FLOW: mechanism, architecture, and simulation evidence

Technical companion to the FLOW announcement. Contents: the full mechanism specification with launch parameters (§1), the system architecture from miner payload to weight vector (§2), the on-chain contract architecture for the trade collateral (§3), the adversarial simulation framework (§4), the statistical failure mode that motivated the gate's block structure (§5), results from a 403-run parameter grid, the ship-spec validation suite, and the capital-at-risk collateral study (§6), reproduction instructions (§7), the file map (§8), and limitations (§9). All simulation code is in `flow_sim/`; every table below regenerates from the commands in §7.

Launch timeline

Date	Event
2026-08-17	FLOW goes live. Validators collect and score payloads; evidence of statistical significance and EWMA history accrue from day one. Emission weight is 0; nothing is paid during this window.
2026-09-14	After four weeks of payload collection, emissions turn on at a **25% pool allocation**. The flip is a deliberate one-line config change, not an automatic switch.

Very strong miners may already have passed the gate by the time emissions turn on.

TRADE-MIX is deprecated in this release. Historical miner embeddings associated with it will be removed from the datalog.

---

1. Mechanism specification

1.1 Submission

One prediction per regime per epoch (epoch = 1h). Regimes are independent horizon bands: A (intraday) = [1, 24]h, B (short) = [24, 48]h, C (medium) = [72, 168]h, D (long) = [168, 336]h. Four bands exist because real strategies are layered: microstructure edges usually play out over one to two days but sometimes run longer, swing positions get hedged with small short-timeframe bets against them, and a serious book runs several of these at once. Independent bands give concurrent and complex strategies that optionality instead of forcing one clock on everyone. A prediction is the tuple

$$\pi = (r, d, f, E, SL, TP_1, TP_2, h, t)$$

with regime $r \in \{A, B, C, D\}$, direction $d \in \{\text{long}, \text{short}\}$, Kelly fraction $f \in [0.01, 0.25]$ (the max bet: no single trade can put more than a quarter of posted collateral at risk), entry price E (validator-assigned at the observed open; see wire format), stop SL, targets TP_1, TP_2, horizon h within the regime bounds, submission time t.

Submission format. The payload is 28 floats, seven per regime: $[d, f, sl, tp_1, tp_2, h, id]$ with id a miner-chosen positive integer. The stop and targets are not prices: each is submitted as a distance from entry, expressed as a fraction of the entry price ($sl = 0.01$ puts the stop 1% away), and no level may sit more than 50% from entry. The entry price itself is recorded on the validator side, never submitted by the miner, so no payload field can encode an absolute price level. Semantics are constant-emission: every payload restates the current tuple, and a trade opens at the first row where a regime's id changes, so a dropped payload delays an open by at most one sampling interval and never cancels a live trade; a stale id never reopens. Opens are rate-limited to one per hour per regime.

Timelock. Submissions are drand-timelocked for one week and the validator holds raw payloads for one week (`PAYLOAD_MATURITY_BLOCKS` = 50,400) before attempting decryption: a payload becomes publicly readable

one week after its entry. That covers regimes A-C entirely; a regime-D trade at the top of its band (up to 336h) can still be live when its opening payload decrypts, a cost accepted. Entry prices are recorded at block arrival, before decryption, which ensures E is independent of the miner payload. The owner path decrypts immediately for trade execution and contract management; only public visibility waits. The cryptographic mechanism is specified in detail in §2.2.

1.2 Validity

Ordering: long requires $SL < E < TP_1 < TP_2$; short mirrored. Odds ratio $b = |TP_1 - E|/|E - SL|$. Implied win probability from the Kelly inversion

$$p = \frac{fb + 1}{b + 1}$$

must satisfy $p \in [0.01, 0.99]$; otherwise the submission is rejected.

1.3 Resolution

At $t + h$, validators walk 1-minute klines from ≥ 4 venues S (wick-feed rollout status and the close-only fallback: §6.14). For a long: the stop triggers on $\max_{s \in S}$ low (hardest to trigger: every venue must have traded through it), take-profits award on $\min_{s \in S}$ high (hardest to award); shorts mirrored. For the exits to count, they must be reached by the consensus of the exchanges. First terminal event in path order wins; a candle printing both SL and a target resolves stop-first. The resolution event is one of $\{SL, TP_1, TP_2, F\}$ where F is the horizon-end close. Maximum adverse excursion along the path, in the miner's risk unit:

$$R_{MAE} = \min\left(\frac{MAE}{|E - SL|},\ 1\right)$$

(capped at 1 because the stop truncates deeper drawdown).

1.4 Score

R-multiple at price P: $R(P) = (P - E)/(E - SL)$ for longs, mirrored for shorts. Base score by resolution event:

$$S_{base} = \begin{cases} R(TP_2)\,(1 + \delta) & TP_2 \text{ hit} \\ R(TP_1)\,\lambda & TP_1 \text{ hit, } TP_2 \text{ not} \\ -1 & SL \text{ hit} \\ R(F) & \text{neither} \end{cases}$$

Path penalty, tail amplifier, Kelly weight:

$$S_{path} = S_{base} - \gamma R_{MAE}, \qquad S_{tail} = S_{path}\left(1 + \tau \mathbb{1}[\sigma_{24h} > 2\sigma_{30d}]\right), \qquad S_w = f\, S_{tail}$$

where $\sigma_{24h}, \sigma_{30d}$ are realized vols at resolution time. The amplifier applies to losses as well as gains.

1.5 Aggregation and emission weight

Per-miner, per-regime EWMA over resolved trades with update weight α; regime EWMAs sum additively, and negative regimes count against the total (see §6.13 for why). Emission score decays with time since the miner's last *resolution* Δt_i:

$$S_{emission,i} = S_{total,i}\, e^{-\Delta t_i/\kappa}$$

so an open long-horizon trade does not decay. Pre-gate weights $w_i = S^+_{emission,i} / \sum_j S^+_{emission,j}$ over positive scores. New keys receive no emissions for the first 168h of history (probation): no return statistic is informative on day 3, so nothing pays before a key has at least a week on the tape. This exists as a backstop in case the gate logic is ever reconsidered; under the current architecture miners are not rewarded until at least four weeks from first submission anyway ($n_{min} = 4$ blocks, §1.7).

1.6 Collateral: skin in the game

Real skin in the game. Miners post subnet alpha as collateral behind their hotkey in the `FlowStake` contract (§3.2): minimum 1 alpha, no maximum, contract-owned stake pulled through the staking-v2 precompile after a miner-side approval, so only a coldkey can post collateral. The first funder controls the position and fixes the refund coldkey, so a hotkey hijack cannot redirect the alpha, and one coldkey can run positions behind any number of hotkeys, each independent. Two consequences:

Collateral is active the moment it lands. From the moment alpha is added it backs the Kelly sizing of every future trade; no past trade or resolved claim changes, so a top-up after a winning week buys nothing retroactively and no pending tier is needed. Validators read collateral at a block height rounded to a shared boundary, so every validator pricing the same trades sees the identical snapshot.

Scoring. The emission score is multiplied by posted collateral: real capital, not Kelly fractions on a paper account assumed to be the same size for everyone. Zero collateral earns zero. The significance gate (§1.7) is the deliberate opposite: it measures the return t-statistic on that same assumed account, so evidence stays collateral-free and capital without a proven record buys the dust tier at most.

Settlement. Every 7 days the alpha lost on losing trades is distributed pro rata to the winners. The execution model is one sentence: **miners post their own bets, by hand, on chain — the owner can only resolve them later.** Resolution is live, per trade, because the owner key path (§1.1) decrypts every payload at arrival; the figure walks the whole lifecycle by actor.

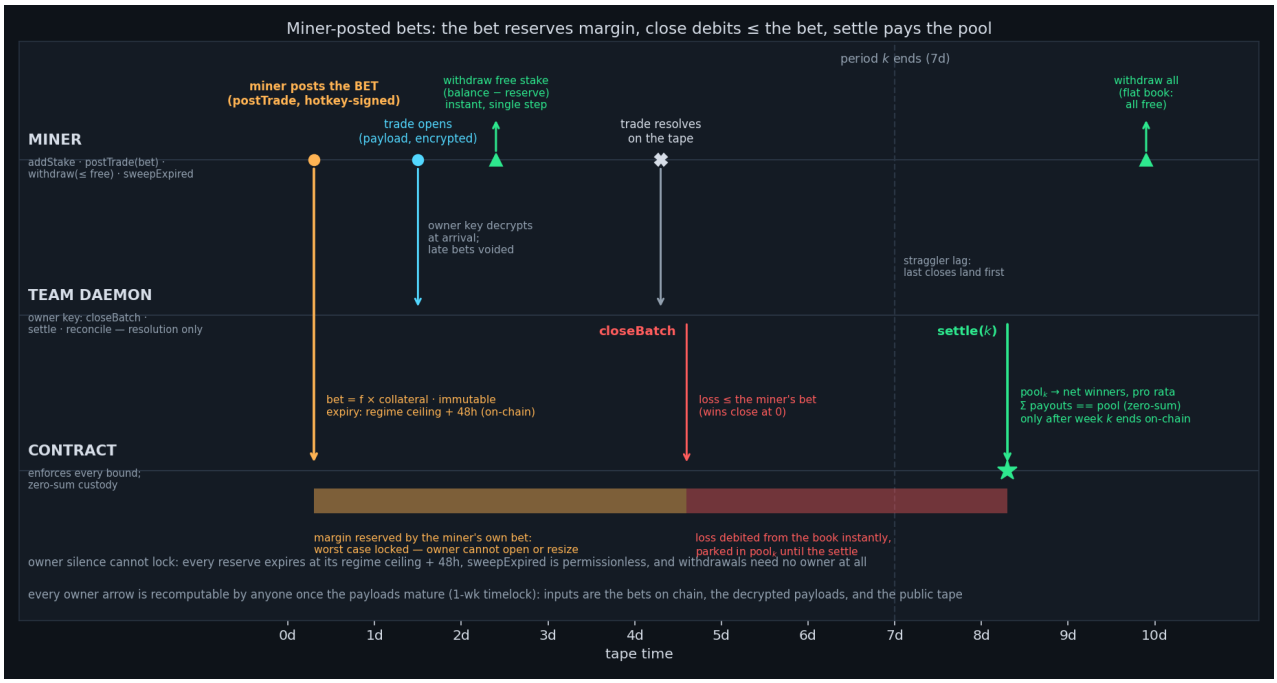


Figure 1: settlement lifecycle

The bet. Alongside each encrypted intent, the miner posts the trade's worst case on-chain themselves: `postTrade(hotkey, tradeKey, stake)`, normally $f\times$ collateral (a stop is exactly -1R, so the bound is tight), either from the EVM key that funded the position or — the native path — authored by the hotkey itself

(`postTradeSigned`: the chain's sr25519 precompile verifies the signature against the very 32-byte key the position is booked to, so any account may relay the transaction without gaining authority; the signed digest binds chain, contract, hotkey, trade key, size, and a strictly increasing nonce, making every signed bet single-use). The contract reserves the bet inside the position and enforces the max bet itself: no single bet may exceed a quarter of the book (`EXPOSURE_CAP_BPS`). A bet is immutable once posted — no resize, no cancel, because the poster knows the direction while everyone else waits on the timelock — and its expiry derives on-chain from the regime encoded in the trade key (horizon ceiling plus 48h), so a bet cannot be swept out from under its own resolution. There is no owner path that creates, resizes, or extends a bet: what is never posted can never lose — or win — a single rao. Settlement credits only bets that were on chain when their trade opened; a bet posted after the fact is voided both ways (the free-look guard), since a late poster has seen the tape.

