.3 — Worked finalization with real protocol rates.

The leader keeps exactly its pre-fee 11,000 USDC, and followers bear the full 300 USDC fee. Valued through the floored π_f , the leader's shares are worth 10,999.989861 USDC and the followers' 21,699.980000 USDC; the 0.030139 USDC difference is per-share rounding retained in equity. A deposit of 1,000 USDC settled in this epoch mints $\lfloor 10^9 \times 10^9 / 1,084,999 \rfloor = 921,659,835,631$ units, 921.659835631 shares. §9 Fees specifies the fee rules, including the founding waiver, in full.

3.7 Caps and capacity

Table 3.4 lists the bounds that apply to strategies and their flows, with the instruction that enforces each. A is a strategy's AUM at the last final NAV and Q its pending deposits; platform totals are kept in Config.

Bound	Value	Enforced in	Condition
Strategy cap	250,000 USDC	<code>create_strategy</code> , <code>request_deposit</code>	$A + Q + a \leq 250,000 \text{ USDC}$
Platform cap	2,000,000 USDC	<code>create_strategy</code> , <code>request_deposit</code>	platform AUM + platform pending + $a \leq 2,000,000 \text{ USDC}$
Strategy count	10	<code>create_strategy</code>	registry of 10 slots
Leader stake	5% of equity	deposits, leader withdrawals, settlement, emergency exits	§3.8
Leverage	3×	<code>create_strategy</code> ; signal admission	sum of target weights $\leq \min(\text{declared}, 30,000) \text{ bps}$
Withdrawals per epoch	20% of net NAV	<code>finalize_epoch</code> , <code>settle_share</code>	$B = \lfloor E'/5 \rfloor$
Notional per execution	250,000 USDC	<code>execute</code>	single request
Pool flow per market	250,000 USDC per minute	<code>execute</code> , reductions	gross opening plus reduction notional per market per protocol minute
Collateral per execution	at least 10 USDC	<code>execute</code>	collateral floor shared with the executor
Execution fees	configured share of platform AUM per day	fill recording	daily ceiling

Table 3.4 — Caps and bounds.

The executor additionally limits each strategy's flow per market to a slice, 250,000 USDC per minute in the protocol defaults, and splits larger orders into slices (§5 The Sealed Executor). Because target weights are fractions of equity, the slice bounds how much equity a strategy can rebalance in a given time. For a change of direction Δw (2 for a full long-to-short flip) at leverage λ , a per-minute slice q and T minutes allowed per rebalance, the capacity estimate is

$$A_{\max} \approx \frac{q T}{|\Delta w| \lambda}$$

A full flip at 3× rebalanced over 10 minutes gives $250,000 \times 10/6 \approx 417,000$ USDC. This is an estimate, not a protocol constant; v0 caps each strategy at 250,000 USDC and the platform at 2,000,000 USDC.

3.8 Leader stake

A leader must hold at least 5% of the strategy's equity, valued at the last final NAV per share π , counting every pending request against it. With w_L the leader's shares pending withdrawal, a deposit a is admitted only if

$$10^4 \cdot \left\lfloor \frac{(S_L - w_L) \pi}{10^9} \right\rfloor \geq 500 (A + Q + a)$$

A leader's withdrawal request for s shares requires

$$10^4 \cdot \left\lfloor \frac{(S_L - s) \pi}{10^9} \right\rfloor \geq 500 \left(A + Q - \left\lfloor \frac{s \pi}{10^9} \right\rfloor \right)$$

and settlement of s^* leader shares requires $10^4 (S_L - s^*) \geq 500 (S - s^*)$. Leader emergency exits apply the same check, including pending leader withdrawals and follower deposits. The strategy's reserved equity can therefore never exceed 20 times the leader's equity. A stake worth 10,000 USDC supports at most 200,000 USDC of reserved equity; reaching the 250,000 USDC cap requires 12,500 USDC of leader equity. The leader holds its stake on the same terms as followers: it shares every fill and pays the same execution and venue costs per share, and the compensation in §3.6 only offsets a performance fee that followers alone owe.

3.9 Strategy states and the shadow period

The public status of a strategy is derived from its account and Config, in this order:

1. **Paused** if the protocol is paused, if its latest attested drawdown has reached its declared kill level, or if it has been deactivated.
2. **Shadow period** if the current time is before `shadow_ends_at`.
3. **Capped** if $A + Q \geq 250,000$ USDC.
4. **Open** otherwise. Deposits are accepted only in this state.

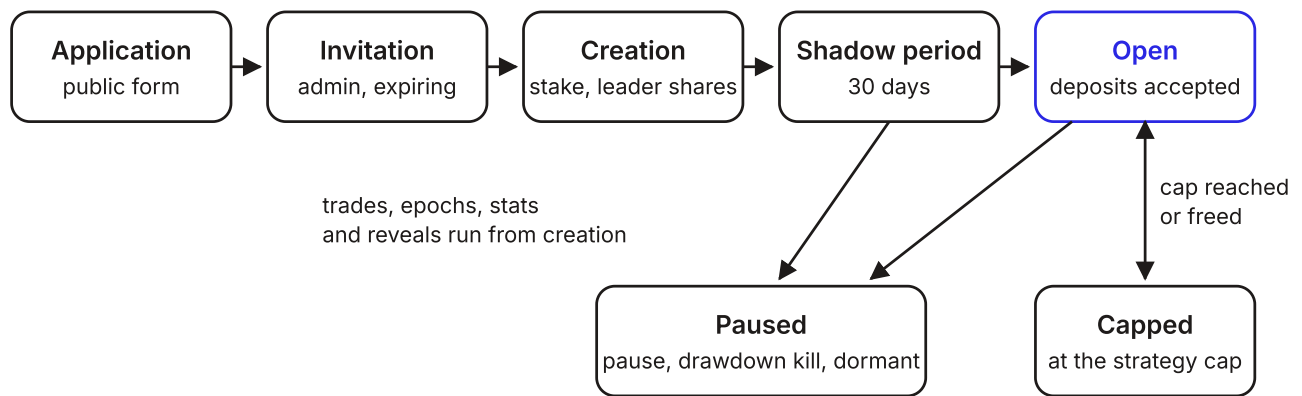


Figure 3.1 — Strategy states. A strategy trades its leader's capital from creation; deposits open only after the 30-day shadow period, and only while the strategy is neither capped nor paused.

The **shadow period** lasts 30 days (2,592,000 seconds) from creation. During it the strategy trades the leader's stake under the same rules as any other strategy: epochs are proposed, verified and finalized, hourly statistics are attested, and executions follow the reveal rule. Deposits are refused until `shadow_ends_at`, so when a strategy opens to followers it already has a month of verified epochs and attested statistics.

A **drawdown kill** is the declared `drawdown_kill_bps`, bounded at creation by the configured protocol maximum. The published maximum drawdown is monotonic; once it reaches the declared level, deposits are refused, the strategy shows as paused, and the executor admits no new risk for it. A **deactivated** strategy is one that a permissionless `deactivate_zero_strategy` has retired: it requires zero AUM, no active epoch or pending cash flow and a flat pool, and it freezes the commitment count after which no execution can be attributed to that strategy. Its registry slot remains.

3.10 Application and admission

Leaders. A prospective leader submits a public application with a name of at most 80 characters, the leader kind (human or agent), one or more distinct markets, the size of its own book (`under_50k`, `50k_250k`, `250k_2m` or `over_2m`), an optional wallet and a contact of at most 120 characters. The route requires an idempotency key, accepts ten writes per IP per minute and returns only an identifier, a status and a timestamp. An accepted applicant receives an invitation: the admin's `invite_leader` writes a `LeaderInvitation` account at `["leader_invitation", leader]` with an expiry and a founding flag. `create_strategy` requires an unused, unexpired invitation for the signing wallet and marks it used; it grants the 90-day protocol-share waiver only if the invitation is marked founding. Creation also checks every parameter bound in Table 3.1, the strategy count, the caps, transfers the stake, mints the leader's shares, registers the signal key, appends the strategy to the registry in Config and raises the registry's longest-ever reveal delay if `D` exceeds it.

Followers. A follower submits an access request with its wallet, an optional contact and the literal attestations `notUSPerson: true` and `notInEU: true`. The request alone grants nothing. Screening

runs through the Range risk API when its key is configured, and otherwise through an allowlist the operator manages; the admin then writes an `Eligibility` account at `["eligibility", owner]` holding an allowed flag and an expiry. `request_deposit` requires that account to be allowed and unexpired. Eligibility gates deposits only: withdrawal requests, cancellations, claims and emergency withdrawals need nothing but the owner's signature.

3.11 The shared pool

All strategies share one `Pool` account at `["pool", venue_program]` and its USDC vault. The pool separates three kinds of balance. Executable cash is the cash available to trading, from which collateral is drawn. Protected withdrawal reserves are the sum of every strategy's 20% reserve; no execution and no ordinary execution fee can spend them. Escrowed deposits sit in the vault but are outside executable cash until they settle, so pending funds cannot finance trades.

Each strategy's own share of the cash is its `liquid_cash`, checked at every finalization. Before an order is authorized, the verifier checks that the selected strategy's own liquid cash covers its withdrawal reserve, prospective fees, the new margin and every earlier authorization still able to land; another strategy's idle cash cannot satisfy that check. Independently, the program admits new pool risk only if it fits a conservative envelope computed over the complete strategy registry at the smallest active strategy's capacity, without knowing which strategy the order belongs to (§5 The Sealed Executor, §12 On-chain programs). The pool also stores the head and length of the commitment hash chain that every execution extends (§6 Exec accounts and commitments).

4. Signals

A signal is the only instruction a leader gives the protocol: the portfolio the strategy should hold, as target weights of equity, and the conditions under which the request may be carried out. This chapter specifies target weights, the signed fields and their canonical encoding, deployment binding, sealing, admission and receipts, sequence and validity, sizing into slices, lifecycle states and retries.

4.1 Target weights

A target weight names a market, a side and a weight in basis points of strategy equity. `SOL long 15000` means a long SOL position with notional equal to 150% of equity. Leverage is already contained in the weight and is not applied again. A signal carries one to three targets, each market at most once, and the sum of the weights must not exceed the strategy's declared maximum leverage, itself at most 30,000 bps (3×).

A signal describes the strategy's whole book, not an order. The executor trades only the difference between target and current position; a weight of 0 closes its market, and a market in the signal's scope that the targets omit is driven to zero (§4.8). Because weights are fractions of equity, signals scale with AUM, and resending one cannot double a position (§4.10).

For market m , let w_m be the signed weight (positive for long, negative for short, zero when omitted), E the strategy's equity at sizing and c_m its current signed filled notional. The target notional and the difference to trade are

$$n_m^* = \text{trunc}\left(\frac{E \cdot w_m}{10^4}\right), \quad \delta_m = n_m^* - c_m$$

With 30,000 USDC of equity and an existing SOL long of 15,000 USDC, the signal `SOL long 15000, ETH short 5000` gives $n_{\text{SOL}}^* = 45,000$ and $\delta_{\text{SOL}} = +30,000$ USDC, and $n_{\text{ETH}}^* = \delta_{\text{ETH}} = -15,000$ USDC. The weights sum to 20,000 bps, so the strategy must have declared a maximum leverage of at least 2×.

4.2 Signal fields

Live signals use version 2. Table 4.1 lists the fields of a signed signal as it is sealed and submitted.

Field	Encoding	Rule
version	JSON number 2	Version 1 refused for live execution
genesisHash, programId, executorPublicKey	base58, 32 bytes each	Cluster genesis hash, unseen program, registered enclave key
strategyId	64 lowercase hex characters	An existing strategy
sequence	decimal u64 string	Positive; above every earlier sequence of the strategy
targets	1–3 objects {market, side, weightBps}	Distinct declared markets among SOL, ETH, wBTC; long or short; sum within declared leverage
validUntilSlot	decimal u64 string	Above the current slot at admission
maxOracleDeviationBps	decimal string	0 to 10,000
signalKey	base58, 32 bytes	The strategy's registered signal_key
signature	base64, 88 characters	Ed25519 over the canonical bytes of §4.3

Table 4.1 — Fields of a signed signal (version 2).

4.3 Canonical signing encoding

The signal key signs a fixed UTF-8 JSON serialization of every field except version and signature, preceded by a domain string. The keys appear in exactly this order, with no whitespace and no trailing newline:

```
{
  "domain": "unseen-signal-v2",
  "genesisHash": "<base58>",
  "programId": "<base58>",
  "executorPublicKey": "<base58>",
  "strategyId": "<64 hex>",
  "sequence": "7",
  "targets": [
    {
      "market": "SOL",
      "side": "long",
      "weightBps": "15000"
    },
    {
      "market": "ETH",
      "side": "short",
      "weightBps": "5000"
    }
  ],
  "validUntilSlot": "<decimal slot>",
  "maxOracleDeviationBps": "30",
  "signalKey": "<base58>"
}
```

Target objects have the keys market, side, weightBps in that order, and targets keep the order the leader listed them in. Integers are canonical decimal strings: digits only, no sign, no leading

zero except `0`. Every value is base58, lowercase hex, decimal digits or a fixed market or side name, so no JSON escaping applies, and `JSON.stringify` in a browser and `serde_json` in the Rust executor and verifier produce identical bytes.

The signal key signs these bytes with Ed25519, and the executor verifies with strict Ed25519 verification. The signed-signal hash, referenced by the receipt, the policy evidence and the commitment, covers the bytes and the raw 64-byte signature:

$$h_{\text{signal}} = H(\text{bytes}(\text{signal}) \parallel \text{sig}_{64})$$

Version 1 used the domain `unseen-signal-v1` and omitted the three deployment fields. Its encoder is kept so that historical V1 proofs remain verifiable; the executor refuses version 1 for any live signal.

4.4 Deployment binding

The deployment fields bind a signal to one cluster (`genesisHash`), one deployment of the program on it (`programId`) and one attested executor identity (`executorPublicKey`). A devnet signal fails on mainnet, a second deployment refuses the first one's signals, and signals made for an enclave key are refused once that key is rotated. The executor compares all three with its chain snapshot: the genesis hash, which must also equal `Config.deployment_genesis`, the program ID and `Config.enclave`. The verifier repeats the comparison from its own RPC at admission. The same three values are bound into the sealed envelope's authenticated data (§4.5), the receipt (§4.6), the V2 commitment preimage (§6 Exec accounts and commitments) and the leader-view proof (§4.11). A client reads them from `GET /v1/deployment` and checks them against independently pinned values before signing.

4.5 Sealing and transport

A signal is never sent in clear. The client seals the signed signal to an encryption key that only the attested executor holds.

1. **Attestation.** The client verifies the executor's Intel Trust Authority token against a pinned trust policy (issuer, signing keys, TDX measurement and workload policy identifiers). Its report data must equal the SHA-512 hash of this runtime binding, which ties the executor's Ed25519 key and P-256 encryption key to the deployment (§5 The Sealed Executor):

```
unseen-tdx-runtime-
v1\n<genesisHash>\n<programId>\n<executorPublicKey>\n<encryptionPublicK
ey>
```

2. **Key agreement.** An ephemeral P-256 key pair and the executor's encryption key give an ECDH shared secret; HKDF-SHA256 with salt $H(\text{aad})$ and info `unseen-sealed-signal-v1` derives a

256-bit key.

3. **Encryption.** AES-256-GCM encrypts the JSON of the signed signal under a random 12-byte nonce with a 128-bit tag and the authenticated data `unseen-sealed-signal-v1\n<genesisHash>\n<programId>\n<executorPublicKey>`.
4. **Envelope.** `{version: "unseen-sealed-signal-v1", ephemeralPublicKey, nonce, ciphertext}`, holding the 65-byte uncompressed key, the nonce and the ciphertext with its tag, each in standard base64.

The envelope goes to the executor's `POST /v1/signals/sealed`, or through the API relay at the same path for a browser that cannot reach the VM. The relay requires a leader session, accepts 60 envelopes per wallet per minute, checks only the envelope's shape (canonical base64, a 65-byte key beginning with `0x04`, a 12-byte nonce, 17 to 16,384 ciphertext bytes) and forwards it with a 15-second timeout; it has no key that could decrypt it. The executor refuses envelopes while its attestation is expired, decryption under another deployment's authenticated data fails, and a production executor refuses unsealed signals.

4.6 Admission and the receipt

Inside the VM, the executor decrypts the envelope and applies the following checks in order. The first failure ends admission.

1. The signature verifies under `signalKey`; `version` is 2 and `sequence` is positive.
2. If the strategy already has a signal with this sequence, identical canonical bytes are an exact replay and return the original acknowledgment (§4.10), and different bytes are refused as `REPLAY_DETECTED`. Otherwise the sequence must exceed every earlier sequence of the strategy.
3. The strategy exists, `signalKey` is its registered key, and the deployment fields match (§4.4).
4. `validUntilSlot` is above the current slot and `maxOracleDeviationBps` is at most 10,000.
5. The strategy is not paused, and the targets satisfy Table 4.1 with their sum within `min(max_leverage_bps, 30,000)`.
6. The strategy's drawdown, recomputed from current equity, is below its declared kill level.
7. The signal key is within its admission rate, a configured number per minute charged only after the signature and key checks (60 in the local development profile).

The executor then assigns a receipt. Its sequence r is one more than the largest it has issued across all strategies, and its time is

$$t_r = \max(t_{\text{now}}, t_{r-1} + 1) \quad \text{in Unix microseconds}$$

so receipt times strictly increase with receipt sequence. The enclave key signs the receipt bytes, the newline-joined lines:

```

unseen-receipt-v2
<genesisHash>
<programId>
<executorPublicKey>
<receiptSequence>
<receivedAtUnixMicros>
<signedSignalHash>

```

The executor persists the record in its sealed journal and submits it to the verifier, authenticated with the enclave key. The verifier admits it only if the receipt sequence is exactly one more than the last it admitted and the receipt time is strictly later; it then rechecks the signal against its own RPC (registered key, deployment, signature, signed-signal hash, receipt signature, markets and leverage), freezes the receipt's market scope (§4.8) and persists the record in its own journal. Only after this does the executor acknowledge the signal.

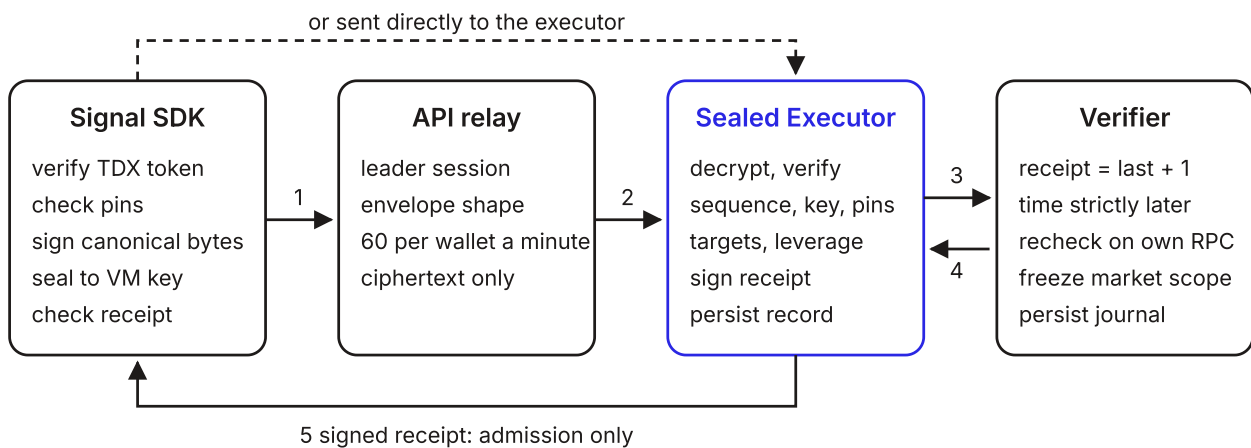


Figure 4.1 — Admission path. (1) The SDK sends the sealed envelope to the relay, or directly to the executor. (2) The relay forwards ciphertext. (3) The executor submits the receipted record to the verifier. (4) The verifier admits it in strict receipt order. (5) The executor returns the signed receipt, which the SDK checks.

The acknowledgment (`RealSignalReceipt` , `acceptedForExecution: true`) returns the signal as admitted, always in state `received` , with the `receipt` : deployment fields, `receiptSequence` , `receivedAtUnixMicros` , `signedSignalHash` and `receiptSignature` . The SDK accepts it only if the deployment fields match, `signedSignalHash` equals its own computation and the receipt signature verifies under the executor key.

A receipt proves admission and a place in the global order, not sizing or fills, and a replay returns the same document. Its sequence, time and signed-signal hash enter the V2 commitment of every execution the signal opens, so after reveal anyone can check which signal caused an execution and in what order competing signals arrived (§6, §10).

4.7 Sequence, validity and deviation

Sequence. Each strategy's sequences start at 1 and strictly increase; gaps are allowed. The pair (strategy, sequence) is the signal's identity for idempotency, and the sequence appears in every commitment and opening the signal produces.

Validity. `validUntilSlot` is checked at admission, before sizing (an expired signal is dropped), before each action is submitted, and on-chain: `execute`, `partial_close` and `close_exec` each require the current slot not to exceed the validity slot they carry, which for ordinary actions is the signal's and is bound into the policy evidence. A request already funded at the venue stays reserved after expiry until it fills or is refunded at its retry deadline (§4.10).

Oracle deviation. Every venue request carries one common, configured wire slippage bound, so the public request reveals nothing about the leader's tolerance. The leader's `maxOracleDeviationBps` is enforced privately: while a request is pending, the executor compares the oracle price p with the request's reference price p_{ref} and cancels when

$$\left\lceil \frac{|p - p_{\text{ref}}| \cdot 10^4}{p_{\text{ref}}} \right\rceil > \text{maxOracleDeviationBps} \implies \text{cancel the request}$$

The cancellation is a dual-signed refund-only close returning all collateral to the pool; the signal's remaining actions in that market end with `oracle_deviation`. The verifier checks the bound independently.

4.8 From signal to slices

The verifier freezes each receipt's **market scope** at admission: the markets in its targets, every market where the strategy has a pending or filled position, and every market still held by an unfinished earlier receipt of the strategy. A market in scope with zero gap and no pending work completes at once as a no-op and never reopens for that receipt. The executor sizes each remaining market with the formula of §4.1:

- **Same direction, larger:** open δ_m of new exposure on fresh execution accounts.
- **Same direction, smaller:** reduce existing execution accounts by $|\delta_m|$, closing each account entirely when the reduction covers it; a partial reduction keeps the account and its entry price.
- **Direction flip:** close the existing exposure, then open n_m^* on a fresh execution account.

Reductions precede openings, so collateral is released and gross exposure falls first. Openings are split into slices of at most the configured slice size (250,000 USDC in the protocol defaults), each with collateral $\lceil n\rho/10^4 \rceil$ at the common collateral ratio ρ . Just before it is sent, a slice is resized to $\min(n_{\text{slice}}, |\delta_m|)$; if δ_m has reached zero or reversed, it completes unsent.

Continuing the example of §4.1, $\delta_{\text{SOL}} = +30,000$ USDC is one opening slice, carrying 10,002 USDC of collateral at the local development ratio of 3,334 bps. Had the signal been `SOL short`

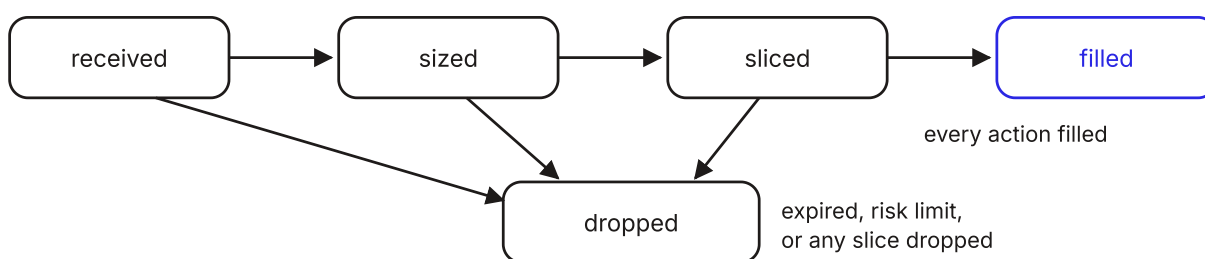
10000, the executor would close the 15,000 USDC long and then open a 30,000 USDC short on a fresh execution account.

A later receipt's action never overtakes an unfinished action in the same market from an earlier admissible receipt (§5 The Sealed Executor).

4.9 Lifecycle states and reasons

A signal moves through five states and each of its slices through four (Figure 4.2): `received` (admitted, not yet sized), `sized` (actions computed from current equity and positions), `sliced` (actions being dispatched), and the terminal `filled` or `dropped`.

Signal



Slice

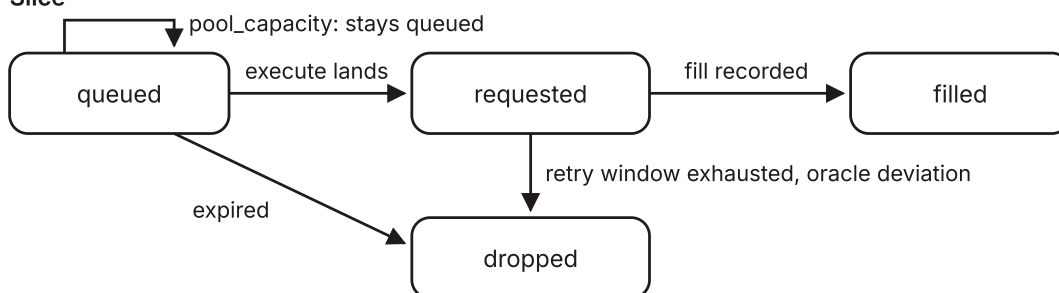


Figure 4.2 — Signal and slice states. A signal is terminal when every action is terminal: filled if all filled, dropped if any was dropped. A slice waiting for pool capacity stays queued with a reason until capacity returns or the signal expires.

Reason	Applies to	Cause
<code>expired</code>	Signal or slice	<code>validUntilSlot</code> passed before the action was submitted
<code>oracle_deviation</code>	Slice and its market	Oracle moved beyond the leader's bound while pending; request cancelled and refunded
<code>retry_window_exhausted</code>	Slice	Not filled before the retry deadline; dropped and refunded
<code>risk_limit</code>	Signal or slice	A risk bound of the strategy stopped the action
<code>pool_capacity</code>	Queued slice, not terminal	Does not yet fit the pool's common envelope; retried unchanged as capacity returns

Table 4.2 — Reasons reported in the leader console.

The executor's journal records three finer causes: `drawdown_limit` when the strategy has reached its declared drawdown kill level and its unfinished actions are stopped, `target_reached` when an opening slice becomes unnecessary and completes as filled, and `request_refunded` when an unfilled request is cancelled with its principal returned to the pool.

4.10 Retries

Submission. Without an acknowledgment, the client resends the identical signed signal, with the same sequence, targets, `validUntilSlot` and deviation bound, and therefore the same bytes. The executor recognizes the (strategy, sequence) pair and returns the original acknowledgment. The replay is not charged against the key's rate, and it completes the verifier's admission of the stored record if that had not finished; no intent ever receives a second receipt. A client must not turn an uncertain response into a new signal with a higher sequence, which would be a second order. The CLI prints the `validUntilSlot` it signed so that a retry reproduces the bytes.

Response	HTTP	Cause	Client action
Acknowledgment	200	New signal admitted, or exact replay of an admitted one	None; read progress through the leader console
INVALID_SIGNATURE	401	Signature fails verification	Sign the same bytes with the registered key
FORBIDDEN	403	Signal key not registered for the strategy	Register or rotate the key, then sign a new signal
REPLAY_DETECTED	409	Sequence reused with different bytes, or not above an earlier sequence	Use a higher sequence only for a deliberately new intent
RATE_LIMITED	429	Per-key or relay limit reached	Resend the identical signed signal later
VALIDATION_ERROR	400	Expired, bounds, markets, leverage, deployment, paused or at drawdown limit	Correct the cause and sign a new signal
SERVICE_UNAVAILABLE	503	Attestation expired, chain snapshot or sealed store unavailable	Resend the identical signed signal

Table 4.3 — Admission responses and the client action for each.

Venue. An unfilled funded request is retried until the execution's authenticated creation slot plus the retry window N , a configured parameter stored as `Config.retry_slots` (300 slots in the local development profile). It is then dropped, its collateral returns to the pool and the slice ends with `retry_window_exhausted`. A keeper fill that lands before the cancellation is recorded instead, through a dual-signed reconciliation that requires the original admission, commitment, policy hash and venue fill.

Capacity. A slice that does not fit the pool's common envelope stays queued with `pool_capacity` at its exact signed size, keeps its place in the market's receipt order, and is dropped as `expired` if its signal's validity ends first.

Key rotation. After a wallet-signed `rotate_signal_key`, signals must use the new key. Old-key work that has not landed is retired, and a never-filled request is refunded through the principal-only cancellation path even after its signal expired. Filled openings keep their original signature, receipt and commitment and stay valid for NAV and public replay.

4.11 Reading the lifecycle

A leader reads current signals, slices, reasons, positions, fills and fees only through a leader-view proof: canonical JSON with the domain `unseen-leader-view-v2` binding the deployment fields, the strategy, the leader wallet, a random 32-byte nonce, `issuedAt` and an `expiresAt` at most 60 seconds later. The leader wallet signs it, not the signal key. The leader posts the proof to the API's `POST /v1/leader/strategies/:strategyId/console`, which forwards it unchanged to the executor, and the executor verifies the signature, the deployment, on-chain leadership of the strategy and the single use of the nonce. A revealed signed signal therefore grants no access to the strategy's later activity (§15 Interfaces).

5. The Sealed Executor

The Sealed Executor is the component that decrypts signals and keeps the strategy-to-account mapping. Before reveal, plaintext signals and the mapping exist only inside it and, for verification, at the independent verifier. It is a Rust service running inside an Intel TDX confidential VM. It decrypts sealed signals, signs receipts, sizes one pooled order per signal, slices it, orders it against every other strategy's work and submits the transactions that move collateral from the pool into fresh exec accounts. It also proposes each strategy's daily NAV (§8 Epochs and settlement) and posts openings once their reveal conditions hold (§10 Reveal).

Its authority is narrow by construction. The program moves funds for it only along pool → exec → approved venue and back to the fixed pool vault (§12 On-chain programs). Every execute, fill record, reduction, close and stats transaction also carries the independent verifier's signature, and the verifier signs only after replaying the policy from the leader's signed signal (§5.5). The executor holds no follower key and no admin key, and no instruction lets it withdraw.

5.1 The confidential VM

The production executor runs as a container image on a cloud confidential VM with Intel TDX, such as a GCP C3 confidential VM or an Azure DCesv5 instance. TDX measures the VM's initial image into MRTD, a 48-byte register, and later boot and runtime components into the RTMRs. Unseen pins MRTD directly and pins the measured OS and executor workload, RTMRs included, through Intel Trust Authority (ITA) policies.

The executor generates its Ed25519 enclave signing key inside the trust domain on first start and keeps it on the VM's encrypted volume; the secret never leaves the VM. This key, registered on-chain as the `EnclaveKey`, signs receipts, transactions, proposals, stats and reveals. The P-256 key that opens sealed signals is derived from it under the domain `unseen-sealed-signal-key-v1`, so no second secret exists. A symmetric key sealed to the VM encrypts the private store `state.enc` (signals, receipts, openings, policy histories, flow reservations, job state) with XChaCha20-Poly1305. Each save is anchored to a monotonic counter kept by the verifier, whose signed replies bind the ciphertext hash, so an older ciphertext cannot be restored. The public database never receives the strategy-to-account mapping.

