Zylith Whitepaper

Protocol Specification v1.0-rc1

Tanya Arora
tanya@zylith.fi

May 2026

Abstract

Non-custodial trading on public blockchains exposes order intent before execution, trade history after settlement, and wallet behavior through the full onchain record. Zylith is a call-auction darkpool on Starknet that addresses these properties at the mechanism level rather than solely through cryptography.

The core observation is that a fixed-epoch call auction avoids a class of timing leaks that continuous matching systems cannot eliminate through proof privacy alone. In a continuous system, settlement events are reactive: they occur when matches are found, so their timing is a direct function of private order arrival. In a fixed-epoch auction, the settlement event is tied to the protocol clock, which removes the reactive link between private order submission and public settlement timing as a structural property of the mechanism.

Beyond timing, the structure of individual settlement transactions can reveal information even when proofs are zero-knowledge. In a continuous matching system each match produces a discrete onchain event with countable outputs and readable fee structure. Zylith's batch settlement carries Merkle root transitions and a padded output bundle reference rather than individual note ciphertexts, so transaction structure does not reveal fill count, participant count, or match shape.

A further problem persists: even when individual settlement proofs are zero-knowledge, the sequence of public events around many settlements can reveal information. Settlement cadence, claim timing, gas-payer patterns, client retrieval behavior, and scheduler rhythm form a parallel information channel. This paper calls that channel the market access-pattern problem, defines a taxonomy of its leak surfaces, and describes the defense stack Zylith implements to address them.

1. Introduction

Darkpool execution on a public blockchain requires solving three distinct problems. First, order contents must be hidden from other traders, block proposers, and the public record. Second, the settlement transcript must not reconstruct those contents through calldata shape. Third, the sequence of settlements must not leak through timing, shape, or behavioral patterns that survive individual-proof privacy.

A zero-knowledge proof hides order parameters inside a single statement. The surrounding sequence of transactions forms a second information channel whose timing, shape, and correlation structure must be controlled separately.

Zylith addresses these surfaces separately. The execution mechanism is a fixed-epoch uniform-price call auction. Settlement is a root-only ZK-STARK state transition verified by AuctionVerifier on Starknet. The access-pattern defense stack treats the public event sequence as a protocol surface and applies structured defenses to each identified leak.

The venue is designed for flow damaged by pre-trade exposure: large spot trades, treasury rebalances, market-maker inventory movement, private BTC and stablecoin flow, vault rebalances, accumulation and distribution programs, repeated execution schedules, and block-style all-or-none fills. On a public chain, every settlement artifact, gas payment, root transition, deposit, and withdrawal is permanently observable, making the design requirements more demanding than for a traditional private venue.

This paper defines the wallet and note model, the auction lifecycle, the Cairo settlement statements, the market access-pattern surface, and the trust boundary. Appendices state the public leakage boundary, defense matrix, proof binding summary, Cairo statement sketches, and bounded MAPP proof sketch.

2. Protocol State

2.1 The Private Wallet

Private balances, recognized notes, active orders, scheduler state, offline renewal packages, recovery records, and execution reports are maintained by an embedded wallet running locally. The coordinator is not the canonical owner of this state. It receives order commitments and routing metadata, maintains batch schedules, and serves public artifacts; it does not receive the recovery seed, note preimages, spend authority, withdrawal authority, or scheduler plans.

The wallet is derived from a 32-byte recovery seed, exportable as a 24-word BIP39 mnemonic. The seed deterministically derives a narrow key hierarchy under domain-separated labels:

sk_spend_auth *authorizes note consumption in orders*
sk_note_recog *recognizes encrypted output notes*
sk_cancel *authorizes order and renewal cancellation*
sk_withdraw_auth *authorizes public asset withdrawal*
sk_view *local viewing and audit functions*
sk_recovery *encrypts recovery artifacts*

The note recognition key is sufficient for passive bundle scanning and is not sufficient to spend or withdraw. Offline renewal authority is represented by exact-slot preauthorization material, not by exporting the spend or withdrawal key.

The wallet is not committed as a single global onchain object. AuctionVerifier tracks note, nullifier, renewal, fee, batch, and output roots. The wallet reconstructs private state by decrypting local vault data, scanning encrypted output bundles with ***sk_note_recog***, and tracking recognized spendable notes. Deposits and settlement outputs create note commitments. Spends register nullifiers. Renewals update the renewal root.

2.2 Notes and the Commitment Tree

A note is a shielded token claim with committed fields:

(***asset_id, amount, owner_public_key, spend_authority, withdraw_authority, blinding, nonce, metadata_commitment***)

The ***spend_authority*** and ***withdraw_authority*** fields are public authorities corresponding to ***sk_spend_auth*** and ***sk_withdraw_auth***. A valid spend proves authorization under the spend key without revealing the key. The ***metadata_commitment*** binds note-local recovery and settlement metadata, such as deposit or output context, without placing that metadata in public fields.

Its commitment is a Poseidon hash with an explicit domain separator. Spending a note derives a nullifier from the commitment and note-secret material, not from the public spend authority:

$$\begin{aligned}\textit{note\_commitment} = \textit{Poseidon}(&\textit{DOMAIN\_NOTE, asset\_id, amount,}\\ &\textit{owner\_public\_key, spend\_authority,}\\ &\textit{withdraw\_authority, blinding, nonce,}\\ &\textit{metadata\_commitment})\end{aligned}$$

$$\textit{nullifier} = \textit{Poseidon}(\textit{DOMAIN\_NULLIFIER, note\_commitment, blinding})$$

A valid spend proves note membership against the prior note root and sparse non-membership of the nullifier against the prior nullifier root. Settlement registers the nullifier, preventing double-spend.

