Overview

Astria is building a decentralized shared sequencer network that creates a market for transaction ordering on Celestia-based rollups. Decentralizing a shared sequencer means that authority over ordering is distributed across a diverse set of node operators, providing better guarantees for users with regards to value extraction.

The shared sequencer creates a single “meta block” consisting of transactions belonging to one or more rollups. It is a lazy sequencer construction, meaning it treats the data it orders as simply bytes, only ordering and not executing any rollup transctions.

We propose a method of integrating SUAVE bundle construction into our existing shared sequencer transaction flow, enabling rollup developers to harness the benefits of SUAVE and MEV-Share to reduce value leakage occuring on their networks.

At a high level, the construction can be described in three steps:

1. MEV-leaking transactions can be submitted through SUAVE to produce searcher bundles
2. Searcher bundles are then submitted to the shared sequencer, integrating MEV Share with Astria’s blockspace market
3. Rollup nodes deterministically derive state transitions from sequenced transactions, placing SUAVE-generated bundles at the top of the block as ordered by the shared sequencer.

Goals

- Implement a proof-of-concept for a rollup that enshrines SUAVE in its block building process.
- Demonstrate using SUAVE to construct bundles for Astria-based rollups. This enables users and protocols to inherit the benefits of MEV-Share, curbing the value leakage that is currently captured entirely by centralized sequencer operators.

Non-goals

- Implement a production-ready SUAVE integration for Astria-based rollups.
- Demonstrate searching for MEV on Astria-based rollups using SUAVE.
- Implement a sufficiently secure or decentralized SUAPP.

Background: Astria

[

astria

620×920 24.4 KB

](https://collective.flashbots.net/uploads/default/original/2X/5/5a597c32c86d526f488d603ddddd42b12368e0db.png)

Write Path: Transaction Submission

1. Users sign a transaction via a wallet, submitting it to the rollup node (e.g. Astria’s modified Geth), which is then gossiped to the rollup’s mempool.
2. The Composer

is a developer tool we have created to act as a gas station. It retrieves pending transactions from the rollup node’s mempool, then wrapping them in an Astria transaction and submitting them to Astria’s mempool.

1. Astria’s shared sequencer constructs a cross-rollup “meta-block”, finalizing an ordering over the data posted to multiple rollups’ namespaces at once.
2. The finalized shared sequencer blocks are posted to Celestia, allowing rollups to leverage strong guarantees over data availability for cheaper.

Read Path: Block Retrieval and Execution

1. Each rollup runs its own instance of the Conductor

. The Conductor

reads and validates rollup data from finalized Astria blocks.

1. Filtered blocks are passed to the rollup node (e.g. Geth) using Astria's Execution API, and are then executed by it, resulting in an updated rollup state and state root. This is how a rollup derives its state from the sequenced data posted to and made available by the data availability layer.

Conductor

[

conductor

620×840 19.6 KB

](https://collective.flashbots.net/uploads/default/original/2X/8/8479d79ec9327b2d6a388901e58aed743f71488c.png)

The [Astria Conductor

](https://hackmd.io/@astriaorg/HJ6cCpp9T#22-The-Astria-Conductor) is a sidecar process that a rollup node operator runs alongside the rollup node. It acts as a light client of the shared sequencer and data availability networks, pulling in Astria and Celestia blocks, validating them and feeding them into the rollup node using the Execution API.

- Data flow and verification spec
- Sequencer inclusion proofs spec

The Astria Execution API

Astria integrates block production infrastructure into rollup architectures by running a Conductor

instance as a sidecar to the rollup node. The sidecar reads and validates finalized Astria blocks for the rollup's data and communicates with the rollup-specific software using Astria's Execution API.

While an in-depth explanation of the API and the specific request semantics can be found in thespecification document and the protobuf definitions, the main request of interest in this situation is ExecuteBlock

.