5.2 Quote and key binding

The attestation has to prove that the keys a client is about to use belong to the measured executor of this deployment. The executor writes a binding string, lines separated by LF:

```
unseen-tdx-runtime-v1
<genesis hash>
<unseen program ID>
<executor Ed25519 public key, base58>
<P-256 encryption public key, base64 SEC1 uncompressed>
```

Its SHA-512 digest, exactly 64 bytes, becomes the TDX report data. The executor obtains a quote through the Linux configs-tsm interface under `/sys/kernel/config/tsm/report`: it requires the provider `tdx_guest`, writes the report data to `inblob` and reads the quote from `outblob`, accepting it only if the entry's generation advanced by exactly one. The quote carries MRTD, the RTMRs and the report data under the platform's signature. A relay that substitutes its own encryption key changes the digest and no longer matches the quote.

The executor posts the quote and the binding string as runtime data to ITA with its API key, naming the configured policy IDs, requiring every policy to match and asking for a PS384-signed token. Before publishing the token it verifies it itself, with the same pins a client uses (§5.3) plus a freshness bound. It then publishes the token, both public keys, the MRTD, the report data and the token's issue and expiry times as `Deployment.executor.attestation` on the public route `GET /v1/deployment`, next to the cluster, program IDs and USDC mint, and refreshes the short-lived token as it ages.

On chain the enclave key enters the registry only through the admin. The Squads multisig calls `schedule_enclave(key, expiry, measurement_hash, report_hash)`; the program requires a key different from the current one, nonzero 32-byte hashes, and an expiry more than 172,800 seconds ahead. After the 48-hour timelock anyone may call `activate_enclave`, which creates `EnclaveKey` at `["enclave", key]` and replaces `Config.enclave`. The guardian revokes instantly with `set_pause(paused, revoke_enclave = true)`. Every enclave-signed instruction checks that the signer equals both `EnclaveKey.public_key` and `Config.enclave`, that neither record is revoked, and that the logical clock is before both expiries. The key lifetime is a configured deployment parameter.

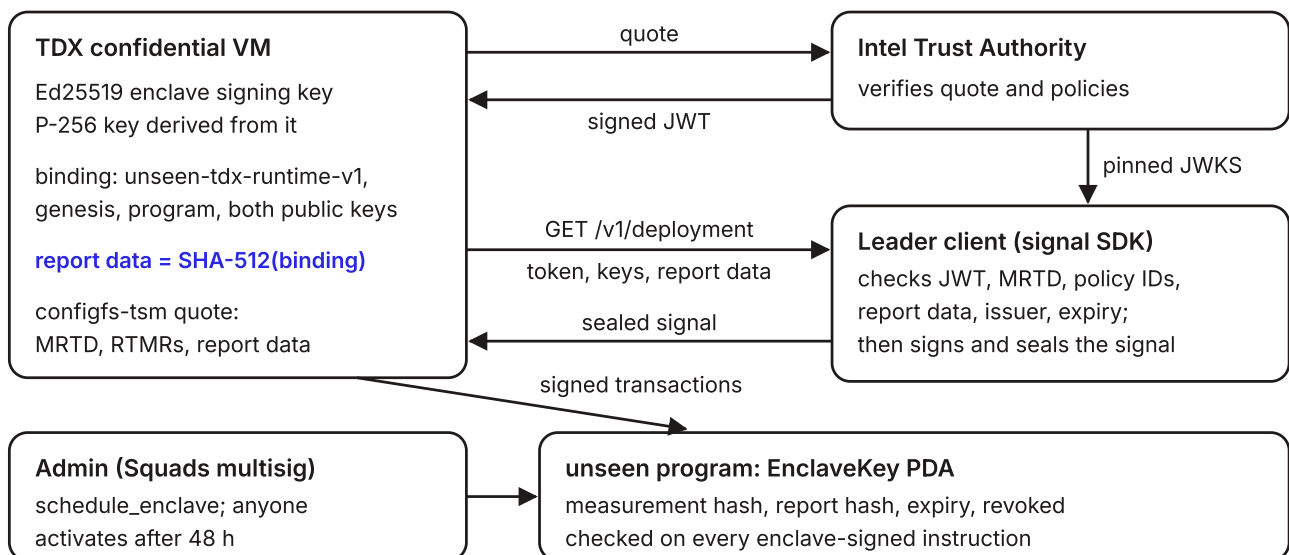


Figure 5.1 — Key binding and attestation. Both executor public keys enter the report data of the TDX quote; ITA turns the quote into a signed token that clients check against pinned values; the admin registers the same signing key on-chain behind a 48-hour timelock.

5.3 How clients check the attestation

The signal SDK runs on Web Crypto alone, in browsers and in the CLI. Its trust policy is configured independently of the executor: the deployment pins (genesis hash, program ID, executor public key), the ITA issuer and JWKS URL, the expected MRTD and the expected ITA policy IDs. A key URL carried inside the token is never followed. Before the signal key signs anything, the SDK applies every check in Table 5.1 and stops on the first failure.

Check	Rule
Identity	<code>mode = intel_tdx</code> , <code>provider = intel-trust-authority</code> , <code>publicKey</code> equals the pinned executor key, <code>measurement</code> equals the pinned MRTD
Encryption key	65 bytes, first byte <code>0x04</code> (uncompressed P-256 point)
Key binding	<code>reportData</code> equals SHA-512 of the binding string built from the pinned genesis, program and executor key and the published encryption key
Token form	At most 65,536 characters, three sections; header <code>alg</code> is PS384 or RS256, <code>typ</code> is JWT, <code>kid</code> is a string, no <code>crit</code>
Token signature	Exactly one RSA key with that <code>kid</code> in the pinned JWKS, fetched without cache, credentials or redirects, verifies the signature
Issuer and time	<code>iss</code> equals the pinned issuer; <code>iat</code> and <code>nbf</code> at most 5 s in the future; <code>exp</code> in the future and after <code>iat</code> ; <code>issuedAt</code> / <code>expiresAt</code> equal <code>iat</code> / <code>exp</code>
TDX claims	<code>attester_type = TDX</code> , <code>attester_tcb_status = UpToDate</code> , <code>tdx_is_debuggable = false</code> , <code>tdx_mrtid</code> equals the pinned MRTD, <code>tdx_report_data</code> equals <code>reportData</code>
Policies	Every expected policy ID appears in <code>policy_ids_matched</code> ; <code>policy_ids_unmatched</code> is empty

Table 5.1 — Attestation checks run by the signal SDK before every signature.

Once the attestation passes, the SDK signs the canonical V2 signal bytes (§4 Signals), verifies the signature, and seals the signed signal: an ephemeral P-256 key agrees a secret with `encryptionPublicKey` by ECDH, HKDF-SHA256 (salt = SHA-256 of the associated data, info `unseen-sealed-signal-v1`) derives an AES-256-GCM key, and the JSON signal is encrypted under a random 12-byte nonce with the associated data `unseen-sealed-signal-v1`, genesis hash, program ID and executor key, joined by LF.

The envelope `{version, ephemeralPublicKey, nonce, ciphertext}` names no wallet, strategy, market, target or sequence. The SDK posts it to the executor's HTTPS endpoint at `/v1/signals/sealed`, or, when a browser cannot reach the VM, to the API at the same path, which

relays it unchanged. Whichever TLS connection carries the envelope, only the VM that produced the quote holds the key that opens it; this is the attested channel. The SDK accepts the response only if its receipt repeats the signal's deployment fields, its `signedSignalHash` equals SHA-256 of the signing bytes followed by the 64-byte signature, and the receipt signature verifies under the pinned executor key. Attestation is re-checked before every signature, so an expired token stops signing.

5.4 Admission and receipts

After decryption the executor applies the admission checks of Table 5.2, then assigns a receipt. The receipt sequence is one more than the highest sequence it has ever issued, across all strategies. The receipt time is in Unix microseconds and is forced to be strictly increasing: $t_n = \max(\text{clock}, t_{n-1} + 1)$. The enclave key signs

```
unseen-receipt-v2
<genesis hash>
<program ID>
<executor public key>
<receipt sequence>
<received at, Unix microseconds>
<signed signal hash>
```

with LF separators and no trailing newline. The record is saved to the sealed store and sent to the verifier, which checks both signatures and the registered signal key, requires the receipt sequence to be exactly one more than the last it admitted with a strictly later time, and freezes the markets the receipt affects. Only then does the executor acknowledge, with an immutable `received` acknowledgement carrying the receipt (§4 Signals). Receipt sequence and time are inside every V2 commitment (§6 Exec accounts and commitments), so once openings are public anyone can check that executions followed receipt order.

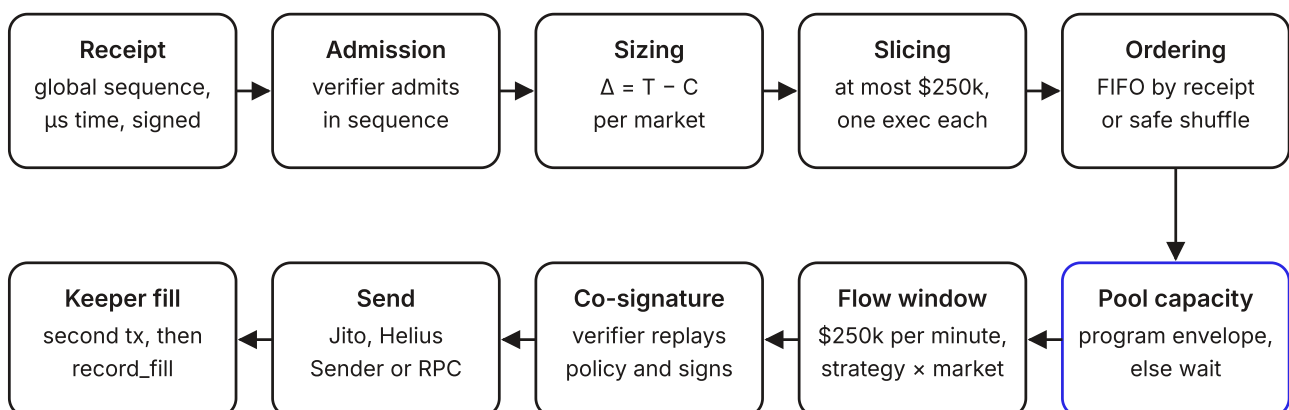


Figure 5.2 — The path of one receipt. Sizing and slicing run once per receipt; capacity, flow and co-signature are checked again for every slice before it is sent.

5.5 The risk engine

The risk engine is the set of rules that every signal and every slice must pass. The executor applies them first. The verifier applies them again, independently, from the signed signal, the authenticated openings and its own RPC reads before it co-signs a transaction. The program enforces the parts that need no attribution. Table 5.2 lists where each rule runs.

Rule	Condition	Enforced by
Signature and binding	V2 bytes signed by the registered signal key; genesis, program and executor key equal the live deployment	Executor, verifier
Sequence	Greater than any earlier sequence of the strategy; a repeated sequence only with identical content	Executor
Validity	<code>valid_until_slot</code> not yet reached, at admission and at every action	Executor, verifier, program
Targets	1 to 3 distinct markets among SOL, ETH, wBTC, each allowed by the strategy	Executor, verifier, program (every active strategy's mask)
Leverage	$\sum_m w_m \leq \min(\text{declared max}, 30,000)$ bps	Executor, verifier
Drawdown	Below the declared kill level, itself at most the protocol maximum	Executor, verifier
Strategy state	Not paused; no proposal of this strategy awaiting finalization	Executor, verifier, program (pause)
Slice size	$0 < n \leq 250,000$ USDC; collateral $c = \lceil nr/10^4 \rceil \geq 10$ USDC	Executor, verifier, program
Direction	An opening has the sign of Δ and $n \leq \Delta $	Verifier
Own cash	Liquid cash covers reserve, fees, new margin and every landable authorization	Verifier
Oracle deviation	Within the leader's bound of the request's reference price, else the unfilled request is cancelled	Executor, verifier
Pool envelope	Open notional and locked collateral within every active strategy's bound (§5.8)	Executor (pre-check), program
Flow	Per strategy and market per window; per pool and market per minute	Executor, program
Execution fees	Per logical day at most $\text{platform AUM} \times \text{cap bps} / 10^4$	Program

Table 5.2 — Risk rules and where they are enforced.

Drawdown. With E the recomputed NAV and S the share supply, the NAV per share is $p = \lfloor E \cdot 10^9 / S \rfloor$. The reference peak P is the largest of the genesis NAV, the high-water mark, the last final NAV per share and the highest value the executor has observed. Drawdown never decreases:

$$\text{dd} = \max\left(\text{dd}_{\text{recorded}}, \left\lfloor \frac{(P - p) \cdot 10^4}{P} \right\rfloor\right)$$

When dd reaches the strategy's declared `drawdown_kill_bps`, new signals are refused, queued work is dropped with reason `drawdown_limit`, unfilled requests are cancelled and filled positions are closed. The declared level is bounded on-chain by `Config.max_drawdown_bps` (configured; local development value 2,000 bps).

Own cash. Pooled idle cash from another strategy never funds an order. The verifier recomputes the strategy's liquid cash L , which is its NAV less the collateral of its pending and filled positions and less their unrealized PnL, and requires

$$L \geq \left\lfloor \frac{E}{5} \right\rfloor + F_{\text{perf}} + \sum_{j \in A} \left(c_j + \left\lfloor \frac{2n_j}{10^4} \right\rfloor + \phi(n_j) \right) + \sum_{k \in P} \left(\left\lfloor \frac{2n_k}{10^4} \right\rfloor + \phi(n_k) \right)$$

where $\lfloor E/5 \rfloor$ is the next epoch's 20% withdrawal reserve, F_{perf} the prospective performance fee, A the candidate plus every authorization that can still land without an existing exec (deduplicated by exec PDA, across all markets and sides), P the strategy's funded pending requests whose principal is already deducted from L , and ϕ the venue opening-fee bound of the production adapter (the development venue uses $\lfloor 6n/10^4 \rfloor$).

Forced reductions. Three conditions authorize a reduction without a new leader signal: drawdown at the kill level, cash below the settlement reserve (`withdrawal_liquidity`) and leverage above the declared maximum after pending withdrawals (`withdrawal_leverage`). The verifier then signs an `unseen-risk-reduction-v2` certificate binding the original signal hash and receipt, the exec and its next operation sequence, the reduced notional, the reason, an expiry within the retry window and the recomputed NAV, cash and drawdown. Its hash becomes the reduction's policy hash (§6). A certificate can only reduce or close an existing filled position.

5.6 Sizing

A signal states target weights in basis points of strategy equity; a weight already includes leverage, so 15,000 bps means a position worth 1.5 times equity. For market m the executor computes, with E the strategy NAV recomputed from chain data and openings, w_m signed (shorts negative), and n_x the remaining notional of each filled exec x of the strategy in that market:

$$T_m = \text{trunc}\left(\frac{E \cdot w_m}{10^4}\right), \quad C_m = \sum_x \sigma_x n_x, \quad \Delta_m = T_m - C_m$$

where $\sigma_x = +1$ for a long and -1 for a short. Three cases follow.

1. **Increase in the same direction** ($C_m = 0$, or T_m and C_m share a sign with $|T_m| > |C_m|$): open $|\Delta_m|$ on the side of Δ_m .
2. **Reduction in the same direction** ($|T_m| < |C_m|$, same sign): reduce $|C_m| - |T_m|$, walking the existing execs, with a partial close where the reduction is smaller than an exec's remaining notional and a full close otherwise.
3. **Sign flip**: close all of $|C_m|$, then open $|T_m|$ on the new side from fresh exec accounts.

All reductions and closes of a signal are ordered before its openings, so collateral is released before new margin is posted. Each opening is cut into slices of at most 250,000 USDC (\$5.7), each with collateral $c = \lceil n \cdot r / 10^4 \rceil$, where r is the common collateral ratio (configured within 3,334–10,000 bps; local development value 3,334 bps). Immediately before a slice is submitted, the executor repeats the computation at the current NAV and position and trims the slice to $\min(n, |\Delta_m|)$; if the target has been reached or the sign has turned, the slice is marked done with reason `target_reached`.

Quantity	Value
Strategy equity E , position	250,000 USDC, flat
Signal	SOL long 15,000 bps
Target T , delta Δ	375,000 USDC, +375,000 USDC
Slices	250,000 and 125,000 USDC, two exec accounts
Collateral at $r = 3,334$ bps	83,350 and 41,675 USDC
Cash needed for both	50,000 reserve + 125,025 margin + 75 execution-fee bound + 225 venue-fee bound = 175,325 USDC

Table 5.3 — Sizing one signal, with the local development collateral ratio and venue fee bound and no performance fee due.

5.7 Slicing and the per-minute flow limit

Slicing bounds how fast the pool trades a market. Three limits apply.

- **Per action.** No exec opens with more than 250,000 USDC notional; the program rejects larger `execute` calls.
- **Per strategy and market.** Before a slice, partial close or close is submitted, the executor reserves its notional in a flow window keyed by strategy and market. Openings and reductions consume the same absolute budget of 250,000 USDC per window. The production window is 60 seconds (local development: 5 seconds). A slice that does not fit waits for the next window.
- **Per pool and market.** The program keeps `flow_minute[m]` and `flow_notional[m]` and charges every `execute`, every `partial_close` and every full `close_exec` (a refund-only cancellation is not charged); gross notional per market per logical minute may not exceed 250,000 USDC. Because the pool counter covers every strategy, each strategy's flow is bounded as well.

A rebalance that moves gross notional G in one market therefore takes at least $\lceil G/250,000 \rceil$ minutes, which is what bounds a strategy's capacity (§7.5).

A slice that has landed as a venue request waits for the keeper. Its on-chain expiry is the creation slot plus `Config.retry_slots`, a configured parameter (local development value 300 slots). If it is still unfilled when the window ends, or when the signal expires, or when the oracle has moved further from the request's reference price than the leader's bound, the executor cancels it through the principal-only refund path, all collateral returns to the pool, and the slice is reported as dropped with its reason.

5.8 Pool capacity before dispatch

The program cannot know which strategy owns a new exec, so it checks each risk increase against every possible owner. Every `execute` passes the complete strategy registry, in registry order. For each active strategy s the program computes a conservative equity

$$\hat{E}_s = \max(0, \text{AUM}_s - V_s^{\text{wd}} - (R - R_s) - \Lambda - \Phi)$$

where V_s^{wd} is the value of its pending withdrawals at the last final NAV, $R - R_s$ the pool's realized losses and execution fees since the strategy's last final checkpoint, Λ an upper bound on the sum of all unrealized losses in the pool, and $\Phi = \lceil 16 \cdot N/10^4 \rceil$ a fee bound on aggregate notional N including the candidate. Λ uses per-market long and short notional and the worst entry extremes, widened by the common wire slippage at entry, and never nets one position's profit against another's loss. The candidate passes only if

$$N_{\text{open}} + n \leq \min_s \left\lceil \frac{\hat{E}_s \cdot \text{lev}_s}{10^4} \right\rceil \quad \text{and} \quad K_{\text{locked}} + c \leq \min_s (\hat{E}_s - \text{reserve}_s)$$

with K_{locked} the original collateral of all pending and filled requests and reserve_s the strategy's protected withdrawal reserve. The market must be allowed by every active strategy, and pool cash net of protected reserves must cover c .

The executor evaluates the same envelope before dispatching an opening, adding the pending requests' fee reservations, the remaining minute flow and the remaining daily fee budget. An opening that does not fit is not clipped: it keeps its size with reason `pool_capacity` and resumes when capacity returns, or is dropped when its signal expires. Reductions and funded requests never wait.

Registry	SOL long slice	$\min_s \lfloor \hat{E}_s \text{lev}_s / 10^4 \rfloor$	$\min_s (\hat{E}_s - \text{reserve}_s)$ vs collateral	Result
A: 250,000 USDC, 3×, reserve 50,000	250,000	741,374.26	197,124.75 vs 83,350	Dispatched
A as above, plus B: 100,000 USDC, 2×, reserve 20,000	250,000	194,249.50	77,124.75 vs 83,350	Waits (<code>pool_capacity</code>)

Table 5.4 — The shared envelope for an empty pool, with the local development SOL price of 100 USD, wire slippage of 100 bps and collateral ratio of 3,334 bps. $\Lambda = 2,475.25$ and $\Phi = 400$ USDC in both rows.

Capacity is deliberately shared at the bound of the smallest active strategy. A strategy with zero equity would block all new risk, so `deactivate_zero_strategy` lets anyone retire one that has zero AUM, no active epoch, no pending cash flows and a flat pool; it freezes `active_until_sequence` at the current commitment count.

5.9 FIFO ordering and batching

Priority is the global receipt sequence. A receipt's next action may be dispatched only if no earlier receipt still has an unfinished action in the same market. An earlier receipt with several markets reserves all of them, including markets behind its current head, and an earlier receipt that has not been sized yet reserves every market in its targets. Receipts of a paused strategy, or of a strategy whose proposal is awaiting finalization, reserve nothing, so one strategy's frozen state does not block another strategy that shares its market.

Order within FIFO is relaxed in one case. When two or three fresh opening heads in distinct markets are ready together, the executor computes the pool resources at the final state of the whole batch: cash after collateral and nominal fees, open-interest ceiling, minute flow per market and daily execution-fee budget. Every reservation is non-negative, so if the final state fits, every prefix of every permutation fits too, and the executor shuffles the batch with a random permutation before dispatch. If any input is scarce, missing or malformed, or if a reduction is involved, receipt order is kept. Reductions get no credit for cash they will release until they have landed.

A batch, for fee purposes, is the set of actions from receipts received in the same chain slot. Its identifier is SHA-256 of `unseen-batch-v2`, genesis hash, executor key and receipt slot, joined by

LF. A batch price-impact fee I over total notional N is split by notional with a prefix-floor rule that conserves the exact total and gives the remainder to members in receipt order:

$$I_i = \left\lfloor \frac{I \cdot (N_{<i} + n_i)}{N} \right\rfloor - \left\lfloor \frac{I \cdot N_{<i}}{N} \right\rfloor$$

where $N_{<i}$ is the notional of the members before i . A fee of 100 over members of 2, 3 and 5 gives 20, 30 and 50; a fee of 2 over three equal members gives 0, 1 and 1. Each member's policy evidence records the batch and its own allocation (§6).

5.10 Signing and sending

An unsigned-transaction bridge outside the VM assembles transactions and holds no role key. The executor and the verifier each decode the prepared bytes and rebuild the expected instruction data, PDAs, signers and deployment from their own RPC reads; the verifier also re-runs the policy, records the authorization in its encrypted journal and returns a partial signature, and the enclave key adds its own. Priority settings are in the message before either role signs: a compute-unit limit (at most 1,400,000), a compute-unit price and, for Jito or Helius, a tip transfer to a published tip account. Sending never changes bytes. The sender first simulates with signature verification on its own HTTPS RPC, because the bundle path skips preflight.

TX_SENDER	Submission	Tip	Notes
rpc	<code>sendRawTransaction</code> , preflight on, no RPC-side retries	Must be 0	The only mode on devnet
jito	<code>sendBundle</code> to the block engine, with <code>x-jito-auth</code> when configured	At least 1,000 lamports to <code>TIP_ACCOUNT</code>	A bundle receipt is not confirmation; status is reconciled on the RPC
helius	<code>sendTransaction</code> to Helius Sender, preflight skipped, no retries	At least 5,000 lamports to <code>TIP_ACCOUNT</code>	The returned signature must equal the transaction's first signature

Table 5.5 — Sender modes. Compute-unit limit, price and tip are configured values.

Each request times out after 15 seconds, and a retry resends the identical signed bytes, so an ambiguous outcome never becomes a second economic action. The opening `execute` and the keeper's fill are separate transactions: `execute` creates the venue request, the keeper fills it at the oracle price, and the executor and verifier then record the fill with `record_fill`, which charges the 2 bps execution fee (§7 Venue: Jupiter Perps). No NAV can be proposed for a strategy while one of its landed fills is unrecorded.

6. Exec accounts and commitments

Every position Unseen opens lives in its own program-derived account, funded from the shared pool vault and addressed by a random 64-bit index. The account shows the trade, which is public by necessity, and a 32-byte commitment to its owner, which stays sealed until reveal.

Commitments are appended to a single pool-wide hash chain, so the order and number of executions are fixed on-chain at the moment they happen. This chapter specifies the accounts, the two commitment encodings, the chain, how a position is reduced without changing account, and how accounts are closed and their rent recovered. The reveal conditions themselves are in §10 Reveal.

6.1 Accounts created per position

One `execute` instruction creates the accounts of Table 6.1. Every address is derived from fixed seeds; the only free input is the index.

Account	Seeds and owner	Purpose	Lifetime
Exec	<code>["exec", pool, index_u64_le]</code> , unseen program	Public record of exactly one position	Until <code>reclaim_exec</code> , then a 40-byte tombstone forever
Exec vault	<code>["exec_vault", exec]</code> , SPL token account in USDC, authority = exec PDA	Holds collateral in transit between pool and venue	Closed at <code>reclaim_exec</code>
Venue accounts	Owned by the venue program, with the exec PDA as position owner; the development venue uses one request account at <code>["request", exec]</code>	The order and, once filled, the position	Closed once flat and reclaimed
CommitmentReceipt	<code>["commitment", commitment]</code> , unseen program	Makes each commitment unique and records its creation clock	Permanent; holds the execution archive after cleanup

Table 6.1 — Accounts created by one `execute`.

The pool itself is the PDA `["pool", venue_program]`, and its USDC vault is `["pool_vault"]` with the pool as token authority. The exec PDA signs for the venue request and for its own vault with the

seeds `["exec", pool, index_u64_le, bump]`; no key exists for it. The enclave key pays the rent of the accounts created in `execute`. Rent recovered at cleanup goes to the pool PDA (§6.8).

6.2 The random 64-bit index

The index is the only variable seed of an exec address. For each new position the executor draws a 64-bit integer from the operating system's random source and rejects it if an execution with that index already appears in its chain snapshot or in its private positions, drawing up to 32 times. The program provides the hard guarantee: `Exec` is created with Anchor `init`, which fails if the address already holds an account, and a reclaimed exec is never deleted but shrunk to a tombstone (§6.8), so an index can never be used twice, even after cleanup.

A random index carries no information. There is no per-strategy counter, so the address of an exec reveals neither which strategy opened it, nor how many positions that strategy has opened, nor in what order. The exec account stores no strategy ID and no signal sequence; the index space has 2^{64} values, shared by every strategy in the pool.

6.3 What an exec account shows

`Exec` is a fixed-size Anchor account of 341 bytes: an 8-byte discriminator and 333 bytes of fields.

Group	Fields
Identity	<code>pool</code> , <code>index</code> , <code>executor</code> (enclave key that created it), <code>bump</code>
Commitment	<code>commitment</code> , <code>previous_head</code> , <code>resulting_head</code> , <code>chain_sequence</code>
Position	<code>market</code> (0 SOL, 1 ETH, 2 wBTC), <code>side</code> (0 long, 1 short), <code>notional</code> , <code>collateral</code> , <code>original_notional</code> , <code>original_collateral</code>
Fees	<code>execution_fee</code> , <code>opening_execution_fee</code>
Policy	<code>policy_evidence_hash</code> , <code>last_policy_hash</code> , <code>operation_sequence</code>
Lifecycle	<code>state</code> (0 pending, 1 filled, 2 closed, 3 dropped), <code>created_at</code> , <code>created_slot</code> , <code>filled_at</code> , <code>closed_at</code> , <code>revealed</code>

Table 6.2 — Fields of `Exec`.

Every public value is a function of the pooled order and of deployment-wide parameters, never of a strategy's own parameters. Collateral is $\lceil n \cdot r / 10^4 \rceil$ with one common ratio r for the whole pool, and the venue request carries one common wire slippage bound. A leader's declared leverage and private oracle-deviation limit are enforced off-chain by the executor and the verifier (§5.5) and never select an on-chain field. The policy hashes are 32-byte digests whose preimages are sealed with the opening.

6.4 Commitment V1

The original encoding, `unseen-sha256-v1`, commits to the four values named in the product brief. Its preimage is exactly 80 bytes; integers are little-endian.

Offset	Bytes	Field
0	32	<code>strategy_id</code>
32	8	<code>exec_index</code> , u64
40	8	<code>signal_seq</code> , u64
48	32	<code>salt</code>

Table 6.3 — V1 opening, 80 bytes; commitment = SHA-256(preimage).

The salt is 32 bytes from the operating system's random source, fresh for every exec. It is what makes the commitment hiding: strategy IDs are public, the exec index is visible in the account, and signal sequences are small integers, so without 256 bits of salt an observer could test every strategy against every commitment. V1 remains supported for legacy contract fixtures and is opened by the `reveal` instruction, which recomputes the hash on-chain. New live executions use V2.

6.5 Commitment V2

V2 commits to the deployment, the executor, the leader's signed signal, the executor's signed receipt and the policy evidence that sized the trade. Its preimage is 273 bytes.

Offset	Bytes	Field
0	17	"unseen-opening-v2" , UTF-8, no terminator
17	32	Genesis hash of the cluster
49	32	Unseen program ID
81	32	Executor (enclave) public key
113	32	strategy_id
145	8	exec_index , u64 LE
153	8	signal_sequence , u64 LE
161	8	global_receipt_sequence , u64 LE
169	8	received_at_unix_micros , u64 LE
177	32	signed_signal_hash
209	32	salt
241	32	policy_evidence_hash

Table 6.4 — V2 opening, 273 bytes; commitment = SHA-256(preimage).

The two nested hashes extend the commitment to the full evidence without enlarging it.

- **Signed signal hash:** SHA-256 of the canonical V2 signal signing bytes (§4 Signals) followed by the raw 64-byte Ed25519 signature. It binds the targets, validity slot, oracle-deviation bound and signal key, and the signature itself.
- **Policy evidence hash:** SHA-256 of a fixed 392-byte binary record, "unseen-policy-v2" (16 bytes) followed by: genesis, program, executor, strategy and signed signal hash (32 bytes each); receipt sequence (u64); batch ID (32); batch index and batch size (u32 each); ten u64 amounts and slots, namely batch total notional, batch impact fee, member notional, original notional, allowed reduction notional, allocated impact, venue and execution fees, sizing slot and sizing equity; prior signed notional (i128); target weight in bps (i64); declared maximum leverage, declared drawdown kill level and reveal delay (u64 each); prior policy hash (32); and the validity slot (u64). The commitment and the policy record are fixed binary layouts; only the signal's signing bytes are canonical JSON.

The receipt fields sit in the preimage, and the published evidence also carries the enclave's signature over them: unseen-receipt-v2 , genesis, program, executor, receipt sequence, receipt time and signed signal hash (§5.4). A revealed V2 opening is therefore checkable without trusting the executor's word on any field: the commitment recomputes from the preimage, the signal

signature verifies under the registered signal key, the receipt signature under the enclave key, and the policy record reproduces the sizing. The shared Rust and TypeScript implementations are pinned to the same vector:

```
genesis = 01×32, program = 02×32, executor = 03×32, strategy = 04×32
exec_index = 18446744073709551615, signal_sequence = 7
receipt_sequence = 42, received_at_unix_micros = 1700000000123456
signed_signal_hash =
6b8c341de4a2a743241b35fe48799a0345f74a223a2b54d55430e78d3c6eb9fd
salt = 05×32
policy_evidence_hash =
91629885f9214898bc0358e8df5fc646fbd5a579fd915f42e4c2ed2e42eafdbc
commitment =
878235a4eb7fff0c354d75ca8ad2bb337682acabe93a7367a9474a9f7cd798ed
```

On-chain, `reveal_v2` takes the opening fields as instruction data, recomputes the commitment with the stored `Config.deployment_genesis`, the program's own ID and the `executor` recorded in the exec at creation, and requires equality with the `CommitmentReceipt`. It also requires the supplied policy hash to equal `Exec.policy_evidence_hash`. Binding the genesis hash, program and executor means an opening from one deployment or executor key cannot be presented as an opening in another.