Notes originate from deposits into the protocol, settlement outputs from matched orders, change notes returned on partial fills, and consolidation outputs. The note commitment tree is append-only. The wallet reconstructs balances by fetching encrypted output bundles from the indexer, attempting recognition with ***sk_note_recog***, and accumulating recognized spendable notes locally.

Withdrawal from a settlement output is a separate AuctionVerifier action. The withdrawer proves membership of an output note in the batch output note root, satisfies the configured claim delay, and supplies a domain-bound withdrawal authorization signed under ***sk_withdraw_auth***. The authorization binds the adapter, batch identifier, note commitment, asset, amount, and public recipient; asset release is routed through the shielded asset adapter. AuctionVerifier marks the output note commitment withdrawn before executing the release.

2.3 System State

Component	State
Embedded wallet	encrypted vault, recognized notes, active orders, scheduler state, offline renewal packages, execution reports, pending withdrawals
AuctionVerifier	note_root, nullifier_root, renewal_root, fee_root, batch registry, output bundle roots, withdrawn output notes, proof program config and lock state, gate and timing config, pause state
Coordinator	order commitments, receipts, batch manifests, public artifacts, artifact publication state
Prover ingress	ingress key material, encrypted payload processing, witness assembly, proof facts

Deposits advance the note root. Settlements consume current roots and write new roots atomically. Note consolidation consumes already-owned notes and advances note and nullifier roots without changing auction state. Withdrawals mark an output note commitment withdrawn and release assets through the adapter. Renewal cancellation and child use update the renewal root.

3. Auction Lifecycle

3.1 Order Submission

The core order is a private batch limit order. An order intent binds:

(***pair_id, batch_id, side, order_type, maker_curve_commitment?, limit_price, amount, min_fill, time_in_force, expiry_epoch, order_nonce, parent_order_commitment?, parent_child_index?, parent_secret_commitment?, parent_cancel_authority?, parent_authorization_secret?, funding_note_ref, funding_nullifier, recipient_owner_public_key, recipient_spend_authority, recipient_withdraw_authority, residual_withdraw_authority, auditor_view_allowed?***)

The order commitment is a Poseidon-domain-separated hash over these fields. For maker-curve orders, the curve points are committed through ***maker_curve_commitment***; for heartbeat-cover orders, the order commitment is still checked but funding-note authorization is not required. For multi-note funding, despite the singular field names, ***funding_note_ref*** is the aggregate input-set commitment and ***funding_nullifier*** is the aggregate nullifier commitment over the bounded input-note set; single-note orders keep the note commitment and nullifier directly. The public coordinator receives the commitment and normalized routing metadata. The prover ingress receives the encrypted payload containing the preimage; the plaintext preimage exists inside the prover ingress boundary at witness assembly time.

Epochs are fixed-duration and pair-specific. An order is accepted only within its target batch window and expires if not admitted. Fill-or-kill is a time-in-force constraint. Block-style all-or-none execution is encoded as $\textit{min\_fill} = \textit{amount}$ rather than as a distinct public order class, keeping the visible order surface minimal.

Submission smoothing delays the encrypted private payload by a bounded random interval inside the batch window, applied uniformly to limit orders, maker curve submissions, scheduler children, and safe cancel-replace paths. The delay is not applied when it would push the submission past the batch safety buffer.

3.2 Batch Clearing

At batch close, the prover ingress reconstructs the admitted order set and computes the clearing price. For admitted orders O and candidate price p:

$$eligible\_buy(p) = \{\ o \text{ in } O : side = buy \text{ and } limit\_price >= p\ \}$$
$$eligible\_sell(p) = \{\ o \text{ in } O : side = sell \text{ and } limit\_price <= p\ \}$$

The clearing price ***p_star*** maximizes matched base volume subject to order constraints, maker curve capacity, minimum fill, and fill-or-kill requirements, with deterministic imbalance and lower-price tie-breaks. Fills are allocated at ***p_star***. Residuals return as change notes.

Uniform pricing gives a specific information property: a filled buy reveals only that its private limit was at least ***p_star***, and a filled sell reveals only that its private limit was at most ***p_star***. The exact bound, original order size, and funding note ownership remain in the witness and local wallet records. The clearing price is a statistic over the full admitted order set, not a direct record of an individual order, provided the batch has sufficient independent participation. The privacy-quality gates in Section 3.3 encode that condition.

Fixed epochs make settlement timing a protocol clock rather than a function of private order arrival. Any reactive settlement system, meaning one that creates an onchain event when a match occurs, makes the timing of public events a function of private order arrival. The proof certifies correctness of the computation; it does not make the computation's occurrence unobservable. A fixed-epoch system decouples the occurrence of a public artifact from the occurrence of a match by producing artifacts on schedule regardless. Enabled pairs therefore produce heartbeat, no-op, and no-cross artifacts alongside matched settlements, so the existence of an artifact does not by itself identify private order flow.

3.3 Privacy-Quality Gates

A thin or dominated batch can let a participant infer too much from its own fill and the public clearing price. Gate conditions define when the auction is permitted to finalize private fills. The configured gate witness checks:

min_batch_base_liquidity
min_batch_participants
min_eligible_orders
max_single_order_fill_bps
max_single_owner_fill_bps
min_maker_participants
max_maker_fill_bps

If gate conditions are not met, private fills are not finalized. The epoch settles through a gate-failure no-fill, no-op, no-cross, or heartbeat-cover path. Scheduler and renewal logic may resubmit into a later epoch. Gate thresholds are deployment market parameters; changing them changes the allowed leakage function.