Resolve. When a trade resolves, the owner posts `closeBatch` with the realized loss: debited immediately, bounded by the bet the miner posted and clamped to the position, into the running period's pool. Wins and flats close at loss 0 and just release their reserve. A trade belongs to the period its resolution falls in; a straggler that misses its period lands in the period open at posting time — and no debit can land in a period that has not started on the chain's own clock.

Settle. Once per period the accumulated pool pays out, attributed net per hotkey: books the live feed over-debited (gross losses beyond their net) are refunded first inside the settle batch, then net winners split the remaining pool pro rata by net P&L. The contract enforces the batch zero-sum in rao (losses can only move to winners, never the treasury) and consumes period ids strictly in order, so a batch cannot be replayed. The week itself is chain state: period k covers $[\,\text{periodZero} + (k{-}1)\cdot 168\text{h},\ \text{periodZero} + k\cdot 168\text{h}\,)$ with `periodZero` fixed forever at deployment, and `settle(k)` reverts until week k has ended (`PeriodNotOver`) — so settlement velocity is bounded to one real week per period id, whatever the owner does. A period with no winners rolls its pool into the next.

The money. Per resolved trade, monetary P&L is realized R $\times$ the posted bet, money rather than score: a stop is exactly -1R and no scoring adjustment $(\delta, \lambda, \gamma, \tau)$ applies. Open-time pricing means a deposit backs only trades opened after it and alpha that leaves stops backing new trades the moment it goes; emission weight (§1.5) reads current collateral, so the two move together. A hotkey's loss per trade is capped at its own posted bet, and a period circuit breaker caps its losses in aggregate: once a book has lost a quarter of its collateral inside one settlement period, its remaining trades that period are void for money, in both directions, so there is no free option after the breaker trips. Scores still count; only the money stops. The contract enforces the same bound independently (§3.2) — and because the week clock is on-chain, "per period" means per real week at any posting velocity. Winners split exactly what the losers forfeited, which can be more or less than their notional win. Resolution numbers are computed off-chain by the owner, but every posted close and settle is recomputable by anyone: the inputs are on-chain (the bets, collateral at the open block, period ids) or public once the payloads mature (decrypted payloads, the tape), so within the timelock week every posting is checkable, and the contract bounds what a bad post could do in the meantime (a debit can never exceed the miner's own bet or the position, and a pool can only pay collateralized winners).

The exit. At any moment, the summed worst case of the miner's own posted bets cannot be withdrawn; everything else can, instantly, in one transaction. Losses debit at close, so no exit can outrun one. If the owner goes silent, reserves self-release at their on-chain expiry and `sweepExpired` is permissionless. The price of collateralized bets is visibility: bet size per trade id appears on-chain the moment the miner posts it, and realized losses appear at resolution; direction, levels, and horizon stay sealed until the payloads mature (§3.4).

Rollout status. The contract (`FlowStake.sol`), the validator weighting, and the settlement daemon (`flow_settlementd.py`) ship with this release; collateral weighting turns on when the validator is pointed at the deployed contract (`FLOW_STAKE_ADDRESS`), no later than the emission turn-on. The grid simulations of §4 implement an earlier per-epoch draft of this layer (lock $f \cdot \min(A, A_{max})$, per-epoch loss pool, bootstrap credit); the study of §6.15 implements the shipped weekly-settlement semantics directly, with exits modeled one settlement period behind (conservative: the shipped contract releases free margin instantly).

1.7 Payment layer

Gate statistic. Per key, per-trade S_{base} values are summed into consecutive calendar blocks of length $L = 168$h keyed on resolution timestamp: $B_k = \sum_{j:\lfloor t_j/L \rfloor = k} S_{base,j}$. With $n \geq n_{min} = 4$ blocks, define the window statistic

$$t(W) = \frac{\bar{B}_W}{s_{B_W}} \sqrt{|W|}, \qquad s_B = \mathrm{sd}(B_k,\ \mathrm{ddof} = 1),$$

and the gate statistic as the better of the full record and the recent one:

$$t_{stat} = \max\big(t(\text{all blocks}),\ t(\text{last } n_{min} \text{ blocks})\big).$$

The rolling window lets a strong recent month clear without waiting out a long cold prefix; the full history keeps a long consistent record cleared through a flat month. The statistic is computed on S_{base}, i.e. raw realized R with the δ/λ event weights but **before** γ, τ, f; scoring parameters therefore cannot change who clears or when (verified empirically in §6.12).

Clearance and latch. A key clears at $t_{stat} \geq z_{in} = 1.25$ and, once cleared, stays cleared while $t_{stat} \geq z_{out} = 0.5$. An alternative population-deflated threshold $z = \Phi^{-1}(1 - q/N_{keys})$ is implemented but not used in the ship spec.

The rolling window also multiplies a noise key's chances: every key retries on each new month, and eviction requires both windows below the latch. That cost is priced in the results: it is §6.4's extra pass and about 1.3 points of leak, §6.8's rows are the price of retrying across many keys, and §6.15's drain is what each surviving key pays to collect. The governed response, if live behavior outruns the simulated envelope, is the alpha statistic (§6.7) and the threshold.

Clearance makes two distinct demands. The threshold itself demands **evidence of statistical significance**: at least four independent week-blocks of realized R whose mean stands 1.25 standard errors above zero, over the whole record or over the last four blocks. Significance alone is not the paid claim: the block structure (§5) strips out the significance a leveraged beta strategy can manufacture from overlapped trades, and the adversarial results (§6) price what survives. What a cleared key holds is **evidence of genuine market intelligence**: an edge that stands after de-overlapped exposure and an adaptive adversary population, not leverage or submission frequency.

Split. Cleared keys split $(1 - d)$ of the pool pro rata by w_i; uncleared keys with positive score split the dust tier $d = 0.02$ pro rata by score. The remainder, the full main share whenever no key is cleared, **burns**. There is no escrow and no retroactive credit: a key's proving period earns dust only, by design.

No entry bond. Registration carries no deposit beyond the chain's registration fee; deterrence is priced in §6.3 and carried by the collateral settlement (§6.15).

1.8 Launch parameters

Symbol	Value	Role
δ	0.30	TP2 completion bonus
λ	0.50	TP1 partial factor
γ	0.25	path penalty on R_{MAE}
τ	1.00	tail amplifier (2× multiplier)
α	0.05	EWMA update weight
κ	672h	emission decay constant ($\geq 2\times$ the longest horizon)
probation	168h	pre-emission history requirement
L	168h	gate block length

Symbol	Value	Role
n_{min}	4	minimum blocks before statistic defined
z_{in}	1.25	clearance threshold (on the max of full-history and rolling last-4 statistics)
z_{out}	0.50	exit latch
d	0.02	dust tier fraction
period loss cap	0.25	circuit breaker: max fraction of a book that can be lost to settlement in one period (validator and contract both enforce it)
period clock	`periodZero` $+k \cdot 168$h	on-chain week boundaries, fixed at deployment: no debit into an unstarted week, no settle before a week ends — the breaker binds per real week
entry bond	none	deterrence lives in the gate and the collateral settlement layer (§6.3, §6.15)

---

2. System architecture

FLOW runs inside the existing MANTIS validator as one challenge on subnet 123 (256 UIDs, 12s blocks). One central validator collects payloads and publishes the archives; every other validator runs the same binary against the same public inputs. Convergence does not depend on validators talking to each other: every scoring quantity, including gate latch state, is recomputed from scratch each weight epoch as a pure function of the shared datalog plus pinned chain snapshots (§2.4, §3.4), so identical inputs force identical weights.

2.1 Component map

```
miner strategy / miner_dashboard
      |
      |  generate_v2: 28-float intent -> encrypted envelope (2.2)
      v
R2 bucket object (key == hotkey)  <--- commit URL registered once
      |                                on subtensor (netuid 123)
      |  fetched every sampled block (60s)
      v
+- validator.py ------------------------------------------------+
|                                                               |
|  append_step: price row (fixes E) + raw payload -> datalog    |
|                          |                                    |
|  decrypt loop (5s): payloads older than 50,400 blocks         |
|      drand signature -> tlock decrypt -> embeddings           |
|                          |                                    |
|  weight worker (every 1,000 blocks):                          |
|      expanding panel -> flow.py scoring (2.4)              |
|          skin weight <- FlowStake.activeStakeOf (EVM, 3.2)  |
|                          |                                    |
|      multi_salience -> set_weights (every 360 blocks)         |
+---------------------------------------------------------------+
```

```
      |
      v
emissions (weight 0 until 2026-09-14, then 25% pool share)
```

The miner-facing surface is exactly three operations: keep one encrypted object in an R2 bucket current, keep collateral posted on the EVM, and post each trade's bet on-chain before (or as) the payload that carries it uploads (once the collateral layer is live). The miner dashboard does all three in one submit. Everything else is validator-side.

2.2 Submission path and payload cryptography

A miner registers a commitment on subtensor once: the URL of one R2 object. The validator enforces the URL shape (R2 host, path exactly the hotkey, no directories) and refetches the object every sampled block. Rotating content in place is the whole protocol; there is no per-submission chain transaction.

The object is a v2 envelope produced by `generate_and_encrypt.py`:

```
alg = x25519-hkdf-sha256+chacha20poly1305+drand-tlock

{
  "v": 2,
  "round":    r,           drand quicknet round ~1 week out
  "hk":       ss58,        submitting hotkey
  "owner_pk": hex32,       team HPKE public key
  "C":        {nonce, ct}  ChaCha20-Poly1305(key) over the plaintext,
                           AAD = binding
  "W_owner":  {pke, nonce, ct}
                           payload key wrapped to owner_pk via
                           X25519(ske, owner_pk) + HKDF, AAD = binding
  "W_time":   {ct}         tlock.tle(r, ske || key): opens when drand
                           publishes round r's signature
}

binding   = SHA256(hk : round : owner_pk : pke)
plaintext = canonical JSON {"FLOW": [28 floats], ..., "hotkey": hk}
```

Properties the envelope buys:

- **Commitment before entry.** The tuple is sealed under the timelock before the entry price it will receive exists; combined with validator-recorded E and fractional brackets (§1.1), no field can encode an absolute price and no entry can be self-reported.
- **Copy resistance.** `binding` is authenticated data in both ciphertexts and commits to the hotkey and round. Re-uploading someone else's envelope under a different hotkey fails decryption (`obj["hotkey"]` is also re-checked against the submitting key after decrypt), and the one-week round means even the owner path reveals nothing to other miners.
- **Two decryption paths.** `W_time` opens publicly when drand reaches round r (quicknet, 3s period; r is computed from the beacon's genesis time for `lock_seconds` = 604,800). `W_owner` lets the team decrypt immediately for copy-trade execution; public visibility still waits the full week.

2.3 Validator pipeline

Three concurrent loops plus a worker thread, all in `validator.py`:

Cadence	Work
every sampled block (5 blocks = 60s)	fetch prices (`price_service`), fetch all commit objects, `datalog.append_step`: one price row per challenge (this is the row that fixes E; for FLOW it also captures the service's per-asset wick keys, `BTC_HIGH`/`BTC_LOW`, when present and sane), one raw payload row per hotkey
every 5s (decrypt loop)	select raw payloads older than `PAYLOAD_MATURITY_BLOCKS` = 50,400, group by drand round, fetch signatures (memory + SQLite cache), tlock-decrypt, validate, write embeddings, delete raw rows
every 1,000 blocks (weight worker)	stream the expanding panel per challenge from SQLite, run `multi_salience`, which dispatches FLOW to `flow.py` with the collateral snapshot (§3.3)
every 360 blocks	set weights on subtensor from the last computed salience
every 100 blocks	metagraph sync

Scoring reads stop `TASK_INTERVAL` = 500 blocks behind the head so every validator scores against a settled prefix of the log. A payload whose drand signature cannot be fetched is retained and retried on the next pass rather than consumed as zeros; a round that never resolves is force-consumed after a one-week grace window (`DRAND_RETAIN_GRACE_BLOCKS` = 50,400) so a garbage round number cannot pin the table.