6.6 The on-chain hash chain

The pool keeps a running head and count. Both start at zero, and every successful `execute` updates them in the same instruction that creates the exec:

$$h_0 = 0^{32}, \quad h_k = \text{SHA-256}(h_{k-1} \parallel c_k), \quad \text{Pool.count} = k$$

where c_k is the commitment of the k -th execution. The exec stores `previous_head` = h_{k-1} , `resulting_head` = h_k and `chain_sequence` = k , and the `ExecutionCommitted` event carries the exec address, index, commitment, both heads, notional, market, side, slot and timestamp. Only `execute` extends the chain; reductions and closes change positions, not commitments.

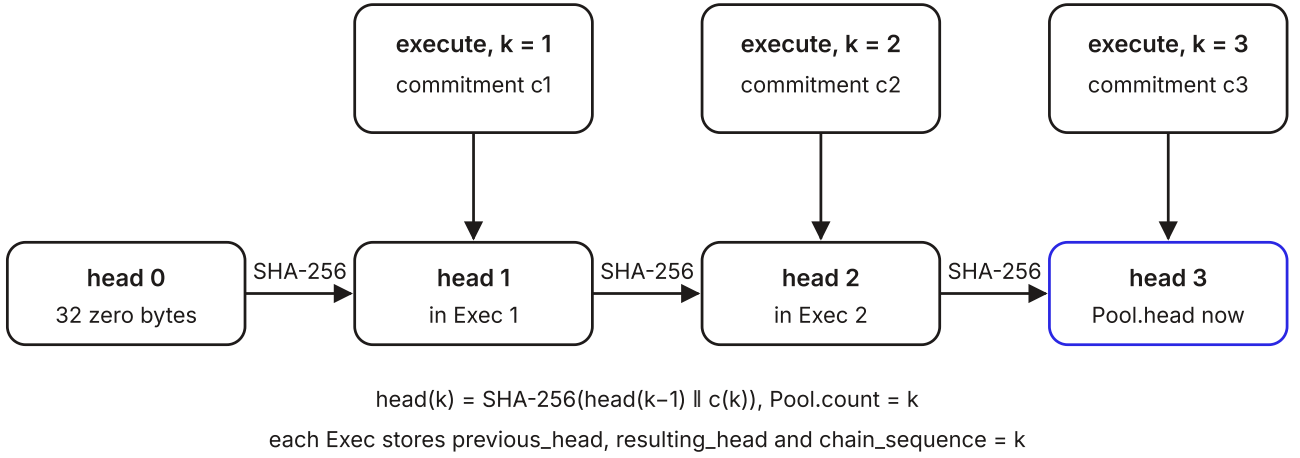


Figure 6.1 — The pool-wide commitment chain. Each execute hashes its commitment into the running head; the exec it creates records the heads before and after.

The chain fixes a total order over all executions of all strategies at the time they land. Any observer can recompute it from the `ExecutionCommitted` events, or from the archives after cleanup, and compare the result with `Pool.head` and `Pool.count`; an omitted, inserted or reordered commitment changes every later head. The public API reports the head and count alongside platform AUM. Two later checks use the sequence number. A dormant strategy's `active_until_sequence` is frozen at the pool count when it is deactivated, and both reveal instructions require `chain_sequence ≤ active_until_sequence` and a strategy created no later than the commitment, so no later execution can be attributed to a dormant strategy and no strategy can claim an execution from before it existed.

6.7 Partial reductions on the same account

A reduction that keeps the position's direction stays on the same exec. The executor and verifier sign `partial_close(reduce_notional, operation_sequence, policy_evidence_hash, valid_until_slot)`, and the program requires a filled exec, $0 < r < n$ for reduced notional r and current notional n , `operation_sequence` equal to the stored value plus one, a nonzero policy hash and the slot no later than `valid_until_slot`. The reduction counts against the pool's per-market minute flow. The venue reduces the position at the oracle price and returns margin plus realized PnL less its closing fee to the exec vault, and the whole amount moves on to the pool vault. With current collateral c :

$$m = \left\lfloor \frac{c \cdot r}{n} \right\rfloor, \quad c' = c - m, \quad n' = n - r, \quad \text{fee} = \left\lfloor \frac{2r}{10^4} \right\rfloor$$

The released margin m leaves `Pool.locked_collateral`; if the returned amount is below m , the shortfall is added to the pool's realized losses; the 2 bps execution fee on the reduced notional goes to the protocol's pull claim. The exec keeps its entry price, and `last_policy_hash` and `operation_sequence` advance. An `ExecutionAdjusted` event records the policy hash, operation

sequence, reduced notional, collateral released, amount returned, realized PnL, venue fee, execution fee, oracle price, slot and time.

Keeping the account means fees are charged only on the change. Closing a 100,000 USDC position and reopening 90,000 USDC would pay venue fees on 190,000 USDC and the execution fee again on 90,000 USDC; a partial close pays them on 10,000 USDC. Each reduction's own signed signal, receipt and policy record become an entry of the opening's `policyHistory`. Entries must have kind `reduce` or `close`, strictly increasing operation sequences, non-decreasing receipt and signal sequences, and a notional equal to their policy record's member notional, and each verifies as its own opening against the same commitment fields. Forced reductions carry a verifier-signed risk certificate in `riskHistory` instead (§5.5). A position whose target changes sign is not reduced through zero: the old exec is closed and the new direction opens from a fresh random index.

6.8 Closure, cancellation and rent

An exec ends in one of two terminal states.

- **Closed (2)** through `close_exec` with `cancel = false`, signed by the enclave and the verifier. The exec must be filled; the full notional counts against the minute flow; everything the venue returns moves to the pool vault; the loss against collateral, if any, enters realized losses; the 2 bps fee on the closed notional goes to the protocol claim; notional and collateral are set to zero and `closed_at` is stamped.
- **Dropped (3)** through `close_exec` with `cancel = true`, the refund-only path for a request that never filled. The venue request must be unfilled with zero fees and zero PnL, the amount returned must be at least the original collateral, no fee is charged, no flow is consumed, and the zero policy hash is allowed because no new leader intent is involved.

After enclave expiry or revocation only the guardian can unwind, with `guardian_close`. Funds still return only to the pool vault, and guardian closes are exempt from execution fees and flow admission so that an exhausted budget cannot hold collateral. Whenever the last open notional in every market reaches zero, the program stamps `Pool.last_flat_at`, which the reveal certificates use (§10).

An opening is revealed only once its exec is closed or dropped and the reveal conditions of §10 hold. After the reveal, anyone may call `reclaim_exec(index)`. It requires a terminal exec, a revealed `CommitmentReceipt` and an empty exec vault, and then:

1. writes an `ExecutionArchive` of 363 bytes into the `CommitmentReceipt`: executor, pool, index, commitment, both heads, chain sequence, market, side, original notional and collateral, execution fees, timestamps and slots, state, both policy hashes, operation sequence, entry price, venue fee, realized PnL and amount returned;
2. closes the flat venue request and the exec vault, sending their rent to the pool PDA;
3. shrinks `Exec` from 341 to 40 bytes and rewrites it as an `ExecutionTombstone`, a distinct 8-byte discriminator followed by the 32-byte commitment, and moves every lamport above the 40-byte rent-exempt minimum to the pool PDA.

The `CommitmentReceipt` is created at its full 454 bytes in `execute`, archive space included, so archiving needs no reallocation, and it is never closed. The tombstone keeps the exec address occupied, so the index cannot be reused, and the receipt keeps the commitment occupied, so it cannot be committed again. Historical NAV reconstruction reads the archives together with the `ExecutionAdjusted` events (§10).

7. Venue: Jupiter Perps

Unseen v0 trades one venue: the SOL, ETH and wBTC perpetual markets of Jupiter Perps, long and short, up to 3×. The choice follows from the product. Jupiter Perps is priced by oracles: Jupiter-run keepers execute position requests at the oracle price, and follower flow does not move that price. A pooled order of a strategy and its followers therefore fills where the leader's own order would have filled, and front-running one's own followers earns nothing. Jupiter publishes its fee and price-impact formulas, which makes the cost and the capacity of a strategy computable in advance.

This chapter describes how an exec account holds a Jupiter position, how fills happen, how collateral is posted on each side, and what the venue charges. Devnet has no Jupiter Perps; the devnet profile deploys the unseen program with the development venue program and a test USDC mint, and runs the same flow end to end. Mainnet uses Jupiter Perps and real USDC.

7.1 Venue binding

The venue is fixed when the protocol is initialized. `initialize` records the venue program in `Config.venue` and creates the pool at `["pool", venue_program]`, so one pool serves one venue. Every instruction that moves a position's funds, namely `execute`, `partial_close`, `close_exec`, `guardian_close` and `reclaim_exec`, requires the venue program account to equal `Config.venue`, and every transfer it performs ends in the pool vault, an exec vault or that venue. The mainnet deployment is configured with the Jupiter Perps program ID, its pool account, the custody accounts of the SOL, ETH and wBTC markets, and a validated per-market oracle configuration. The program's market index maps 0 to SOL, 1 to ETH and 2 to wBTC.

An `execute` that performs the Jupiter CPI needs an address lookup table to fit Solana's 1,232-byte transaction limit; protocol initialization creates the lookup tables for the static accounts. Each `execute` handles one market, which keeps it within the 64-account limit. The compute budget per `execute` is estimated at 400,000 units, against Solana's per-transaction maximum of 1,400,000.

7.2 The request path from an exec PDA

Jupiter Perps changes positions through requests. The position owner creates a request account that states the change, and a keeper executes it in a later transaction. In Unseen the owner of every position is an exec PDA. The PDA has no private key; the unseen program signs for it with the seeds `["exec", pool, index_u64_le, bump]` inside its own instructions, so only those instructions can act on the position.

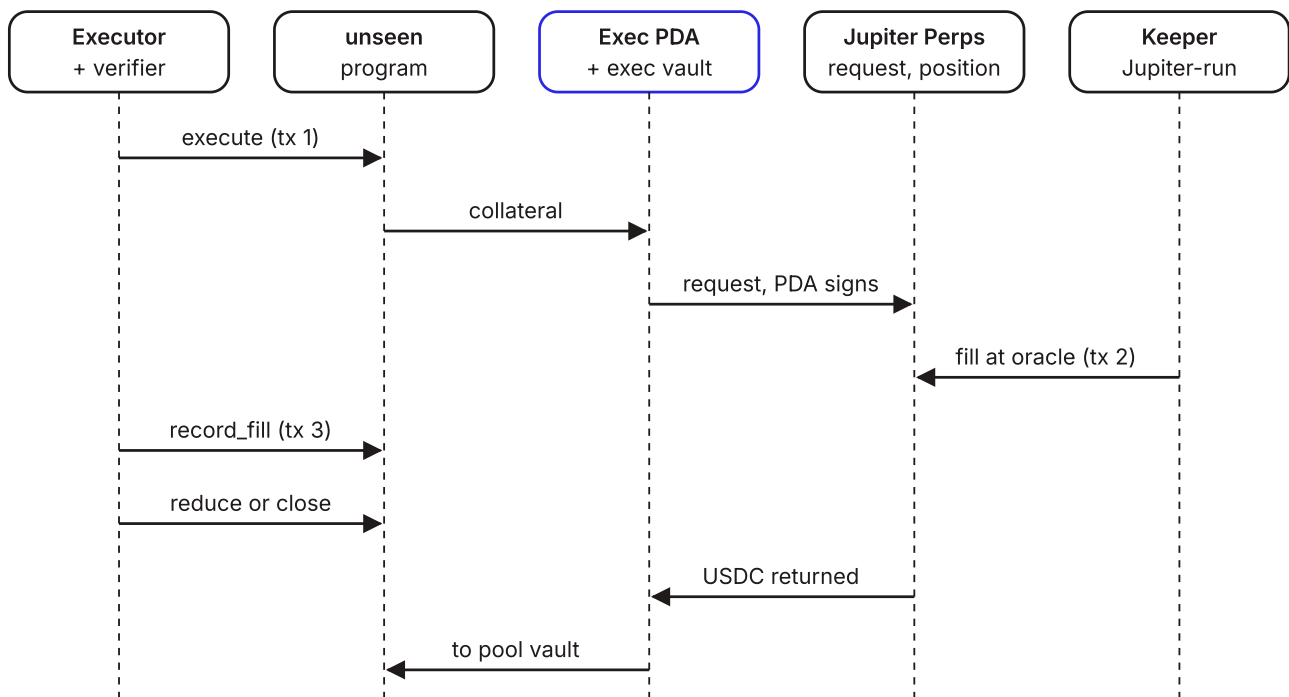


Figure 7.1 — A position's life on the venue. The exec PDA is the position owner; collateral leaves the pool vault only into its exec vault and returns only to the pool vault.

1. **Request.** The executor and verifier sign `execute` (§5). After its risk checks, the program transfers the collateral from the pool vault to the exec vault and, signing as the exec PDA, creates the venue request: market, side, notional, collateral, an expiry of the current slot plus `Config.retry_slots`, and the common wire slippage bound.
2. **Fill.** A keeper executes the request at the oracle price, provided the price is within the request's slippage bound of its reference price and the expiry has not passed. The venue's opening fee is taken from collateral.
3. **Record.** The executor and verifier sign `record_fill`. The program requires the exec to be pending and the request filled, charges the 2 bps execution fee on the notional from pool cash into the protocol fee vault, and marks the exec filled.
4. **Adjust.** `partial_close` and `close_exec` decrease the position through the venue. What the venue returns, margin plus realized PnL less its closing fee, lands in the exec vault and moves on to the pool vault; no other destination is accepted.
5. **Cancel.** A request that has not been filled when its retry window ends, when its signal expires or when the oracle has moved beyond the leader's bound is cancelled through the refund-only path; the venue returns all collateral and the exec becomes dropped (§6.8).

Until the keeper fills it, the request is a public account. It shows what the exec account shows: a market, a side, a size and the common pool parameters, with the exec PDA as owner. Every request in the pool has the same kind of owner and the same collateral ratio and slippage bound, so nothing in it identifies a strategy.

Jupiter Perps allows six positions per owner, one per market and side, and a minimum collateral of 10 USD. Because every position has its own exec PDA as owner, the per-owner limit never applies to the pool, and the program's floor of 10 USDC of collateral per exec equals the venue minimum. The executor never submits an opening below that floor; a residual target smaller than it is left unexecuted and can be certified complete only by the verifier.

7.3 Keepers and the oracle price

Fills come from Jupiter-run keepers at the oracle price, with Edge, Chainlink and Pyth as price sources. The pooled order does not move that price, so the cost of size is the deterministic impact fee of \$7.5, not slippage against a book. Two bounds limit how far the price may move between request and fill.

- **Common wire slippage.** Every request carries `Config.common_wire_slippage_bps` (configured; local development value 100 bps). It is identical for every strategy, so the public request reveals nothing about the leader's preferences.
- **Leader bound.** Each signal carries a signed `maxOracleDeviationBps`, at most 10,000. Before each fill attempt the executor compares the current oracle price with the request's reference price; if the deviation exceeds the leader's bound, it cancels the unfilled request. The verifier checks the same bound independently.

A keeper fill lands on its own, independently of the executor. The executor reconciles it with `record_fill`, or with a dual-signed `reconcile_fill` when the fill raced a cancellation; in both cases the program requires the request to be filled on the venue, and the strategy cannot propose an epoch while a landed fill is unrecorded. NAV valuation uses the oracle prices captured in the epoch proposal (§8 Epochs and settlement).

7.4 Collateral for longs and shorts

All collateral leaves the pool as USDC and is sized by one ratio:

$$c = \left\lceil \frac{n \cdot r}{10^4} \right\rceil, \quad 3,334 \leq r \leq 10,000 \text{ bps}$$

The ratio r is set at initialization (`COMMON_COLLATERAL_RATIO_BPS` ; local development value 3,334 bps, a margin fraction just above one third). At 3,334 bps a position's notional is at most $10,000/3,334 \approx 2.9994$ times its collateral, under the 3× protocol maximum. Since r is the same for every strategy and both sides, no public collateral amount encodes a strategy's declared leverage. Leverage is expressed by the leader's target weights and bounded by the risk engine (§5.5). The program also requires pool cash, after protected withdrawal reserves, to cover the collateral, so withdrawal money is never posted as margin.

Jupiter Perps collateralizes a long position in the market's underlying asset. The pool holds only USDC, so the production adapter's long path converts the exec vault's USDC collateral into the custody asset within the position request, and the decrease path returns USDC to the exec vault. A

short posts USDC collateral directly. Either way the pool's accounting stays in USDC: the program records collateral, returned amounts and realized PnL in USDC base units, and every return lands in the pool vault.

7.5 Venue fees and price impact

Venue fees go to Jupiter and its liquidity providers. Followers pay them in proportion to their shares, as the leader does, because the order is pooled.

Charge	Rate	Notes
Open fee	0.06% of size	Taken from collateral at fill
Close fee	0.06% of size	Taken from the returned amount
Linear price impact	1 bp per started ~\$375k of SOL notional	Deterministic; scales with trade size
Additive impact	Applies once net opened-minus-closed size on an asset within roughly a minute passes \$1.5M (SOL, ETH) or \$750k (wBTC)	Counts all traders on the venue, not only Unseen
Cap	All venue fees combined at most 0.50% of trade size	
Unseen execution fee	2 bps of filled notional	Paid to the protocol, not the venue (\$9 Fees)

Table 7.1 — Charges on a Jupiter Perps trade, from Jupiter's published fee schedule and the protocol's execution fee.

For a SOL trade of size n USD, with Jupiter's step of approximately \$375k, the linear impact charge is

$$\text{impact}(n) = n \cdot \frac{\lceil n/375,000 \rceil}{10^4}$$

so a slice up to about \$375k pays 1 bp, and a larger one 2 bps or more on its whole size. Unseen's per-action limit of 250,000 USDC (\$5.7) keeps every slice inside the first step.

Item	Amount (USDC)	Share of notional
Notional	250,000	
Collateral at $r = 3,334$ bps	83,350	33.34%
Venue open fee, 0.06%	150	0.06%
Linear impact, 1 started block	25	0.01%
Unseen execution fee, 2 bps	50	0.02%
Total charged at open	225	0.09%

Table 7.2 — Opening one 250,000 USDC SOL slice, with the local development collateral ratio. Closing the slice pays the 0.06% close fee and the execution fee again, with impact according to Jupiter's schedule.

When several strategies' actions land in one batch, the batch's impact fee is split by notional with the prefix-floor rule of \$5.9, so each strategy pays for its own share of the size and the total is conserved to the base unit. The verifier's cash check reserves each pending opening's future venue fee, bounded at 6 bps in the development venue and by the production adapter's bound on Jupiter, so a strategy cannot commit more margin than it can pay fees on.

Capacity. Fees and flow limits together bound how much strategy money the venue can carry. Unseen limits each strategy to 250,000 USDC of flow per market per minute and the whole pool to 250,000 USDC of gross flow per market per logical minute, below every one of the venue's additive-fee thresholds. A strategy that rebalances over M minutes supports roughly $250,000 \times M / (|\Delta w| \times \text{leverage})$ of AUM; a full long-to-short flip at 3× over ten minutes gives about \$417k (an estimate). v0 caps each strategy at \$250,000 and the platform at \$2,000,000. Across the three JLP markets, where the additive thresholds are shared with every other trader and correlated rebalancing bursts are what bind, Unseen estimates a ceiling of roughly \$30–60M of strategy AUM at its assumed turnover; Jupiter's per-custody position limits also apply. Growth beyond that requires further venues.

8. Epochs and settlement

Deposits and withdrawals never execute at a price the executor chooses on its own. They settle only at the close of a strategy's daily epoch, at a NAV that the executor proposed, the independent verifier recomputed and co-signed, and no one successfully challenged within the challenge window. Each strategy runs its own sequence of epochs, so a stalled proposal affects only that strategy. This chapter specifies the schedule, the proposal, the verifier's recomputation and co-signature, challenges, finalization, the mint and burn of shares, the 20%-per-epoch withdrawal budget and its queue, pull-based claims, and the paths that retire a proposal that cannot finalize. Figure 8.1 shows the states an epoch passes through.

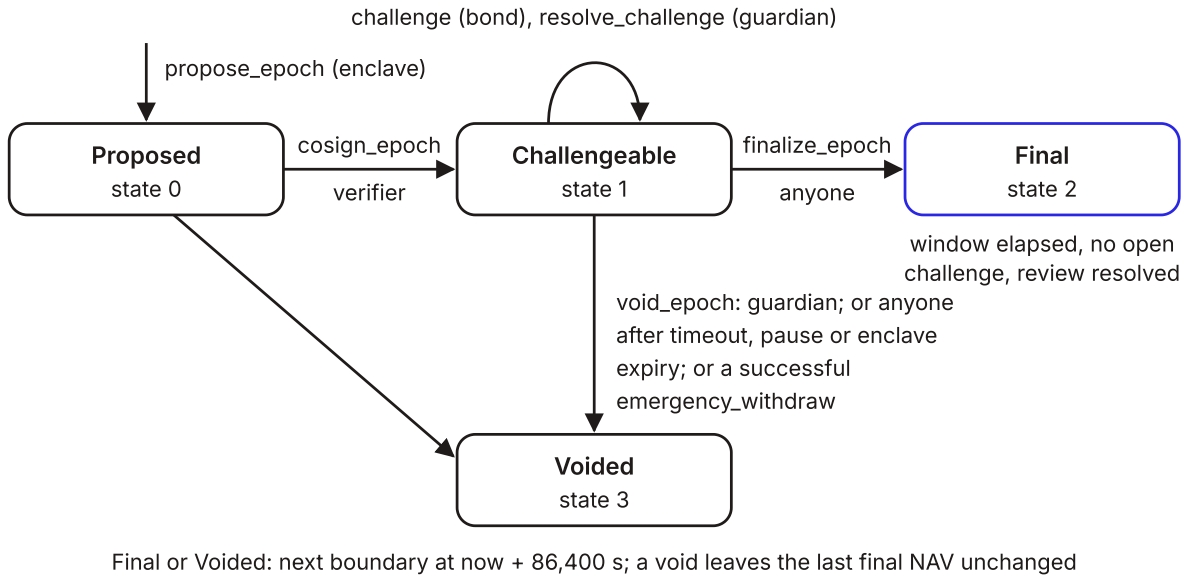


Figure 8.1 — Epoch states. Only a co-signed epoch whose window has elapsed without an open challenge or unresolved review becomes Final; every other path out of a proposal is a void.

8.1 Units and rounding

All amounts are integers. USDC has 6 decimals, so one USDC is 1,000,000 base units. Shares have 9 decimals; one whole share is 10^9 share units. NAV per share π is in USDC base units per whole share, and the genesis value π_0 is a deployment parameter of at least 1,000,000, one USDC per share. With equity E and supply S , the three conversions are

$$\pi(E, S) = \left\lfloor \frac{E \cdot 10^9}{S} \right\rfloor, \quad \text{shares}(a, \pi) = \left\lfloor \frac{a \cdot 10^9}{\pi} \right\rfloor, \quad \text{value}(s, \pi) = \left\lfloor \frac{s \cdot \pi}{10^9} \right\rfloor$$

Every conversion rounds down, intermediate products are checked 128-bit integers, and rounding residue stays in the strategy's equity for the remaining holders. Among share and NAV computations, the only rounding up is the leader's fee compensation, by at most one share unit (§3.6).

8.2 The daily schedule

Each `Strategy` carries a timestamp `next_epoch_at` and a slot `next_epoch_slot`. At creation they are the creation time plus 86,400 seconds and the current slot plus `Config.epoch_duration_slots`; after every finalization or void they are reset the same way from the current clock and slot. A proposal for epoch n is accepted only when

$$t \geq \text{next_epoch_at} \quad \text{and} \quad \text{slot} \leq \text{next_epoch_slot} + \text{proposal_deadline_slots}$$

so the executor cannot postpone the valuation moment indefinitely. If a window is missed, anyone can call `reset_epoch_schedule` once there is no active proposal and either the slot deadline has passed or $t \geq \text{next_epoch_at} + \text{epoch_timeout_seconds}$; it schedules the next boundary one day ahead and changes no NAV. The slot cadence, proposal deadline and epoch timeout are configured parameters (local development values: 75 slots, 1,000 slots and 172,800 logical seconds).

Accounting epochs are anchored to each strategy's own finalizations. They are distinct from the UTC days used for reveal periods (§10 Reveal). Epochs run from creation onward, including the 30-day shadow period, during which only the leader's own capital is in the strategy.

Deposit and withdrawal requests target the next epoch that has not yet been proposed: a request made while a proposal is open waits for the following one. A deposit is escrowed in the pool vault at request time but does not enter the pool's executable cash until it settles, so pending money never finances a trade. Each `Share` holds at most one pending request, sequenced by an owner nonce.

8.3 The proposal

The executor values each strategy at its boundary and signs `propose_epoch(number, nav, liquid_cash)`. Before proposing, it deleverages the strategy if pending withdrawals require cash or would push leverage above the declared maximum (§5.5). The program checks:

- the protocol is not paused and the enclave key is valid;
- `number` is the strategy's last epoch number plus one and no proposal is active;
- the time and slot window of §8.2 holds;
- every request captured by the previous epoch has been settled, carried or cancelled (`unsettled_requests = 0`);
- $\text{liquid_cash} \leq \text{nav}$.

It then creates `Epoch` at `["epoch", strategy, number_u64_le]` and freezes the checkpoint the proposal will be judged by: scheduled boundary time and slot, actual proposal time and slot, proposed NAV and liquid cash, the venue's oracle prices for all three markets, the strategy's cumulative deposits, withdrawals and performance fees, the pre-fee share supply, and the

withdrawal shares captured. It computes the proposed NAV per share $\pi_g = \pi(\text{nav}, S)$ and sets the guardian review flag when the move from the last final value π_{last} exceeds 15%:

$$\text{review} \iff |\pi_g - \pi_{\text{last}}| \cdot 10^4 > \pi_{\text{last}} \cdot 1,500$$

Settlement of earlier epochs is forbidden while the proposal is active, and the next proposal is forbidden until this one's captured requests are processed. A late burn therefore cannot change the supply under a proposed NAV. While the proposal is open, the verifier refuses new risk for that strategy only.

8.4 Verifier recomputation and co-signature

The verifier is an independent operator with its own key, its own RPC endpoint and its own state. It obtains the private openings from the executor at `POST /internal/verifier/openings` with a proof signed by the registered verifier key over `unseen-verifier-openings-v1`, a fresh nonce and the issue time; proofs expire after 60 seconds and nonces cannot be replayed. It authenticates every opening against its on-chain exec and commitment: a missing, duplicated or relabelled opening of a real execution fails the audit, and an opening reserved for an exec that never landed is ignored.

It then rebuilds the chain state as of the proposal slot. Executions created after the slot are excluded. A request filled after the slot is valued as pending: its original collateral at face value, with no fees and no PnL. A request cancelled at or before the slot counts as closed with zero amounts. Reductions after the slot are ignored. Prices are the oracle prices frozen in the `Epoch`, and capital flows are its cumulative checkpoints. On that state it computes the strategy's equity exactly as defined in §3.3:

$$E = D - W - P + \sum_x (R_x - f_x^{\text{exec}} - f_x^{\text{venue}}) + \sum_{x \text{ filled}} U_x$$

and its liquid cash, which is equity less the remaining collateral of pending and filled positions and less the unrealized PnL of filled positions. The audit passes only if all of the following hold:

1. E equals the proposed NAV exactly.
2. Liquid cash covers the next withdrawal reserve and the prospective performance fee: $L \geq \lfloor E/5 \rfloor + F$.
3. After the captured withdrawals, the strategy is within its declared leverage lev , as defined below.

With $E^- = E - F$ and captured withdrawal shares S_w , the withdrawal the next budget can pay and the leverage condition are

$$W = \min\left(\left\lfloor \frac{S_w \cdot E^-}{S} \right\rfloor, \left\lfloor \frac{E^-}{5} \right\rfloor\right), \quad \sum_{x \text{ filled}} n_x \leq \left\lfloor \frac{(E^- - W) \cdot \text{lev}}{10^4} \right\rfloor$$

The verifier then hashes its evidence: SHA-256 of a JSON object with the domain `unseen-epoch-evidence-v2`, the genesis hash, the program ID, the epoch, the NAV and the account and transaction evidence it read. It signs `cosign_epoch(expected_nav, liquid_cash, evidence_hash)`. The program requires the signer to be `Config.verifier`, `expected_nav` and `liquid_cash` to equal the proposal exactly, a nonzero evidence hash, the epoch to be in state Proposed, and the slot to be no later than the proposal slot plus `proposal_deadline_slots`. It stores the evidence hash, moves the epoch to Challengeable and opens the window: `challenge_ends_at = cosigned_at + Config.challenge_seconds`.

A verifier that disagrees does not sign; the epoch stays Proposed until it is voided (§8.9), and other strategies continue. Historical epochs keep the verifier key and signature of their co-sign transaction after the verifier key is rotated, so past epochs remain checkable.

8.5 Challenge window and bonds

While an epoch is Challengeable and $t < \text{challenge_ends_at}$, any wallet may call `challenge`. It transfers `Config.challenge_bond` in the protocol's USDC mint from the challenger into the pool vault, creates a `ChallengeReceipt` at `["challenge", epoch, challenger]`, and marks the epoch challenged, which blocks finalization. The receipt allows one challenge per wallet per epoch, including after a dismissal; while one challenge is unresolved no second one can be filed. The window length and the bond amount are configured parameters with no production default (local development bond: 1 USDC).

The guardian resolves with `resolve_challenge(approve)`. Approving the epoch dismisses the challenge: the bond is forfeited to the protocol's pull claim, the challenge flag clears and the epoch may finalize. Upholding the challenge credits the bond back to the challenger's pull claim, `claim_bond`, and leaves the epoch blocked, so its only exit is a void. The same instruction clears a guardian review flag raised by a move of more than 15%; a review-only resolution involves no challenger and no token account.

8.6 Finalization

`finalize_epoch` is permissionless. It requires the protocol not to be paused; the epoch to be co-signed and Challengeable with its window elapsed, not challenged and, if flagged for review, resolved; and the epoch to be the strategy's active epoch and its next number. The fee and share computations are those of §3.6, with E the proposed NAV, S the pre-fee supply, S_L the leader's shares and φ the fee rate:

$$F = \left\lfloor \frac{\text{value}(S - S_L, \max(0, \pi_g - \text{HWM})) \cdot \varphi}{10^4} \right\rfloor, \quad F_p = \left\lfloor \frac{F}{5} \right\rfloor, \quad F_L = F - F_p$$

with $F_p = 0$ during a founding waiver. Then $E' = E - F$, the leader receives compensation shares ΔS_L that preserve its pre-fee equity, $S' = S + \Delta S_L$, $\pi_f = \pi(E', S')$, $\text{HWM}' = \max(\text{HWM}, \pi_f)$, and the epoch's withdrawal budget is

$$B = \left\lfloor \frac{E'}{5} \right\rfloor$$

Finalization checks liquidity before it commits: pool cash, proposed liquid cash and proposed NAV each at least F ; proposed liquid cash at least $F + B$; and pool cash net of all strategies' protected reserves at least F plus any increase of this strategy's reserve. It credits F_L to the leader's fee claim and F_p to the protocol claim, sets the strategy's liquid cash to $L - F$, its AUM to E' , `last_nav` to π_f and its loss checkpoint to the pool's current realized losses, and replaces its protected withdrawal reserve with B in both the strategy and the pool totals. The epoch becomes Final with `final_nav` = E' , `nav_per_share` = π_f , its performance fee and its budget, and the next boundary is scheduled one day ahead.