3.4 Hidden Maker Curves

A hidden maker curve is passive liquidity across private price-depth bands. A maker specifies (***price, depth***) points without publishing a public orderbook. The maker curve commitment is part of the order preimage. Settlement can consume eligible curve depth at ***p_star***, but the public record does not expose the curve shape, unfilled depth, or inventory bands; only the net root transition reflects the settlement.

Valid maker curves specify at least three strictly increasing price-depth bands, satisfy pair-specific minimum band depth and minimum outer-band price range, and bind the curve commitment into the order preimage. Maker fee eligibility requires renewal-backed child authorization. A one-shot maker curve order is valid only as taker-fee flow; it does not receive maker-fee treatment.

Maker commitments are fresh per epoch. Resting maker strategies keep the private curve in the embedded wallet and materialize each child with bounded price and depth rotation before funding selection and private ingress submission. Bounded renewal windows, randomized slicing, maker exposure limits, and dominance caps reduce repeated-shape fingerprints; persistent economic effects can still be inferred statistically over long histories.

Offline delegated renewal uses exact-slot preauthorization. A renewal operator receives authority for specific authorized slots and does not receive the wallet spend key, withdrawal key, or reusable future authority. Parent cancellation updates the renewal root so future children cannot be accepted after cancellation. Child replay is rejected by renewal nullifier state.

3.5 Scheduled Execution

TWAP, VWAP, repeat-private, randomized slicing, and encrypted resting renewal are wallet-private schedulers. They do not define public order types. Each scheduler holds a parent strategy locally and materializes a fresh child order for each eligible epoch. The child binds:

(***pair_id, epoch_id, batch_id, funding_note_ref,***
recipient_authorities, parent_order_commitment,
parent_child_index, parent_secret_commitment,
parent_cancel_authority, parent_authorization_secret)

The parent strategy, remaining schedule, total parent size, and slicing seed are not public fields. Child indices are bounded by the preauthorization. Fresh child commitments remove direct commitment linkage across epochs. They do not make economically regular behavior statistically undetectable.

4. Settlement

4.1 Cairo Statement Structure

Zylith uses scoped ZK-STARK statements proved through the Starknet prover endpoint. The admission and auction-result statements are proved and submitted sequentially. The settlement transition is decomposed into three statement programs: settlement root/value constraints, nullifier sparse-root updates, and renewal sparse-root updates. Together they bind to the same transcript commitment.

The admission statement proves that each admitted order commitment correctly binds its preimage, that each non-cover order has valid funding-note-set authorization, that non-cover funding nullifiers are correctly derived and aggregated, and that the admission summary root is correctly computed. Heartbeat-cover orders are protocol-generated order commitments and are admitted without funding-note binding. The statement's public outputs are ***batch_id***, ***order_commitment_root***, and ***admission_root***.

The auction-result statement takes ***admission_root*** as input and proves allocation feasibility at the committed clearing price, consistency of hidden maker curve commitments with the allocation, satisfaction of the matched-volume clearing rule, passage of all configured privacy gates, or validity of a no-cross, gate-failure no-fill, empty no-op, or heartbeat-cover path. Its public outputs are ***batch_id***, ***order_commitment_root***, ***admission_root***, and ***transcript_commitment***.

The settlement statement takes ***transcript_commitment*** and proves pair and epoch binding, consumed note membership against the prior note root, order and output-note formation, output bundle binding, fee computation, value conservation, and the public settlement commitment verified by AuctionVerifier. The nullifier statement proves 128-depth sparse non-membership and insertion against the nullifier root. The renewal statement proves 128-depth parent-cancel absence and child insertion against the renewal root.

Note consolidation is a separate maintenance statement. It consumes a set of already-owned notes of the same asset, proves membership, authorization, nullifier insertion, output-note formation, value conservation, and output-bundle binding, and updates the note and nullifier roots without changing auction state.

AuctionVerifier enforces the relationship between these statements before state changes. Settlement is accepted only when the recorded ***transcript_commitment*** for the batch, the ***batch_id***, the ***order_commitment_root***, and the current roots match the settlement calldata; the settlement proof facts carry the settlement, nullifier, and renewal messages bound to the same transcript commitment; and the recorded transcript commitment is itself accepted only after it has been bound to the recorded ***admission_root***.

The admission statement proves order preimage binding, funding-note-set binding, funding authorization, and nullifier derivation in Cairo. The auction-result statement proves allocation feasibility, privacy gates, and clearing-price optimality through ***assert_best_clearing_price***.

4.2 Root Chaining and Aggregate Settlement

AuctionVerifier maintains root families for notes, nullifiers, renewals, and fees. Each settlement atomically consumes current roots and writes new roots. The batch registry records prepared and settled batches to prevent double settlement.

Native aggregate settlement chains ordered transitions. If batch ***i*** writes roots ***R_i***, batch ***i + 1*** must prove against ***R_i***. An aggregate proof over ***n*** batches proves the member transitions in one virtual Cairo execution:

R0 -> R1 -> ... -> Rn

The resulting proof-facts object contains the settlement, nullifier, and renewal messages for each ordered member, and AuctionVerifier applies them sequentially against its current roots. Two batches prepared independently from the same prior roots cannot be aggregated without rebuilding one witness against the other's transition.

4.3 AuctionVerifier

AuctionVerifier verifies proof facts against the configured virtual program hash, checks proof freshness against the validity block window, resolves pair and epoch from the batch registry rather than from caller-provided calldata, records admission roots, records auction-result transcript commitments only when they bind to the recorded admission root, enforces that the recorded ***transcript_commitment*** for the batch matches the settlement calldata before state mutation, validates root continuity, records root transitions atomically, stores output bundle roots, and prevents double settlement.

