Getting Started

To start searching on MEV-Share, you will first need to connect to a MEV-Share Node. As a reminder, the MEV-Share Node is responsible for receiving transactions and bundles from users, and selectively sharing information ("hints") about them with searchers. When a searcher wants to include a transaction in their bundle, they use that transaction's hash. The MEV-Share Node replaces it with the original transaction before sending the bundle to a block builder.

Connect to MEV-Share Node

Flashbots operates an MEV-Share Node on the Ethereum mainnet. This Node provides endpoints for user transactions and searcher bundles. Additionally, it offers an Server-Sent Events (SSE) streaming endpoint, broadcasting pending events/transactions to searchers.

The simplest way to connect to the Flashbots MEV-Share Node is to use a client library . For this guide, we'll refer to mev-share-client-ts .

- Typescript
- Rust

Add library to your project:

yarn add @flashbots/mev-share-client Use the following code to import the library (Replace ALL_CAPS placeholders with your data):

import

{ Wallet , JsonRpcProvider }

from

'ethers' ; import MevShareClient ,

{ BundleParams , IPendingBundle , IPendingTransaction , TransactionOptions , }

from

'@flashbots/mev-share-client' ;

const provider =

new

JsonRpcProvider (RPC_URL) ; const authSigner =

new

Wallet (FB_REPUTATION_PRIVATE_KEY , provider) ; const mevShareClient = MevShareClient . useEthereumMainnet (authSigner) ; Connecting to Goerli:

const mevShareClient = MevShareClient . useEthereumGoerli (authSigner) ; To use custom network parameters, you can instantiate a new MevShareClient instance directly. This example is what the client uses to connect to mainnet:

// connect to MEV-Share on mainnet const mevShareClient =

new

MevShareClient (authSigner ,

{ name :

'mainnet' , chainId :

1 , streamUrl :

'https://mev-share.flashbots.net' , apiUrl :

'https://relay.flashbots.net' , }) ; Further documentation on the client library can be found in the mev-share-client-ts . For Rust users, thanks to Paradigm's effort, we have a Rust MEV-Share client mev-share-rs .

If you want a complete MEV bot framework, could also directly go for Artemis , which has built-in support for MEV-Share as outlined in this example .

A note on other languages

If you're coding in a language that doesn't yet have a MEV-Share Node client library, you can send transactions and bundles directly with theJSON-RPC endpoint . To listen for transactions, all you need is an HTTP client. More details on that in theEvent Stream page.

Now you should be connected to the Flashbots MEV-Share Node. Continue reading on the next page to learn how tolisten for hints about pending transactions shared by the MEV-Share Node.

A note for experienced searchers getting acquainted with MEV-Share Searching on MEV-Share is different from searching on the mempool in that only certain parts of a transaction are shared with searchers. In the mempool, we can see all parts of a transaction, such as its calldata or who the transaction is from. But on MEV-Share, a transaction might only reveal its function selector, making a traditional arbitrage calculation infeasible.

There are three primary strategies for searching on MEV-Share:

- Probabilistically: send many bundles that probabilistically backrun MEV-Share orderflow.
- On-chain: perform more of your searching on-chain instead of off-chain.
- Existing: only search on transactions which share all the information you need.

To maximally leverage MEV-Share searchers will need to employ new strategies.Edit this page Last updatedonJan 30, 2024 Previous For Orderflow Providers Next Event Stream