The reserve is physical: execution collateral and ordinary execution fees can be paid only from pool cash above the sum of protected reserves, so the 20% set aside at finalization cannot be spent on trades before the strategy's withdrawals are paid.

8.7 Settlement: mint, burn and the withdrawal queue

Settlement runs per share account with the permissionless `settle_share`, after finalization and while no new proposal is active. It processes a request captured by the final epoch that has not yet been settled for it.

Deposit. The deposit a mints $\text{shares}(a, \pi_f)$, which must be positive. The amount is added to the owner's cost basis, the strategy's AUM and liquid cash, the platform AUM and the pool's executable cash.

Withdrawal. Withdrawals form a FIFO queue per strategy. Each request receives a sequence number and a `WithdrawalTicket` at `["withdrawal", strategy, sequence_u64_le]`, and only the request at `next_withdraw_sequence` can settle. With B_{used} already paid from this epoch's budget, the head request of s shares receives

$$b = \min(B - B_{\text{used}}, L, \text{reserve}, \text{pool cash}), \quad s^* = \min(s, \text{shares}(b, \pi_f)), \quad \text{payout} =$$



The program burns s^* , lowers the owner's cost basis by $\lfloor \text{basis} \cdot s^* / s_{\text{held}} \rfloor$, reduces the strategy's AUM, liquid cash and reserve and the pool's reserves and cash by the payout, and credits the payout to the owner's `claimable_withdraw`. If $s^* < s$, the remainder stays at the head of the queue and carries to the next epoch; if the budget is exhausted the request is carried whole. A leader's withdrawal must leave at least 5% of the supply with the leader. A request that is worth zero base units at the last final NAV is refused when made; if a later NAV makes a queued remainder worth zero, settlement releases only that reservation, keeps the owner's shares and cost basis, and advances the queue. A cancelled request leaves its ticket marked cancelled, and anyone can skip it with `advance_withdrawal_queue`, so a cancelled head never blocks the owners behind it.

The 20% budget bounds how much equity any single epoch can release. The executor prepares for it by deleveraging, the verifier refuses to co-sign a proposal whose liquid cash or post-withdrawal leverage cannot support it, and the program enforces it through the reserve.

8.8 Pull-based claims

No settlement step pushes tokens to a user. Payouts accumulate as claims and leave the pool vault only when their owner pulls them:

Claim	Instruction	Signer	Destination
Withdrawal payouts and deposit refunds	<code>claim_withdrawal</code>	Share owner	A USDC token account owned by that owner
Leader's fee share	<code>claim_leader_fee</code>	Leader	A USDC token account owned by the leader
Protocol fees and forfeited bonds	<code>buyback_and_burn</code>	Anyone	<code>Config.protocol_fee_vault</code> (the <code>protocol</code> PDA's USDC account), then swapped for \$UNSEEN and burned in the same transaction
Returned challenge bond	<code>claim_bond</code>	Challenger	A USDC token account owned by the challenger

Table 8.1 — Pull claims. Each instruction zeroes the claim before the pool PDA signs the transfer.

A frozen or missing token account delays only its own owner's claim; it cannot block finalization, the queue or another follower's payout.

8.9 Voids, timeouts and emergency exits

A proposal that cannot finalize is retired by `void_epoch`. The guardian may void any non-final epoch; anyone may void it when the protocol is paused, the enclave key is revoked or expired, or $t \geq \text{proposed_at} + \text{epoch_timeout_seconds}$. A void sets the epoch to state Voided, moves any unresolved bond to the challenger's claim without adjudicating it, clears the active proposal and its request counters, consumes the epoch number and schedules the next boundary one day ahead. The last final NAV is unchanged. Requests captured by the voided proposal are captured again by the next one. The executor's epoch job finalizes a finalizable proposal first; otherwise it voids at the on-chain timeout, refreshes state, deleverages and proposes again.

Two owner-signed paths keep exits open whatever the state of a proposal.

- `emergency_withdraw(shares, nonce)` pays at the last final NAV from the remainder of that epoch's budget, at any time, including while a proposal is open or the protocol is paused. It must be the queue head; the payout must be positive and within the remaining budget, the strategy's liquid cash and reserve, and pool cash; a leader must keep 5% of equity, counting pending requests. A pending deposit of the same owner is refunded to its claim without counting against the budget. If a proposal is open, the same transaction voids it, and only if the positive burn and payout succeed. Once the reserve is spent, further withdrawals wait for another verified epoch; neither a void nor a pause creates a second budget.
- `cancel_request(nonce)` withdraws a pending request. A deposit becomes a refundable claim; a withdrawal's ticket is marked cancelled. If the request was captured by an open proposal, only the proposal's request counters change; its NAV and share supply stay as proposed.

Guardian pause stops execution, proposals and finalization. Settlement of an epoch that is already final, claims and emergency withdrawals remain available.

8.10 Worked example

This example continues the finalization of Table 3.3 (§3 Strategies, shares and the pool): a genesis NAV of 1,000,000, a leader with 10,000 shares and followers with 20,000 shares, of which follower A holds 12,000 and follower B 8,000, a 1,500 bps fee, no waiver and a declared maximum leverage of 3×. Before the proposal, A requests a withdrawal of 8,000 shares and a new follower C requests a deposit of 5,000 USDC; both are captured by epoch n . The strategy holds one filled SOL long of 30,000 USDC notional with 10,002 USDC of collateral and 998 USDC of unrealized gain.

Step	Computation	Result
Proposal	NAV 33,000 USDC; liquid cash $33,000 - 10,002 - 998 = 22,000$ USDC	$\pi_g = 1,100,000$
Review flag	$ 1,100,000 - 1,000,000 \cdot 10^4 = 10^9$ vs $1,000,000 \cdot 1,500 = 1.5 \cdot 10^9$	Not flagged
Verifier cash	$\lfloor 33,000/5 \rfloor + 300 = 6,900 \leq 22,000$	Passes
Verifier leverage	$W = \min(\lfloor 8,000 \cdot 32,700/30,000 \rfloor, \lfloor 32,700/5 \rfloor) = \min(8,720, 6,540); 30,000 \leq \lfloor (32,700 - 6,540) \cdot 3 \rfloor = 78,480$	Passes; co-signed
Finalization	$F = 300, F_L = 240, F_p = 60; E' = 32,700; \Delta S_L = 138.248847927$ shares	$\pi_f = 1,084,999, B = 6,540$ USDC
Liquidity gate	$22,000 \geq F + B = 6,840$	Passes; liquid cash becomes 21,700 USDC
C's deposit	$\lfloor 5,000 \cdot 10^6 \cdot 10^9 / 1,084,999 \rfloor$	4,608.299178155 shares
A's withdrawal	$b = \min(6,540, L, 6,540, \text{pool cash}) = 6,540;$ $s^* = \lfloor 6,540 \cdot 10^{15} / 1,084,999 \rfloor$	6,027.655325027 shares burned; 6,539.999999 USDC claimable
Carry	$8,000 - 6,027.655325027$	1,972.344674973 shares stay at the queue head
After settlement	$E = 32,700 - 6,539.999999 + 5,000; S = 30,138.248847927 + 4,608.299178155 - 6,027.655325027$	31,160.000001 USDC over 28,718.892701055 shares; $\pi = 1,085,000$

Table 8.2 — One epoch from proposal to settlement. Amounts in USDC unless stated; the execution and venue fees of the open position are already included in the NAV.

A then pulls 6,539.999999 USDC with `claim_withdrawal`, the leader pulls 240 USDC with `claim_leader_fee`, and anyone can move the protocol's 60 USDC to its fee vault. The one base unit of USDC by which the payout falls short of the budget, and the one by which the NAV per share rises after settlement, are rounding residue retained in equity. In epoch $n + 1$ A's remaining 1,972.344674973 shares, worth about 2,140 USDC, are first in the queue; at an unchanged NAV the new budget is about $\lfloor 31,160/5 \rfloor = 6,232$ USDC, so they settle in full. Had the verifier refused to co-sign or the proposal been challenged and upheld, the epoch would have been voided at its

timeout, both requests would have rolled into the next proposal at the unchanged last final NAV, and A could still have exited up to the last epoch's remaining budget with `emergency_withdraw`.

9. Fees

A strategy carries three kinds of cost, all in USDC: a performance fee on follower gains above the high-water mark, a protocol execution fee on every change in position notional, and the venue's own trading fees. v0 has no management fee. Every amount is an unsigned integer in USDC base units (six decimals), shares carry nine decimals, and every intermediate product is checked `u128` arithmetic in the shared `unseen_domain` crate. The program, the Sealed Executor and the verifier all link that crate, so the three compute the same integers.

Table 9.1 — Fee schedule.

Fee	Base	Rate	Borne by	Recipient	Charged in
Performance fee	Follower gain in NAV per share above the high-water mark	Set by the leader at creation, 0–2,500 bps; default 1,500 bps	Follower shares only	80% leader, 20% protocol; 100% leader during a founding waiver	<code>finalize_epoch</code>
Execution fee	Changed notional: opened at fill, reduced, or closed	2 bps	Strategy equity, pro rata by shares	Protocol	<code>record_fill</code> , <code>partial_close</code> , <code>close_exec</code>
Venue fees	Trade size	Jupiter Perps: 0.06% to open, 0.06% to close, plus price impact; all fees capped at 0.50% of trade size	Position collateral	Jupiter and its liquidity providers	The venue, inside the exec account's position
Management fee	—	None in v0	—	—	—

9.1 Performance fee and the high-water mark

Each `Strategy` account stores `performance_fee_bps` and `high_water_mark`. `create_strategy` rejects a fee above 2,500 bps, and no instruction changes it afterwards. The high-water mark is a NAV per whole share in USDC base units. It starts at the deployment's genesis NAV (`Config.genesis_nav`, which `initialize` requires to be at least 1,000,000, that is 1 USDC per share) and never falls.

The fee is computed once per epoch, in `finalize_epoch`, from values frozen when the epoch was proposed (see §8 Epochs and settlement). Let N be the proposed NAV, T the total shares, L the leader's shares, $F = T - L$ the follower shares, H the high-water mark and f the fee in basis points. The proposal records the pre-fee NAV per share

$$p = \left\lfloor \frac{N \cdot 10^9}{T} \right\rfloor$$

and finalization computes the follower gain and the fee:

$$G = \left\lfloor \frac{F \cdot \max(0, p - H)}{10^9} \right\rfloor, \quad \Phi = \left\lfloor \frac{G \cdot f}{10\,000} \right\rfloor$$

Only follower shares enter G . The leader's stake pays no performance fee to itself. Because Φ leaves equity that the leader shares with followers, the program also mints compensation shares to the leader, so that the leader's equity after the fee equals its equity before it:

$$E_L = \left\lfloor \frac{N \cdot L}{T} \right\rfloor, \quad c = \left\lceil \frac{E_L \cdot F}{N - \Phi - E_L} \right\rceil - L$$

Rounding up gives the leader at most one share base unit of dust, and no USDC is created.

Finalization then sets the strategy's AUM to $N - \Phi$, adds c to both L and T , computes the final NAV per share $p' = \lfloor (N - \Phi) \cdot 10^9 / T \rfloor$ and raises the mark to $H \leftarrow \max(H, p')$.

The mark is a single per-share value for the whole strategy, so every follower share is measured against it. A gain is charged once, in the epoch that first carries NAV per share above the mark. Deposits always mint at the final NAV per share, which never exceeds the mark, so a share minted below the mark pays nothing while NAV climbs back to it.

Finalization charges the fee only if the strategy can pay it in cash. It requires `Pool.cash`, the proposal's verified liquid cash and the proposed NAV each to be at least Φ . It also requires liquid cash to cover Φ plus the new 20% withdrawal reserve, and the pool's unprotected cash to cover Φ plus any increase in that reserve. The fee is then booked as two pull claims,

`Strategy.leader_fee_claim` and `Pool.protocol_claim`, and `Pool.cash` falls by Φ . Both `Epoch.performance_fee` and `Strategy.cumulative_performance_fees` record it. The cumulative

figure is part of every later epoch's capital checkpoint, and both the verifier and the public NAV reconstruction subtract it (see §10.10). Because payouts are claims, a missing or frozen leader token account cannot hold up an epoch.

9.2 The 80/20 split and the founding waiver

The split is fixed in code: the protocol share is $\lfloor \Phi/5 \rfloor$ and the leader receives $\Phi - \lfloor \Phi/5 \rfloor$, so rounding never favours the protocol. At the v0 strategy cap, take a leader with 100,000 USDC of their own and 150,000 USDC from followers, a 30% gross annual return, a 15% fee and the 20% protocol share (the brief's assumptions). Followers gain 45,000 USDC and owe 6,750 USDC, of which the leader keeps 5,400 USDC on top of the 30,000 USDC earned on their own capital.

Founding leaders pay no protocol share for 90 days, and that waiver can only be granted on-chain. The admin issues a `LeaderInvitation` (`invite_leader(expires_at, founding)`), and `create_strategy` accepts `founding = true` only when that invitation carries the founding flag. The strategy then stores `waiver_ends_at = created_at + 7,776,000` seconds (90 days); non-founding strategies store zero. While the finalization timestamp is below `waiver_ends_at`, the protocol share is zero and the leader receives the whole fee. The 30-day shadow period starts at the same moment, and `request_deposit` refuses deposits until it ends, so a founding waiver covers the follower-funded epochs from day 30 to day 90. The waiver leaves the execution fee unchanged. The public `Strategy` record exposes `protocolPerformanceShareBps` and `foundingFeeWaiverEndsAt`.

9.3 Execution fee

The protocol charges 2 bps on notional that actually changes:

$$\phi_{\text{exec}} = \left\lfloor \frac{\Delta N \cdot 2}{10\,000} \right\rfloor$$

where ΔN is one of three amounts:

- the exec account's opened notional, charged in `record_fill` once the keeper has filled the request;
- the reduced notional in `partial_close`, which keeps the same exec account and entry price;
- the remaining notional in `close_exec`.

A request that never filled and is cancelled through `close_exec(cancel = true)` pays nothing, and neither does a guardian rescue through `guardian_close`. A same-side reduction pays only on the delta. A sign flip closes the old exposure and opens a fresh exec account, so it pays on both legs, which together equal the full change from $+x$ to $-x$.

Every charge passes `charge_execution_fee`. That function keeps a per-UTC-day counter (`Pool.fee_day`, `Pool.execution_fees`) and rejects the transaction if the day's total would exceed $\lfloor \text{platform AUM} \cdot \text{execution_fee_epoch_cap_bps} / 10\,000 \rfloor$. The cap rate is a configured

parameter. Its local development value, 100 bps, allows 20,000 USDC per day at the 2,000,000 USDC platform cap. The function also adds the fee to `Pool.total_realized_losses`, which the anonymous risk envelope deducts from every active strategy's conservative equity until that strategy's next final checkpoint (see §12.3). Each charge also requires unprotected pool cash, `Pool.cash - Pool.withdrawal_reserves`, to cover the fee.

The fee comes out of pooled strategy cash, so it reduces NAV and every share bears it in proportion, the leader's included. The signed policy evidence of each execution carries `allocatedExecutionFeeUsdc` as an authorized ceiling. The indexer requires every revealed adjustment's actual fee to be at or below that ceiling. Execution-fee revenue pays for the enclave, keepers, RPC and Jito tips.

9.4 Venue fees

Jupiter Perps charges 0.06% to open and 0.06% to close. It adds a linear price-impact fee of about 1 bp per started ~375,000 USDC of SOL notional, and an additive fee once net opened-minus-closed size on an asset over roughly a minute passes 1.5 million USDC (SOL, ETH) or 750,000 USDC (wBTC). All fees together are capped at 0.50% of trade size. These fees are taken from the position's collateral inside the exec account and go to Jupiter and its liquidity providers. Followers pay them with or without Unseen; §7 Venue: Jupiter Perps gives the request path. The local fake venue models the same structure: at fill it takes 6 bps of notional from the request's collateral, and at a reduction or close it takes $\min(\lfloor 6 \cdot \Delta N / 10\,000 \rfloor, \text{gross returned})$.

When several strategies' orders in one market land in the same batch, the batch's price-impact fee I is split pro rata by notional. Members are taken in signed receipt order, and P_i is the notional of the members before member i :

$$a_i = \left\lfloor \frac{I \cdot (P_i + n_i)}{\sum_j n_j} \right\rfloor - \left\lfloor \frac{I \cdot P_i}{\sum_j n_j} \right\rfloor$$

The sum telescopes to exactly I , and the remainder falls by receipt order. The domain tests pin $I = 100$ over notionals 2, 3 and 5 to 20, 30 and 50, and $I = 2$ over 1, 1 and 1 to 0, 1 and 1. Each member's share is written into its policy evidence (`allocatedImpactFeeUsdc`, `allocatedVenueFeeUsdc`, `allocatedExecutionFeeUsdc`) and published with its opening.

Fees are reserved before an order is sent. The verifier requires the strategy's own cash to cover future opening fees, $\lfloor 2N / 10\,000 \rfloor$ for the protocol plus the venue's opening fee (the fake venue's $\lfloor 6N / 10\,000 \rfloor$ locally). The program's envelope subtracts a 16 bps ceiling-rounded bound on outstanding venue and execution fees over the pool's aggregate notional, the candidate order included.

9.5 Worked example: performance fee over three epochs

The parameters are a genesis NAV of 1 USDC per share, a leader stake of 10,000 USDC and follower deposits of 20,000 USDC (the local development profile's stake and deposit), the default 1,500 bps fee and no waiver. The opening state is $T = 30,000$ shares, $L = 10,000$ and $H = 1,000,000$.

Table 9.2 — Three consecutive finalizations.

Epoch	Proposed NAV	p (pre-fee)	Mark before	Follower gain G	Fee Φ	Leader / protocol
1	33,000.000000	1,100,000	1,000,000	2,000.000000	300.000000	240.000000 / 60.000000
2	31,000.000000	1,028,593	1,084,999	0	0	0 / 0
3	34,000.000000	1,128,134	1,084,999	862.700000	129.405000	103.524000 / 25.881000

NAV and fees are in USDC, p and the mark in base units per whole share, and c in shares. In epoch 1 the strategy is up 10% gross. Followers own 20,000 shares that each gained 0.1 USDC, so $G = 2,000$ and $\Phi = 300$. The leader receives 138.248847927 new shares, and the 32,700 USDC of equity divides as exactly 11,000 USDC to the leader (its pre-fee equity) and 21,700 USDC to followers: a net 8.5% on 20,000 USDC. In epoch 2 NAV per share is below the mark and no fee accrues. In epoch 3 the fee applies only to the climb from 1,084,999 to 1,128,134, that is 43,135 base units on each of the 20,000 follower shares.

9.6 Worked example: fees over a position's life

Take a strategy at the v0 cap of 250,000 USDC trading SOL at unchanged oracle prices, with sizing equity held at 250,000 USDC for simplicity. A 15,000 bps long target is 150% of equity, that is 375,000 USDC of notional. `execute` accepts at most 250,000 USDC per call, and pool flow is capped at 250,000 USDC per market per logical minute, so the opening goes out as two fresh exec accounts in two minutes.

Table 9.3 — Execution and venue fees for one sequence of targets.

Step	Target	Changed notional	On-chain actions	Execution fee (2 bps)	Jupiter open/close (0.06%)
1	Long 15,000 bps	375,000	Two <code>execute</code> + <code>record_fill</code> : 250,000 and 125,000	75	225
2	Long 10,000 bps	125,000	Full close of the smaller exec, or a partial close of the larger	25	75
3	Short 10,000 bps	500,000	<code>close_exec</code> on the 250,000 long, then a fresh short exec of 250,000	100	300
4	Flat	250,000	<code>close_exec</code> on the short	50	150
Total		1,250,000		250	750

Each of the six trades is below 375,000 USDC, so each pays about 1 bp of price impact, roughly 125 USDC in all. None comes near Jupiter's 1.5 million USDC per-minute additive threshold. A leader holding the 5% minimum bears 12.50 USDC of the execution fees and 37.50 USDC of the open and close fees.

9.7 Where each fee is charged on-chain

Table 9.4 — Instructions, ledgers and destinations.

Charge	Instruction	Signer	Debited	Credited	R
Execution fee on opening	<code>record_fill</code>	Enclave and verifier	<code>pool_vault</code> (<code>Pool.cash</code>)	<code>Config.protocol_fee_vault</code> , transferred at once	E E E
Execution fee on reduction	<code>partial_close</code>	Enclave and verifier	<code>Pool.cash</code>	<code>Pool.protocol_claim</code>	E E
Execution fee on close	<code>close_exec(cancel = false)</code>	Enclave and verifier	<code>Pool.cash</code>	<code>Pool.protocol_claim</code>	A
Refund of an unfilled request	<code>close_exec(cancel = true)</code>	Enclave and verifier	Nothing	—	E P
Guardian rescue	<code>guardian_close</code>	Guardian	Nothing (exempt)	—	E h
Performance fee, leader part	<code>finalize_epoch</code>	Anyone	<code>Pool.cash</code>	<code>Strategy.leader_fee_claim</code>	E
Performance fee, protocol part	<code>finalize_epoch</code>	Anyone	<code>Pool.cash</code>	<code>Pool.protocol_claim</code>	S
Leader payout	<code>claim_leader_fee</code>	Leader	<code>pool_vault</code>	A leader-owned USDC account	C
Protocol buyback	<code>buyback_and_burn</code>	Anyone	<code>pool_vault</code> , then the protocol fee account	<code>Config.protocol_fee_vault</code> , swapped for \$UNSEEN and burned	C C
Venue fees	Venue CPI in <code>execute</code> , <code>partial_close</code> , <code>close_exec</code> ; keeper fill	Venue	Position collateral in the exec account	Venue fee vault	E v

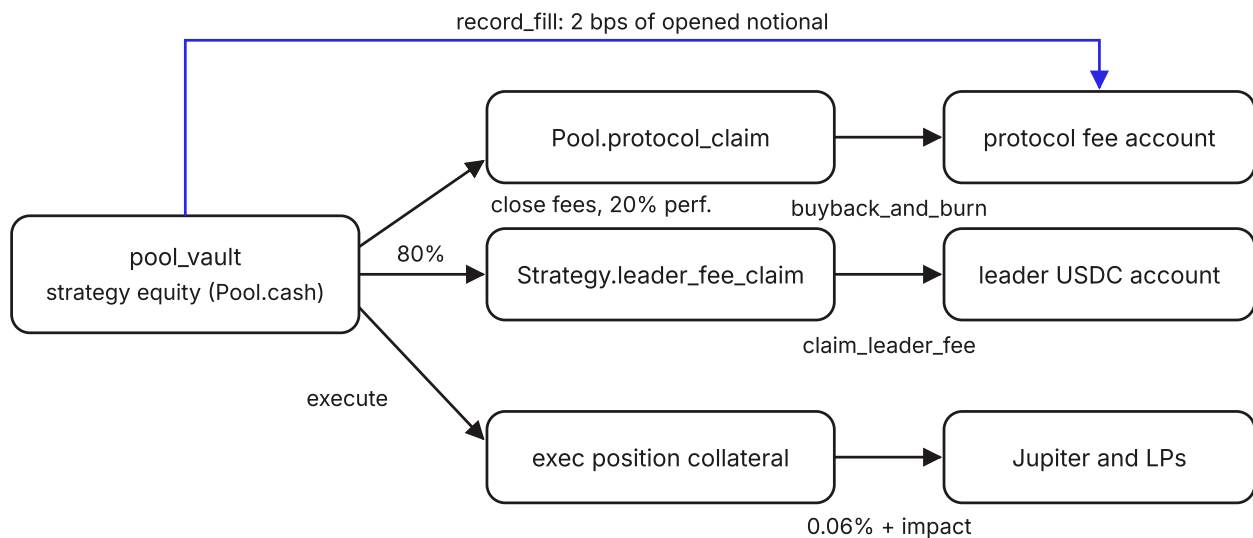


Figure 9.1 — Fee flows. The opening execution fee goes straight to the protocol fee account. Reduction and close fees and both parts of the performance fee become claims on the pool vault; the protocol's part leaves only through `buyback\_and\_burn`. Venue fees leave from the exec account's position collateral.

9.8 The protocol's share: buyback and burn of \$UNSEEN

Every fee the protocol earns ends in one account, the protocol fee account (`Config.protocol_fee_vault`, the USDC token account of the keyless `protocol` PDA). Opening execution fees are transferred there by `record_fill` at once; execution fees on reductions and closes, the protocol's 20% of each performance fee and forfeited challenge bonds accrue in `Pool.protocol_claim`. No key can sign for the `protocol` PDA, and no instruction pays its USDC to a wallet: the only way out is `buyback_and_burn`.

All of it buys **\$UNSEEN** and burns it. `buyback_and_burn(min_out, route_data)` is permissionless and atomic. It moves the whole `Pool.protocol_claim` into the protocol fee account; invokes the configured swap program — Jupiter v6 on mainnet — with an exact-in USDC → \$UNSEEN route whose authority, source and destination it checks against the `protocol` PDA's own token accounts, signing only as that PDA; requires at least `min_out` newly received \$UNSEEN; and burns the PDA's entire \$UNSEEN balance. Claim, swap and burn succeed or roll back together, so the account never keeps the token and the supply only falls. `Config` keeps the cumulative USDC spent and \$UNSEEN burned, and `GET /v1/protocol` publishes them with the last buyback's signature.

The protocol's worker cranks it every hour once at least 1 USDC has accrued: it quotes the route from Jupiter's swap API (at most 48 accounts, 1% slippage by default), sets `min_out` to the quote's minimum output, and sends claim, swap and burn as one versioned transaction. The \$UNSEEN mint and the swap program are set through the admin path with the 48-hour timelock (`schedule_buyback`, then `activate_buyback`). \$UNSEEN has a fixed supply of 1,000,000,000 and was launched 100% fair: the whole supply went to the public launch on equal terms, with no presale, no team or investor allocation, no airdrop, no reserved pools and no vesting.

Table 9.5 — What each fee contributes to the buyback.

Fee	Protocol's part	Example	To the buyback
Performance fee	20% of Φ (0% during a founding waiver)	Followers gain \$10,000 above the mark at a 15% fee: $\Phi = \$1,500$	\$300
Execution fee	All of it, 2 bps of changed notional	A \$100,000 fill	\$20

Users never hold or spend the token to use Unseen: deposits, NAV, every fee as charged and every payout are USDC, and strategy shares are a ledger in the `Share` account, separate from \$UNSEEN.

10. Reveal

Every Unseen execution is public from the moment it lands: its exec account, market, side, notional, collateral, fills and fees are ordinary Solana state. Only its owner is hidden, behind the 32-byte commitment the account carries, until the strategy's reveal delay D has passed (24 hours to 30 days, default 7 days). A reveal publishes the commitment's opening: the preimage that names the strategy, plus the signed evidence that ties the trade to the leader's signal. This chapter specifies when an opening may be published, how the program enforces that without knowing who owns which account, what an opening contains, how it is posted, and how anyone checks it and rebuilds a strategy's NAV from it.

10.1 Why reveals are grouped by period

If each opening were published at its own strategy's D , revealing would leak attribution. Every exec account of a day is visible and unattributed. As the openings of short-delay strategies come out, the day's still-hidden accounts can only belong to strategies with longer delays. Once a single long-delay strategy is left, its trades are identified before its own delay ends. This reveal-by-elimination property was found in review and settled by a product decision on 2026-10-15: publish a period only once every strategy active in it has passed its delay, even if short-delay strategies then reveal later than their own delay. No opening is ever published before its own strategy's delay.

10.2 Periods

A reveal period is a UTC day on the program's clock:

$$\text{day}(t) = \left\lfloor \frac{t}{86\,400} \right\rfloor, \quad \text{period } d = [86\,400\,d, 86\,400\,(d+1))$$

The program uses Euclidean division, so days before 1970 are well defined. Its clock is

`logical_genesis + (unix_timestamp - real_genesis) · clock_scale + clock_offset`.

`initialize` accepts a `clock_scale` other than 1 only in a local-testing build, so in production the period boundary is the Solana clock's UTC midnight. Reveal periods are independent of each strategy's accounting epochs (see §8 Epochs and settlement). A strategy is *active* in a period if it traded in it or held an open position at any time during it.

10.3 The rule

The openings of period d may be published once every strategy active in d is past its own delay. On top of that, each revealed execution must be terminal: closed, or a cancelled request that never filled. An open position is never revealed. A position that spans several days is active in each of them, so its opening waits for every day it was held, from the day of its first fill to the day it closed. Each of those days matures at its end plus a delay at least its own strategy's, so no opening is ever published before its own strategy's delay.

10.4 How the program enforces it

The set of strategies active in a day is the mapping Unseen keeps secret: no account links a strategy to an exec account. The program never learns the set. It learns one number about it, certified by the two parties who can compute it: D_P , the longest reveal delay among the strategies active in day d .

After the day ends, the executor computes D_P from its sealed openings — every execution filled in the day or held through it, and the delays of the strategies they belong to — and proposes it. The independent verifier reads the complete execution history from its own RPC, checks that authenticated sealed openings account for every execution, recomputes which strategies were active and their longest delay from the on-chain strategy accounts, and signs only a matching value. `seal_reveal_period_certified(day, D_P, evidence_hash)` requires both registered signatures and creates the day's `RevealPeriod` account (seeds `"reveal_period"`, the pool, d as i64 little-endian):

$$\text{mature_at}(d) = 86\,400(d + 1) + D_P, \quad \text{max_reveal_delay}(d) = D_P$$

It accepts $24\text{ h} \leq D_P \leq 30\text{ d}$, no more than the longest delay ever registered, and a non-zero evidence hash. The hash commits to the private openings, salts included, so observers can't enumerate the short public strategy list to recover membership. A certified day can't be rewritten. The certificate publishes the day, D_P , the opaque hash and its status — never the active set.

Neither an unrelated open position nor a registered strategy that didn't trade that day moves the certificate: a day on which only a 24-hour strategy was active matures 24 hours after it ends, even while a 30-day strategy is registered.

An understated D_P publishes nothing early. Every reveal checks that its own strategy's delay is at most each certificate's D_P (§10.5), so a strategy whose delay exceeds the certified value can't be revealed against it.

The permissionless `seal_reveal_period(day)` remains as a fallback for a day that never gets a certificate. It waits for the first moment the whole pool is flat after the day ends, F , and sets `mature_at` = F plus the longest delay ever registered. A certificate can replace a fallback record; a fallback can never replace or change a certificate.

10.5 Reveal checks

`reveal_v2` publishes a V2 opening and `reveal` publishes a legacy V1 opening. Both are signed by the registered, unexpired, unrevoked enclave key. The two period accounts are not caller-chosen: the instruction's account constraints derive them from the exec account's own timestamps: the day of its first fill for the opening period, `day(closed_at)` for the closing period.

Table 10.1 — Checks in `reveal_v2`.