The settlement calldata is root-only:

batch_id
order_commitment_root
encrypted_orderset_commitment
transcript_commitment
proof_artifact_commitment
clearing_price
price_base_scale
taker_fee_bps
maker_fee_bps
relay_fee_bps
protocol_fee_recipient
relay_fee_recipient
output_bundle_ref
prior_note_root
prior_nullifier_root
prior_renewal_root
prior_fee_root
consumed_note_root
consumed_nullifier_root
renewal_child_root
output_note_root
fee_root
new_note_root
new_nullifier_root
new_renewal_root
new_fee_root
fee_asset_ids
fee_recipients
fee_amounts

AuctionVerifier does not store individual fills, maker curve points, plaintext orders, or per-order settlement details. Fee rows are public root-bound aggregate entries by asset and recipient, not per-order records. The public record after settlement is a verified root transition and an output bundle root.

The fee policy is pair-class based. Speculative pairs use a 4 bps taker fee, 0 bps maker fee, and 2 bps relay fee for makers using the Zylith renewal relay. Conversion pairs use a 2 bps taker fee, 0 bps maker fee, and 1 bps relay fee for makers using the Zylith renewal relay. Makers using self-relay pay no relay fee. Protocol fees accrue to the configured protocol fee recipient through FeeLedger and are released by the configured fee-claim authority. Relay fees accrue to the configured relay fee recipient.

5. Market Access-Pattern Privacy

5.1 The MAPP Problem

A ZK-STARK proof hides the witness for one statement. It does not hide the public sequence of events in which that statement appears. Zylith settlements occur in a public sequence of Starknet transactions with timing, calldata shape, output bucket class, gas payer accounts, deposit events, and withdrawal events. These observations survive even when every

individual proof is zero-knowledge, because they are properties of the transaction sequence, not of the proof contents.

Let $\boldsymbol{L}$ be the allowed leakage function over private order-flow histories. For histories $\boldsymbol{H0}$ and $\boldsymbol{H1}$ where $\boldsymbol{L}(\boldsymbol{H0}) = \boldsymbol{L}(\boldsymbol{H1})$, the public Starknet transcripts produced by $\boldsymbol{H0}$ and $\boldsymbol{H1}$ are computationally indistinguishable to an observer with full chain and indexer access, except through facts in $\boldsymbol{L}$ or through facts accessible only within the prover ingress trust boundary stated in Section 6.

Economically constrained repeated behavior remains the hard case. A TWAP strategy, persistent maker, or recurring accumulation program cannot behave like uniform noise without harming execution. Zylith constrains the protocol-level correlations that make this behavior easy to observe: fixed epochs decouple submission timing from settlement timing, bucketed transcripts prevent exact fill-count inference, delayed artifacts remove real-time fill signals from public routes, and fresh child commitments remove direct commitment-level linkage across epochs.

5.2 Leak Surface Taxonomy

Market access-pattern leak surfaces fall into three categories.

Onchain-observable leakage is visible to any block explorer without privileged access. It includes pair activity timing, settlement artifact cadence, output bundle size class, root transition timing, deposit and withdrawal transactions, gas payer accounts, and any public field that labels an epoch as empty, crossed, or no-op. Heartbeat artifacts normalize empty-epoch labeling for enabled pairs; markets without heartbeat coverage expose that surface directly.

Coordinator and prover-observable leakage requires access to Zylith infrastructure or the prover ingress. It includes order commitment submission timing within a batch window, encrypted payload size patterns, cancellation and replacement frequency, pair monitoring patterns, and plaintext order preimages inside the prover ingress boundary during witness assembly. The public coordinator and prover ingress are separate components with separate access to information: the coordinator handles commitments and manifests; the prover ingress handles encrypted payloads and witness assembly.

Inference leakage is computed from repeated observation rather than read from a single public field. Thin-batch inference occurs when a fill combined with the public clearing price is sufficient for a participant to estimate the counterparty order. Dominance inference occurs when a single order, owner, or maker controls enough of a batch that the clearing result becomes informative. Strategy-rhythm inference occurs when scheduler children appear in a consistent temporal pattern. Maker-curve fingerprinting occurs when repeated curve behavior produces a recognizable effect on clearing outcomes across epochs.

Each category requires different controls. A ZK proof does not address gas payer correlation; encryption does not address settlement cadence; a fixed epoch does not make a public withdrawal recipient private. Appendix B maps each surface to its control.

5.3 Protocol Defenses

Fixed epochs and pair heartbeats. Enabled pairs run on scheduled epochs. Heartbeat and no-op/no-cross artifacts preserve cadence when private order flow is absent. Sentinel values that would label a heartbeat epoch as empty are avoided on the heartbeat path. Batch-close jitter and settlement-submission jitter operate below the epoch clock, preventing close time and proof-submission time from becoming exact signals for order arrival or prover completion.

Root-only settlement transcript. Settlement calldata exposes roots, commitments, and bucketed transcript shape without exposing consumed note arrays, individual output notes, fee rows, renewal records, or per-order fills. Public transcript routes serve only this root-only view after the configured delay; the witness transcript is retained only in restricted infrastructure routes and local prover artifacts.

Output bucketing and dummy slots. Output bundles are padded to configured size buckets. Dummy encrypted slots have the same public structure as real encrypted output records. Exact fill count is hidden within the bucket; the bucket class itself remains public.

Delayed artifact publication. Settlement artifacts and output bundle references are delayed before appearing on public routes. The delay is epoch-progress based and depends on heartbeat publication for quiet markets. This does not substitute for private retrieval, but it prevents the public indexer from functioning as a real-time fill notification channel.