2.4 Scoring dataflow

`flow.py` is a pure function: expanding panel in, pool fractions out. No state survives between weight epochs; the latch, EWMA, and decay are all recomputed from the full history every time, which is what makes cross-validator agreement a property of the inputs rather than a synchronization protocol.

```
expanding hourly panel (datalog)
  X: T x (H x 28) intents        P: T x [close, high, low]
        |
        v
decode_trades (per key, per regime)              sec 1.1
  new trade_id at an observed row -> open
  E = validator price at the open row
  1 open/hour/regime; dropped rows delay, never cancel
        |
        v
_resolve (path walk, wick channels)              sec 1.3
  stop on adverse wick, target on favorable wick
  stop-first ties; event in {SL, TP1, TP2, F}; MAE
        |
        +----------------------------------+
        v                                  v
  score stack (sec 1.4, 1.5)         gate stream (sec 1.7)
    S_base -> gamma path penalty       S_base only, summed into
    -> tau tail amp -> f weight        168h blocks keyed on
    -> per-regime EWMA (alpha)         resolution timestamp
    -> kappa decay, probation          t-stat over n >= 4 blocks
    -> x posted collateral (when live)      max(full, last-4) t
                                            clear 1.25 / hold 0.50
        |                                  |
```

```
      v                                    v
  split (sec 1.7): cleared keys share (1 - d) by weight;
  uncleared positive keys share d = 0.02 by score;
  remainder -> "__burn__";
  trailing-24h mean allocation -> pool fractions
```

The price input carries `[close, high, low]` columns, the venue consensus of §1.3. Exits are conservative: they count only where every exchange agrees the print happened. For a long,

$$SL \text{ triggers at } \max_{s \in S} \text{low}_s, \qquad TP \text{ awards at } \min_{s \in S} \text{high}_s$$

(shorts mirrored): hardest to trigger, hardest to award. Rows without wick data resolve on closes, a degradation priced in §6.14; the wick feed ships from challenge start.

2.5 Storage

```
.storage/
  mantis_datalog.db      legacy challenges (SQLite, WAL)
    blocks, challenge_meta, challenge_data,
    raw_payloads, drand_cache, breakout_state
  flow_datalog.db     ATTACHed as `inv` on every connection
    challenge_data       FLOW rows only (price rows + embeddings)

bucket/
  datalog.db             existing publish flow, unchanged
  flow_datalog.db     VACUUM INTO snapshot (snapshot_for_publish)
```

FLOW rows live in their own SQLite file, attached to every connection as schema `inv` and routed by ticker, because the legacy datalog has grown too large to keep re-shipping for one challenge. Shared state stays in the main DB: one raw payload covers every challenge, so it cannot be split per ticker. Crash and outage behavior is regression-tested in `test_robustness.py` under one rule: transient infrastructure failure never destroys or fabricates miner data.

2.6 Market-context logger

The team privately logs detailed market-microstructure variables across many timeframes, in a tamper-evident record kept off the public codebase and outside scoring. It exists for one question, asked after the fact: does a miner's performance line up with genuine microstructure conditions, or only with luck? It is never a scoring input, and nothing in the public validator depends on it.

---

3. Contract architecture

One contract on the Bittensor EVM, one Foundry package (`flow_stake/`): `FlowStake.sol` (trade stake, §1.6). The write model is asymmetric by design: miners post their own bets on chain (hotkey-signed or from the funding key), and the owner can only resolve them — close at a bounded loss, settle the weekly pool zero-sum, repair rounding dust. Collateral moves through zero-sum settlement and the miner's own instant, margin-gated withdrawal. 53 unit and fuzz tests cover it, plus an anvil integration suite driven through the same Python client the validator uses and the same daemon the team runs, with real sr25519 signatures.

3.1 Custody model

Alpha is not an ERC20; it exists only as stake keyed by `(hotkey, coldkey, netuid)` inside the subtensor runtime. The contract therefore holds miner alpha as *stake owned by its own ss58 mirror coldkey* (`blake2_256("evm:" ++ address)`, recorded once via `setContractColdkey`), staked to the miner's own hotkey, and moved through the staking-v2 precompile at `0x...0805`:

```
miner EVM key (H160) --- mirror ---> miner ss58 coldkey
      |                              owns alpha staked to the hotkey
      |  1. approve(contract, netuid, amount)      on 0x...0805
      |  2. addStake                               on the contract
      v
contract pulls: transferStakeFrom(miner, contract, hotkey, ...)
      |
      v
stake position (hotkey = miner's, coldkey = contract's mirror)
      |                            |
  withdrawal path             settlement moves
  transferStake to            moveStake between hotkeys
  refundColdkey               (contract owned, zero-sum)
```

Consequences of this shape:

- **Only a coldkey can fund.** `transferStakeFrom` pulls from the caller's own mirror coldkey; there is no path for a hotkey to post someone else's alpha.
- **Per-hotkey accounting is inherent.** The stake stays staked to the miner's hotkey; the runtime's own bookkeeping is the ledger, and the miner cannot move it because they do not own it.
- **Receipt is verified, not assumed.** After the pull, the contract reads its own position (`getStake`) and requires the delta within `AMOUNT_TOLERANCE` = 10,000 rao of the requested amount (share-pool rounding slack); a shortfall reverts.
- **First funder controls.** The first `addStake` fixes the depositor and the refund coldkey. A hotkey hijack cannot redirect alpha at exit, and one coldkey can run any number of hotkeys, each position independent.
- **Precompile dispatch.** State-changing precompile calls go through a raw `call` (interface dispatch does not reach the runtime precompile); views go through `staticcall`. Failures revert the whole operation: there is no state in which the contract's books and the runtime's stake disagree.

3.2 FlowStake

One position per hotkey, minimum `MIN_STAKE` = 1 alpha, no maximum, depositor-gated top-ups. Collateral is active the moment it lands; the position's life is a loop of reserve, debit, and payout:

```
 addStake (approve first; >= 1 alpha total)     active immediately
    |
    v
  BALANCE ---- postTrade / postTradeSigned (MINER): the bet reserves
    |            |     its worst case, normally f x collateral;
    |            |     expiry fixed on-chain by the key's regime
    |            v     (horizon ceiling + 48h); immutable once posted
    |         POSTED BET --- closeBatch (owner): loss debited into
    |            |           the period pool (<= the miner's bet,
    |            |           <= balance); wins close at 0
    |            |--- sweepExpired (permissionless): owner went
    |            |    silent past expiry, reserve self-releases
    |            v
    |         reserve released
    |
    |    <-- settle(periodId): period pool pays net winners pro rata
    |        after refunding over-debited books, zero-sum in rao,
    |        period ids strictly ordered, only after the week ends
    |
    |  withdraw(amount <= balance - posted bets)    single step,
    v                                               instant
  paid to the refund coldkey; clamped to the stake physically held
```

```
minus rao parked for unsettled pools; a flat, fully-settled
remainder below 1 alpha exits fully; an emptied position is deleted
```

The miner side creates every trade; the owner side can only resolve. Each operation enforces its invariants on-chain:

1. `postTrade(hotkey, tradeKey, stakeRao)` — **miner (depositor key).** One bet per trade key; size capped at a quarter of the position's balance (`EXPOSURE_CAP_BPS` = 2,500), the spec's max bet enforced on-chain. Immutable once posted: no resize, no cancel, because the poster knows the direction while everyone else waits on the timelock. Expiry is not caller-chosen: it derives from the regime encoded in the trade key (horizon ceiling plus a 48h resolution buffer, at most 384h for regime D), so a bet cannot be swept out from under its own resolution and no reserve can outlive owner silence. The reserve is bookkeeping inside the position: balance does not move.
2. `postTradeSigned(hotkey, tradeKey, stakeRao, nonce, r, s)` — **miner (hotkey), relayed by anyone.** The native path: the hotkey a position is booked to *is* an sr25519 public key, and the chain's verification precompile (`0x...0403`) checks the signature against it. The digest binds chain id, contract address, hotkey, trade key, size, and a strictly increasing per-hotkey nonce, so the relaying account has zero authority: it can neither alter a bet nor replay one (trade keys free up after a close; the burned nonce keeps old signed bets dead forever). Miners need no EVM key of their own to bet — the relayer only pays gas.
3. `closeBatch(periodId, hotkeys[], tradeKeys[], lossRao[])` — **owner.** A close must reference a bet the miner posted (`UnknownTrade` otherwise) and its loss can never exceed that bet (`LossExceedsExposure`) or the position (clamped). The period circuit breaker clamps on top: cumulative debits from one miner within one period never exceed a quarter of its period-start balance (with d already debited and b the current balance, the next debit is capped at $(b - 3d)/4$, which is that bound exactly); the excess is simply not collected. A debit cannot land in a period that has not started on the chain clock (`PeriodNotStarted`), so the breaker's allowance cannot be drawn early. The debit leaves the book immediately and accumulates in `poolAccum[periodId]`; the rao stays physically parked on the loser's hotkey (tracked as `pooledOut`) until the settle sweeps it. `periodId` must be the period awaiting settlement or the one after it.
4. `settle(periodId, winners[], winRao[])` — **owner.** `periodId == lastSettledPeriod + 1` or revert: batches can never be skipped, replayed, or reordered — and `settle(k)` reverts until week k has ended on the chain's own clock (`PeriodNotOver`), so settle velocity is bounded to one real week per period id. Payouts must equal `poolAccum[periodId]` exactly, in rao, or the batch reverts (`NotZeroSum`): settlement moves alpha from losers to winners and nowhere else; nothing mints, nothing routes to the treasury. A period with no winners rolls its pool into the next period. Physical stake moves through `winners[0]` as pivot, each move clamped to the stake actually held on the origin hotkey, because the runtime's share-pool rounding can deliver a few rao less than requested. Books stay exact and authoritative; physical drift is bounded dust, absorbed at withdrawal by the same clamp, and repairable by `reconcile`.
5. `reconcile(from, to, amount)` — **owner.** The drift janitor: moves contract-owned stake from a hotkey holding *more* than its booked total (balance plus parked pool rao) to one holding *less*. Both bounds are enforced on-chain, so it can only move phantom stake toward phantom books; it cannot touch any booked balance, any parked pool, redirect a withdrawal, or create value.

The miner's calls are `addStake`, `postTrade`/`postTradeSigned`, and `withdraw`. Withdrawal is instant up to the free stake (balance minus the miner's own posted bets); the runtime refusing a transfer (a dust amount below its minimum) reverts atomically and a different size goes through, so no request shape can lock a position. `sweepExpired` is callable by anyone, so a miner can always release margin the owner failed to close.