Check	Purpose
<code>Exec.state ≥ 2</code> (closed or cancelled)	An open position is never revealed
<code>Exec.chain_sequence ≤ Strategy.active_until_sequence</code>	A strategy frozen by <code>deactivate_zero_strategy</code> cannot claim later executions
<code>Strategy.created_at ≤ CommitmentReceipt.created_at</code>	A strategy registered after the execution, whose delay a sealed certificate may not cover, cannot claim it
Receipt not yet revealed; <code>index</code> and <code>strategy_id</code> match the accounts	One reveal per commitment
<code>policy_evidence_hash = Exec.policy_evidence_hash</code>	The published policy is the one the execution committed to
<code>commitment_v2(Config.deployment_genesis, program ID, Exec.executor, ...) = CommitmentReceipt.commitment</code>	Deployment, program and executor come from chain state, not from the caller
The first-fill day's and the closing day's certificates have <code>now ≥ mature_at</code> and <code>Strategy.reveal_delay ≤ max_reveal_delay</code>	The period rule for those days, and never before the strategy's own delay
The same for every held day in between whose end is less than 30 days past	The period rule for every day the position was open

The period accounts are not caller-chosen: the first-fill and closing days come from the exec account's own timestamps, and each intermediate account is derived from its day. A held day that ended more than 30 days ago — the protocol's longest delay — is mature under any certificate, so it needs no account, and a reveal never presents more than 30 of them.

10.6 One period, three delays

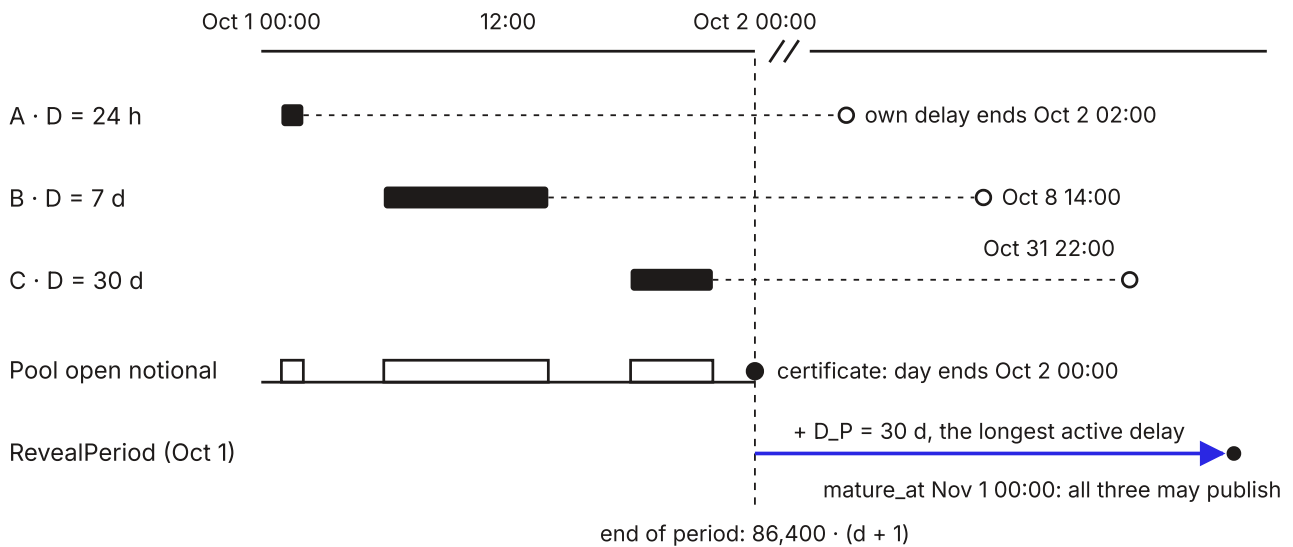


Figure 10.1 — Strategies with 24-hour, 7-day and 30-day delays trade in the same UTC day. Each closes on that day. The day's certificate carries the longest of the three delays, 30 days, and matures 30 days after the day ends; all three openings publish from then on, later than A's and B's own delays. The right half of the axis is not to scale.

Figure 10.1 extends the reveal job's own test case. On 2026-10-01, strategy A ($D = 24$ h) trades from 01:00 to 02:00, B ($D = 7$ d) from 06:00 to 14:00 and C ($D = 30$ d) from 18:00 to 22:00. Measured from each close, their own delays would end on Oct 2 at 02:00, Oct 8 at 14:00 and Oct 31 at 22:00. The executor and the verifier certify the day with $D_P = 30$ days, the longest delay among A, B and C, so `mature_at` = Oct 2 00:00 + 30 d = Nov 1 00:00. On Oct 9 none of the three openings may be published. From Nov 1 00:00 all three may. Had A's and B's openings appeared on Oct 2 and Oct 8, the one unrevealed exec account of Oct 1 would have pointed to the only registered strategy with a longer delay, more than three weeks before C's own delay ended. Had only A traded on Oct 1, the day would have been certified with $D_P = 24$ hours and matured on Oct 3 at 00:00, though C stays registered. The program's tests reject the 24-hour reveal until the 30-day certificate matures, publish a 24-hour-only day after 24 hours, and never publish an open position.

10.7 The opening

A V2 commitment is the SHA-256 of a fixed 273-byte preimage. Integers are little-endian, and keys and hashes are raw 32 bytes.

Table 10.2 — V2 opening preimage (`unseen-sha256-v2`).

Offset	Bytes	Field	Bound to
0	17	"unseen-opening-v2"	Domain separation, UTF-8, no terminator
17	32	genesis	Config.deployment_genesis
49	32	program	The Unseen program ID
81	32	executor	Exec.executor , the enclave key that executed
113	32	strategy_id	Strategy.id
145	8	exec_index	Exec.index , the random 64-bit PDA index
153	8	signal_sequence	The leader's signed sequence
161	8	global_receipt_sequence	The executor's global receipt order
169	8	received_at_unix_micros	Receipt time
177	32	signed_signal_hash	SHA-256 of the canonical V2 signing bytes followed by the raw 64-byte leader signature
209	32	salt	32 random bytes
241	32	policy_evidence_hash	SHA-256 of the 392-byte PolicyEvidenceV2 encoding

The legacy V1 commitment (`unseen-sha256-v1`) is the SHA-256 of 80 bytes: `strategy_id` || `exec_index` || `signal_seq` || `salt` . V1 remains verifiable for legacy fixtures, and all new live executions use V2.

The published evidence envelope, `OpeningV2` , carries the preimage fields and everything needed to check them:

- `signedSignal` , the complete deployment-bound signal: strategy, sequence, target weights in submitted order, `validUntilSlot` , `maxOracleDeviationBps` , signal key and Ed25519 signature (see §4 Signals);
- `receiptSignature` , the enclave's Ed25519 signature over the `unseen-receipt-v2` lines: genesis, program, executor, receipt sequence, receipt time and signed-signal hash;
- `policyEvidence` , 26 fields: batch identity, index and size, member and original notional, allowed reduction, allocated impact, venue and execution fees, sizing slot and equity, prior signed notional, target weight, declared leverage, drawdown kill level and reveal delay, prior policy hash and validity slot;
- `policyHistory` , one witness for each later leader-signed reduction or close, each with its own signal, receipt and policy evidence and its per-exec operation sequence;

- `riskHistory`, any `unseen-risk-reduction-v2` certificates by which the verifier forced a reduction for withdrawal liquidity, post-withdrawal leverage or drawdown;
- `openingPeriodDay` and `closingPeriodDay`, which are derived metadata outside the preimage.

10.8 Publication

The executor's background job publishes in four steps, after trading and keeper reconciliation in each tick. Per tick it attempts at most one period seal, at most two evidence pages and at most three background chain operations, and its round-robin cursors advance on failure as well as success, so one stuck item cannot starve the rest.

1. **Seal.** `seal_reveal_period` for the creation day and the closure day, once each is eligible.
2. **Header.** `reveal_v2` passes the fixed fields as instruction data and emits `OpeningRevealedV2`: strategy ID, index, signal sequence, salt, receipt sequence and time, signed-signal and policy hashes, executor, genesis, commitment, exec, timestamp and slot. It sets `revealed` on the Exec account and the `CommitmentReceipt` and stores nothing else.
3. **Evidence pages.** `reveal_evidence(page, pages, data)` splits the UTF-8 JSON envelope into pages of at most 400 bytes, each emitted as a `RevealEvidence` event. The program accepts pages only after the header and while the exec account is terminal. The job's progress record is bound to the SHA-256 of the exact evidence bytes, so a lost acknowledgement retries the identical page.
4. **Cleanup.** `reclaim_exec`, which anyone may call once the opening is revealed and the exec vault is empty, archives the exact public execution fields in `CommitmentReceipt.archive`. It closes the venue request and the SPL exec vault to the pool, and shrinks the Exec account to a 40-byte tombstone that keeps its random index irreusable.

Public reads expose a reveal only once it is complete and mature. `GET /v1/strategies/:strategyId/revealed-trades` and `GET /v1/revealed-executions/:executionId` return the `RevealedExecution` record: the unattributed execution, the opening, reveal time, slot and transaction, exact execution and venue fees, and a `revealBarrier` holding both period certificates under the policy name `daily_closed_cohort`. An unrevealed execution ID returns the same `NOT_FOUND` as an unknown one. `revealSchedule.nextBoundaryAt` is the same fixed daily boundary for every strategy and is never derived from a hidden execution.

10.9 Verification in the browser

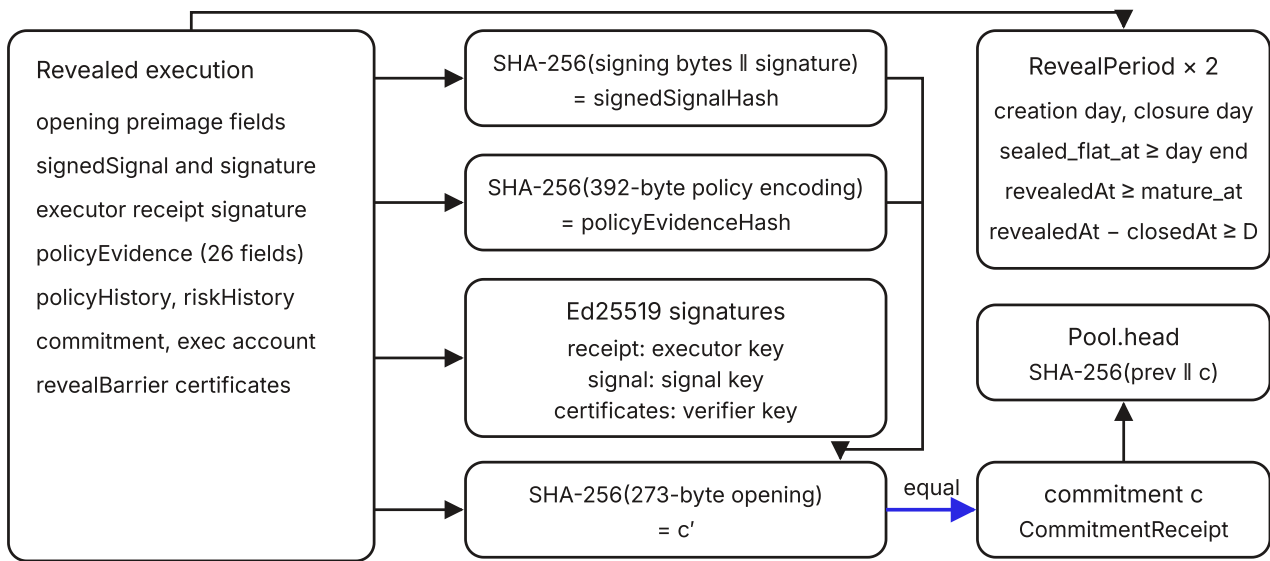


Figure 10.2 — Verification path for a V2 opening. The two hashes computed from the evidence feed the 273-byte preimage. Its SHA-256 must equal the commitment the exec account carried on-chain, that commitment must sit in the pool's hash chain, and the reveal must satisfy both period certificates.

The browser check needs only the public record and Web Crypto. The strategy page fetches each revealed execution and calls `verifyOpening(opening, commitment)` from the contract package. It shows *Opening matches its commitment* when every step passes:

1. Encode the preimage (Table 10.2), hash it, and compare the result with the commitment.
2. Check cross-field bindings: genesis, program, executor and strategy are identical in the opening, the signed signal and the policy evidence; the signal's sequence equals `signalSeq`; the receipt sequence and signed-signal hash agree with the policy evidence; and the policy's `validUntilSlot` equals the signal's.
3. Recompute `signedSignalHash` from the canonical signing bytes and the 64-byte signature, and `policyEvidenceHash` from the 392-byte encoding.
4. Verify the executor's receipt signature with the executor key.
5. Walk `policyHistory` in order, requiring strictly increasing operation sequences, non-decreasing receipt and signal sequences and each witness's notional equal to its policy's member notional, and verify each witness the same way.
6. Verify each `riskHistory` certificate's Ed25519 signature against its verifier key, with operation sequences unique and increasing.
7. Verify the leader's Ed25519 signature over the signal with the signal key.

A second, chain-side layer runs before the indexer projects any reveal. It reads the actual accounts and transactions from its RPC and checks the following:

- the full commitment chain: sequence numbers 1 to n with no gap, each `resulting_head` equal to SHA-256 of `previous_head` || commitment, and the last head equal to `Pool.head`;
- each certificate: matured at the reveal's time, with a maximum delay no shorter than the revealed strategy's, for the first-fill day, the closing day and every held day in between that ended less than 30 days before;
- each reveal: the envelope equals the on-chain header, the policy hash equals the exec account's, and every leader-authorized adjustment has exactly one matching witness or risk certificate whose notional, fees and validity agree with the landed transaction;
- terminal state: zero remaining notional and `verifyOpening` passing.

Checking the signal key's historical registration and replaying the sizing policy need chain history, and the independent verifier performs them.

10.10 NAV reconstruction from revealed data

`GET /v1/revealed-strategies/:strategyId/nav`, served from the same public replay as the verifier's `public-nav --strategy <id>` command, rebuilds a strategy's NAV at each finalized epoch. It uses only public accounts, signed transactions and complete revealed evidence. It never reads the executor's store and needs no signing key. An epoch is included only when it is final, its boundary plus the strategy's D has passed, and **every** pool execution created at or before the epoch's valuation slot has a public opening. Until the whole pool's openings for that window are out, nobody can prove that some hidden execution does not belong to this strategy, so the epoch is left out rather than estimated.

For an included epoch, the replay reconstructs every execution's state at the valuation slot, the actual proposal slot recorded in `Epoch.proposal_slot`. A request that had not filled by then counts as refundable collateral at face value, with no fees or PnL. Adjustments count only up to that slot, from the original archived notional and collateral. Prices are the epoch's captured `Epoch.prices`. It authenticates the complete opening set: one opening per execution, each commitment checked, and market and side equal to the chain's. Then, over the strategy's openings, with S_k the epoch's cumulative capital checkpoints:

$$\text{NAV} = S_{\text{dep}} - S_{\text{wd}} - S_{\text{perf}} + \sum_e \left(R_e - \phi_e^{\text{exec}} - \phi_e^{\text{venue}} + U_e \right) - \Phi_{\text{epoch}}$$

Here R_e is realized PnL, the two ϕ terms are the actual execution and venue fees, U_e is the unrealized PnL of a still-filled position at the epoch price, and Φ_{epoch} is the epoch's own performance fee. The replay then checks a second, independent composition:

$$\text{cash} = S_{\text{dep}} - S_{\text{wd}} - S'_{\text{perf}} + \sum_e R_e - \sum_e \phi_e - C, \quad \text{cash} + C + \sum_e U_e = \text{NAV}, \quad \text{cash} \geq$$

C is the collateral still at the venue, and S'_{perf} includes Φ_{epoch} . Each point carries `navUsdc`, `totalShares`, `boundarySlot`, `valuationSlot`, the itemized reconstruction, the transaction signature and slot of every source (execution, keeper fill, reveal, adjustment), and `commitmentSetHash`, the SHA-256 of the strategy's commitments sorted and concatenated. The API accepts a point only if its NAV equals the epoch's on-chain final NAV and its share count matches. Each point therefore checks the verifier's co-signed number against data anyone can fetch.

11. Attested statistics

Before any of its trades is revealed, a strategy's public page shows three numbers: return, maximum drawdown and an exposure band. They are published hourly into the strategy's own on-chain account and computed only from finalized epochs. They are quantized to 100 bps, delayed at least one hour, and written only by a transaction that both the registered enclave key and the registered independent verifier key sign. This chapter defines each number, the observation that produces it, the joint signing path, the checks the program applies, and what the label *attested* asserts.

11.1 What is published

Table 11.1 — The attested statistics and where they are stored.

Statistic	Definition	<code>post_stats</code> argument	Stored in <code>Strategy</code>
Return	Change in NAV per share since genesis, in bps, truncated toward zero to the quantum	<code>return_bps: i64</code> , a multiple of 100	<code>stats_return_bps</code>
Maximum drawdown	Largest peak-to-trough fall in NAV per share, in bps, floored to the quantum	<code>drawdown_bps: u16</code> , $\leq 10,000$, a multiple of 100	<code>stats_drawdown_bps</code> , never decreasing
Exposure band	Gross notional of open positions relative to NAV, in four bands	<code>exposure: u8</code> , 0 flat, 1 low, 2 moderate, 3 high	<code>stats_exposure</code>
Observation hour	The UTC hour the values describe	<code>observation_hour: i64</code> , a multiple of 3,600	<code>stats_observation_hour</code>
Evidence hash	SHA-256 of the canonical stats evidence	<code>evidence_hash: [u8; 32]</code> , non-zero	<code>stats_evidence_hash</code>
Signers and time	Enclave and verifier keys of the posting transaction	Transaction signers	<code>stats_executor</code> , <code>stats_verifier</code> , <code>stats_at</code> , <code>stats_slot</code>

The statistics are computed from NAV per share, not from NAV, so deposits and withdrawals change a strategy's size but not these figures.

11.2 The observation

Let t be the chain clock and δ the publication delay. The delay is a configured parameter (`STATS_PUBLICATION_DELAY_SECONDS`, local development value 3,600 seconds) and is at least one hour. The observation hour is

$$h = 3600 \left\lfloor \frac{t - \delta}{3600} \right\rfloor$$

The input set is the strategy's Final epochs whose `finalized_at` is at or before h , ordered by number. Let p_0 be the genesis NAV per share, $p_1 \dots p_n$ the final NAV per share of those epochs, and q the quantum (`STATS_QUANTIZATION_BPS`, a configured parameter that the program requires to be a multiple of 100 bps, local development value 100).

$$r = q \cdot \text{trunc} \left(\frac{1}{q} \cdot \frac{(p_n - p_0) \cdot 10\,000}{p_0} \right)$$

$$m = q \left\lfloor \frac{1}{q} \max_{k \leq n} \left\lfloor \frac{(P_k - p_k) \cdot 10\,000}{P_k} \right\rfloor \right\rfloor, \quad P_k = \max(p_0, p_1, \dots, p_k)$$

Return is truncated toward zero, so a fall of 150 bps publishes -100. Drawdown is floored.

For exposure, let g be the summed remaining notional of the strategy's exec accounts that were filled at the valuation slot of the last included epoch, and N_n that epoch's final NAV. Then $x = \lfloor g \cdot 10\,000 / N_n \rfloor$ and the band is:

- flat if $g = 0$;
- low if $x \leq X_{\text{low}}$;
- moderate if $x \leq X_{\text{mod}}$;
- high otherwise.

The high band reaches up to the strategy's declared maximum leverage, at most 30,000 bps. Both thresholds are configured parameters (`EXPOSURE_LOW_UPPER_BPS`, `EXPOSURE_MODERATE_UPPER_BPS`), common to all strategies, with local development values of 5,000 and 15,000 bps. Exposure is the one statistic that depends on the private mapping: computing g requires knowing which filled exec accounts belong to the strategy.

The evidence hash binds the values to the deployment and the enclave key. It is the SHA-256 of UTF-8 JSON with its keys in alphabetical order:

```
{
  "domain": "unseen-stats-v2",
  "executorPublicKey": "...",
  "exposureBand": "low",
  "genesisHash": "...",
  "maxDrawdownBps": "200",
  "observationHour": "...",
  "programId": "...",
  "returnBps": "700",
  "strategyId": "..."
}
```

All quantities are decimal integer strings. The example values are illustrative.

11.3 Why quantized and delayed

Exact statistics would leak attribution. A strategy's NAV moves by exactly its fees at each of its fills. If a return figure updated with every fill, an observer could match a 1 bp step in a strategy's return to a public, unattributed fill in the same minute, and with it the exec account to the strategy. Three rules close this.

1. **Finalized state only.** The input set contains only finalized epochs, so intra-epoch execution fees never produce a per-fill update. A strategy's statistics change at most once per daily epoch, and only after the verifier has recomputed and co-signed that epoch's NAV.
2. **Quantization.** Return and drawdown are published in steps of the quantum, which the program fixes at a minimum of 100 bps. A strategy's fees or small price moves within an epoch fall below the step.
3. **Delay.** An observation is posted at least an hour after its hour begins, and observation hours strictly increase. The public series is therefore coarse in time as well as in value.

The exposure band is coarse by construction: four common bands, with thresholds shared by every strategy, so a band does not identify a strategy's declared leverage. The posting cadence beyond the hourly series is a configured parameter (`STATS_ATTESTATION_CADENCE_SECONDS`). The local development executor polls every five seconds and posts each new hour once.

11.4 The joint signing path

The executor and the verifier each compute the statistics, and both must sign the same transaction.

1. **Executor proposal.** When its cadence elapses, the Sealed Executor computes each strategy's payload (hour, return, drawdown, band and evidence hash) from its chain snapshot and the strategy's openings. It builds the unsigned `post_stats` transaction. It sends the verifier an authorization request carrying the operation, payload, openings context and exact transaction bytes, and signs the request with the enclave key.
2. **Independent recomputation.** The verifier reads its own snapshot through the RPC it chose and checks the request's signature against the enclave key registered on-chain. It rejects an hour that is negative, not aligned to 3,600 seconds, younger than δ , or not later than the strategy's last published hour. It then recomputes the payload for that hour from its own snapshot and requires return, drawdown, band, hour and evidence hash all to match exactly.

3. **Transaction check.** The verifier decodes the prepared transaction. It confirms the compute-budget instruction, this operation with exactly these arguments, the expected program-derived accounts and the signer set. It then records the authorization in its journal and returns the transaction with its signature added.

4. **Submission.** The executor adds the enclave signature and sends the transaction.

A mismatch at any step stops the observation. It affects no other strategy and no trading.

11.5 What the program checks

Table 11.2 — Checks in `post_stats`.

Check	Effect
Enclave key registered in <code>EnclaveKey</code> and <code>Config</code> , unexpired and unrevoked	Only the attested executor can propose
Second signer equals <code>Config.verifier</code>	The independent verifier must co-sign
<code>drawdown_bps</code> $\leq 10,000$, and <code>drawdown_bps</code> and <code>return_bps</code> divisible by 100	Quantization floor of 100 bps
<code>exposure</code> ≤ 3	One of the four bands
<code>observation_hour</code> divisible by 3,600	Hourly observations only
<code>now</code> $\geq$ <code>observation_hour + 3,600</code>	At least one hour of delay
<code>observation_hour</code> $>$ <code>Strategy.stats_observation_hour</code>	Strictly increasing hours; no re-posting
<code>evidence_hash</code> $\neq 0$	Evidence is always bound
Stored drawdown becomes <code>max(stored, drawdown_bps)</code>	Published maximum drawdown never decreases

The stored drawdown also feeds a protocol rule. `request_deposit` requires `stats_drawdown_bps` $<$ `drawdown_kill_bps`. The kill level is declared at creation and bounded by the protocol ceiling `Config.max_drawdown_bps`, whose local development value is 2,000 bps. Once an attested drawdown reaches the kill level, the strategy stops accepting deposits, and because the stored maximum never falls, it stays closed. Separately, when the verifier authorizes orders it computes a fresh drawdown in NAV per share against genesis, the high-water mark and its own observed peak, and can issue a signed risk-reduction certificate with reason `drawdown` (see §5 The Sealed Executor and §13 Security design).

11.6 What "attested" means

The API labels a statistic *attested* only after it has checked the posting transaction itself. For the strategy's `stats_slot`, the chain adapter performs four checks:

1. It fetches the confirmed, successful transaction that called `post_stats`.
2. It verifies every signature in that transaction.
3. It requires the signer set to include both the recorded `stats_executor` and the recorded `stats_verifier`.
4. It decodes the instruction and requires its hour, return, drawdown, exposure and evidence hash to equal the stored values.

Only then does it attach a `StatsProofV2`: version, genesis hash, program ID, strategy, observation hour, `quantizationBps: "100"`, evidence hash, enclave and verifier keys, transaction signature, slot and `verified: true`.

Before the indexer projects any snapshot, it recomputes the evidence hash from the published values and the deployment. It checks that the values are multiples of 100 bps, that the hour is aligned and at least an hour old, and that the slot is finalized. It rejects the whole snapshot otherwise. A strategy without a verified proof shows `verification: "pending"` with zero values, never an unproven number. The contract's label set also reserves `publicly_recomputed` and `stale`. The hourly series (`GET /v1/strategies/:strategyId/stats`, interval `hour`) stores one point per strategy per hour, and only for verified proofs.

Attested therefore asserts three things:

- two registered keys, held by different parties, signed one transaction carrying these exact values;
- each party computed the values independently from finalized epochs whose NAV the verifier had already recomputed and co-signed (see §8 Epochs and settlement);
- the program enforced the format, quantization, delay, strictly increasing hours and the monotonic drawdown.

The figures are also open to check without either party. Return and drawdown are functions of public Epoch accounts, so anyone can recompute them at any time. The exposure band becomes checkable when the strategy's openings for the referenced epoch are revealed (see §10 Reveal), and from then on the whole series can be checked against chain data.

12. On-chain programs

This chapter is the reference for what the chain enforces. It lists every account the `unseen` program owns, with its PDA seeds and main fields, and every instruction with its signer and the checks it applies. It then covers the anonymous risk envelope that `execute` evaluates, transaction format and address lookup tables, and the three governance roles: the admin with its 48-hour timelock, the guardian, and the upgrade authority. Earlier chapters explain why each rule exists. Here each rule is stated in the terms the program checks it.

12.1 Programs and builds

`unseen` is an Anchor 0.31.1 program compiled to SBF. It holds all protocol state: configuration, strategies, the share ledger, epochs, the pooled vault, exec accounts, commitments, reveal certificates and statistics. Venue access goes through a venue adapter called by CPI. The local and devnet adapter is `fake-venue`, which has the production request shape: a request account owned by the exec PDA, a keeper fill at the oracle price in a separate transaction, and open and close fees. In production the adapter is the Jupiter Perps position-request path, with the exec PDA as position owner (see §7 Venue: Jupiter Perps).

The build is feature-gated. A `production` build takes its program ID from the configured `UNSEEN_PROGRAM_ID`. A `local-testing` build carries the local ID, the accelerated logical clock and `local_advance`. The two are mutually exclusive by compile error, and a guard rejects a non-local ID combined with `local-testing`. A `devnet-venue` build runs the fake venue at clock scale 1 so the whole flow can run on devnet, which has no Jupiter Perps. The IDL is generated from the Rust account and instruction definitions.

Amounts are `u64` USDC base units, shares are `u128` with nine decimals, and every product goes through checked `u128` or `i128` arithmetic in the shared domain crate. Any overflow aborts the transaction with `overflow`. Anchor checks owners, seeds, signers, exact mints and token authorities on every account. Account contexts are boxed to stay within SBF stack limits.

12.2 Accounts

Table 12.1 — Program-derived accounts. Integers in seeds are little-endian.

Account	Seeds	Created by	Lifetime
Config	"config"	initialize	Singleton
EnclaveKey	"enclave", public key	initialize, activate_enclave	One per registered key; revocable
Pool	"pool", venue program	initialize	One per venue
pool_vault (SPL token)	"pool_vault"	initialize	Authority Pool, mint Config.mint
Strategy	"strategy", id[32]	create_strategy	Permanent; can be frozen as dormant
Share	"share", strategy, owner	create_strategy for the leader, request_deposit	Permanent, non-transferable
Epoch	"epoch", strategy, number u64	propose_epoch	Permanent; final or voided
WithdrawalTicket	"withdrawal", strategy, sequence u64	request_withdraw	Permanent FIFO tombstone
ChallengeReceipt	"challenge", epoch, challenger	challenge	One per challenger per epoch
Exec	"exec", pool, index u64	execute	Shrunk to a tombstone by reclaim_exec
exec_vault (SPL token)	"exec_vault", exec	execute	Authority Exec; closed by reclaim_exec
Venue request (venue program)	"request", exec	Venue CPI in execute	Owned by Exec; closed by reclaim_exec

Account	Seeds	Created by	Lifetime
CommitmentReceipt	"commitment", commitment[32]	execute	Permanent; holds the archive after cleanup
RevealPeriod	"reveal_period", pool, day i64	seal_reveal_period	Immutable
LeaderInvitation	"leader_invitation", leader	invite_leader	Single use, expiring
Eligibility	"eligibility", owner	set_eligibility	Expiring screening outcome
ExecutionTombstone	The original Exec address	reclaim_exec	40 bytes: discriminator and commitment

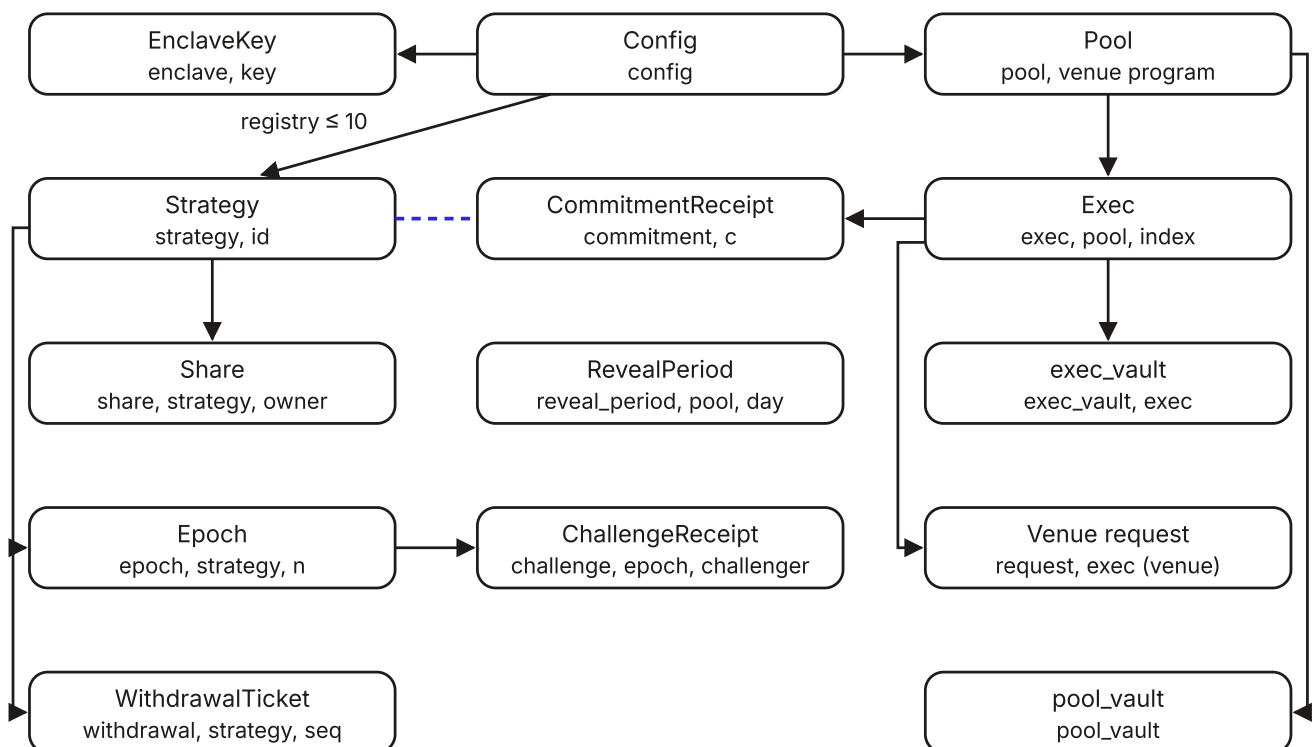


Figure 12.1 — Account relationships. Arrows point from an account to the accounts whose seeds or fields it anchors. The strategy family on the left and the execution family on the right share no account reference. The dashed line is the only link between a strategy and an execution: the opening of the commitment, published at reveal. RevealPeriod is keyed by the pool. The wallet-keyed LeaderInvitation and Eligibility are written by the admin and omitted.