Multi-pair artifact bundles. Public artifact bundles aggregate commitments across pairs and epoch ranges. Pair heartbeats give every enabled pair a regular clock; multi-pair bundles reduce pair-level activity disclosure within the bundle.

Privacy-quality gates. The gate conditions in Section 3.3 prevent thin or dominated batches from settling as private fills. Below the thresholds, the batch takes a gate-failure no-fill, no-op, no-cross, or heartbeat-cover path.

Submission smoothing. Bounded random delay on private payload submission within the batch window prevents the coordinator and prover ingress from observing exact submission timing correlated with a trader's local decision to act.

Fresh child commitments. Scheduler and renewal children use fresh commitments per epoch. Consecutive child commitments are not directly linkable through commitment reuse; statistical linkage from regular behavior remains an inference surface rather than a commitment-reuse surface.

Client query cadence. The app polls public batch state on a fixed cadence and scans output artifacts over epoch ranges rather than single fill-driven requests. This reduces query-pattern leakage to the indexer without claiming private retrieval for the default user-facing path.

Cancellation and replacement. Cancellation is accepted only while the target batch is still open and replacement orders use fresh commitments. Cancellation endpoints remain metadata-bearing coordinator operations; they are rate-limited and commitment-bound rather than made anonymous.

Claim windows, withdrawal buckets, and paymaster relay. A configured claim delay separates batch settlement from note redemption. Withdrawal amounts are compared against denomination buckets. Gas for public exits routes through a

paymaster, decoupling the gas-paying account from the trading wallet. The public withdrawal recipient remains visible at exit; these defenses reduce timing and gas correlation rather than making the withdrawal unlinkable.

Local execution analytics. Post-trade reports and TCA are computed inside the wallet from decrypted output records. Private order parameters, fill details, balances, and scheduler events are not transmitted to third-party analytics services by the protocol.

6. Trust Boundary

Private order payloads are encrypted to the prover ingress boundary. The public coordinator receives commitments and routing metadata and does not receive order preimages. At batch close, the prover ingress decrypts admitted payloads, assembles witnesses, computes auction outputs, requests native proofs from the configured Starknet prover endpoint, and returns proof facts to AuctionVerifier.

The prover ingress determines witness construction and liveness for the batches it processes; AuctionVerifier determines what the chain accepts. With split-proof enforcement active, the correctness of order preimages, spend authorizations, funding-note-set bindings, nullifier derivation, allocation feasibility, privacy gates, and clearing-price optimality is proved inside Cairo and bound onchain before settlement. The residual ingress boundary is privacy, availability, and inclusion: the ingress sees plaintext order preimages during witness assembly and decides which admitted payloads enter the witness.

AuctionVerifier accepts a state transition only when proof facts match the configured proof program hash, satisfy the proof freshness window, bind to the batch registry, and match the verifier's current roots. The proof program can be locked after deployment. With split-proof enforcement active, the recorded admission root and auction-result transcript commitment must also match the settlement transcript before any state mutation is accepted. The operational controls on the ingress are key pinning, receipt signing, private payload no-logging with short retention windows, request size caps, rate limits, and monitoring.

Administrative authority is separated from fee custody in the protocol surface. AuctionVerifier supports emergency pause, a pause guardian, two-step admin transfer, proof-program locking, bounded claim-delay configuration, pair-fee configuration, protocol-fee-recipient configuration, and relay-fee-recipient configuration. FeeLedger supports a separate fee-claim authority. Pair fee changes are timelocked for 24 hours after the initial configuration. Protocol fee recipient, relay fee recipient, and fee-claim authority changes are timelocked for seven days. The verifier exposes no general upgrade entrypoint; after the proof program is locked, changing the proof program requires redeployment rather than an admin call.

Transport privacy is split by path. OHTTP is enabled by default on the prover-runtime-to-native-prover endpoint path and can be explicitly disabled by configuration. Tor, mixnets, and PIR for trader-to-coordinator and trader-to-indexer paths are not default protocol dependencies; they can be added at the client or infrastructure layer without changing the settlement or proof statements.

7. Conclusion

Zylith is a call-auction darkpool on Starknet built around three interlocking properties. Execution is a fixed-epoch uniform-price auction whose settlement timing follows the protocol clock rather than private order arrival. Settlement is a root-only ZK-STARK state transition that carries Merkle root transitions and padded bundle references rather than individual fills, fees, or note ciphertexts. Access-pattern controls treat the public event sequence as a protocol surface, addressing settlement cadence, transcript shape, claim timing, gas-payer correlation, and scheduler rhythm through protocol-level controls rather than relying solely on external transport.

The public chain sees enabled markets, scheduled epochs, proof facts, root transitions, bucketed output bundle references, delayed artifact availability, deposits, note consolidations, withdrawals, and gas paid by protocol or paymaster accounts. It does not see plaintext order parameters, note ownership, maker curve shape, or scheduler parent strategy, except through the bounded leakage function in Appendix A and the prover ingress boundary in Section 6.

Appendix A: Public Leakage Boundary

After the defense stack in Section 5.3, the allowed public leakage function L contains the enabled pairs and epoch schedules; fixed epoch duration and deployment configuration; proof program hashes and verifier configuration; batch identifiers and root transition sequence; bucketed output bundle size class per epoch, not the exact fill count within the bucket; delayed artifact availability after the configured delay; deposit transactions and public fields from the selected funding rail; note consolidation proof facts, root transitions, and output bundle references; withdrawal transactions, public recipients, and public adapter fields; coarse timing of paymaster-relayed withdrawals; and gas paid by protocol, coordinator, or paymaster accounts.