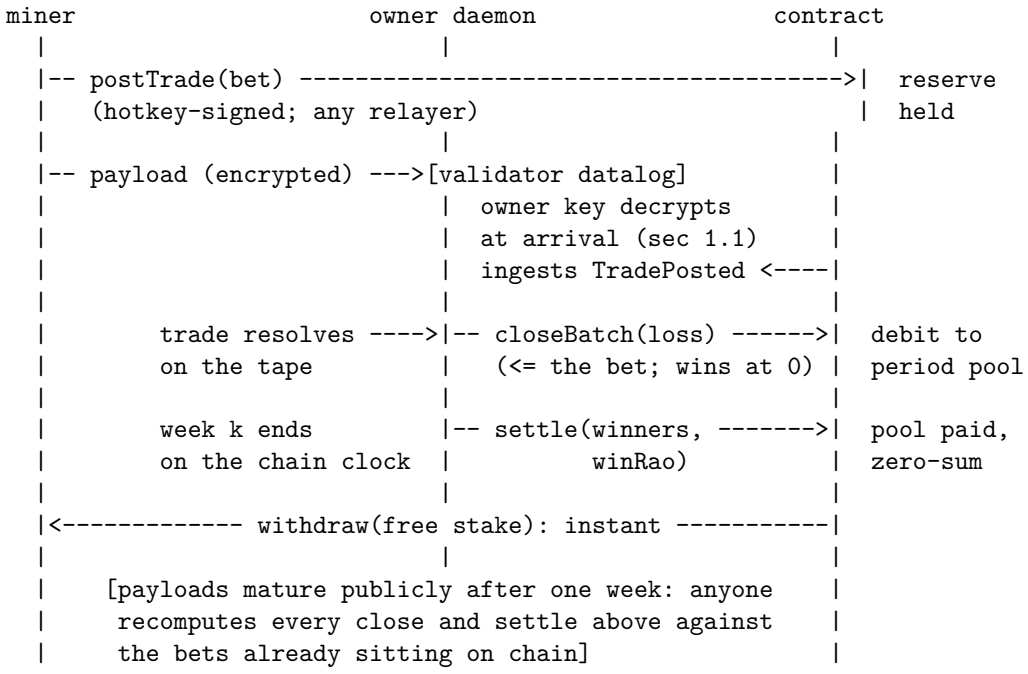
There is no other write path: posted collateral leaves a position through a settlement debit to that period's winners or the depositor's own withdrawal, and nothing can ever be at risk that the miner did not personally post. What the owner's remaining resolution power is worth is the subject of §3.4.

The per-batch rao rounding (`pro_rata_payouts` in `flow_stake.py`) floors each winner's share and gives the remainder to the largest claim, so posted settles are exactly zero-sum by construction and recomputable as a pure function.

Validators never write to this contract. They read `activeStakeOf` for the hotkey set at a block height rounded down to `PIN_BLOCKS` = 300 (one hour), so every validator sampling within the same window prices skin from the identical chain state; reads are fail-static (on RPC failure the last good snapshot is served). No contract configured means f-only weights (§1.6 rollout).

3.3 Validator and EVM interaction

The miner writes the bets; the owner writes only resolutions. The settlement daemon (`flow_settlementd.py`, run by the team) never opens anything: each cycle it ingests the miners' `TradePosted` events from chain — that sync *is* how trades enter it — replays the full expanding panel through `flow.trade_events`, matches each decoded trade against its posted bet, and posts the resolutions. Validators are read-only against the contract. What each party can do is bounded by the contract, not by policy (§3.4). The daemon is a replay-diff-post loop, stateless in spirit (its posted-set cache rebuilds from contract event logs) and idempotent by construction.



Path	Direction	Cadence	Failure mode
`activeStakeOf` batch	validator reads stake	every weight calc, pinned to 300-block boundaries	fail-static snapshot
`postTrade` / `postTradeSigned`	**miner writes bets**	alongside each payload upload, before the trade opens	an unposted bet carries no money, ever — the trade scores but cannot win or lose a rao; a bet posted late (after its trade opened) is voided both ways by the free-look guard
`closeBatch`	daemon writes debits	minutes-hours after resolution	unposted close leaves the reserve held until the bet's on-chain expiry, then it self-releases; the loss is simply never collected

Path	Direction	Cadence	Failure mode
`settle`	daemon pays the pool	weekly, only after the week ends on-chain	unposted settle just delays the payout; periods stay ordered, pools roll
`reconcile`	team repairs drift	as needed, dust scale	can only move phantom stake toward phantom books (both bounds on-chain)

Resolution and drift repair are the team's entire on-chain surface; nothing the team signs can create risk.

The daemon runs on two boxes. Both hold the owner key and re-sync their posted sets from contract event logs every cycle, so neither re-posts the other's work; the standby differs in exactly one way: it posts only work that has gone stale (a close whose resolution is older than its takeover threshold, 30 minutes by default, still absent from the chain, or a settle overdue by the same margin). Liveness is measured by unposted work rather than heartbeats, so there is no election to misconfigure, a returning primary simply finds the backlog posted, and a true race is harmless: both boxes compute identical numbers from the same bets and the same panel, so the contract rejects the duplicate (`UnknownTrade` / `BadPeriod`) at the cost of one reverted transaction. Note what the standby no longer has to protect: under the old owner-posted-opens design, a late open was a window in which a watching miner could withdraw collateral that should have been reserved. That window is structurally gone — the miner's own bet reserves the margin before the payload even uploads — so daemon lag now only ever delays debits and payouts, never custody.

3.4 Trust model

Stated honestly: the owner is trusted for *resolution*, and for nothing else. The team runs the tape, decrypts at arrival, and computes the loss and settlement numbers — but it cannot create risk. Every trade the money layer touches exists because a miner personally posted its worst case, signed by the miner's own key; the owner's entire write surface is resolving those bets and paying the weekly pool, inside bounds the contract enforces:

Power	Held by	On-chain constraint
`postTrade` / `postTradeSigned`	**miner only**	one bet per trade key, capped at a quarter of the book; immutable once posted; expiry fixed by the key's regime; hotkey signature verified by the chain (`0x...0403`), nonce burned per signed bet — a relayer can neither alter nor replay
`closeBatch`	owner	must reference a bet the miner posted; loss capped at that bet, at the position, and by the period circuit breaker (at most a quarter of a period-start balance leaves one miner per period); cannot debit into a week that has not started; debits go to the period pool only
`settle`	owner	zero-sum against the accumulated pool, in rao; period ids strictly sequential; cannot run before the week ends on the chain clock; can only pay collateralized hotkeys

Power	Held by	On-chain constraint
`reconcile`	owner	source must hold more than its booked total, destination less; cannot touch any booked balance or parked pool
ownership	owner	plain governance setter, evented

What a fully dishonest owner could still do, exhaustively: fail to close a bet (its reserve self-releases at the on-chain expiry; the loss is never collected — the pool eats it, no miner does), debit a real losing bet up to the worst case its own miner posted, and misroute a week's pool among collateralized miners — at most a quarter of any book per real calendar week, because the week boundaries are chain state. It cannot open, resize, or extend a bet; it cannot touch a book whose miner posted nothing; it cannot settle faster than real weeks; nothing mints and nothing routes to a treasury.

The check on the remaining resolution power is that everything the owner posts is public and recomputable: every close and settle is a deterministic function of the bets on chain, the decrypted payloads (public after one week), the tape, and collateral at the trade's open block, so a bad post is provable by anyone, in public, within the timelock week — and the free-look guard is symmetric, applied to the owner's daemon by code and verifiable from the same public data. Bounded resolution, verified after the fact, is the honest description; a miner who cannot accept a one-week verification window should size their bets accordingly.

The bet layer also reveals something before public decryption: bet size per trade id (posted by the miner themselves) and realized losses per hotkey, in near real time. Direction, levels, and horizon stay sealed until the payloads mature; what leaks early is that a pseudonymous key risked some amount and how much of it resolved away.

---

4. Simulation framework

4.1 Tape and units

Real multi-venue BTC 1-minute tape, two years: Binance, Bybit, OKX, and Coinbase, fetched and assembled by `flow_sim/fetch_tape.py` into `flow_sim/cache_btc_multi_1m.npz`. Resolution follows the conservative-execution rule of §1.3 on venue-consensus channels: the low channel is the max of venue lows, the high channel the min of venue highs, so a simulated exit prints only where every venue agrees it traded; closes (decisions, F exits, returns) are the Binance print. Epochs are hourly; the pool emits 1.0 unit per epoch, 24 per day, so all money quantities below are in **pool-days** (1 pool-day = one day of total challenge emissions; the tape holds ~720) and scale to any TAO rate. The final 336h of the tape (one maximum regime-D horizon) are excluded from submission so every prediction can resolve. Rolling $\sigma_{24h}, \sigma_{30d}$ are computed from log returns and drive both bracket sizing and the tail condition.

4.2 Mechanism implementation

`flow_mechanism.py` implements §1.1–1.5 exactly (validation, path resolution with stop-first tie-break, full scoring stack, EWMA, decay) plus the per-epoch collateral draft noted in §1.6 (loss-pool settlement, bootstrap credit, track-record cap); `rational.py` implements §1.7. Stops and targets trigger on the consensus channels of §4.1, the same conservative-execution rule the validator applies.

4.3 Rational entry model

A prospect pool of 240 agents, each holding one strategy from the adversary roster (§4.4), spread evenly across the pool. The simulated momentum strategies are overfit to the test tape itself: our estimate of a lucky leveraged-beta strategy, not of a realistic entrant. The rest of the roster is basic noise. Patience $T_i \sim \text{LogNormal}(\ln 60, 0.5)$ days, clipped to [20, 240]. Once per day, up to `entries_per_day` prospects may register. Agent i registers iff

$$\hat{\rho}\, T_i - c_{op} T_i - B > 0$$

where B is a counterfactual entry burn in units ($B = 0$ at ship; §6.3 sweeps it), $c_{op} = 0.5$ units/day is operating cost, and $\hat{\rho}$ is the *marginal* observer estimate of per-key capture: with k active noise keys earning per-day rates $\rho_1 \dots \rho_k$ (public on-chain data, measured over a trailing 14d window), $\hat{\rho} = \sum_j \rho_j/(k+1)$; cold-start prior $\hat{\rho} = 24/(N_{act}+1)$. No belief noise (`belief_sigma = 0`): entrants are exactly rational on public information. An active agent quits when tenure $\geq$ 14d and its own trailing 14d capture rate is below c_{op}. An agent that busts its collateral re-registers iff the same EV test passes over its remaining patience; otherwise it quits.

4.4 Adversary strategies

The adversary in all gate studies is the **leveraged beta adversary**: the in-sample-optimized momentum pair (`tmom72`, `tmom168`). Trailing-return sign over a 7-day lookback, **no trend threshold**, brackets $SL = 0.5\sigma_h$, $TP_1 = 1\sigma_h$, $TP_2 = 2\sigma_h$ (vol-scaled to horizon $h \in \{72, 168\}$h), fixed $f = 0.05$, the aggressive end of ordinary 1–5% account sizing, submitting every 8h, i.e. heavily overlapping positions. These parameters were grid-searched on a full year of BTC and held fixed; the strategy stays profitable before mechanism costs across both test years, including the 2026 crash. Coinflip and lottery opportunists exist in the wider roster and are strictly dominated by tuned momentum; gate studies run against the full roster during design confirmed nothing changes when they are included.

4.5 Skilled reference

`OracleSkillMiner`: a scripted-outcome profile answering "suppose a strategy with win rate w at 2:1 exists" without modeling signal generation. Regime C only, $h = 168$h, setup checks every 12h with an 84h post-trade cooldown (realized rate 97–99 resolutions/yr $\approx$ 1.9 trades/week), brackets $SL = 0.35\sigma_h$, TP_1 at 1R, TP_2 at 2R, $f = 0.03$: ordinary account sizing, well inside the 0.25 cap and below the adversary's 0.05. Wins/losses are drawn to hit w among taken trades; discretionary setup-skipping keeps the realized rate at the target. The reference profiles are $w = 0.70$ (strong) and $w = 0.50$ (moderate; per-trade expectation $\approx +0.5$R gross of penalties). Zero-edge at a 2:1 bracket is $w = 1/3$.

4.6 Accounting

Adversary net = emissions + max(alpha P&L, 0) − operating cost − the counterfactual entry burn where §6.3 sweeps one (bootstrap credit treated as free protocol money; loss-pool wins count: the attacker-favorable convention). Skilled net = emissions. The capital-at-risk study of §6.15 replaces the max(·, 0) convention with the shipped weekly settlement: real collateral, real losses, in TAO.