[

executeblock

320×580 13.2 KB

](https://collective.flashbots.net/uploads/default/original/2X/8/8f82bea776fd97daffacb8a8c449e9b8c12f5183.png)

SUAVE-Enabled Astria Rollup

When ingesting SUAVE produced blocks, the write/read path of a single Astria rollup is modified to look like the following:

Write Path

[

write

1083×764 56.1 KB

](https://collective.flashbots.net/uploads/default/original/2X/4/4a335aa9ec14107c14a5329358d28d08912b8da2.png)

1. User connects to the rollup node (e.g. Geth) and signs a transaction and submits transactions via a wallet, submitting it to the rollup node (e.g. Geth), gossiping the transaction to which land in the rollup's mempool.
2. Some of the rollup's nodes run SUAVE wallets which pull all the transactions that reach thei rollup's mempool and posts them to a SUAPP as a SUAVE transaction with the user's rollup transaction as a confidential input. Nodes that

do not operate a SUAVE wallet forward the user transactions to the Composer

as described above.

- Alternatively, the user can maintain both rollup and SUAVE wallets and sign and submit directly to SUAVE instead of passing through a rollup node. However, our proof of concept implementation does this from the rollup node to simplify things.

- Alternatively, the user can maintain both rollup and SUAVE wallets and sign and submit directly to SUAVE instead of passing through a rollup node. However, our proof of concept implementation does this from the rollup node to simplify things.

- The Kettle validates the SUAVE transactions and stores the rollup transaction it received as confidential inputs in its local confidential store. The public part of the SUAVE transaction is gossiped to other Kettles.
- A SUAVE block is finalized with the SUAVE transaction, and the Kettle executes it against the SUAPP. The SUAPP accesses the rollup transaction from the confidential inputs and passes it to a precompile that sends a bundle back to a Composer

instance.

1. POST requests with bundles are received in the Composer

, which then serializes bundles and puts them into an Astria transaction to be submitted to the Astria sequencer.

1. The ordered sequencer blocks are posted to the data availability layer, Celestia, similar to the previous path.

Read Path

1. The Conductor receives individual blocks and filters these to get only the data for its particular rollup's namespace.
2. Filtered blocks are passed to the rollup node via Astria's Execution API.
3. The rollup deserializes the transactions it receives into either SUAVE bundles or regular transactions.
4. The rollup node verifies the Kettle signatures accompanying all the SUAVE bundles.
5. The rollup then executes transactions from SUAVE bundles first and regular transactions afterwards, resulting in the updated rollup state.

Use Case & Examples

SUAPP-generated Top of Block

As described in the implementation above, this construction allows rollup developers to incorporate a SUAVE-driven Top of Block Auction into their protocol design.

This is implemented by adding additional middleware to the Execution API server in the rollup node (Geth) that does the following:

1. Filter out the SUAVE bundles from the rest of the transactions
2. Validate Kettle operator signatures for each bundle
3. Unbundle the transactions from inside the SUAVE bundles and prepend them in order to the non-SUAVE transactions

Afterwards, we end up with a block ordering that can be succinctly described as (Bundle_0, ..., Bundle_k, Transaction_0, ... Transaction_n)

, where each bundle is then unbundled into the constituent transactions.

Importantly, this ordering is deterministcally derived from the order finalized by the sequencer.

OVAL-style DeFi mechanisms

Uma's OVAL allows protocols to manage the value leaked by oracle-related actions.

Specifically, liquidations in lending markets typically follow an oracle price update which adjusts the LTV ratios for all users, leading to newly available liquidations. Currently, this happens entirely out of protocol, leaking value from lenders to

searchers and builders.

Oval attaches repayment instructions to the searcher’s bundles that will require any excess value to be repaid back to the protocol. Integrating SUAVE into the shared sequencer allows rollup developers to restrict certain types of transaction flow only to SUAVE bundles. This allows for the design of DeFi mechanisms that internalize this value extraction, reducing the overall value leakage and making DeFi markets more efficient.