The public transcript is designed not to reveal plaintext order side, size, or limit price; funding note ownership or private balances; output note ownership before withdrawal; exact private fill count inside a padded output bucket; hidden maker curve shape, inventory bands, or unfilled depth; scheduler parent strategy, remaining schedule, or total parent size; the association between a child commitment and its parent; or the notes, amounts, and owner involved in a consolidation operation.

For private order-flow histories $H0$ and $H1$ where $L(H0) = L(H1)$, the public Starknet transcripts produced by $H0$ and $H1$ are computationally indistinguishable to an observer with full chain and public indexer access, except through facts in L or through facts accessible only within the prover ingress trust boundary.

Deposit entry, user-facing transport, and private retrieval are protocol-edge surfaces; Appendix B keeps them explicit.

Appendix B: Defense Matrix

Surface	Defense
Settlement timing	fixed epochs and pair heartbeats; batch-close and settlement-submission jitter
Pair activity presence	all-enabled-pair heartbeats; delayed multi-pair artifact bundles; all-enabled-market sync
Empty epoch identification	heartbeat, no-op, and no-cross artifacts with cover orders and nonzero cover prices; normalized metadata
Output note count	padded output bundles to configured size buckets; dummy encrypted slots with identical public structure
Recipient inference	recipient-private note encryption; nonzero empty output-note root binding
Settlement transcript structure	root-only calldata; delayed public transcript routes; full transcripts behind bearer-auth internal routes
Artifact publication timing	delayed public artifact availability after configured epoch-progress threshold; heartbeat-dependent for quiet markets
Gas payer identity	protocol-side settlement account for batch settlement; paymaster relay for withdrawals
Deposit entry correlation	Starknet Privacy funding rail; deposit-note activation; amount bucketing as client policy
Claim and exit timing	claim delay window; withdrawal window policy
Withdrawal amount	denomination bucket policy
Note consolidation timing	root-only maintenance statement; padded output bundle; delayed artifacts; wallet policy can batch consolidation away from sensitive trading windows
Order submission timing	submission smoothing within batch window; batch safety buffer
Encrypted payload metadata	private payload size caps; padded ingress responses; coarse status errors
Client query patterns	fixed polling cadence; all-enabled-pair checks; epoch-range artifact scanning
Thin-batch inference	participant, liquidity, eligible-order, and dominance gates; gate-failure no-fill artifacts
Clearing-price sequence	proof-bound clearing rule via ***assert_best_clearing_price*** in the auction-result statement; privacy gates; delayed and bucketed public metrics
Long-lived strategy rhythm	fresh child commitments; randomized slicing; exact-slot renewal preauthorization; parent cancellation registry; child replay protection
Maker curve fingerprinting	fresh per-epoch curve commitments; bounded renewal windows; per-epoch rotation; maker dominance caps; exposure controls
Cancellation timing	open-batch-only cancellation windows; fresh replacement commitments
Order preimage at witness assembly	encrypted ingress; key pinning; receipt signing; no private payload logging; short retention; request caps
Client private state	encrypted local vault; local note scanner; encrypted recovery artifacts
Recovery and state sync	seed-bound recovery auth; encrypted recovery artifacts; epoch-range fetches with count buckets
Execution analytics	execution analytics computed locally from decrypted wallet records; no fill data transmitted externally
Public metrics	delayed and bucketed metrics; no real-time private-volume data
Prover transport metadata	default OHTTP from Zylith prover runtime to native Starknet prover endpoint
Output bundle retrieval*	PIR or ORAM-style private retrieval for output bundle scanning
User-facing transport*	Tor or OHTTP on trader-to-coordinator and trader-to-indexer paths

* Outside the default deployment scope.

Appendix C: Proof Binding Summary

AuctionVerifier accepts settlement only after the proof-facts checks bind the admission statement, auction-result statement, and settlement transition to the same batch transcript. The settlement submission carries settlement, nullifier, and renewal statement messages bound to the same transcript commitment. The authorized settlement account is the protocol-controlled account configured in AuctionVerifier.

In each call, proof-facts validity means the transaction carries non-empty ***proof_facts***, the proof variant and output format are the configured virtual-SNOS format, the ***virtual_program_hash*** equals the configured proof program hash, the base block is inside ***proof_validity_blocks***, and ***message_to_l1_hashes*** equals the expected domain-separated message, or message sequence, for the statement being accepted.

C.1 Admission Root

Interface. ***record_admission_root_with_proof_facts***(***batch_id, order_commitment_root, admission_root***), with all fields encoded as felt252.

Requires. Authorized settlement caller, nonzero inputs, batch-registry binding between ***batch_id*** and ***order_commitment_root***, and an admission proof-facts message over the configured proof program, verifier, batch id, order root, and admission root.

Effect. Store ***admission_root*** for the batch.

C.2 Auction Result

Interface. ***record_auction_result_with_proof_facts***(***batch_id, order_commitment_root, admission_root, transcript_commitment***).

Requires. Authorized settlement caller, nonzero inputs, batch-registry binding to the same order root, a recorded admission root equal to ***admission_root***, and an auction-result proof-facts message over the proof program, verifier, batch id, order root, admission root, and transcript commitment.

Effect. Store ***transcript_commitment*** as the verified auction transcript for the batch.