---

5. Why the gate aggregates blocks

A per-trade statistic $t = \bar{R}/s_R \cdot \sqrt{n}$ over individual resolutions treats n as the evidence count. That is the wrong unit for markets: two trades opened hours apart into the same multi-day horizon ride the same price path, so their returns are one draw counted twice. The tuned adversary submits every 8h into 72–168h horizons; on the test tape its per-trade t ran an order of magnitude above its block statistic, enough to clear any usable threshold within weeks on pure correlated exposure.

Previous subnets have seen where this leads: an operator spreads a number of highly correlated bets, effectively the same volatility ticket, across many miner keys; variance hands a few of them a passing record; the operator then concentrates into the survivors' position. The evaluation was passed by the spread, not by an edge. Block aggregation is part of FLOW's defense against this: summing S_{base} into non-overlapping 168h calendar blocks makes the evidence unit one week of market exposure, regardless of submission frequency, so correlated bets collapse into the same few blocks instead of counting as independent evidence, and a manufactured record has to keep surviving week after week rather than once. The rest of the defense is per key: crowding dilutes the spread's emission take (§6.8) and every surviving key must hold collateral inside the settlement drain to collect (§6.15).

§6.5 and §6.6 quantify the failure when blocks are too short or too few, and price the 336h block that would fully de-overlap the top of regime D. This is the split the gate lives on: the t-statistic supplies evidence of statistical significance, and the block unit is what makes that significance evidence of genuine market intelligence rather than of correlated exposure.

---

6. Results

All tables: real BTC year, ship spec (§1.8) unless a column overrides it, rational entry per §4.3, skilled joins an empty pool on day 0. Per-seed data: `out/papergrid.csv`, `out/finalspec.csv`, `out/stakeecon.csv`. Columns, used throughout:

Column	Meaning
Leak %	emissions captured by leveraged beta adversaries, as % of everything the pool *could* have emitted over the two-year tape (~720 pool-days; 10% = 72 pool-days)
Adv EV	net for one leveraged beta *participant*, per entrant, in pool-days: emissions + positive collateral P&L − operating cost (attacker-favorable, §4.6). The EV of participating, not of the strategy, which is positive on this tape by construction (§4.4)
Clear	days until the skilled key clears the gate (mean/med/max over seeds)
Steady	the skilled key's steady-state pool share
Passes	distinct noise keys ever clearing the gate, summed over seeds
Crowd	equilibrium count of concurrently registered adversary keys

Two accounting layers, read together. §6.1–6.14 price the *emission layer only* (f-only weights, no collateral). §6.15 prices the *capital-at-risk layer*: pool weight requires posted collateral, collateralized beta drains to stronger books through weekly settlement, and the period circuit breaker bounds what any book can lose in one period at a quarter of its collateral.

6.1 Ship-spec headline (10 seeds)

Profile	Cleared	Clear day mean/med/max	Steady share	Leak %	Adv EV
$w = 0.70$	10/10	28.0 / 29.0 / 36.0	0.80	9.1%	+1.07
$w = 0.50$	10/10	35.5 / 29.0 / 58.0	0.73	12.1%	+1.52

The figure: daily emission share, 10-seed mean. Green = skilled, red = leveraged beta (paid), grey = withheld and burned (~the whole pool minus dust until first clearance). The strong profile clears within 36 days in every seed, and the moderate one within 58: the rolling window turns the worst moderate seed from 218 days (full-history statistic) into 58. The red band is the late-2024 rally buying roughly one transient pass per seed; its whole take is inside the Leak column, and the two-year tape then hands it the 2026 crash. The positive Adv EV is the emission layer only; §6.15 flips it.

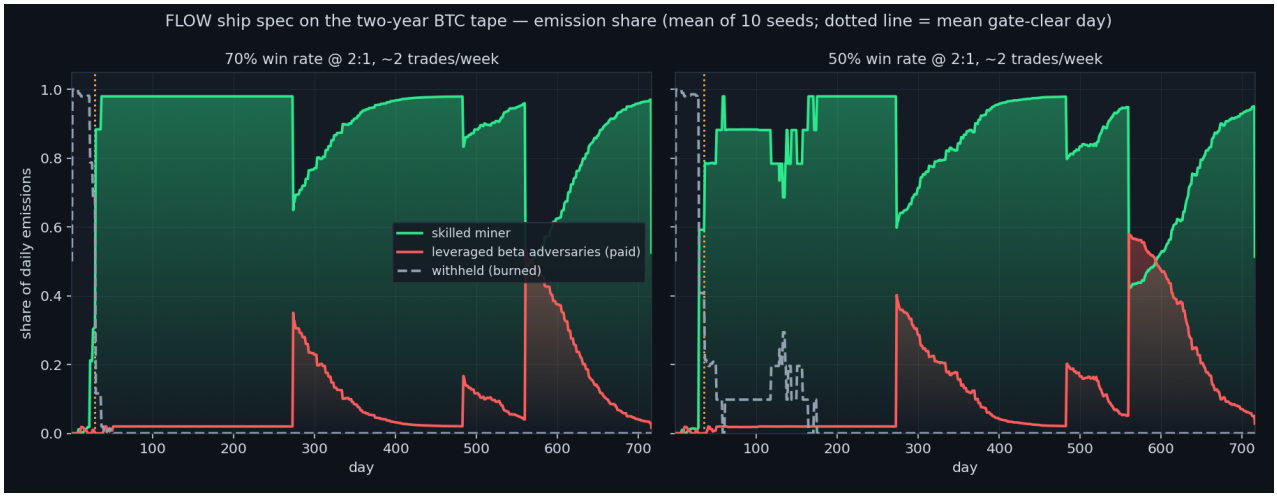
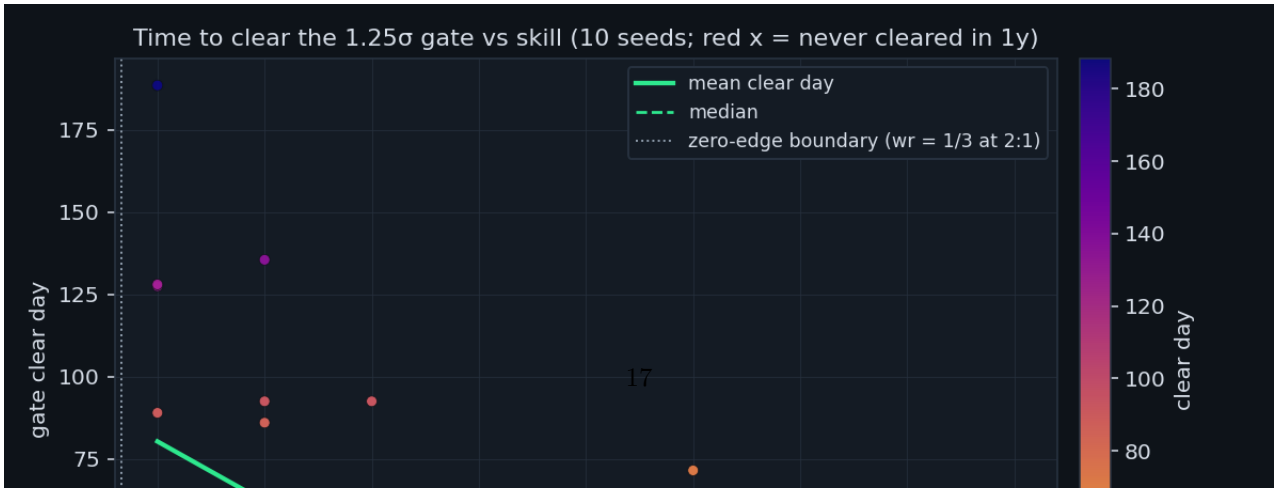


Figure 2: ship-spec ramp

6.2 Skill detection boundary (win-rate grid, 10 seeds)

w	E[R]/trade	Cleared	Clear mean/med/max	Steady	Leak %	Adv EV
0.35	+0.05	10/10	80.3 / 55.8 / 188.5	0.31	30.1%	+4.10
0.40	+0.20	10/10	62.1 / 50.5 / 135.5	0.58	17.9%	+2.36
0.45	+0.35	10/10	47.2 / 47 / 92.5	0.67	14.2%	+1.84
0.50	+0.50	10/10	35.5 / 29 / 58	0.73	12.1%	+1.52
0.55	+0.65	10/10	35.5 / 29 / 58	0.76	10.9%	+1.33
0.60	+0.80	10/10	35.8 / 29 / 71.5	0.77	10.1%	+1.22
0.65	+0.95	10/10	30.5 / 29 / 47	0.79	9.4%	+1.12
0.70	+1.10	10/10	28.0 / 29 / 36	0.80	9.1%	+1.07
0.75	+1.25	10/10	27.6 / 29 / 36	0.81	8.7%	+1.01

(E[R]/trade is the naive expectation $2w - (1 - w) = 3w - 1$ at the 2:1 bracket, before the TP1/TP2 split, path penalty, and partial fills.)



rolling window pays a real edge as soon as it shows a strong month. Leak grows as the whale weakens because with no dominant cleared key absorbing the pool, transient passers capture the full main share instead of a sliver; §6.15 prices those same worlds far more harshly for the adversary.

6.3 Counterfactual entry burn ($w = 0.50$, 5 seeds)

There is no entry bond at ship. This sweep prices what one would have bought if there were, by charging each entrant a sunk burn B at registration:

Burn (× daily pool)	0	0.25	0.5	0.75	1.0	1.5	2.0	3.0	5.0
Adv EV	+1.48	+1.23	+0.98	+0.73	+0.48	−0.02	−0.52	−1.52	−3.52

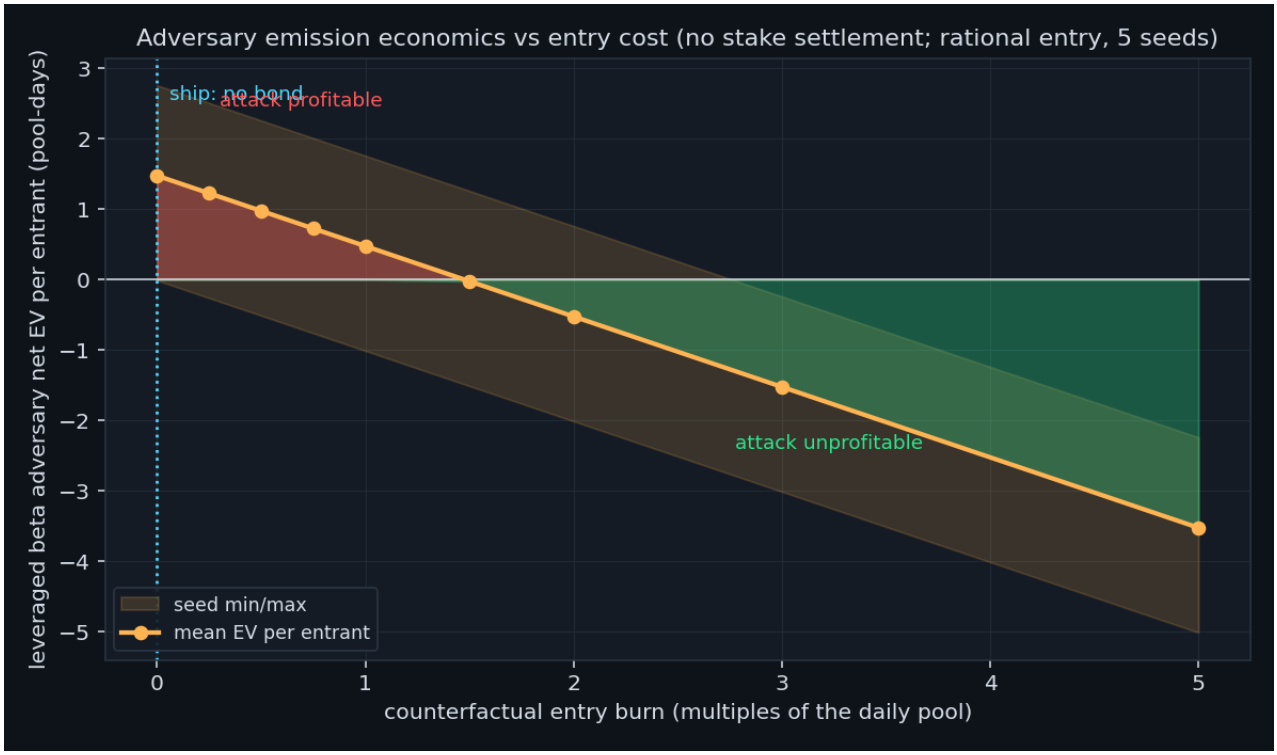


Figure 4: EV vs entry burn

The table and figure: per-entrant EV against a sunk registration burn. Exactly linear, $\mathrm{EV}(B) = 1.48 - B$; clear times and leakage are burn-invariant because the 1/day entry cap binds at every level, so an entry cost moves only where the participant's net lands. Why none ships: alpha cannot be burned by an EVM contract, so the real instrument is a refundable deposit, and an entrant that prices in the refund behaves like a zero-bond entrant. What a burn would buy, settlement already delivers: with collateral weighting live the aware book runs −1.7 to −2.9 TAO per key across the whale band, −6.5 to −7.4 for a naive whale book (§6.15), beside which every deposit size that could matter is a week-one charge on honest entrants. Sybil capital-per-key is carried by stake instead: k hotkeys split stake weight k ways at linear $\eta = 1$, gaining nothing.