Table 12.2 — `Config` fields.

Group	Fields
Roles	<code>admin</code> , <code>guardian</code> , <code>verifier</code> , <code>enclave</code> , <code>enclave_expiry</code> , <code>enclave_revoked</code> , <code>paused</code>
Timelocked changes	<code>pending_enclave</code> , <code>pending_enclave_expiry</code> , <code>pending_measurement</code> , <code>pending_report</code> , <code>enclave_activate_at</code> ; <code>pending_verifier</code> , <code>pending_guardian</code> , <code>roles_activate_at</code>
Deployment	<code>deployment_genesis</code> , <code>mint</code> , <code>venue</code> , <code>protocol_fee_vault</code> (the <code>protocol</code> PDA's USDC account), <code>genesis_nav</code>
Buyback	<code>buyback_mint</code> , <code>buyback_swap_program</code> , their pending values and activation time, <code>total_buyback_usdc_spent</code> , <code>total_buyback_tokens_burned</code> , <code>last_buyback_slot</code>
Registry and caps	<code>registry[10]</code> (ordered Strategy addresses), <code>strategies</code> (count), <code>platform_aum</code> , <code>pending_deposits</code>
Common wire policy	<code>common_collateral_ratio_bps</code> (3,334–10,000), <code>common_wire_slippage_bps</code> ($\leq 10,000$)
Risk and schedule	<code>max_drawdown_bps</code> , <code>max_reveal_delay_ever</code> , <code>epoch_duration_slots</code> , <code>proposal_deadline_slots</code> , <code>epoch_timeout_seconds</code> , <code>execution_fee_epoch_cap_bps</code>
Challenge and venue	<code>challenge_seconds</code> , <code>challenge_bond</code> , <code>retry_slots</code>
Clock	<code>clock_scale</code> (1 in production), <code>real_genesis</code> , <code>logical_genesis</code> , <code>clock_offset</code>

Table 12.3 — `Pool` fields. Arrays are indexed by market: 0 SOL, 1 ETH, 2 wBTC.

Field	Meaning
<code>venue</code> , <code>vault</code>	The venue program and the pool vault address
<code>cash</code>	Executable cash held for all strategies in <code>pool_vault</code>
<code>withdrawal_reserves</code>	Sum of every strategy's protected 20% withdrawal reserve
<code>head</code> , <code>count</code>	Commitment hash-chain head and length
<code>open_notional[3]</code>	Aggregate notional per market; pending requests count
<code>long_notional[3]</code> , <code>short_notional[3]</code>	Notional per market side
<code>max_long_entry[3]</code> , <code>min_short_entry[3]</code>	Conservative entry extrema, reset only when that side is flat
<code>locked_collateral</code>	Original principal of all pending and filled requests
<code>total_realized_losses</code>	Cumulative realized losses plus execution fees
<code>flow_minute[3]</code> , <code>flow_notional[3]</code>	Gross flow per market in the current logical minute
<code>fee_day</code> , <code>execution_fees</code>	Execution fees charged in the current UTC day
<code>protocol_claim</code>	Protocol pull claim
<code>last_flat_at</code>	Last clock at which every market's open notional was zero

Table 12.4 — `Strategy` fields.

Group	Fields
Identity	<code>id</code> , <code>name</code> (≤ 64 bytes), <code>kind</code> (0 human, 1 agent), <code>leader</code> , <code>signal_key</code>
Declared policy, immutable	<code>performance_fee_bps</code> ($\leq 2,500$), <code>reveal_delay</code> (86,400–2,592,000 s), <code>markets</code> (bitmask 1–7: bit 0 SOL, bit 1 ETH, bit 2 wBTC), <code>max_leverage_bps</code> ($\leq 30,000$), <code>drawdown_kill_bps</code> ($\leq \text{Config.max_drawdown_bps}$)
Shares and NAV	<code>aum</code> , <code>total_shares</code> , <code>leader_shares</code> , <code>last_nav</code> , <code>high_water_mark</code> , <code>liquid_cash</code> , <code>withdrawal_reserve</code> , <code>loss_checkpoint</code>
Requests and queue	<code>pending_deposits</code> , <code>pending_withdraw_shares</code> , <code>pending_leader_withdraw</code> , <code>pending_requests</code> , <code>future_requests</code> , <code>future_request_epoch</code> , <code>unsettled_requests</code> , <code>withdraw_sequence</code> , <code>next_withdraw_sequence</code>
Schedule	<code>created_at</code> , <code>shadow_ends_at</code> (creation + 30 days), <code>waiver_ends_at</code> (creation + 90 days, or 0), <code>next_epoch_at</code> , <code>next_epoch_slot</code> , <code>epoch_number</code> , <code>active_epoch</code> , <code>last_final_epoch</code>
Totals and claims	<code>cumulative_deposits</code> , <code>cumulative_withdrawals</code> , <code>cumulative_performance_fees</code> , <code>leader_fee_claim</code>
Statistics	<code>stats_return_bps</code> , <code>stats_drawdown_bps</code> , <code>stats_exposure</code> , <code>stats_observation_hour</code> , <code>stats_evidence_hash</code> , <code>stats_executor</code> , <code>stats_verifier</code> , <code>stats_at</code> , <code>stats_slot</code>
Lifecycle	<code>active_until_sequence</code> (<code>u64::MAX</code> while active)

A Strategy holds no exec account, position or signal sequence. Its public accounting is the final checkpoint of each epoch.

Table 12.5 — `Share` and `Epoch` fields.

Account	Group	Fields
Share	Holding	strategy, owner, shares (u128), cost_basis
	Request	pending_deposit, pending_withdraw, pending_epoch, request_sequence (owner nonce), queue_sequence (FIFO position), requested_at
	Settlement	withdrawal_requested_shares, withdrawal_settled_usdc, withdrawal_accounting_epoch, last_settled_epoch, claimable_withdraw
Epoch	Identity	strategy, number, state (0 Proposed, 1 Challengeable, 2 Final, 3 Voided)
	Schedule	boundary_at, boundary_slot, proposed_at, proposal_slot
	Immutable proposal	proposed_nav, proposed_liquid_cash, nav_per_share, total_shares, pre_fee_shares, prices[3], cumulative_deposits, cumulative_withdrawals, cumulative_performance_fees, remaining_withdrawal_shares, requested_withdrawals, unsettled_requests
	Verification	evidence_hash, cosigned, cosigned_at, challenge_ends_at
	Challenge and review	challenged, challenger, bond, bond_claim, bond_claimed, guardian_review, guardian_resolved
	Finalization	finalized_at, final_nav, performance_fee, withdrawal_budget, settled_withdrawals, deposits

Table 12.6 — Exec and CommitmentReceipt fields.

Account	Fields
Exec	pool, index, executor, commitment, previous_head, resulting_head, chain_sequence, market, side, notional, collateral, original_notional, original_collateral, execution_fee, opening_execution_fee, policy_evidence_hash, last_policy_hash, operation_sequence, created_at, created_slot, filled_at, closed_at, state (0 requested, 1 filled, 2 closed, 3 cancelled), revealed
CommitmentReceipt	commitment, exec, created_at, created_slot, revealed, archived, archive: executor, pool, index, commitment, both heads, chain sequence, market, side, original notional and collateral, execution fees, every timestamp and slot, state, both policy hashes, operation sequence, and the venue's entry price, fee, realized PnL and returned amount

An Exec account never stores a strategy ID or a signal sequence. Its market, side, notional and collateral are public, and the collateral follows the deployment's common ratio, so it carries no leader-specific leverage. Partial reductions keep the account and its entry price.

Table 12.7 — Smaller accounts.

Account	Fields
EnclaveKey	public_key, measurement_hash, attestation_report_hash, expires_at, revoked, registered_slot
RevealPeriod	pool, day, sealed_flat_at, mature_at
WithdrawalTicket	strategy, sequence, owner, canceled
ChallengeReceipt	epoch, challenger
LeaderInvitation	leader, expires_at, used, founding
Eligibility	owner, allowed, expires_at
ExecutionTombstone	commitment

12.3 The anonymous execution envelope

`execute` must bound risk without knowing which strategy owns the new position. It therefore requires the caller to pass the complete `Config.registry`, in order and owned by the program, as remaining accounts. It then evaluates every active strategy as if the whole pool's risk were its own.

Let n be the candidate notional, and let U be the summed loss upper bound over the three markets after the candidate is added:

$$U = \sum_m \left\lceil \frac{\ell_m (\hat{e}_m - x_m)^+}{\hat{e}_m} \right\rceil + \left\lceil \frac{s_m (x_m - \check{e}_m)^+}{\check{e}_m} \right\rceil$$

Here ℓ_m and s_m are long and short notional, x_m the oracle price, and $\hat{e}_m$ and $\check{e}_m$ the highest long and lowest short entry bounds. Entry bounds widen the price by the common wire slippage: $\lceil x(10\,000 + \sigma)/10\,000 \rceil$ for longs and $\lfloor x(10\,000 - \sigma)/10\,000 \rfloor$ for shorts. Profitable positions never offset losses, since they may belong to different owners. For each active strategy s :

$$E_s = \max(0, A_s - W_s - (\Lambda - \lambda_s) - U - \lceil 16 \cdot \sum_m (\ell_m + s_m)/10\,000 \rceil)$$

Here A_s is its AUM, W_s its pending withdrawal shares valued at the last final NAV, Λ the pool's total realized losses and execution fees, and λ_s its checkpoint at its last finalization. The 16 bps term bounds outstanding venue and execution fees, the candidate included. The program then requires:

$$\sum_m \text{open}_m + n \leq \min_s \left\lfloor \frac{E_s \cdot \text{lev}_s}{10\,000} \right\rfloor, \quad \text{locked} + c \leq \min_s (E_s - R_s)$$

Here $c = \lceil n \cdot \rho/10\,000 \rceil$ is the candidate's exact collateral at the common ratio ρ , and R_s the strategy's withdrawal reserve. Every active strategy must also allow the market. A strategy frozen by `deactivate_zero_strategy` is skipped, and an active strategy with zero equity blocks new pool risk. The smallest active bound limits the whole pool, so any hidden strategy's exposure is bounded by its own conservative ceiling. The per-strategy cash check, under which another strategy's idle cash cannot fund an order, belongs to the independent verifier (see §5 The Sealed Executor).

12.4 Instructions

The instructions are grouped by role below. Signer names refer to §12.5.

Table 12.8 — Governance and roles.

Instruction	Signer	Checks and effects
<code>initialize(settings)</code>	Admin, equal to the ProgramData upgrade authority	Genesis NAV $\geq 1,000,000$; $0 < \text{max drawdown} < 10,000$ bps; collateral ratio 3,334–10,000; slippage $\leq 10,000$; epoch slots, proposal deadline, epoch timeout, fee cap, challenge window, bond, retry slots and key lifetime all positive; genesis, measurement and report hashes non-zero; <code>clock_scale = 1</code> outside local testing. Creates Config, Pool, EnclaveKey and <code>pool_vault</code> . The protocol fee vault must use the configured mint
<code>schedule_enclave(key, expiry, measurement, report)</code>	Admin	New key differs from the current one; hashes non-zero; expiry later than now + 172,800 s; activation at now + 172,800 s
<code>activate_enclave</code>	Any payer	Now $\geq \text{enclave_activate_at}$. Creates the new EnclaveKey, switches <code>Config.enclave</code> and its expiry, clears revocation
<code>schedule_roles(verifier, guardian)</code>	Admin	Both keys non-default; activation at now + 172,800 s
<code>activate_roles</code>	Admin	Now $\geq \text{roles_activate_at}$; installs both keys
<code>set_pause(paused, revoke_enclave)</code>	Guardian	Immediate. Revocation marks both Config and the EnclaveKey revoked
<code>set_eligibility(allowed, expires_at)</code>	Admin	Writes a wallet's screening and allowlist outcome
<code>invite_leader(expires_at, founding)</code>	Admin	Expiry in the future; single-use invitation; the founding flag gates the 90-day waiver
<code>local_advance(seconds)</code>	Admin	Local-testing builds with <code>clock_scale > 1</code> only

Table 12.9 — Strategies and participants.

Instruction	Signer	Checks and effects
<code>create_strategy(...)</code>	Invited leader	Not paused; fewer than 10 strategies; invitation unused and unexpired, founding only if invited as founding. $0 < \text{stake} \leq 250,000$ USDC; name ≤ 64 bytes; kind < 2 ; fee $\leq 2,500$ bps; delay 86,400–2,592,000 s; markets 1–7; $0 < \text{leverage} \leq 30,000$ bps; $0 < \text{drawdown kill} \leq \text{protocol maximum}$. Platform AUM plus pending deposits plus stake $\leq 2,000,000$ USDC; stake mints a positive number of shares at genesis NAV. Raises <code>max_reveal_delay_ever</code> , appends to the registry, starts the 30-day shadow period
<code>rotate_signal_key(key)</code>	Leader	Strategy's recorded leader
<code>request_deposit(amount, nonce)</code>	Owner	Not paused; strategy active with positive NAV; attested drawdown below the kill level; shadow period over. Eligibility allowed and unexpired; amount > 0 . Strategy AUM plus pending plus amount $\leq 250,000$ USDC, platform $\leq 2,000,000$ USDC. Leader equity at last NAV, net of its pending withdrawal, $\geq 5\%$ of the reserved total. Nonce = <code>request_sequence + 1</code> , no other pending request. Escrows the tokens
<code>request_withdraw(shares, nonce)</code>	Owner	$0 < \text{shares} \leq \text{held}$; no pending request; nonce; value at last NAV > 0 ; a leader keeps $\geq 5\%$ afterwards. Creates the next WithdrawalTicket
<code>cancel_request(nonce)</code>	Owner	Nonce; a pending request exists. A deposit becomes a claimable refund; a withdrawal's ticket is marked canceled. Counters of an active proposal are adjusted in place. Needs no token account
<code>emergency_withdraw(shares, nonce)</code>	Owner	Prices at the last final epoch; owner at the FIFO head; payout $\leq$ the remaining 20% budget, pool cash, the strategy's liquid cash and its reserve; leader 5% check including pending flows. Refunds the owner's pending deposit. Voids an active proposal atomically only on a successful, positive payout

Instruction	Signer	Checks and effects
<code>claim_withdrawal</code>	Owner	Claim > 0; destination owned by the owner, on the configured mint
<code>claim_leader_fee</code>	Leader	Claim > 0; destination owned by the leader
<code>claim_bond</code>	Recorded challenger	Bond claim > 0

Table 12.10 — Execution. Every instruction here except `guardian_close` needs both the enclave and the verifier.

Instruction	Checks and effects
<code>execute(index, commitment, market, side, notional, collateral, policy_hash, valid_until_slot)</code>	Not paused; market < 3, side < 2; $0 < \text{notional} \leq 250,000$ USDC; collateral equals $[\text{notional} \cdot \text{ratio} / 10,000]$ and is ≥ 10 USDC; policy hash non-zero; slot $\leq \text{valid_until_slot}$. Envelope (§12.3) over the full registry; per-market flow in the logical minute $\leq 250,000$ USDC; unprotected cash $\geq$ collateral. Creates Exec, CommitmentReceipt and <code>exec_vault</code> (creation fails on reuse); extends the hash chain; moves collateral pool $\rightarrow$ exec vault; venue request expires at slot + <code>retry_slots</code> with the common slippage; emits <code>ExecutionCommitted</code>
<code>record_fill</code>	Exec requested and venue request filled. 2 bps fee under the daily cap, paid from unprotected cash to the protocol fee vault; state filled; <code>ExecutionFilled</code> . Also records a keeper fill that landed on its own
<code>partial_close(reduce, op_seq, policy_hash, valid_until_slot)</code>	Filled; $0 < \text{reduce} < \text{notional}$; <code>op_seq</code> = previous + 1; policy hash non-zero; slot valid; flow limit. Returned funds go to the pool, proportional original margin is released, losses are booked, 2 bps on the reduction goes to the protocol claim; <code>ExecutionAdjusted</code>
<code>close_exec(cancel, op_seq, policy_hash, valid_until_slot)</code>	<code>op_seq</code> and slot as above. Close: filled, policy hash non-zero, flow limit, 2 bps on the remaining notional. Cancel: requested, venue request unfilled with zero fees and PnL, returned $\geq$ original collateral. The whole exec-vault balance returns to the pool; <code>last_flat_at</code> updates when every market is flat; state closed or cancelled
<code>guardian_close(cancel)</code> (guardian only)	Enclave revoked or expired; state matches. Funds return only to the pool; no execution fee and no flow limit; zero policy hash

Table 12.11 — Epochs and settlement.

Instruction	Signer	Checks and effects
<code>propose_epoch(number, nav, liquid_cash)</code>	Enclave	Not paused; number = <code>epoch_number + 1</code> ; no active epoch; now $\geq$ <code>next_epoch_at</code> ; slot $\leq$ <code>next_epoch_slot + proposal_deadline_slots</code> ; no unsettled requests; liquid cash $\leq$ NAV. Captures NAV per share, prices, supply, capital checkpoints and pending withdrawals; flags guardian review if NAV per share moved by more than 1,500 bps
<code>cosign_epoch(expected_nav, liquid_cash, evidence_hash)</code>	Verifier	Values equal the proposal; evidence non-zero; slot $\leq$ <code>proposal_slot + deadline</code> ; Proposed $\rightarrow$ Challengeable; window closes at now + <code>challenge_seconds</code>
<code>challenge</code>	Anyone	Challengeable, unchallenged, inside the window; escrows <code>Config.challenge_bond</code> ; one receipt per challenger per epoch
<code>resolve_challenge(approve)</code>	Guardian	Not final. Approve: bond to the protocol claim, challenge cleared, review resolved. Reject: bond to the challenger's claim, epoch stays blocked
<code>finalize_epoch</code>	Anyone	Not paused; co-signed; window elapsed; not challenged; review resolved if flagged; the strategy's active epoch. Fee, cash and reserve checks (§9.1); leader compensation; final NAV, high-water mark, next schedule; <code>EpochFinalized</code>

Instruction	Signer	Checks and effects
<code>settle_share</code>	Anyone	Latest final epoch, no active proposal; share pending at or before it and not yet settled. Mints deposits at final NAV per share; pays withdrawals in FIFO order within min(budget left, liquid cash, reserve, pool cash); releases zero-value requests; carries remainders
<code>void_epoch</code>	Guardian, or anyone once paused, revoked, expired or past <code>epoch_timeout_seconds</code>	Not final. Unresolved bond refunded to the challenger's claim; Voided; next schedule set
<code>reset_epoch_schedule</code>	Anyone	No active epoch; slot deadline or timeout passed; reschedules without changing NAV
<code>advance_withdrawal_queue</code>	Anyone	Ticket at the queue head is canceled
<code>deactivate_zero_strategy</code>	Anyone	AUM 0, no epoch or pending flows, whole pool flat; freezes <code>active_until_sequence</code> at <code>Pool.count</code>

Table 12.12 — Reveal, statistics and cleanup.

Instruction	Signer	Checks and effects
<code>seal_reveal_period_certified(day, max_reveal_delay, evidence_hash)</code>	Enclave and verifier	Day ended; $24\text{ h} \leq \text{delay} \leq 30\text{ d}$ and $\leq$ the registered maximum; evidence hash non-zero; <code>mature_at</code> = day end + delay (§10.4)
<code>seal_reveal_period(day)</code>	Any payer	Fallback for a day with no certificate: registered maximum delay > 0 ; flat fence $\geq$ day end (§10.4)
<code>reveal_v2(...)</code> , <code>reveal(...)</code>	Enclave	Terminal, mature, commitment matches; Table 10.1
<code>reveal_evidence(page, pages, data)</code>	Enclave	Revealed and terminal; page $<$ pages; data ≤ 400 bytes; <code>RevealEvidence</code>
<code>reclaim_exec(index)</code>	Anyone	Terminal, revealed, exec vault empty. Archives, closes the venue request and vault to the pool, leaves a tombstone, returns excess rent to the pool
<code>post_stats(...)</code>	Enclave and verifier	Table 11.2
<code>buyback_and_burn(min_out, route_data)</code>	Anyone	Buyback mint set; claim moved to the protocol fee account; configured swap program; route accounts are the <code>protocol</code> PDA's own; at least <code>min_out</code> received; whole \$UNSEEN balance burned (§9.8)
<code>schedule_buyback(mint, swap_program)</code> , <code>activate_buyback</code>	Admin	48-hour timelock between the two

The program emits eight events: `StrategyCreated`, `ExecutionCommitted`, `ExecutionFilled`, `ExecutionAdjusted`, `EpochFinalized`, `OpeningRevealed`, `OpeningRevealedV2` and `RevealEvidence`. Before reveal, none of them connects an exec account to a strategy (see §15 Interfaces).

12.5 Signer predicates

- **Enclave.** The signer equals `EnclaveKey.public_key` at `["enclave", signer]`, that record is unrevoked and unexpired, and the signer equals `Config.enclave` with `Config.enclave_revoked` false and the clock before `Config.enclave_expiry`. The enclave pays rent for the accounts `execute` creates, and `reclaim_exec` returns the reclaimable part of it to the pool.
- **Verifier.** A second signer constrained to `address = Config.verifier`. Both the enclave and the verifier sign execution, keeper-fill recording, reductions, closes and statistics.
- **Guardian.** The signer is `Config.guardian`.
- **Admin.** The signer is `Config.admin`.
- **Owner.** Share accounts carry `has_one = owner`. Owner token accounts must belong to the signer and use `Config.mint`, so claims pay only to the owner. Wallet signatures alone authorize deposits, withdrawal requests and claims. No service holds a follower key.

The enclave cannot withdraw. Its transfers run only pool → exec vault → approved venue, with returns going to the fixed pool vault.

12.6 Transactions and address lookup tables

Solana limits a transaction to 1,232 bytes and 64 accounts, and allows 1.4 million compute units.

`execute` names 16 accounts and passes up to 10 registry Strategy accounts as remaining accounts. That is 26 before the venue adapter's own accounts, each a 32-byte key in a static message, which is why `execute` with the Jupiter CPI needs an address lookup table. The protocol initialization creates the table. It holds the accounts every execution repeats:

- `Config`, `Pool`, `pool_vault` and the USDC mint;
- the current `EnclaveKey` record;
- the token and system programs;
- the venue program and its fixed accounts: for Jupiter Perps, the configured pool, custodies and price accounts;
- every registered Strategy account.

Signers (enclave and verifier) and the accounts created fresh for each execution (Exec, CommitmentReceipt, `exec_vault` and the venue request) stay static keys, so no table entry ever announces a future exec account. The compute budget per `execute` is estimated at no more than 400,000 compute units.

Both signers check every prepared message before signing it. The first instruction must set a compute-unit limit above zero and at most 1,400,000. In production that limit must equal the configured `COMPUTE_UNIT_LIMIT`, followed by the configured compute-unit price and, when a tip is configured, a transfer of exactly `TIP_LAMPORTS` to the configured tip account. The program instruction must carry the expected arguments and program-derived accounts. Fees and tips are part of the message before either authority signs, and sending never changes the bytes. The sender is configured as plain RPC, Jito or Helius Sender. Devnet uses RPC, a Jito bundle needs a tip of at least 1,000 lamports, and Helius Sender at least 5,000. A message that loads addresses

from a lookup table is accepted only through an address resolver that reads the table from the signer's own RPC. No key is ever guessed.

12.7 Admin, timelock and upgrade authority

The admin is a Squads multisig, configured as `ADMIN_MULTISIG_ADDRESS` under `SQUADS_PROGRAM_ID`. `initialize` binds it. The instruction requires the admin signer to equal the upgrade authority recorded in the program's ProgramData account, so an arbitrary first caller cannot become admin, and the multisig that controls upgrades is the multisig that administers the protocol.

The program enforces a 172,800-second (48-hour) timelock on the two changes that alter who may act: registering an enclave key, together with its TDX measurement hash, attestation report hash and expiry, and rotating the verifier and guardian. Both are two-step. `schedule_*` writes the pending values into Config, where anyone can read them, and the change can take effect only once the timelock has passed. A new enclave key must also outlive the timelock. The admin acts immediately only on admission data: `set_eligibility` for screening and allowlist outcomes, and `invite_leader` for invitations.

Beyond these, no instruction lets the admin move pool funds, sign as the enclave or verifier, or edit a strategy's declared parameters. The protocol's economic constants are fixed in code:

- the strategy and platform caps of 250,000 and 2,000,000 USDC;
- the 5% leader stake;
- the 2 bps execution fee and the 1/5 protocol share;
- the 30-day shadow period and the 90-day waiver;
- the 1,500 bps review threshold;
- the 20% reserve;
- the 250,000 USDC per-minute flow limit;
- the ten-entry registry.

The deployment settings written by `initialize`, such as the common collateral ratio, the challenge window and bond, and the retry slots, have no setter either. Changing any of them requires a program upgrade. The upgrade authority is the same admin multisig under its 48-hour timelock, with no durable-nonce pre-signing of admin actions. The brief cites a precedent for that rule: Drift lost about 285 million USD to pre-signed admin actions under a zero-timelock council. A production deployment compares the SHA-256 of the deployed program bytes with the locally built artifact, and production artifacts are built only against a configured non-local program ID.

12.8 The guardian

The guardian is a second multisig (`GUARDIAN_MULTISIG_ADDRESS`) with fast, narrow powers.

- **Pause.** `set_pause` stops `execute`, `propose_epoch`, `finalize_epoch`, `create_strategy` and `request_deposit` at once. Closes, reductions, fill recording, withdrawal requests, settlement of

already final epochs, cancellations, claims, reveals and emergency withdrawals at the last final NAV stay open.

- **Revoke.** `set_pause(_, true)` revokes the enclave key in Config and in its EnclaveKey record. Every enclave-signed instruction fails, `guardian_close` becomes available and `void_epoch` becomes permissionless.
- **Resolve.** `resolve_challenge` settles a challenge or a review flag on a NAV move above 15%, and `void_epoch` retires a proposal at any time.
- **Rescue.** `guardian_close` unwinds positions once the enclave key is revoked or expired. Funds return only to the pool vault, free of execution fees and flow limits.

The guardian cannot reinstate a revoked key, since a replacement key must pass the admin's 48-hour schedule. It cannot rotate roles, move funds anywhere but the pool, sign executions or statistics, or make final an epoch the verifier has not co-signed. A void keeps the last final NAV, and the strategy's existing 20% reserve stays available to emergency withdrawals.

12.9 Errors

Table 12.13 — Program errors.

Error	Meaning
Config	Protocol configuration invalid
LocalOnly	Local test facility unavailable in this build
Paused	Protocol paused, or the strategy invitation limit is reached
Bounds	Declared protocol bounds violated
Cap	Capacity reservation exceeds a cap
LeaderStake	Leader stake would fall below 5%
Unauthorized	Wrong signing authority or account
EnclaveExpired	Enclave key expired or revoked
Replay	Signal or request replay, or a stale sequence
DepositsClosed	Deposits closed for this strategy
Ineligible	Follower not eligible
State	Wrong lifecycle state
Liquidity	Insufficient idle pool liquidity
Verification	Verification or challenge gate not satisfied
Queue	Withdrawal queue order not reached
Commitment	Commitment preimage mismatch
TooEarly	A delay, maturity or deadline condition is not met
Timelock	Admin timelock has not passed
Overflow	Exact integer arithmetic overflow

13. Security design

Unseen divides authority among parties that hold different keys and run on different machines. For each party this chapter states what it can do, what it cannot do, and the mechanism that draws the line: a program check, a second required signature, a timelock, an encoding that binds a message to one deployment, or a rule about where data is stored. Account layouts and full instruction checks are in §12 On-chain programs, the epoch lifecycle in §8 Epochs and settlement, and byte encodings in §16 Parameters and encodings.

13.1 The custody loop

USDC moves only along the paths in Figure 13.1. The pool vault is a token account at the PDA `["pool_vault"]` whose authority is the `pool` PDA, so only the program signs transfers out of it, and every instruction that does so fixes the destination through account constraints.

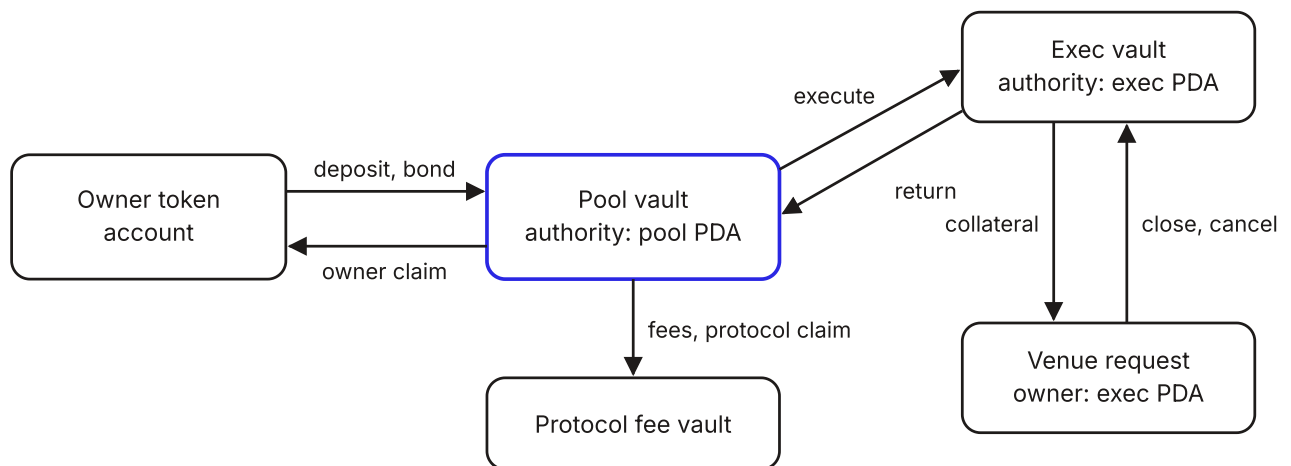


Figure 13.1 — The only token paths the program allows. Exec vaults return to the pool vault; payouts to people are claims the owner pulls.

Deposits, leader stakes and challenge bonds enter from a token account whose authority is the signer. `execute` moves exact collateral to a new exec vault and on into the venue request; partial closes, closes, cancellations and guardian closes return everything released to the pool vault. Execution fees and protocol claims go to the protocol fee vault fixed in `Config` at initialization. Withdrawals, refunds, leader fees and returned bonds are claimable balances, paid by `claim_withdrawal`, `claim_leader_fee` and `claim_bond` to a token account in the configured USDC mint whose authority is the claimant. No instruction pays an account chosen by the executor, verifier, guardian, admin or API.

13.2 Keys

Key	Holder and location	Authorizes
Leader wallet	Leader's wallet	Strategy creation, stake requests, signal-key rotation, leader-fee claims, leader-view proofs
Signal key	Leader's client or agent host	Version 2 signals only
Follower wallet	Follower's wallet	Requests, cancellations, emergency withdrawals, claims, challenges
Enclave key (Ed25519)	Generated inside the Intel TDX trust domain; persisted on its encrypted volume	Executions, fills, reductions, closes, proposals, stats, reveals, signal receipts
Encryption key (P-256)	Derived inside the trust domain from the enclave key	Decryption of sealed signals
Verifier key	Verifier service only	<code>cosign_epoch</code> ; the second signature on executions, fills, reductions, closes and stats; admissions; risk certificates; counter replies
Guardian	Multisig (<code>GUARDIAN_MULTISIG_ADDRESS</code>)	Pause, revocation, challenge resolution, voids, <code>guardian_close</code>
Admin	Squads multisig (<code>ADMIN_MULTISIG_ADDRESS</code>), also the upgrade authority	Initialization, key scheduling, role rotation, eligibility, invitations, upgrades

Table 13.1 — Signing keys. The API and the indexer hold none.