C.3 Settlement

Interface. ***submit_settlement_with_proof_facts***(...), carrying the batch id, order root, encrypted order-set commitment, transcript commitment, proof artifact commitment, clearing price, price-base scale, taker, maker, and relay fee bps, protocol and relay fee recipients, output bundle reference, prior roots, consumed roots, renewal child root, output note root, fee root, new roots, fee asset ids, fee recipients, and fee amounts. Fields are felt252 except ***clearing_price***, ***price_base_scale***, ***taker_fee_bps***, ***maker_fee_bps***, ***relay_fee_bps***, and ***fee_amounts***, which are u128.
Requires. Authorized settlement caller; proof artifact commitment binding the verifier and transcript commitment; settlement proof message over that artifact; unsettled prepared batch; current verifier roots matching supplied prior roots; registry binding for the order and encrypted order-set roots; configured pair fee bps and fee recipients matching contract storage; fee entries root recomputed from fee asset ids, recipients, and amounts equal to ***fee_root***; public settlement commitment recomputed from the batch id, registry pair and epoch, order roots, clearing price, price scale, fee bps, fee recipients, output bundle reference, and root transitions equal to the transcript commitment; nullifier proof message binding the transcript commitment, ***prior_nullifier_root***, ***consumed_nullifier_root***, and ***new_nullifier_root***; renewal proof message binding the transcript commitment, ***prior_renewal_root***, ***renewal_child_root***, and ***new_renewal_root***; and a verified auction transcript equal to the settlement transcript.
Effect. Atomically update note, nullifier, renewal, and fee roots; store the output note root; record settlement metadata; and mark the batch settled.

C.4 Note Consolidation

Interface. ***submit_note_consolidation_with_proof_facts***(...), carrying the consolidation id, proof artifact commitment, output bundle reference, prior note and nullifier roots, consumed note and nullifier roots, output note root, and new note and nullifier roots.
Requires. Authorized settlement caller; unpaused verifier; current verifier note and nullifier roots matching the supplied prior roots; nonzero consumed and output roots; unique consolidation id; proof facts over the note-consolidation commitment; and root recomputation matching the supplied new roots.
Effect. Atomically update note and nullifier roots, store the consolidation output note root, record settlement metadata, and record a consolidation note-root transition without changing auction, renewal, or fee state.

C.5 Binding Property

A settlement cannot finalize unless the admitted order set, auction result, and root transition all refer to the same batch and transcript. This binding enforces the critical relationship across the auction pipeline while keeping each native prover request within the intended statement size.

Appendix D: Cairo Statement Sketches

The following sketches name the verifier-bound public fields, private witness material, and checked relations for the native Cairo statements. The Cairo source is normative for the full constraint definitions.

D.1 Admission Statement

The public interface is ***batch_id***, ***order_commitment_root***, and ***admission_root***. The witness contains the settlement payload, admitted order preimages, funding note preimages for the bounded input set and spend-authorization material, plus recipient, renewal, and maker-curve material where applicable. The admitted order summary is the vector of order commitment, side, order type, maker-curve commitment, limit price, amount, minimum fill, time-in-force, funding-note aggregate amount, and funding-note owner key. The statement proves that ***batch_id*** and ***order_commitment_root*** are read from ***settlement_payload***; order vectors are length-consistent and order commitments are unique; the ordered commitment root of the admitted commitments equals ***order_commitment_root***; each order commitment equals the hash of its preimage; each non-cover order has a valid funding authorization under the committed spend authority; each input note commitment, the aggregate note reference, and the aggregate nullifier are mutually consistent, with each nullifier derived from the note commitment and note-secret material; input-asset compatibility follows order side; time-in-force, fill-or-kill, expiry, recipient authority, parent renewal, and maker-curve fields are well formed, including minimum band count, pair-specific minimum price range, per-band minimum depth, and parent renewal fields where relay or maker-fee eligibility is claimed; and ***admission_root*** is the Poseidon root of the admitted order summaries.

D.2 Auction-Result Statement

The public interface is ***batch_id***, ***order_commitment_root***, ***admission_root***, and ***transcript_commitment***. The witness contains the settlement payload, admitted order summaries, allocation vector, matched-order bindings, maker-curve summaries, and privacy-gate parameters. The statement proves that ***transcript_commitment*** is the transcript hash of ***settlement_payload***; ***order_commitment_root*** matches the ordered root of admitted commitments; ***admission_root*** matches the admitted order summary root; maker curve commitments are consistent with the curve summaries used in allocation; private fills, when present, match the committed clearing price, balance buy and sell base volume, satisfy minimum-fill and fill-or-kill constraints, and maximize matched base volume subject to order constraints under deterministic tie-break rules; no-fill paths have zero allocations and either no executable auction, a valid privacy-gate failure, or a heartbeat-cover path; and configured privacy gates succeed for private fills.

D.3 Settlement Statement

The public interface is ***transcript_commitment***; batch, order-root, encrypted order-set, clearing-price, price-scale, taker, maker, and relay fee bps, protocol and relay fee recipients, and output-bundle fields; prior note, nullifier, renewal, and fee roots; consumed note and nullifier roots; ***renewal_child_root***; ***output_note_root***; ***fee_root***; and the four new roots. The witness contains matched order and funding-note-set material, consumed nullifier commitments, note membership proofs,

output and residual note material, output recovery records and dummy commitments, and fee computation material. The statement proves that settlement header fields are nonzero and domain-separated; matched order commitments, consumed notes, consumed nullifiers, renewal children, and output note commitments are unique; matched order preimages, funding note input preimages, funding authorizations under committed spend authorities, aggregate and per-input funding nullifiers, maker curve capacity, and parent renewal links are valid; each consumed note is a deposit note or prior settlement output included in ***prior_note_root***; output and residual notes are correctly formed; output recovery records and dummy commitments bind to ***output_bundle_ref***, while ciphertext-to-output authentication is checked during witness construction before proof request; protocol and relay fees, buy/sell conservation, minimum-fill, fill-or-kill, and maker-curve capacity constraints hold; consumed-note, consumed-nullifier, renewal-child, output-note, fee, new-note, and new-fee roots recompute from the witness; and ***public_settlement_commitment*** over the calldata fields equals ***transcript_commitment***. The settlement statement binds ***new_nullifier_root*** and ***new_renewal_root*** into the public commitment; their sparse updates are proved by the separate nullifier and renewal statements.