6.4 Clearance threshold × exit latch ($w = 0.50$, 5 seeds)

z_{in}	z_{out}	Clear mean/med/max	Steady	Leak %	Adv EV	Passes
1.25	0.5	37.6 / 36 / 58	0.74	11.7%	+1.48	6
1.25	none	37.6 / 36 / 58	0.74	11.7%	+1.47	6
1.50	0.5	37.6 / 36 / 58	0.74	10.4%	+1.26	5
1.50	none	37.6 / 36 / 58	0.74	7.8%	+0.88	5
1.75	0.5	41.3 / 40 / 58	0.74	7.8%	+0.88	3
2.00	0.5	68.9 / 54 / 174.5	0.98	1.9%	−0.01	0
2.50	0.5	77.6 / 58 / 174.5	0.98	1.9%	−0.01	0

($z_{out} \in \{0.25, 1.0\}$ rows differ from the 0.5 rows by under 0.3% of the pool throughout; omitted.)

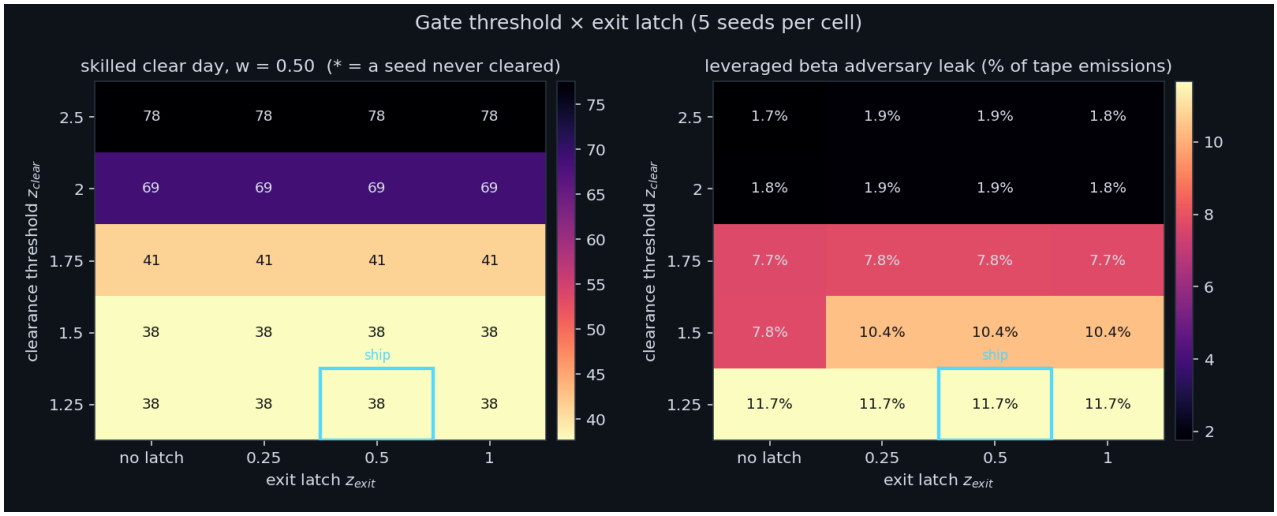


Figure 5: threshold × latch heatmaps

The heatmaps: clear day (left) and leak (right) over the threshold × latch grid; the ship cell is 1.25 / 0.5. At the emission layer the ship row costs about 1.3 leak points and one extra pass over 1.5; what it buys is at the bottom of the whale band, where 1.25 roughly halves the proving period ($w = 0.40$: 68 vs 143 mean days; $w = 0.35$: 76 vs 163, §6.15). With a strong collateralized key committed at launch, every uncleared week is a week the pool burns. At $z_{in} \geq 2.0$ the adversary never passes but moderate clearance roughly doubles. The latch is close to free at the ship threshold (11.7% leak with or without it: at 1.25 a passer that drops out re-clears through the rolling window anyway); its value is the cleared key's continuity, and the collateral layer (§6.15), where a latched noise key must keep collateral posted inside the drain to collect, is what keeps it cheap.

6.5 Block length (5 seeds)

The strip: §6.5–6.9 at a glance, one governed knob at a time at $w = 0.50$. Red = leakage (left axis), grey = mean clear day (right axis), ring = ship value. Leakage buys down with longer blocks and more required evidence at the price of proving time; entry pressure and dust raise leakage with proving time flat.

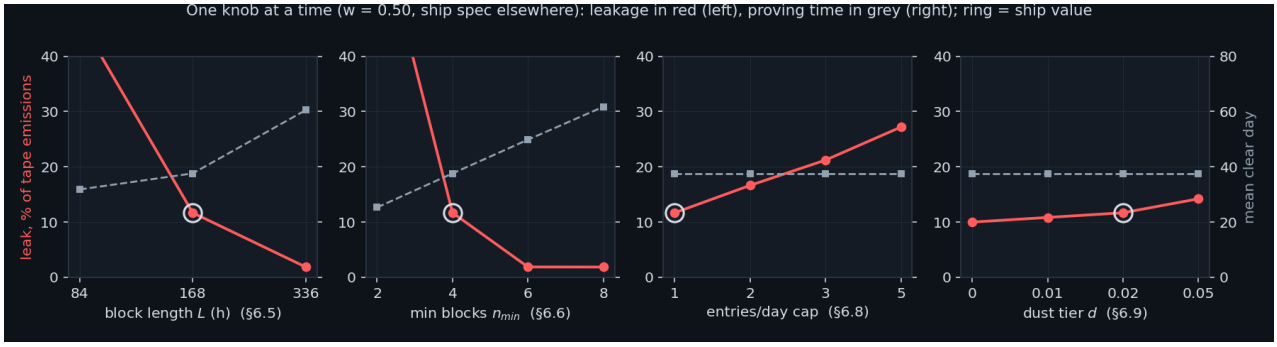


Figure 6: sensitivity strip

w	L	Clear mean/max	Steady	Leak %	Adv EV	Passes	Crowd
0.50	84h	31.8 / 50.5	0.43	46.7%	+1.63	212	5.6
0.50	168h	37.6 / 58	0.74	11.7%	+1.48	6	0.9
0.50	336h	60.5 / 96.5	0.98	1.9%	−0.01	0	0.9
0.70	84h	21.2 / 25.5	0.48	32.5%	+1.93	125	3.3
0.70	168h	26.9 / 29	0.79	8.9%	+1.04	6	0.9
0.70	336h	48.3 / 50.5	0.98	1.9%	−0.01	0	0.9

84h blocks re-admit the autocorrelation blocks exist to remove (passes ×35 under the rolling gate, crowd ×6); 336h blocks eliminate passes but grow the moderate clear by half. 168h fully de-overlaps the 72–168h horizons the tested adversaries live in; the 336h row prices the governed response if long-horizon farming shows up live.

6.6 Minimum evidence n_{min} ($w = 0.50$, 5 seeds)

n_{min}	Clear mean/med	Steady	Leak %	Passes	Crowd
2	25.3 / 25	0.19	65.2%	522	8.3
4	37.6 / 36	0.74	11.7%	6	0.9
6	49.8 / 43.5	0.98	1.9%	0	0.9
8	61.7 / 57.5	0.98	1.9%	0	0.9

$n_{min} = 2$ is structural failure, and the rolling window makes it lethal (a rolling two-block statistic clears on any lucky fortnight, forever: 522 passes, crowd 8.3, two-thirds of the pool leaks). n_{min} is both the evidence floor and the rolling window length; each increment past 4 buys leakage linearly in waiting time.

6.7 Gate statistic: return-t vs alpha-t (5 seeds)

w	Statistic	Clear mean/med/max	Leak %	Adv EV	Passes
0.50	ret	37.6 / 36 / 58	11.7%	+1.48	6
0.50	alpha	64.4 / 43.5 / 116	1.9%	−0.01	0
0.70	ret	26.9 / 29 / 29	8.9%	+1.04	6
0.70	alpha	44.8 / 43.5 / 50.5	1.9%	−0.01	0

The alpha statistic (OLS intercept t of block R on market and momentum benchmarks) removes exactly the factor exposure momentum monetizes: zero passes at every threshold tested. Its cost is degrees of freedom, 40–60% longer waits. Ship spec takes the return statistic and the transient passes; the alpha statistic is implemented as the governed switch if live leakage exceeds the simulated range.

6.8 Entry pressure ($w = 0.50$, 5 seeds)

Entries/day cap	Entries/yr	Leak %	Adv EV	Passes	Crowd
1	24	11.7%	+1.48	6	0.9
2	48	16.6%	+0.95	11	1.9
3	72	21.2%	+0.79	18	2.8
5	120	27.2%	+0.58	28	4.7

Leakage is sub-linear in entry pressure (5× pressure → 2.3× leak) and per-entrant EV *falls* past ~2 concurrent keys: entrants dilute each other in the dust tier and the pass window. With collateral weighting live, each extra key also carries stake into the settlement drain (§6.15); no crowd size makes entry collectively profitable.

6.9 Dust tier ($w = 0.50$, 5 seeds)

d	Steady	Leak %	Adv EV	Entries/yr
0.00	0.75	10.0%	+1.22	24
0.01	0.75	10.8%	+1.35	24
0.02	0.74	11.7%	+1.48	24
0.05	0.72	14.2%	+1.86	24

Leakage rises roughly linearly with d; at 0.05 the waiting room starts to pay. $d = 0.02$ is ≈ 0.48 units/day split across *all* uncleared keys, under the 0.5/day operating cost for any crowd > 1: the waiting room stays strictly loss-making, and the probation week keeps a fresh key's first days dust-free. (The pressure and dust panels of the §6.5 strip are this pair of tables drawn.)