13.3 Leader

A leader creates one strategy per invitation: `create_strategy` consumes an unused, unexpired `LeaderInvitation` and checks every parameter against protocol bounds (§3 Strategies, shares and the pool). Afterwards the leader can rotate the signal key, sign signals, withdraw stake, claim fees and read the private console.

A leader cannot change the reveal delay, markets, leverage or drawdown level after creation; the signal key is the only mutable leader field. The stake cannot fall below 5% of equity: deposits require

$$V_{\text{leader}} \cdot 10000 \geq (\text{AUM} + \text{pending deposits} + \text{amount}) \cdot 500$$

with V_{leader} valued net of the leader's pending withdrawal, and leader withdrawals apply the same inequality to the remaining position.

Trades outside declared limits are refused at three layers. The executor rejects signals with undeclared or repeated markets, more than three targets or gross weight above declared leverage. The verifier replays the signed signal, sizing, reservations and the strategy's own cash before signing. The program bounds total pool open notional by the smallest active strategy's conservative equity times its declared leverage (§5 The Sealed Executor). Public fields carry no strategy parameter: collateral must equal $\lceil n \cdot r / 10000 \rceil$ for the deployment-wide ratio r , every request uses the common wire slippage, and a leader's tighter oracle bound is enforced privately by cancellation. Signal sequences must strictly increase; reusing a sequence with different content returns `REPLAY_DETECTED`.

13.4 Follower

A follower deposits into an open strategy past its shadow period when its `Eligibility` PDA is allowed and unexpired; the deposit is escrowed immediately and reserves strategy and platform capacity. A follower can request and cancel withdrawals, exit at the last final NAV at any time, and claim payouts.

Shares are a `u128` field in the follower's `Share` PDA, not a token, and cannot be transferred. An owner nonce equal to `request_sequence + 1` and mutually exclusive pending deposit and withdrawal stop replays and double reservation. A follower cannot block others: payouts are pulled, so a frozen token account stops only its owner's claim, and a cancelled withdrawal leaves a `WithdrawalTicket` tombstone that the permissionless `advance_withdrawal_queue` skips.

13.5 Sealed Executor

The executor decrypts sealed signals, assigns receipts, sizes and slices orders, proposes epoch NAV, posts statistics and publishes openings.

It cannot withdraw; every transfer it triggers runs along §13.1. It cannot act alone: `execute`, `record_fill`, `partial_close`, `close_exec` and `post_stats` require a second signer at `Config.verifier`. It cannot act with an expired or revoked key: each enclave instruction checks the signer against `Config.enclave` and the `EnclaveKey` record, neither revoked, before both expiries. Each `execute` is bounded without attribution: notional at most 250,000,000,000 base units, collateral at least 10 USDC and exactly the common ratio, the slot within the authorization's validity, per-market flow at most \$250,000 per logical minute, collateral drawn only from cash above protected reserves, and execution fees of 2 bps on filled notional under a configured daily ceiling. A proposal must use the next epoch number, arrive within `proposal_deadline_slots` of the boundary slot, follow settlement of all earlier requests and report liquid cash no greater than NAV;

its slot and prices are immutable, and a NAV-per-share change above 1,500 bps flags guardian review.

Attestation and key registration. The enclave key is generated inside the trust domain and written with mode 0600 to the VM's encrypted volume; the secret never leaves the VM. The executor builds the runtime binding

```
"unseen-tdx-runtime-v1" \n genesis_hash \n program_id \n executor_public_key \n encryption_public_key
```

requests a TDX quote through configfs-tsm with SHA-512 of that binding as report data, and submits it to Intel Trust Authority with pinned policy IDs. It accepts the PS384 token only if it verifies against the pinned JWKS and issuer and reports the same report data, the pinned 48-byte MRTD, TCB status `UpToDate`, a non-debuggable TD and every pinned policy matched; clients repeat the check (§15.7 Signal SDK and CLI). On-chain, only the admin signs `schedule_enclave`, which requires a new key, non-zero measurement and report hashes and an expiry more than 172,800 seconds ahead; `activate_enclave` succeeds only 172,800 seconds later.

13.6 Verifier and challenges

The verifier reads accounts, signed transaction bytes and program events from an RPC endpoint it chooses, rejects malformed Borsh, wrong owners, forged logs and missing history, and only then signs. `cosign_epoch(expected_nav, liquid_cash, evidence_hash)` is accepted only from `Config.verifier`, for an epoch in state Proposed, with both amounts equal to the proposal, a non-zero evidence hash and a slot within `proposal_deadline_slots` of the proposal; it opens the challenge window. The verifier admits each signal receipt in order and reads openings before reveal through a registered-key proof over a fresh nonce that expires after 60 seconds.

It cannot open a position alone, pay itself, or skip the challenge window. Its `unseen-risk-reduction-v2` certificates authorize only bounded reductions of filled positions. Rotation runs through `schedule_roles` and `activate_roles` with the 172,800-second timelock.

Anyone can challenge before `challenge_ends_at`. `challenge` escrows the configured bond in the pool vault and creates a `ChallengeReceipt` at `["challenge", epoch, challenger]`, so a wallet challenges an epoch at most once, even after a dismissal. A challenged epoch cannot finalize. Dismissal moves the bond to a protocol claim; an upheld challenge makes it the challenger's refund and leaves the epoch blocked until voided. `void_epoch` succeeds for the guardian at any time and for anyone once the protocol is paused, the enclave key is invalid, or `epoch_timeout_seconds` have passed since the proposal; it keeps the last final NAV and refunds any unresolved bond.

13.7 Guardian

`set_pause` stops `create_strategy`, `request_deposit`, `execute`, `propose_epoch` and `finalize_epoch`; its `revoke_enclave` flag revokes the enclave key in the same slot. The guardian resolves challenges and review flags and voids proposals. After revocation or expiry,

`guardian_close` cancels unfilled requests or closes filled positions, exempt from execution fees and flow admission. Every released token goes to the pool vault; the guardian names no destination, cannot touch shares or NAV, cannot register keys and cannot block exits.

13.8 The withdrawal budget

`finalize_epoch` sets the strategy's protected reserve to $\lfloor (\text{NAV} - \text{fees})/5 \rfloor$, requires verified liquid cash for fees plus reserve, and adds the reserve to pool-wide protected reserves that executions cannot spend. Every payout p requires

$$p \leq B_e - S_e, \quad p \leq \text{cash}_{\text{pool}}, \quad p \leq \text{cash}_{\text{strategy}}, \quad p \leq R$$

for epoch budget B_e , settled withdrawals S_e and reserve R ; excess carries forward.

`emergency_withdraw` draws on the same reserve at the last final NAV at any time, takes the FIFO head and, on a positive payout, voids an open proposal atomically. Voids and pauses never create a second budget.

13.9 Admin

`initialize` succeeds only when the signer equals the upgrade authority in `ProgramData`, so the first caller of a fresh deployment cannot become admin. The admin schedules keys and roles, writes eligibility, invites leaders and upgrades the program through the Squads multisig's timelock, without durable-nonce pre-signing. It has no instruction that moves pool funds, signs executions or edits strategies.

13.10 API operator, RPC, senders, keepers and the public

API operator. The API holds no keys. It stores the exact message it prepared and accepts signed bytes only when the message is identical, the session wallet is the sole signer and the signature verifies; it records the signature before broadcast. Sealed signals pass through it opaque. A bearer session does not open a private console: the executor verifies the leader's signed proof, deployment, on-chain ownership and single-use nonce. The public database holds no strategy-to-exec mapping.

RPC providers and senders. The executor and verifier rebuild instruction data, PDAs and signers from RPC state they read themselves, coherent within 100 slots. Compute price and tips are inside the message before signing. Transactions are simulated on the configured RPC before going to RPC, Jito or Helius Sender, and signatures are reconciled against RPC.

Keepers. A fill counts only through the dual-signed `record_fill`, which requires the request to be owned by the exec PDA. An unfilled request expires after `retry_slots`, and its principal returns through a dual-signed refund-only cancellation.

The public. Anyone can verify openings, replay NAV, challenge, and send `finalize_epoch`, `settle_share`, `advance_withdrawal_queue`, `seal_reveal_period`, `reset_epoch_schedule`, `deactivate_zero_strategy`, `reclaim_exec` and `void_epoch` once its conditions hold. Attribution stays sealed: no public field depends on a strategy parameter, statistics are 100-bps multiples published at least an hour late, and an opening posts only after closure, its own delay D , and maturity of both daily certificates (§10 Reveal).

13.11 Cross-cutting mechanisms

Deployment-bound signatures. Version 2 signals, receipts, leader-view proofs, policy evidence, risk certificates and stats evidence all bind genesis hash, program ID and executor key. Sealed envelopes use the same triple as AES-GCM additional data, and the TDX report data binds the encryption key to it, so a relay cannot substitute a key and a devnet signal does not decrypt on mainnet.

Nonces and idempotency. On-chain, owner nonces order requests, `CommitmentReceipt` makes each commitment unique, an exec PDA at a random 64-bit index is created once, and a 40-byte `ExecutionTombstone` keeps a reclaimed index unusable. Off-chain, signal sequences, durable leader-view and verifier nonces, and wallet-scoped `Idempotency-Key`s cover every write.

Sealed storage. Signals, openings, policy histories and reservations exist only in the executor's `state.enc` (XChaCha20-Poly1305, fresh nonces) and the verifier's private journal. An exclusive writer lease refuses a second executor, and each save advances a monotonic counter held by the verifier's counter service, whose signed replies bind the ciphertext hash, so a restored older ciphertext is refused.

Capability	Holder	Enforcement
Transfer out of the pool vault	Program only	<code>pool</code> PDA signature; destinations fixed by seeds, owner authority, <code>config.protocol_fee_vault</code> (keyless; spent only by <code>buyback_and_burn</code>)
Open, fill, reduce, close	Executor with verifier	Two signers; valid, unexpired, unrevoked <code>EnclaveKey</code>
Trade beyond declared limits	Nobody	Executor admission, verifier replay, on-chain envelope
Encode strategy parameters on the wire	Nobody	Exact common collateral ratio and wire slippage
Choose the valuation slot	Nobody	<code>proposal_deadline_slots</code> ; recorded slot and prices
Finalize an epoch	Anyone, after verification	Exact co-signature, elapsed window, no challenge, review resolved, not paused
Freeze finalization	Any wallet with a bond	Escrowed bond; one <code>ChallengeReceipt</code> per challenger and epoch
Retire a stuck proposal	Guardian; anyone after timeout, pause or key loss	<code>void_epoch</code> conditions
Exit at the last final NAV	Each owner	<code>emergency_withdraw</code> against the protected reserve
Take over 20% per epoch	Nobody	Reserve $\lfloor \text{net}/5 \rfloor$; payout $\leq$ budget – settled
Unwind after key loss	Guardian	<code>guardian_close</code> ; pool vault destination
Register or rotate keys and roles	Admin, after 172,800 s	<code>schedule_*</code> then <code>activate_*</code>
Revoke the enclave key	Guardian, immediately	<code>set_pause</code> with <code>revoke_enclave</code>
Become admin	Upgrade authority	<code>ProgramData</code> check in <code>initialize</code>

Capability	Holder	Enforcement
Replay a signal or request	Nobody	Deployment-bound V2 bytes; increasing sequences; owner nonces
Reuse an index or commitment	Nobody	One-time PDAs; <code>CommitmentReceipt</code> ; tombstone
Reveal early	Nobody	Closed state, D , mature certificates, on-chain commitment check
Publish unverified stats	Nobody	Dual signature, 100-bps multiples, 3,600-s delay, increasing hours
Read private lifecycle	The strategy's leader	Signed proof of at most 60 s, single-use nonce
Read openings before reveal	Registered verifier	Signed nonce-bound proof
Decrypt a signal	Attested executor	P-256 key in TDX report data; AES-256-GCM with deployment data
Roll back executor state	Nobody	Counter compare-and-swap at the verifier
Alter a wallet transaction	Nobody	Identical message, sole wallet signer, pinned signature

Table 13.2 — Capabilities, their holders and the enforcing mechanism.

14. Infrastructure

Unseen runs as one on-chain program and six off-chain services. Each service has a single job, holds only the keys that job needs, and reaches the others over authenticated HTTP. This chapter describes the services and their data flows, the database, how transactions reach the chain, the confidential VM that hosts the executor, the Railway services, and how a deployment is created on devnet or mainnet.

14.1 Services

Service	Code	Runs on	Keys held	Job
web	<code>web/</code> (Next.js)	Railway	None	Site, follower pages, leader console, explorer
api	<code>backend/src</code> (TypeScript, Node 22)	Railway	None	Public reads, sign-in, transaction plans, signal relay, SSE
indexer and jobs	<code>backend/src/indexer-main.ts</code>	Railway	None	Chain projection into PostgreSQL, public event outbox, public jobs
chain bridge	<code>backend/chain/bridge.ts</code>	Railway private network	Role tokens only	Builds unsigned transactions, simulates and sends signed bytes
executor	<code>crates/executor</code> (<code>unseen-executor</code> , Rust)	Intel TDX confidential VM	Enclave key	Sealed signals, sizing, execution, proposals, stats, reveals
verifier	<code>crates/verifier</code> (<code>unseen-verifier</code> , Rust)	Independent operator	Verifier key	Admission, co-signatures, epoch audit, monotonic counter
Postgres	Railway Postgres	Railway	—	Public projection, sessions, intents, intake

Table 14.1 — Off-chain services.

The bridge builds every unsigned transaction and sends every signed one, but it never signs for a role: role tokens only identify which process is asking for a preparation. The executor and the verifier each read chain state from an RPC endpoint they choose, rebuild the expected instruction from that state, and sign inside their own process.

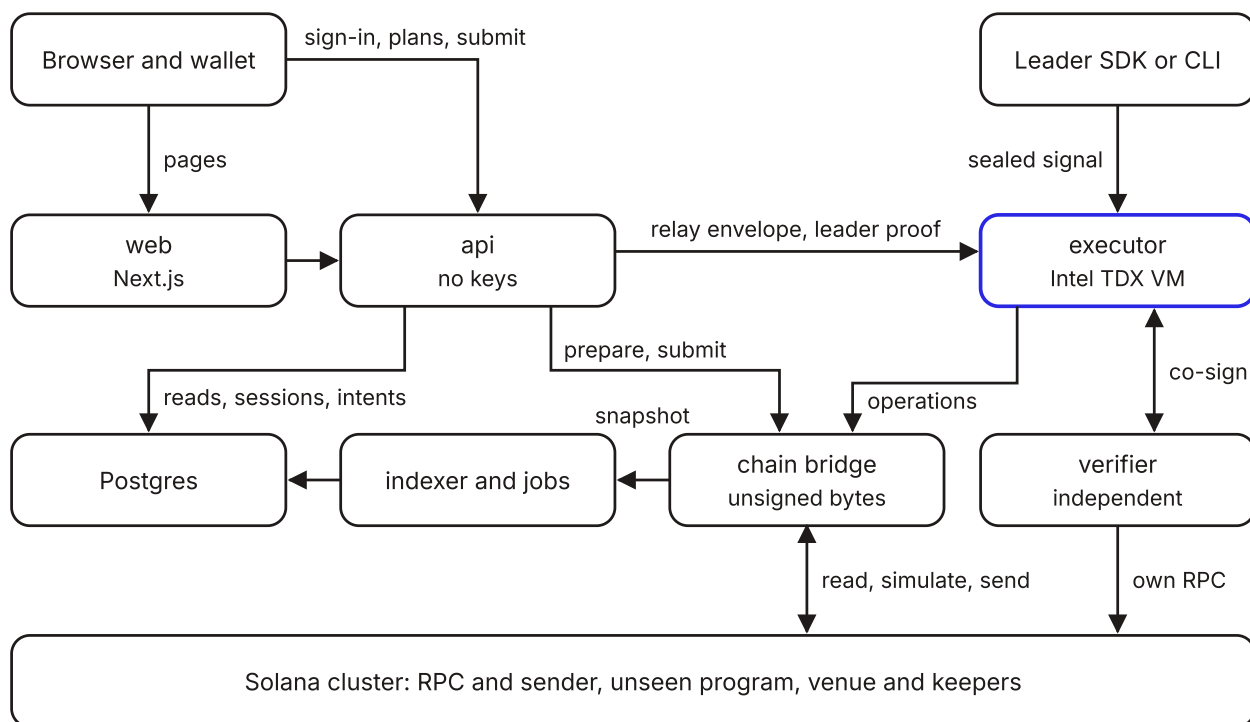


Figure 14.1 — Service architecture. The web app reads the API; the indexer writes the public projection to Postgres; the executor and the verifier read their own RPC, exchange admissions, authorizations and counter replies, and sign inside their own processes.

14.2 Data flows

Public reads. The indexer polls the bridge's `/snapshot`, pausing one second between passes. `validatePublicSnapshot` rejects any snapshot whose commitment chain does not reconcile to the pool's head and count, whose reveals do not hash to their commitments, whose statistics lack verified transaction evidence, or whose reveal-period certificates are inconsistent. A valid snapshot is written in one transaction under advisory lock `75084001`; an older slot never overwrites a newer one. The same transaction appends one `execution.observed` row per execution and state to the `PublicStreamEvent` outbox, keyed by `commitment:state`, and only for slots at or below the finalized slot. Every fifth pass, the indexer fetches each strategy's revealed NAV from the executor's `/public/revealed-nav/{strategyId}` and stores it only after `validatePublicNav` accepts it. The API serves reads from these rows and the event stream from the outbox (§15.5 Server-sent events).

Wallet transactions. The API forwards a validated request to the bridge's `/transactions/prepare`, checks that the returned transaction has the session wallet as fee payer and only signer, and stores

the serialized message, blockhash, last valid block height and a SHA-256 digest of the canonical request body in `TransactionIntent`. The wallet signs those bytes. On submit, the API takes a per-intent advisory lock, checks the message is unchanged and the signature valid, records the expected signature, and hands the signed bytes to the bridge. Status comes from `getSignatureStatuses` with history search; an intent whose block height has passed `lastValidBlockHeight` without a signature status becomes expired (§15.4 Transaction plans).

Signals. The leader's client verifies the executor's attestation, seals the signed signal to the attested encryption key and posts it to the executor's `/v1/signals/sealed`, directly or through the API relay at the same path. The executor decrypts it, checks the signature, deployment, registered signal key, sequence and limits, assigns a global receipt sequence and microsecond time, has the verifier admit the receipt through `/internal/policy/admit`, and returns the signed receipt.

Execution. On each tick the executor sizes ready receipts, asks the bridge's `/operations/prepare` for unsigned bytes, checks them against its own RPC reads, signs, and obtains the verifier's signature through `/internal/policy/authorize`. Keeper fills are recorded the same way. Trading and keeper reconciliation run before background reveal work, which is capped per tick.

Epochs. At each strategy's boundary the executor proposes NAV and liquid cash. The verifier recomputes both from its own RPC and the authorized openings and co-signs the exact tuple with an evidence hash. After the challenge window, `finalize_epoch` and per-owner `settle_share` run permissionlessly, and payouts wait as claims (§8 Epochs and settlement).

Reveals and statistics. After each UTC day the executor proposes the day's certificate and the verifier co-signs it with `seal_reveal_period_certified`; mature closed executions get `reveal_v2` and their evidence in pages of at most 400 bytes; `reclaim_exec` archives the execution and returns rent to the pool. Hourly observations are posted by `post_stats` with both signatures (§10 Reveal, §11 Attested statistics).

14.3 Database

PostgreSQL is accessed through Prisma and split into two schemas. `public` holds the chain projection (`Strategy`, `Share`, `Epoch`, `EpochChallenge`, `StatsAttestation`, `Pool`, `ExecAccount`, `Execution`, `RevealedOpening`, `SignalKey`, `EnclaveKey`, `ProtocolConfig`), indexing state (`IndexerCursor`, `ChainEvent`, `PublicChainSnapshot`, `PublicStreamEvent`), sign-in (`WalletChallenge`, `WalletSession`), `TransactionIntent` and `Job`. `intake` holds `LeaderApplication` and `FollowerAccessRequest`, which no read endpoint returns.

Money and share columns are `Decimal(39,0)` and slots `Decimal(20,0)`, so every amount is an exact integer. `ExecAccount` and `Execution` carry public facts only, with no relation to a strategy; `RevealedOpening` rows are created only from a confirmed, mature on-chain reveal. Sessions store the SHA-256 of the bearer token, challenges store hashes of the nonce and message. `TransactionIntent` is unique on (`wallet`, `action`, `idempotencyKey`), `ChainEvent` on (`transactionSignature`, `instructionIndex`, `eventIndex`), and `Job` on `deduplicationKey`, with a lease time and attempt count for retries. There are no tables for sealed signals or openings. Migrations are applied by `prisma migrate deploy` when a service starts, never during a build.

14.4 RPC and sending

Every service reads the cluster through `SOLANA_RPC_URL` and `SOLANA_WS_URL`, which production requires to be HTTPS. The verifier's RPC is its own choice, and both Rust services refuse an account read whose slots span more than 100.

Sending is configured by `TX_SENDER`:

<code>TX_SENDER</code>	Endpoint	Tip	Submission
<code>rpc</code>	<code>SOLANA_RPC_URL</code>	<code>TIP_LAMPORTS=0</code>	<code>sendRawTransaction</code> with preflight, no client retries
<code>jito</code>	<code>TX_SENDER_URL</code>	at least 1,000 lamports to <code>TIP_ACCOUNT</code>	<code>sendBundle</code>
<code>helius</code>	<code>TX_SENDER_URL</code>	at least 5,000 lamports to <code>TIP_ACCOUNT</code>	<code>sendTransaction</code> , preflight skipped, no retries

Table 14.2 — Sender modes. Devnet requires `rpc`.

`COMPUTE_UNIT_LIMIT` (at most 1,400,000), `COMPUTE_UNIT_PRICE_MICROLAMPORTS` and the tip transfer are added to the message before any authority signs, so sending never changes signed bytes. Every signed transaction is first simulated with signature verification on the configured RPC; a program rejection there is final and is reported with its program reason. Jito and Helius requests time out after 15 seconds, a bundle acceptance is not a confirmation, and the expected signature is always reconciled against RPC. Jito bundles can carry up to five transactions executed atomically in one slot, which suits slices and top-up-plus-execute pairs. Wallet transactions prepared by the bridge carry a 1,200,000 compute-unit limit, and their chain expiry is the blockhash's last valid block height.

14.5 The executor's confidential VM

The executor runs as the production binary, built with `--no-default-features --features production`, which leaves out every local adapter. `UNSEEN_PROFILE` selects `devnet` or `mainnet` and `UNSEEN_SERVICE=executor` selects the role; the role requires the Intel Trust Authority settings `ITA_URL`, `ITA_API_KEY_PATH`, `ITA_JWKS_URL`, `ITA_ISSUER`, `ITA_POLICY_IDS` and `TDX_EXPECTED_MRTD`. Startup refuses loopback URLs, relative or `.local` secret paths, a clock scale other than 1, a collateral ratio outside 3,334–10,000 bps, a stats cadence under 3,600 seconds and an attestation TTL under 60 seconds.

On first start the executor generates its Ed25519 key inside the trust domain and writes it to `UNSEEN_SIGNING_KEY_PATH` on the VM's encrypted durable volume; the P-256 encryption key is derived from it, so a restart keeps the same identity. For each attestation it creates a report directory under `/sys/kernel/config/tsm/report`, checks the provider is `tdx_guest`, writes the 64-byte report data to `inblob`, reads the quote from `outblob`, and requires the generation counter to

have advanced by exactly one. The Intel Trust Authority token is validated as in §13.5; its effective expiry is the earlier of its `exp` and issue time plus `ATTESTATION_TTL_SECONDS`, and it is refreshed every half TTL, never more often than every 30 seconds. `/attestation` publishes the current token with the public key, encryption key, MRTD and report data; `/health` and `/v1/signals/sealed` answer 503 while it is stale, and the unsealed `/v1/signals` route is refused in production. With `--attest-only` the binary writes `public-identity.json` to its state directory, prints the attestation and exits, which is how a new deployment obtains the key it registers.

The deployment plan targets a cloud confidential VM with TDX, such as a GCP C3 confidential VM or an Azure DCesv5 VM, started from a container image and start script. The VM reaches the bridge and the verifier over HTTPS and accepts HTTPS signal traffic.

14.6 Railway services

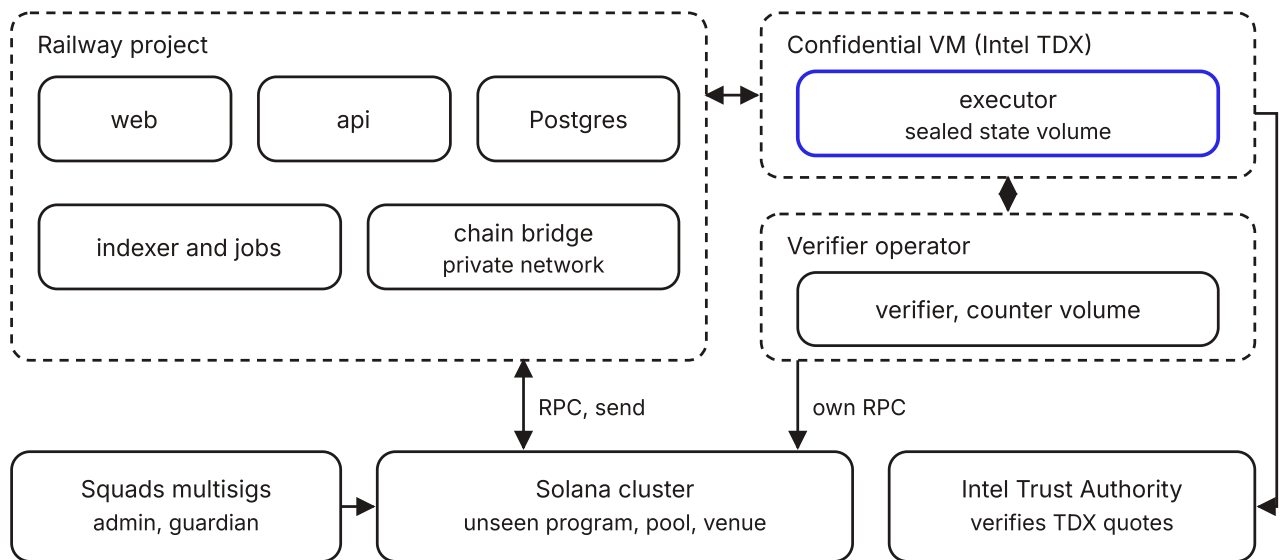


Figure 14.2 — Deployment topology. Unseen's Railway project hosts the web app, API, indexer and jobs, bridge and Postgres; the executor runs in its own TDX VM, reached over HTTPS, and sends quotes to Intel Trust Authority; the verifier runs under an independent operator; admin and guardian act through Squads multisigs.

Each Railway service is created on its own, with settings entered in the dashboard; a `railway.json` inside an upload is not read. Every service root ships a `.railwayignore`, because Railway does not apply `.gitignore` outside git, and each upload must build on its own with the npm version Railpack uses.

Service	Root	Build	Start	Health	Variables
web	Repository root	Build @unseen/contract in backend/, then next build in web/	npm --prefix web run start	/	NEXT_PUB NEXT_PUB
api	backend/	npm ci && npm run build (includes prisma generate)	npm run start: prisma migrate deploy, then the server	/health	DATABASE CORS_ORI SOLANA_R VENUE_PR TRANSACTION
indexer and jobs	backend/	Same as api	prisma migrate deploy, then the indexer	—	Same as (
verifier	backend/, or the operator's own project	Production Rust binary	unseen-verifier with UNSEEN_SERVICE=verifier	/health	Own RPC pins
Postgres	Railway Postgres	—	—	—	—

Table 14.3 — Railway services.

`web` is rooted at the repository root because it depends on the contract package by path, and its watch paths are `web/**` and `backend/packages/contract/**`. `NEXT_PUBLIC_UNSEEN_API_URL` is inlined at build time, so `web` is rebuilt when the API's domain changes. No start script fixes a port; Railway injects `PORT`. Migrations run at start because Railway's build phase has no access to the private network where Postgres lives. The API's production profile requires `BACKEND_MODE=real`, HTTPS origins and service URLs (the bridge may use HTTP on a `*.railway.internal` host), and refuses local development tokens, a local runtime config, an accelerated clock, LiteSVM and fake attestation.

14.7 Program deployment per cluster

The `unseen` program is built in one of three mutually exclusive profiles. `build.rs` requires `UNSEEN_PROGRAM_ID` for devnet and mainnet builds, refuses the local development ID for them, and turns a `local-testing` build aimed at a non-local ID into a compile error.

	Local	Devnet	Mainnet
Program feature	local-testing	devnet-venue	production
Program ID	Fixed local ID	UNSEEN_PROGRAM_ID	UNSEEN_PROGRAM_ID
Venue	Fake venue program	Fake venue program	Jupiter Perps position-request path
USDC	Local fake mint	Devnet test USDC mint	USDC
Clock scale	Accelerated (local 2880)	1	1
Sender	Local RPC	rpc	rpc, jito or helius
Attestation	Local fake	Intel TDX	Intel TDX

Table 14.4 — Cluster profiles.

Devnet has no Jupiter Perps deployment, so the devnet profile deploys the fake venue, whose keeper fills requests at a configured oracle price in a second transaction, together with a devnet test USDC mint; the full flow runs there. Mainnet uses Jupiter Perps through the position-request path, with the exec PDA as position owner; the request instructions come from the vendored Jupiter Perpetuals IDL (`chain/vendor/jupiter-perpetuals.json`), such as

`createIncreasePositionMarketRequest` and `createDecreasePositionMarketRequest`.

`npm run deploy:program -- --cluster devnet|mainnet` builds the artifact into `target/production` with its IDL, prints the SOL the deployer needs before sending anything, deploys with `--use-rpc`, a fixed `--buffer` keypair so an interrupted deploy resumes, and a compute-unit price, then compares the SHA-256 of the deployed bytes with the local `.so`.

14.8 Protocol initialization

`npm run init:protocol -- --cluster devnet|mainnet` is idempotent and skips what exists. It sends `initialize` with a `Settings` value and creates every PDA, token account and lookup table the flows need. `initialize` itself creates `Config`, the `Pool` for the venue program, the `pool_vault` token account and the first `EnclaveKey`, and checks:

- the signer is the program's upgrade authority;
- genesis NAV of at least 1,000,000 USDC base units per whole share;
- a protocol drawdown ceiling above 0 and below 10,000 bps;
- a common collateral ratio of 3,334–10,000 bps and a common wire slippage of at most 10,000 bps;

- positive epoch duration slots, proposal deadline slots, epoch timeout, execution-fee cap, challenge window, challenge bond, retry slots and key lifetime;
- a non-zero deployment genesis hash, measurement hash and attestation report hash;
- a clock scale of 1 outside local testing.

The settings also name the guardian, the verifier and the enclave key. The venue program passed at initialization becomes `Config.venue`, the only venue `execute` will call, and the protocol fee vault becomes the fixed fee destination. Every value the brief leaves open, such as the challenge window, bond, retry slots, key lifetime and slot cadence, is a required input with no default; initialization fails while any is unset.