D.4 Nullifier Statement

The public interface is ***transcript_commitment***, ***prior_nullifier_root***, ***consumed_nullifier_root***, and ***new_nullifier_root***. The witness contains consumed nullifiers and their 128-depth sparse non-membership and insertion paths. The statement proves that consumed nullifiers are unique; ***consumed_nullifier_root*** is the Poseidon root of the consumed nullifier vector; each consumed nullifier is absent from ***prior_nullifier_root*** at its 128-bit sparse key; and ***new_nullifier_root*** is computed by inserting the consumed nullifiers in order. If there are no consumed nullifiers, ***new_nullifier_root*** must equal ***prior_nullifier_root***.

D.5 Renewal Statement

The public interface is ***transcript_commitment***, ***prior_renewal_root***, ***renewal_child_root***, and ***new_renewal_root***. The witness contains matched parent renewal links, child nullifiers, parent-cancel markers, and 128-depth sparse absence and insertion paths. The statement proves that each renewal child is derived from the matched parent order, child index, and authorization secret; the parent cancellation marker is absent from the running renewal root; the child nullifier is then inserted; ***renewal_child_root*** is the Poseidon root of the renewal child vector; and ***new_renewal_root*** is the resulting renewal root. If there are no renewal children, ***new_renewal_root*** must equal ***prior_renewal_root***.

D.6 Note Consolidation Statement

The public interface is ***consolidation_id***, ***consolidation_commitment***, ***output_bundle_ref***, prior note and nullifier roots, consumed note and nullifier roots, ***output_note_root***, ***new_note_root***, and ***new_nullifier_root***. The witness contains input note preimages, spend authorization under the common spend authority, note membership proofs, input nullifiers and 128-depth sparse non-membership and insertion paths, output note preimages, and output recovery records. The statement proves that all input notes are already owned notes of the same asset and spend authority; input note commitments and nullifiers are correctly derived; input notes are included in the prior note root; input nullifiers are absent from the prior nullifier root and inserted into ***new_nullifier_root***; output notes are correctly formed; output recovery records and dummy commitments bind to ***output_bundle_ref***; input value equals output value; consumed-note, consumed-nullifier, output-note, new-note, and new-nullifier roots recompute from the witness; and ***public_note_consolidation_commitment*** over the calldata fields equals ***consolidation_commitment***. The statement does not alter auction, renewal, or fee state.

Appendix E: Bounded MAPP Proof Sketch

This appendix states a bounded indistinguishability claim for the Zylith public protocol transcript.

E.1 Assumptions

ZK. The Cairo ZK-STARK proof system is zero-knowledge for the admission, auction-result, settlement, nullifier, renewal, and note-consolidation statements defined in Appendix D.
ENC. The note encryption scheme is IND-CPA secure under the note recognition key.
PRF. The nullifier derivation function ***H_nullifier*** is pseudorandom.
COM. The note commitment scheme is computationally binding and hiding.
HASH. Poseidon over the Starknet field is collision-resistant.

E.2 Claim

Let L be the allowed leakage function defined in Appendix A. Let ***H0*** and ***H1*** be two private order-flow histories where $L(H0) = L(H1)$. Let $T(Hi)$ denote the public Starknet transcript produced by executing the Zylith protocol on history ***Hi***, including settlement proof facts, note-consolidation proof facts, root transitions, output bundle references, deposit events, and withdrawal events.
Under ZK, ENC, PRF, COM, and HASH, $T(H0)$ and $T(H1)$ are computationally indistinguishable to a probabilistic polynomial-time adversary with full access to the public Starknet chain and public indexer, except through facts in L or through facts accessible only within the prover ingress trust boundary defined in Section 6.

E.3 Proof Sketch

Settlement transcripts. By ZK, each settlement proof fact reveals nothing about its witness beyond what is implied by the public inputs: batch identifier, root transitions, bucketed output bundle class, and clearing price. Since $L(H0) = L(H1)$, those public inputs are identically distributed, so the proof facts are computationally indistinguishable under the ZK assumption.
Note consolidation transcripts. By ZK, each consolidation proof fact reveals nothing about its witness beyond the public consolidation commitment, root transitions, and bucketed output bundle reference included in L. By COM and PRF, the consumed notes, output notes, and nullifiers remain hidden except through the public root transition.
Output notes. By ENC, encrypted output note ciphertexts in the output bundle are indistinguishable without the note recognition key. By COM, note commitments in the root reveal nothing about note contents. Output bundles are padded to standard size buckets, so the bucket class is identically distributed for ***H0*** and ***H1*** by the condition $L(H0) = L(H1)$.
Nullifiers. By PRF, registered nullifiers are pseudorandom and reveal nothing about the consumed notes or their owners.

Scheduler children. By COM and PRF, fresh child commitments per epoch are computationally unlinkable across epochs. The parent strategy, remaining schedule, and total size are not recoverable from the child commitment sequence.

E.4 Scope and Limitations

This claim covers the protocol transcript: the sequence of onchain transactions, proof facts, root transitions, and encrypted artifacts. It excludes network-layer metadata, long-run behavioral inference from economically constrained strategies, facts inside the prover ingress boundary, and deposit, consolidation, and withdrawal events included in $\boldsymbol{L}$. It makes no claim of universal composability, security under adaptive corruption of the prover ingress, or indistinguishability of user behavior patterns under long-run observation.