6.10 Evidence rate: skilled trade frequency (5 seeds)

w	Trades/yr	Clear mean/med/max	Steady
0.50	50	62.8 / 57 / 99	0.74
0.50	99	37.6 / 36 / 58	0.74
0.50	204	32.6 / 31 / 39.3	0.71
0.50	497	32.2 / 26.5 / 55.8	0.73
0.70	50	38.8 / 36 / 43	0.80
0.70	97	26.9 / 29 / 29	0.79
0.70	200	27.9 / 28 / 28	0.80
0.70	485	23.4 / 22.8 / 25.8	0.80

Calendar blocks mean trading more adds no blocks, only lower within-block variance: frequency gains saturate fast for the strong profile (a weekly trader clears in 39 days vs 23 for an every-four-hours one) and bind hardest at the margin (the moderate weekly trader waits 63 mean days vs 38 at ship cadence). The mechanism prices evidence, not volume.

6.11 Adversary-only world (skilled never joins; 10 seeds)

Steady state, emission layer	Value
pool burned	21%
beta capture	~41% (292 pool-days over the tape)
per-entrant EV	**+5.71** pool-days

Steady state, emission layer	Value
same world, collateral weighting live (§6.15)	capture 0.5%, EV −0.41 TAO

The known conditional: an in-sample-optimal adversary, no competing skill, no collateral weighting. Three closures: the premise is counterfactual (a strong collateralized whale is committed at launch, and one cleared $w \geq 0.40$ key collapses capture to the §6.2 leak column); the share prices at most the 25% challenge weight; and with the settlement layer live, meaningful capture requires meaningful collateral, which a break-aware adversary will not hold inside the weekly drain. The pool burns instead of leaking.

6.12 Scoring-parameter invariance ($w = 0.50$, 3 seeds)

$\gamma \in \{0.15, 0.25, 0.40\} \times \tau \in \{0.3, 1.0\}$: clear day identical to the decimal (41.0 / 36 / 58) across all six cells, leakage within ±0.25% of pool, EV within ±0.05. The gate consumes S_{base}, so scoring parameters cannot move clearance (§1.7); their effects are confined to the split among paid keys. Same for α and κ within tested ranges.

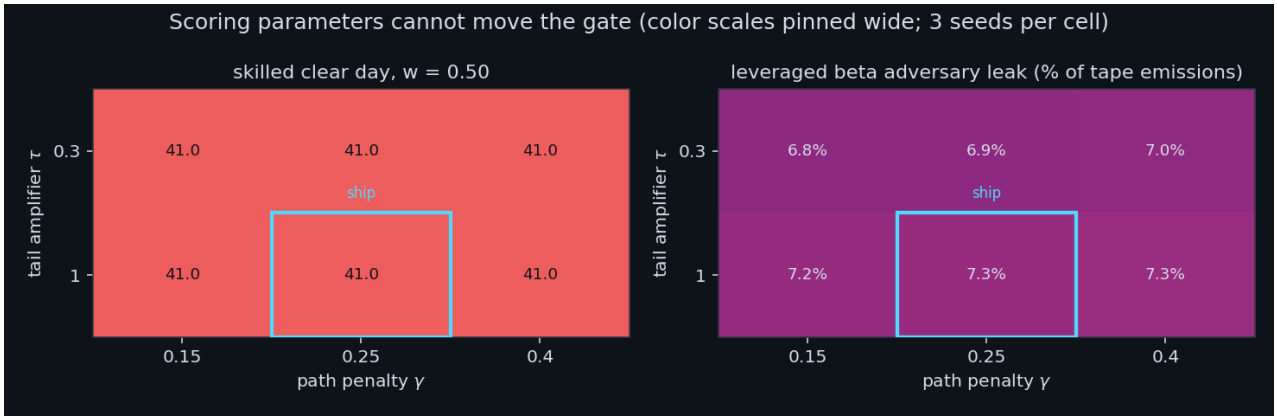


Figure 7: gamma × tau invariance

The heatmaps: clear day and leak over the $\gamma \times \tau$ grid; flat by construction.

6.13 No-gate baseline ($w \in \{0.5, 0.7\}$, 5 seeds)

Score-proportional payment, identical scoring stack, no gate: adversary capture 70–83% of the pool, skilled steady share 0.12–0.23, crowd ~7.5, momentum EV +1.9 to +2.1 (rents dissipated by entry, not returned to signal). This is the configuration the gate replaces; no parameter setting rescues it.

6.14 Production implementation cross-check (`prodsim.py`, 5 seeds)

`prodsim.py` closes the loop: the rational headline scenarios, re-encoded as dim-28 constant-emission payloads and replayed through the shipped validator module (`flow.py`) on hourly bars of the same tape.

	sim	flow.py + wicks	flow.py close-only
70% clear mean / worst (d)	26.9 / 29	25.0 / 34.1	25.4 / 36.0
50% clear mean / worst (d)	37.6 / 58	36.7 / 81.0	40.3 / 83.4
50% steady share	0.74	0.84	0.79
50% leak % of pool	11.7%	11.7%	7.7%
momentum keys ever passing (5 seeds)	6	11	8
momentum latched at tape end	n/a	11	8

The table and figure: simulation verdicts vs the shipped module, wick and close-only columns side by side. The shipped module lands within seed noise of the published clear times and leak, with a higher skilled steady

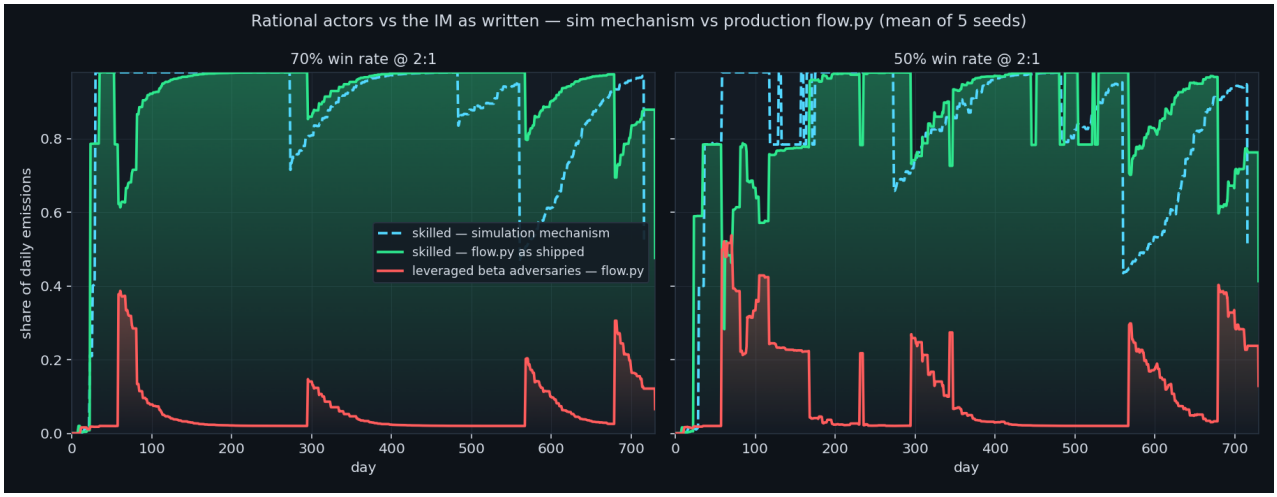


Figure 8: production cross-check ramp

share. Its constant-emission serialization holds one open trade per regime, which under the rolling gate makes transient passes a little easier to keep (11 latched vs the sim's 6); the take stays inside the leak column, which matches the sim's within a tenth of a point. Close-only, the no-kline fallback, is close but blunter (slower moderate clears, fewer passes); the wick feed remains the spec (§1.3) because close-only resolution cannot see intrabar stops and targets at all, and §2.4 wires it from challenge start.

6.15 Capital-at-risk economics (`stakeecon.py`, 5 seeds)

Everything above prices emissions only. This study adds the §1.6 layer as shipped: a real per-miner collateral ledger in TAO, weekly zero-sum settlement with losses capped at the position, emission weight $S_{emission}$ · collateral (linear, $\eta = 1$). The anchor, the launch fact stated plainly:

- **Skilled whale**: 33 TAO of subnet-123 alpha from day 0, run as a *band*, not a compounding book: refilled after settlement losses, gains above target swept out (one period later in the sim; the shipped contract releases free margin instantly). Per-trade $f = 0.03$: ordinary sizing (§4.5), an order of magnitude inside the 0.25 cap. A single week can never put a compounded balance on the line.
- **Break-aware adversary**: the same in-sample-tuned momentum book, sizing collateral over {1, 5, 15, 33} TAO by believed EV under a 4-weeks-per-year macro-shift hazard. Measured at the anchor: enters at 15 TAO, de-risks to the 1 TAO floor after its first losing fortnight, eventually quits; mean posted book at tape end 0.4 TAO per entered key, ~24 entries/yr, roughly one active at a time.
- **Naive whale contrast**: believes its in-sample edge, holds the full 33 TAO band for the whole tape. Posting more only deepens the drain; this column is that bound.
- Pool = 1.88 TAO/day. Whale flows are quoted per year (tape totals halved); adversary figures are per entered key, over that book's life.

The whale band, ship spec (collateral weighting on, no bond), with the 1.5σ gate as contrast:

w	Clear 1.25σ mean/max	Clear 1.5σ	Steady	Leak %	Whale settle TAO/yr	Adv settle TAO	Adv EV TAO
0.35	76 / 128	163 / 240	0.54	7.2%	+11	−0.5	+1.0
0.40	68 / 92	143 / 221	0.81	5.1%	+27	−1.1	−0.2
0.45	57 / 92	96 / 174	0.92	2.8%	+35	−1.5	−1.2
0.50	38 / 58	38 / 58	0.96	2.1%	+41	−1.7	−1.7
0.55	38 / 58	38 / 58	0.96	2.4%	+50	−2.1	−2.0
0.60	40 / 72	41 / 75	0.96	2.0%	+54	−2.2	−2.2

w	Clear 1.25σ mean/max	Clear 1.5σ	Steady	Leak %	Whale settle TAO/yr	Adv settle TAO	Adv EV TAO
0.70	27 / 29	—	0.98	1.3%	+65	−2.7	−2.9

Findings, in decision order:

The 1.25σ rolling gate is priced for the whale band. From $w = 0.35$ to 0.45 it roughly halves the proving period against the 1.5σ full-history contrast (76 vs 163, 68 vs 143, 57 vs 96 mean days), because every week below threshold is a week the pool burns instead of paying the only real book present. From $w = 0.50$ up the two configurations converge: a real edge produces a strong month early either way. Cost: §6.4's extra pass and about 1.3 points of emission-layer leak.

At ordinary sizing the circuit breaker is a backstop, not a governor. With $f = 0.03$ the whale's weekly flows sit naturally inside the cap: its worst realized settlement week is −4.5 TAO against the 8.25 ceiling (a quarter of the 33 TAO band), ruin weeks are zero everywhere, and refill cycling runs 12–33 TAO/yr. What the breaker still prices is behavior outside the band, a book that sizes up or compounds gains into exposure. And because the whale's own bad weeks are small, the drain now runs one way across the entire band: settlement income is positive from +11 TAO/yr at $w = 0.35$ to +65 at 0.70. The naive whale funds most of it (−6.5 EV per key against the strong whale, +162 TAO/yr donated to its book; +122/yr against the moderate one), and the concentration keeps its shape: in periods when many books are on the same strategy, momentum being the obvious one, their losing weeks arrive together and the capped pool flows to whoever is orthogonal to the crowd. The forced crowd-losing month over the 2026 crash moves the aware book to −3.1 and the naive one to −7.4. A break-aware book that de-risks to the 1 TAO floor forfeits little per period, which is exactly why its EV column, not the whale's income, is the deterrence number to read.