14.9 Order of operations

1. Create the Railway project and its Postgres service.
2. Build and deploy the program with `deploy:program`; on devnet also deploy the fake venue and create the test USDC mint. Record the program ID and the cluster's genesis hash.
3. Start the executor in the TDX VM with `--attest-only`. It generates the enclave key, obtains a verified Intel Trust Authority token and writes `public-identity.json`.
4. Run `init:protocol` with that enclave key, its measurement and report hashes, the verifier key, the guardian and admin multisigs and the protocol settings.
5. Start the verifier; its counter service must be reachable before the executor opens its sealed store.
6. Start the executor in service mode.
7. Deploy the bridge, the indexer and jobs, and the API on Railway; the API's first start applies migrations.
8. Deploy `web` with the API's public domain.
9. Later enclave keys go through `schedule_enclave` and, 172,800 seconds later, `activate_enclave`; verifier and guardian rotations go through `schedule_roles` and `activate_roles`.

15. Interfaces

Clients reach Unseen through four interfaces: the public HTTP API with its event stream, the `@unseen/contract` package that types it, the `@unseen/signal-sdk` package and CLI that leaders and agents use to send sealed signals, and the wallet-signed proof that opens a leader's private console. Wallets sign every transaction and every proof; no interface accepts or stores a private key. Examples use `https://<api host>` for the API's base URL.

15.1 Conventions

Envelope. Every JSON response is either `{data, meta}` or `{error, meta}`. `meta` carries `requestId` (also sent as `X-Request-Id`), `asOf`, `slot` and `mode`; accelerated local chains add a `clock` object, which `estimateChainTime(meta)` turns into a chain-time estimate.

```
{
  "error": {
    "code": "IDEMPOTENCY_CONFLICT",
    "message": "This idempotency key was used with different input.",
    "retryable": false
  },
  "meta": { "requestId": "...", "asOf": "...", "slot": "...", "mode": "real" }
}
```

`details`, when present, maps field names to safe validation messages, never to accounts or signals. A program rejection found during preparation or simulation is non-retryable and carries `details.programReason` (for example `Cap`, `LeaderStake`, `TooEarly`) and `details.programCode`.

Code	HTTP	Retryable	Raised when
VALIDATION_ERROR	400 (413 for a body over 32,768 bytes)	No	Malformed input, unknown field, out-of-bounds integer
UNAUTHENTICATED	401	No	A non-public route without a valid session
INVALID_SIGNATURE	401	No	Failed sign-in, transaction or proof signature
CHALLENGE_EXPIRED	401	No	A leader-view proof outside its 60-second life
FORBIDDEN	403	No	Disallowed origin, or a session without the route's role
NOT_FOUND	404	No	Unknown resource, another wallet's resource, or an unrevealed execution
CONFLICT	409	No	Changed plan bytes, expired intent, program state refusal
IDEMPOTENCY_CONFLICT	409	No	A reused key with different input
CAP_EXCEEDED , DEPOSITS_CLOSED , INELIGIBLE	409	No	The matching program refusal
REPLAY_DETECTED	409	No	A reused nonce or signal sequence
RATE_LIMITED	429	Yes	A rate window is full
SERVICE_UNAVAILABLE	503	Yes	A dependency is unreachable; retry identical input
CONFIGURATION_REQUIRED	503	No	The route is not served in this mode

Table 15.1 — Error codes. Retryable failures carry `Retry-After: 2`.

Values. Amounts, shares, slots, sequences, indices and basis points are canonical decimal strings, to be parsed with `BigInt`. USDC has six decimals and shares nine; a weight of `"15000"` means 150% of equity. Strategy IDs, salts, nonces and commitments are lowercase 64-character hex; keys are base58.

Requests. POST bodies must be `application/json` with no unknown fields. Lists take `cursor` and `limit` (1–100, default 50) and return `{items, nextCursor}`; the cursor is opaque and `null` ends the list. Every response is `Cache-Control: no-store`, and browsers are admitted only from configured origins.

Rate limits. Per minute: 600 reads and 60 writes per client IP, 60 writes per wallet, 20 sign-in challenges per IP, and 10 submissions per application form per IP. The executor separately limits signals per registered signal key per minute, charged only after the signature and key are verified.

Idempotency. Every transaction preparation, submission and application form needs an `Idempotency-Key` of 8–128 letters, digits, underscores or hyphens. Keep it across retries of one action and generate a new one for a new action. Transaction keys are scoped to the wallet and action, application keys to the form; a replay with identical input returns the original result.

15.2 Routes

Group	Routes	Access
Deployment	GET /v1/deployment	Public
Platform	GET /v1/platform, GET /v1/executions	Public
Strategies	GET /v1/strategies, /v1/strategies/:strategyId, .../stats?from&to, .../epochs, .../revealed-trades	Public
Reveals	GET /v1/revealed-executions/:executionId, GET /v1/revealed-strategies/:strategyId/nav	Public
Guardian	GET /v1/guardian/outcomes	Public
Events	GET /v1/events?after	Public
Forms	POST /v1/leader-applications, POST /v1/follower-access-requests	Public
Sign-in	POST /v1/auth/challenge, POST /v1/auth/verify, POST /v1/auth/logout	Public; logout authenticated
Account	GET /v1/me, GET /v1/me/portfolio, GET /v1/me/eligibility/:strategyId	Authenticated
Plans	POST /v1/transactions/{deposit, withdrawal, emergency-withdrawal, cancel-request, claim-withdrawal, claim-bond, challenge-epoch, void-epoch, reset-epoch-schedule}	Authenticated
Leader plans	POST /v1/transactions/{create-strategy, signal-key, claim-leader-fee}	Leader
Plan status	GET /v1/transactions/:transactionId, POST /v1/transactions/:transactionId/submit	Owner
Leader	GET /v1/leader/strategies, POST /v1/leader/strategies/:strategyId/console	Leader
Signals	POST /v1/signals/sealed	Leader

Table 15.2 — Public API routes. `routes` in the `contract` package is the authoritative table.

Roles come from chain state: `leader` for a wallet that leads a strategy or holds an invitation, `verifier` for the registered verifier key. The unsealed `POST /v1/signals` and the verifier's

openings route answer `CONFIGURATION_REQUIRED` in real mode; signals travel sealed, and the verifier reads openings through its own authenticated channel. Development routes under `/v1/dev/` exist only on an explicitly local chain and are refused by production configuration.

15.3 Sign-in

A session is a bearer token obtained by signing a one-time message with the wallet.

```
import { createUnseenClient, signatureToBase64 } from '@unseen/contract';

const api = createUnseenClient({ baseUrl: 'https://<api host>' });
const { data: challenge } = await api.request('authChallenge', { body: {
  wallet } });
const signature = await walletAdapter.signMessage(new
  TextEncoder().encode(challenge.message));
const { data: session } = await api.request('authVerify', {
  body: { wallet, challengeId: challenge.challengeId, signature:
    signatureToBase64(signature) },
});
const authed = createUnseenClient({ baseUrl: 'https://<api host>',
  accessToken: session.accessToken });
```

The message names the requesting origin's host and URI, the wallet, a 24-byte random nonce, issue and expiry times and a request ID, and states that no transaction or payment is authorized (layout in §16.2). A challenge lives five minutes and is consumed on first use, successful or not. The session lasts one hour; the database stores only the SHA-256 of the token, and `authLogout` revokes it. Roles are assigned by the server; the client cannot request one.

15.4 Transaction plans

Every on-chain action a person takes follows the same four steps: prepare, sign, submit, confirm.

```

const idempotencyKey = crypto.randomUUID();
const { data: plan } = await authed.request('deposit', {
  idempotencyKey,
  body: { strategyId, amountUsdc: '1000000000' }, // 1,000 USDC
});
if (!plan.executable) throw new Error('Preview only.');
```

```

const unsigned = Uint8Array.from(atob(plan.unsignedTransaction), c =>
c.charCodeAt(0));
const signed = await walletAdapter.signTransaction(unsigned);
await authed.request('submitTransaction', {
  params: { transactionId: plan.id },
  idempotencyKey,
  body: { signedTransaction: btoa(String.fromCharCode(...signed)) },
});
```

```

let status = plan;
while (status.nextAction !== 'none') {
  await new Promise(resolve => setTimeout(resolve, 1000));
  ({ data: status } = await authed.request('transaction', { params: {
transactionId: plan.id } }));
}
```

Preparation checks fresh on-chain bounds before asking for a signature, so a request the program would refuse returns a field error naming the limit. A plan carries the base64 unsigned transaction, its blockhash, `lastValidBlockHeight`, the cluster and `nextAction: sign_and_submit`, then `await_confirmation`, then `none` with state `confirmed` or `failed`. The wallet must sign the exact bytes; changing instruction data, blockhash or payer makes submission fail with `CONFLICT`. The API verifies the signature, records it before broadcast, and on an ambiguous failure accepts only the same signed bytes again. A plan unsigned past its preview TTL or block height expires; prepare a new one with a new key. Confirmation means the instruction landed; a deposit mints shares only when its epoch settles, which `portfolio` shows.

15.5 Server-sent events

`GET https://<api host>/v1/events` streams platform-wide events. Each SSE `data` field holds one JSON `PublicEvent`.

```
: connected
```

```
id: chain:<n>
```

```
event: execution.observed
```

```
data: {"id":"chain:<n>","type":"execution.observed","data":  
{"id":"...", "market":"SOL", "side":"long", "notionalUsdc":"...", "executedAt":"...",  
"slot":"...", "execAccount":"...", "commitment":"...", "transactionSignature":"...",  
"state":"filled"}}
```

```
event: heartbeat
```

```
data: {"id":"heartbeat:...", "type":"heartbeat", "data":  
{"asOf":"...", "slot":"..."}}
```

`execution.observed` carries only the unattributed execution fields shown, and `platform.updated` carries the platform overview. Events come from the durable outbox, cover only finalized executions, and are reduced to a fixed field list before they are sent. A heartbeat is sent every 15 seconds without an SSE `id`, so a reconnect with `?after=chain:<n>` or `Last-Event-ID` resumes from the last durable event. The contract client reads the stream with `fetch`:

```
const stop = new AbortController();  
for await (const event of api.events({ query: { after: lastId }, signal:  
stop.signal })) {  
  if (event.type === 'execution.observed') lastId = event.id;  
}
```

15.6 The contract package

`@unseen/contract` is a dependency-free ESM package that runs in browsers and Node 22. It exports:

- `routes` and `ApiRoutes`: method, path, access, and the params, query, body and response type of every route;
- `createUnseenClient({baseUrl, accessToken?, fetch?})` with `request(name, options)` and `events(options)`, and `UnseenApiError` exposing `status`, `error` and `meta`;
- commitment helpers: `encodeOpening`, `commitmentForOpening`, `verifyOpening` for V1 and V2 openings, and `commitmentSetHash`;
- encoders for signed messages: `encodeSignalForSigning`, `encodeExecutorReceiptForSigning`, `encodePolicyEvidenceV2`, `encodeOpeningV2`, `encodeLeaderViewProofForSigning`, `encodeRiskReductionForSigning`, `encodeStatsEvidenceForHash`, `encodeSealedSignalAdditionalData`, `encodeTdxRuntimeBinding`;

- `verifiers: verifyExecutorReceipt, verifyOpeningEvidenceV2, verifyRiskReductionCertificate;`
- `prepareLeaderViewProof, signatureToBase64` and `estimateChainTime`.

Any visitor can check a revealed trade in the browser:

```
import { createUnseenClient, verifyOpening } from '@unseen/contract';

const api = createUnseenClient({ baseUrl: 'https://<api host>' });
const { data: trade } = await api.request('revealedExecution', { params: {
  executionId } });
const valid = await verifyOpening(trade.opening, trade.commitment);
```

For a V2 opening, `verifyOpening` recomputes the 273-byte preimage hash, then checks the signal signature, the receipt signature, the policy-evidence hash, every later reduction witness and every risk certificate. Authenticated responses must never enter a shared cache.

15.7 Signal SDK and CLI

`GET https://<api host>/v1/deployment` returns the cluster, genesis hash, program IDs, USDC mint, the executor's public key and URL, its current attestation and the signal transport paths. The SDK compares this response with pins the leader obtained independently and refuses to sign when they differ.

```
import { createUnseenClient } from '@unseen/contract';
import { createSealedSignalClient } from '@unseen/signal-sdk';

const api = createUnseenClient({ baseUrl: 'https://<api host>',
  accessToken });
const { data: deployment } = await api.request('deployment');

const signals = createSealedSignalClient({
  deployment,
  trust: {
    genesisHash: PINS.genesisHash, programId: PINS.programId,
    executorPublicKey: PINS.executorPublicKey,
    issuer: PINS.itaIssuer, jwksUrl: PINS.itaJwksUrl,
    expectedMeasurement: PINS.mrtd, expectedPolicyIds: PINS.policyIds,
  },
  signMessage: bytes => signalKey.sign(bytes),
  submitEnvelope: envelope => api.request('sealedSignal', { body: envelope
}),
});

const { data: receipt } = await signals.submit({
  version: 2,
  genesisHash: PINS.genesisHash, programId: PINS.programId,
  executorPublicKey: PINS.executorPublicKey,
  strategyId, sequence: '1',
  targets: [{ market: 'SOL', side: 'long', weightBps: '15000' }],
  validUntilSlot, maxOracleDeviationBps: '30', signalKey: signalPublicKey,
});
```

`submit` performs, in order:

1. **Attestation.** `verifyTdxAttestation` requires HTTPS trust URLs, an `intel_tdx` attestation from `intel-trust-authority` for the pinned executor key and MRTD, and a 65-byte uncompressed P-256 encryption key. It recomputes SHA-512 of the runtime binding and compares it with `reportData`. It fetches the pinned JWKS without credentials or redirects, ignoring any key URL inside the token, verifies the PS384 or RS256 signature with the unique matching `kid`, and checks the issuer, freshness, `attester_type` TDX, TCB `UpToDate`, a non-debuggable TD, `tdx_mrtd`, `tdx_report_data` and that every pinned policy matched with none unmatched.
2. **Signing.** The canonical V2 bytes go to `signMessage`, and the returned signature is verified against the signal key before use.
3. **Sealing.** `sealSignal` generates an ephemeral P-256 key, derives an AES-256-GCM key by ECDH and HKDF-SHA256, and encrypts the signed signal with a random 12-byte nonce under

the deployment's additional data. The envelope holds only the version, ephemeral key, nonce and ciphertext.

4. **Transport.** The envelope goes to the executor's `/v1/signals/sealed`, or through the API relay when `submitEnvelope` is given.
5. **Receipt.** The SDK verifies that the receipt names the same deployment, that its signed-signal hash matches, and that the executor's signature is valid.

The receipt acknowledges admission only. A retry must resend the identical signal with the same sequence, which returns the original acknowledgment; a new target needs a higher sequence. Current sizing, slices, fills and drop reasons are read only through the console proof.

The CLI wraps the same SDK for agents and scripted leaders:

```
npm run leader:signal -- --generate-key .local/my-signal.json
npm run leader:signal -- --strategy <strategy-id> --key .local/my-signal.json --sequence 1 --target SOL:long:15000 --max-oracle-deviation-bps 30
```

`--generate-key` refuses to overwrite a file and prints only the public key and path. `--target MARKET:side:bps` repeats once per market. Expiry defaults to the current slot plus the profile's validity and can be set with `--valid-for-slots` or `--valid-until-slot`; `--config` selects a profile. Before signing, the CLI checks the strategy account, the registered signal key, the deployment pins and the executor identity.

15.8 Leader-console proof

The private console shows a strategy's own positions, signals with their lifecycle and reasons, fills, fees, followers in aggregate and epochs. It is served by `POST https://<api host>/v1/leader/strategies/:strategyId/console` with a proof the leader wallet signs, not the signal key.

```
const proof = await signals.signLeaderView({
  strategyId,
  wallet: leaderWallet.publicKey.toBase58(),
  signMessage: bytes => leaderWallet.signMessage(bytes),
});
const { data: view } = await authed.request('leaderConsoleView', { params:
  { strategyId }, body: proof });
```

`signLeaderView` checks the attestation first, then signs the output of `prepareLeaderViewProof`: a random 32-byte nonce, the wallet, the strategy, the deployment pins, a wall-clock `issuedAt` and an `expiresAt` 60 seconds later. The API rejects a proof that expires too soon or too late, or whose wallet differs from the session, and forwards it unchanged to the executor's

`/internal/leader/view`. The executor verifies the Ed25519 signature, the deployment, that the wallet is the strategy's leader on-chain, and that the nonce is unused, then records the nonce durably. The API stores no part of the response. Every refresh needs a new proof.

16. Parameters and encodings

This chapter collects the protocol's parameters, the byte layouts that signatures and commitments cover, and its vocabulary. Amounts are USDC base units (six decimals) unless marked; shares have nine decimals; basis points use 10,000 as 100%. A *configured* value has no default and must be supplied at initialization or service start; a development value is that of the local profile only.

16.1 Protocol parameters

Parameter	Value	Set in	Enforced in
Leader minimum stake	500 bps of equity	Program	<code>request_deposit</code> , <code>request_withdraw</code> , <code>emergency_withdraw</code>
Performance fee	0–2,500 bps, default 1,500	<code>create_strategy</code>	<code>create_strategy</code> , <code>finalize_epoch</code>
Performance-fee split	Leader 8,000 bps, protocol 2,000 bps	Domain crate	<code>finalize_epoch</code>
Founding protocol-share waiver	7,776,000 s (90 days)	Founding <code>LeaderInvitation</code>	<code>create_strategy</code> , <code>finalize_epoch</code>
Execution fee	2 bps of filled or reduced notional	Program	<code>record_fill</code> , <code>partial_close</code> , <code>close_exec</code>
Execution-fee daily cap	<code>execution_fee_epoch_cap_bps</code> of pool equity; configured (development 100)	<code>initialize</code>	Execution-fee charge
Reveal delay D	86,400–2,592,000 s, default 604,800; immutable	<code>create_strategy</code>	<code>reveal_v2</code> , <code>reveal</code>
Maximum delay ever	Monotonic maximum of registered D	<code>create_strategy</code>	<code>seal_reveal_period</code>
Reveal period	UTC day $[86400\ d, 86400\ (d + 1))$	Program	<code>seal_reveal_period</code> , reveal checks
Maximum leverage	Declared, 1–30,000 bps	<code>create_strategy</code>	Executor admission, verifier, on-chain envelope
Drawdown kill level	1 bps to the protocol ceiling	<code>create_strategy</code>	<code>request_deposit</code> , verifier risk checks
Protocol drawdown ceiling	1–9,999 bps; configured (development 2,000)	<code>initialize</code>	<code>create_strategy</code>

Parameter	Value	Set in	Enforced in
Epoch length	86,400 s	Program	<code>propose_epoch</code> , <code>finalize_epoch</code> , <code>void_epoch</code>
Epoch slots	<code>epoch_duration_slots</code> ; configured (development 75)	<code>initialize</code>	Boundary slot schedule
Proposal deadline	<code>proposal_deadline_slots</code> ; configured (development 1,000)	<code>initialize</code>	<code>propose_epoch</code> , <code>cosign_epoch</code> , <code>reset_epoch_schedule</code>
Epoch timeout	<code>epoch_timeout_seconds</code> ; configured (development 172,800 logical s)	<code>initialize</code>	<code>void_epoch</code> , <code>reset_epoch_schedule</code>
Guardian review threshold	NAV-per-share change above 1,500 bps	Program	<code>propose_epoch</code> , <code>finalize_epoch</code>
Withdrawal budget	$\lfloor (\text{NAV} - \text{fees}) / 5 \rfloor$, 2,000 bps	Program	<code>finalize_epoch</code> , <code>settle_share</code> , <code>emergency_withdraw</code>
Challenge window	<code>challenge_seconds</code> ; configured	<code>initialize</code>	<code>cosign_epoch</code> , <code>challenge</code> , <code>finalize_epoch</code>
Challenge bond	<code>challenge_bond</code> in the USDC mint; configured (development 1,000,000)	<code>initialize</code>	<code>challenge</code>
Strategy cap	250,000,000,000 (\$250,000)	Program	<code>create_strategy</code> , <code>request_deposit</code>
Platform cap	2,000,000,000,000 (\$2,000,000)	Program	<code>create_strategy</code> , <code>request_deposit</code>
Invited strategies	10	<code>Config.registry</code>	<code>create_strategy</code>
Shadow period	2,592,000 s (30 days)	Program	<code>request_deposit</code>

Parameter	Value	Set in	Enforced in
Genesis NAV	At least 1,000,000 per whole share	<code>initialize</code>	Share conversion
Notional per execution	At most 250,000,000,000	Program	<code>execute</code>
Per-market flow	250,000,000,000 per logical minute, opens and reductions	Program; executor slice window 60 s	Flow charge, executor slicer
Execution collateral floor	10,000,000 (10 USDC)	Domain crate	<code>execute</code> , executor
Common collateral ratio	3,334–10,000 bps; configured (development 3,334)	<code>initialize</code>	<code>execute</code>
Common wire slippage	At most 10,000 bps; configured (development 100)	<code>initialize</code>	Venue request
Outstanding fee bound	16 bps of aggregate notional, rounded up	Program	Common envelope
Slice retry window	<code>retry_slots</code> ; configured (development 300)	<code>initialize</code>	Venue request expiry
Admin timelock	172,800 s	Program	<code>activate_enclave</code> , <code>activate_roles</code>
Enclave key lifetime	Configured (<code>ENCLAVE_KEY_LIFETIME_SECONDS</code>)	<code>initialize</code> , <code>schedule_enclave</code>	Every enclave instruction
Attestation TTL	At least 60 s; configured (development 300)	Executor	Executor, SDK
Stats quantization	100 bps	Program	<code>post_stats</code>
Stats delay	At least 3,600 s after the observation hour	Program	<code>post_stats</code>
Stats cadence	At least 3,600 s; configured	Executor	Executor

Parameter	Value	Set in	Enforced in
Exposure bands	Low and moderate upper bounds configured (development 5,000 and 15,000 bps)	Executor	Stats posting
Signal targets	1–3 distinct markets; gross weight within declared leverage	Signal	Executor, API
Oracle deviation	At most 10,000 bps per signal	Signal	Executor, verifier
Signal rate	Per signal key per minute; configured (development 60)	Executor	Executor
Evidence page	At most 400 bytes	Program	<code>reveal_evidence</code>
Leader-view and verifier proofs	At most 60 s, single-use nonce	Contract	API, executor

Table 16.1 — Protocol parameters.

16.2 Byte encodings

Integers are little-endian; signed integers use two's complement; public keys and hashes are raw 32 bytes in binary layouts and base58 or lowercase hex in JSON. All hashes are SHA-256 unless named otherwise.

Commitment V1 (`unseen-sha256-v1`) hashes an 80-byte preimage.

Offset	Size	Field
0	32	<code>strategy_id</code>
32	8	<code>exec_index</code> (u64)
40	8	<code>signal_seq</code> (u64)
48	32	<code>salt</code>

Table 16.2 — Commitment V1 preimage.

Commitment V2 (`unseen-sha256-v2`) hashes a 273-byte preimage.

Offset	Size	Field
0	17	ASCII <code>unseen-opening-v2</code> , no terminator
17	32	Deployment genesis hash
49	32	Program ID
81	32	Executor public key
113	32	<code>strategy_id</code>
145	8	<code>exec_index</code> (u64)
153	8	<code>signal_sequence</code> (u64)
161	8	Global receipt sequence (u64)
169	8	Receipt time, Unix microseconds (u64)
177	32	Signed-signal hash
209	32	<code>salt</code>
241	32	Policy-evidence hash

Table 16.3 — Commitment V2 preimage.

Chain head and commitment set. The pool's head starts at 32 zero bytes, and each execution sets

$$h_k = \text{SHA256}(h_{k-1} \parallel c_k).$$

`commitmentSetHash` is SHA-256 of the 32-byte commitments concatenated in ascending hex order, duplicates kept.

Policy evidence V2 is 392 bytes; its hash is the execution's policy hash.

```

offset size field
  0    16 ASCII "unseen-policy-v2"
  16   32 genesis hash           48   32 program ID           80   32
executor key
  112   32 strategy ID           144   32 signed-signal hash  176   8
receipt sequence
  184   32 batch ID              216   4 batch index (u32)    220   4 batch
size (u32)
  224   80 ten u64: batch total notional, batch impact fee, member
notional,
          original notional, allowed reduction, allocated impact fee,
          allocated venue fee, allocated execution fee, sizing slot,
sizing equity
  304   16 prior signed notional (i128)
  320    8 target weight bps (i64)
  328   24 three u64: declared max leverage bps, declared drawdown kill
bps, reveal delay s
  352   32 prior policy hash     384    8 valid-until slot

```

Signed messages. Text messages join their fields with `\n` and have no trailing newline; JSON messages are UTF-8 with no whitespace and keys in the order listed.

```

signal V2      {"domain":"unseen-signal-
v2","genesisHash","programId","executorPublicKey",
                "strategyId","sequence","targets":
[{"market","side","weightBps"}]},
                "validUntilSlot","maxOracleDeviationBps","signalKey"}
signal V1      {"domain":"unseen-signal-
v1","strategyId","sequence","targets",
                "validUntilSlot","maxOracleDeviationBps","signalKey"}
signed hash    SHA256(signal V2 bytes || 64-byte Ed25519 signature)
receipt V2     unseen-receipt-v2, genesisHash, programId,
executorPublicKey,
                receiptSequence, receivedAtUnixMicros, signedSignalHash
leader view    {"domain":"unseen-leader-view-
v2","genesisHash","programId",

"executorPublicKey","strategyId","wallet","nonce","issuedAt","expiresAt"}
stats V2       alphabetical JSON: domain "unseen-stats-v2",
executorPublicKey, exposureBand,
                genesisHash, maxDrawdownBps, observationHour, programId,
returnBps, strategyId
risk V2        alphabetical JSON with domain "unseen-risk-reduction-v2";
its SHA-256 is the
                adjustment's policy hash, signed separately by the verifier
sealed AAD     unseen-sealed-signal-v1, genesisHash, programId,
executorPublicKey
TDX binding    unseen-tdx-runtime-v1, genesisHash, programId,
executorPublicKey,
                encryptionPublicKey; report data = SHA512(binding), 64
bytes

```

Sealed envelope. The client computes an ECDH P-256 shared secret with the executor's encryption key, derives a 32-byte key with HKDF-SHA256 using salt SHA-256(AAD) and info `unseen-sealed-signal-v1`, and encrypts the JSON of the signed V2 signal with AES-256-GCM, a random 12-byte nonce and a 16-byte tag. The envelope carries base64 fields `ephemeralPublicKey` (65-byte uncompressed point), `nonce` and `ciphertext`. The executor's P-256 secret is SHA-256 of `unseen-sealed-signal-key-v1\0` followed by its Ed25519 secret, rehashed until it is a valid scalar.

Sign-in message.

```
<host> wants you to sign in with your Solana account:  
<wallet>
```

Sign in to Unseen. No transaction or payment is authorized.

```
URI: <origin>  
Version: 1  
Chain ID: solana:<cluster>  
Nonce: <48 hex characters>  
Issued At: <ISO time>  
Expiration Time: <ISO time>  
Request ID: <UUID>
```

16.3 Deployment defaults

The values below are written by `initialize` and carried by the service configuration. The operator can change them before initialization; the services and the on-chain `Config` must agree.

Setting	Default	Purpose
Epoch cadence	UTC daily; nominal 216,000 slots	Wall-clock boundaries, free of slot-duration drift
Challenge window	3,600 s	Time for an independent challenge before finalization
Challenge bond	100 USDC	Escrowed by a challenger
Venue retry window	150 slots	Bounds how long a request may wait to land
Enclave key lifetime	2,592,000 s (30 days)	Renewed through governance
Genesis NAV	1,000,000 base units per whole share	Initial NAV per share; also its floor
Protocol drawdown ceiling	2,000 bps	Upper bound for each strategy's drawdown stop
Proposal deadline	300 slots, within 300 s of the boundary	Bounds valuation timing
Epoch timeout	172,800 s (48 hours)	Releases a stuck proposal
Execution-fee cap	100 bps of equity per day	Bounds aggregate execution charges
Common collateral ratio	3,334 bps	One ceiling-rounded margin for every exec account
Common wire slippage	100 bps	One public slippage bound across strategies
Stats cadence / quantization / delay	3,600 s / 100 bps / 3,600 s	Hourly, coarse and delayed observations
Exposure bands	below 5,000 bps low; below 15,000 bps moderate	Coarse 0.5× and 1.5× boundaries
Oracle age	5 s	Price freshness for new requests
Valuation age	150 slots	Checkpoint freshness for NAV

Setting	Default	Purpose
Venue admission reserve	500 bps	Cushion for venue fees, swaps and carry at admission
Request rent float	0.02 SOL	Funds request accounts; unused SOL returns to the pool
Service and admin floats	3 SOL each	Working rent and fees
Compute budget	1,400,000 units at 1,000 micro-lamports	Explicit transaction budget
Sender tip	10,000 lamports (devnet 0)	Jito or Helius Sender inclusion
Attestation refresh	300 s	Fresh Intel Trust Authority tokens
Signal admission limit	60 per key per minute	Independent budgets per signal key
Range screening	score at most 3, result valid 3,600 s	Hourly wallet screening
Transaction plan lifetime	60 s	Wallet signs a recent plan

16.4 Glossary

Term	Meaning
Admission	The verifier's durable, in-order acceptance of a signal receipt
Attestation	An Intel Trust Authority token proving the executor's measurement and key binding
Chain head	Running SHA-256 over all commitments, stored in <code>Pool</code>
Challenge	A bonded objection that blocks an epoch's finalization
Claim	An on-chain balance its owner pulls to their own token account
Commitment	The 32-byte hash of an opening, posted with each execution
Common envelope	Pool-wide bounds on open notional and collateral, applied without attribution
Co-signature	The verifier's signature on an epoch's NAV, liquid cash and evidence hash
Deployment pins	Genesis hash, program ID and executor key bound into every signed message
Emergency withdrawal	An exit at the last final NAV against the protected reserve
Enclave key	The executor's Ed25519 key, generated inside the TDX trust domain
Epoch	A strategy's daily accounting interval: Proposed, Challengeable, Final or Voided
Exec account	A PDA at a random 64-bit index owning exactly one position
Flat fence	A time at which the whole pool held no open notional
Guardian	The multisig that pauses, revokes, resolves challenges and unwinds
High-water mark	The highest final NAV per share; fees apply only above it
Keeper	A venue agent that fills requests at the oracle price
Leader-view proof	A wallet-signed, single-use, 60-second console authorization

Term	Meaning
MRTD	The 48-byte TDX measurement of the trust domain's initial image
Opening	The preimage of a commitment, published at reveal
Receipt	The executor's signed record of a signal's sequence and arrival time
Reveal delay D	A strategy's immutable wait between closing a position and its reveal
Reveal period certificate	A daily record, signed by the executor and the verifier, of the longest reveal delay among the strategies active that day and the time the day matures
Risk-reduction certificate	A verifier-signed authorization to reduce an existing position
Sealed signal	A signed signal encrypted to the attested executor's key
Shadow period	The first 30 days, when only the leader's capital is invested
Share	A non-transferable ledger entry in a <code>Share</code> account
\$UNSEEN	Unseen's token: 1,000,000,000 fixed supply, launched 100% fair; the protocol's whole fee share buys it on the market and burns it
Signal	Signed target weights with a sequence and an expiry slot
Target weight	Desired exposure in one market and side, in bps of equity
Verifier	The independent party that recomputes NAV and co-signs
Withdrawal reserve	Cash protected at finalization for the epoch's withdrawals

Table 16.4 — Glossary.