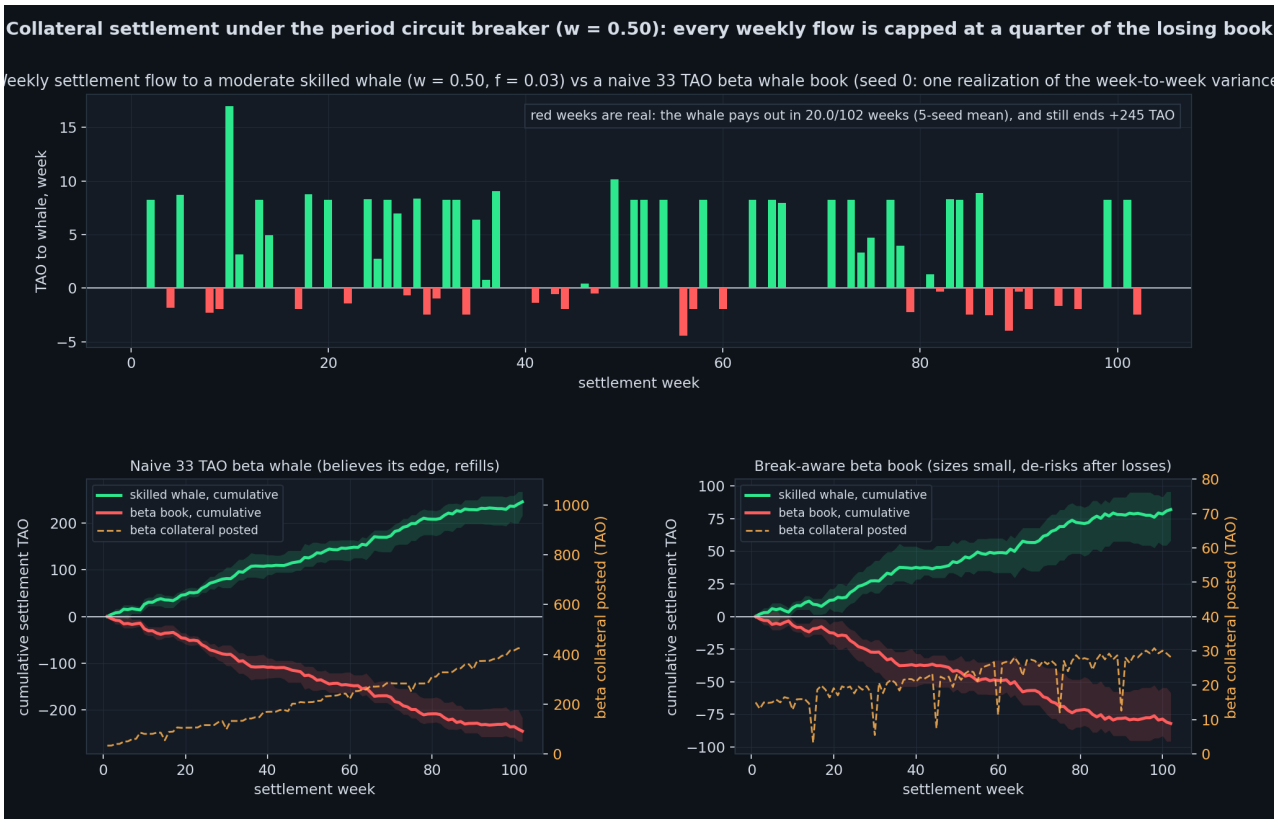


Figure 9: settlement extraction

The figure: a moderate whale ($w = 0.50$, $f = 0.03$), two full years, breaker on. Top: weekly settlement flow against the naive 33 TAO book, one seed. Red weeks are real: the whale pays out in 20 of 102 settlements, and the worst of them is −4.5 TAO, well inside the 8.25 breaker ceiling. Bottom: cumulative transfer, mean and 5-seed envelope: +122 TAO/yr against the naive book, +41/yr against the break-aware one, which sits at the 1 TAO floor and forfeits little. The amber trace is the adversary's posted collateral. The strong whale ($w = 0.70$) runs +162/+65 per year.

Weighting collateral during the significance window is load-bearing. At $w = 0.70$:

Collateral weighting	Leak %	Adv EV TAO
from day 0 (ship)	1.3%	−2.9
only after clearance	2.0%	+0.0
off (f-only)	8.9%	+2.0

Day-0 weighting makes uncleared capture expensive: dust and false-pass income split by collateral the noise does not want to post, and posting it exposes the book to the drain. This is why the §1.6 rollout activates collateral weighting no later than emission turn-on.

The bond adds nothing on top. A 1.5× counterfactual bond moves the aware adversary's EV by ~0.2 TAO per entrant (−3.1 vs −2.9) and moves clear times, steady shares, and leak by nothing; the empty-pool world with weighting live is capture 0.5% at −0.41 TAO EV (§6.11). Meaningful capture requires meaningful collateral, and meaningful collateral is drain exposure.

Miner-side sizing, same table ($w = 0.70$):

Skilled collateral policy	Settle TAO/yr	Note
3% bet, no refill	+65	zero ruin weeks; the book survives unrefilled
half sizing (f 0.015), band	+69	the drain is nearly sizing-invariant this deep inside the cap
3% bet, refill, no sweep	+53	gains compound into exposure and get given back in drawdowns
band: 3% bet, refill + sweep (ship anchor)	+65	15 TAO refilled, 143 swept to the wallet over two years, zero ruin weeks

At ordinary sizing the policy spread nearly closes: ruin is off the table in every row, so the sweep's job is banking gains, not survival (the no-sweep row still gives back 12 TAO/yr of drain income by letting exposure compound). Two honest edges. A below-band whale ($w = 0.35$) clears (5/5 at day 76 mean, the rolling window pays its hot streaks) and even collects +11 TAO/yr of settlement on ~33 TAO/yr of refill cycling, but it holds only 0.54 of the pool, and beside it sits the one adversary-positive row in the band: at 7.2% leak the beta book's EV is +1.0 per key. The floor of the mechanism is therefore $w \approx 0.40$, where adversary EV crosses zero (−0.2) and the whale's share reaches 0.81; from there up every number runs the drain's way.

---

7. Reproduction

```
cd MANTIS
python3 -m flow_sim.fetch_tape binance   # tape: repeat for bybit, okx,
python3 -m flow_sim.fetch_tape build     # coinbase, then build consensus
python3 -m flow_sim.finalspec     # sec 6.1 headline + ramp figure
python3 -m flow_sim.papergrid     # sec 6.2-6.13, 403 runs, 2 figures
python3 -m flow_sim.prodsim       # sec 6.14 production cross-check
```

```
python3 -m flow_sim.stakeecon      # sec 6.15 capital-at-risk study + settlement-flow figure
python3 -m flow_sim.relfigs        # heatmap figures from the CSVs
```

Outputs land in `flow_sim/out/`. `papergrid.py` contains the exact cell definitions for every table above; per-seed rows are in `out/papergrid.csv`, `out/finalspec.csv`, and `out/stakeecon.csv`.

Contract tests (Foundry):

```
cd flow_stake
forge test                              # 43 unit + fuzz tests
python3 test_stake_integration.py       # client + daemon lifecycle on anvil:
                                        # opens/closes/settles, standby takeover,
                                        # drift and no-lock paths
```

8. Files

File	Role
`flow_sim/flow_mechanism.py`	§1.1–1.5 + per-epoch collateral draft: validation, path resolution, scoring
`flow_sim/fetch_tape.py`	downloads the two-year, four-venue 1m tape and builds the consensus channels of §4.1
`flow_sim/rational.py`	§1.7 + §4.3–4.5: gate, latch, rational entry, adversaries
`flow_sim/params.py`	all governance parameters
`flow_sim/finalspec.py`	ship-spec validation suite
`flow_sim/papergrid.py`	the 12-study grid of §6
`flow_sim/prodsim.py`	§6.14: rational actors replayed through the shipped `flow.py`
`flow_sim/stakeecon.py`	§6.15: collateral ledger, weekly settlement, break-aware adversary sizing
`flow_sim/relfigs.py`	heatmap figures from the study CSVs, plus the §1.6 settlement-lifecycle diagram
`flow_sim/miners.py`	`OracleSkillMiner` + momentum Kelly sizing
`flow_sim/README.md`	simulation package overview
`flow.py`	validator implementation of §1.1–1.5 + 1.7 (scoring, gate, payment), plus §1.6 collateral weighting, the per-trade event feed, and settlement computation (off until the collateral contract is configured, see rollout note)
`flow_stake.py`	collateral contract client (pinned collateral snapshots, bet posting incl. the hotkey-signed digest, close/settle posting, miner tooling)
`flow_post.py`	miner-side bet module: post bets from a payload vector (hotkey-signed via the local wallet or depositor-keyed), read them back, and audit from event logs that no debit ever exceeded a bet
`flow_settlementd.py`	settlement daemon: ingests the miners' posted bets from chain, replays the panel, posts closes/settles, voids late-posted bets (§1.6, §3.3); run by the team on two boxes (primary + staleness-gated standby)
`ledger.py`	datalog: price rows + wick channels (§2.4), payload maturity, drand cache, FLOW storage split (§2.5), publish snapshot
`generate_and_encrypt.py`	v2 payload envelope (§2.2)

File	Role
`miner_dashboard/`	local manual-submission cockpit (chart, payload encryption, R2 upload, on-chain bet posting signed by the local hotkey wallet); reference tooling, not part of the validator
`flow_console/`	public read-only settlement console: books and withdrawability (balances, locked margin, pools, settlement history; no per-trade detail) over `eth_call`/`eth_getLogs`, five-minute refresh, plus the data-custody and operations reference (§1.6, §3.3)
`../flow_stake/`	`FlowStake.sol` + deployment (§3, separate Foundry package)
`test_flow.py`	smoke test: decode, drop-robustness, gate, dust cap
`test_flow_edges.py`	resolution precedence, bounds, garbage payloads, latch release, pre-clearance burn
`test_flow_e2e.py`	wiring test: salience dispatch, payload maturity, price rows, storage split, purge
`test_flow_stake.py`	collateral weighting, monetary R, the per-trade event feed, open-time pricing, loss pool, zero-sum rounding
`test_robustness.py`	crash/outage regressions: drand retention, snapshot integrity, corrupt-DB quarantine, atomic dashboard trade log
`../flow_stake/test/`	Foundry unit + fuzz suite for the collateral contract; `test_stake_integration.py` drives client + daemon on anvil

9. Limitations

- One asset, one two-year tape. The in-sample-tuned adversary is conservative in one direction; unrepresented regime mixes are not.
- The skilled reference is scripted, not a signal model: results measure mechanism ramp-up conditional on an edge existing, not the probability that one does.
- §6.11 is the binding caveat at the emission layer: with no cleared real edge and collateral weighting not yet active, a skill-free challenge leaks ~33% of the pool to an in-sample-optimal momentum strategy on a favorable year (while burning ~39%). The closure is §6.15's collateral layer, which makes its rollout deadline (no later than emission turn-on, §1.6) a real dependency, not a preference.
- Proving periods for moderate edges are months, and they are unpaid beyond the dust tier: the cost the mechanism deliberately imposes on unproven keys.
- §6.15's whale-band economics inherit the break-hazard belief assigned to the adversary (4 weeks/yr); a beta book that correctly called the tape's calm stretches would post more collateral and lose more slowly than simulated, though the naive-whale row bounds that case.
- Validator behavior and the fairness effects of τ among several simultaneously cleared keys are specified but not simulated.
- Gate threshold, block length, n_{min}, and dust fraction are governed parameters; §6.4–6.9 are the sensitivity range for reviewing them against live data, and an entry cost can be introduced along §6.3's priced curve if live economics demand one.
- §3.4 is a trusted-operator model with public after-the-fact verifiability, not a trustless one: the team feeds the contract (opens, closes, settles) and holds the owner keys. Every posted number is recomputable and the contract bounds the worst case, but liveness of those posts is a team responsibility, and the live feed publishes per-key exposure and realized losses ahead of public payload decryption.