slug: searching-on-mev-share title: Searching On MEV-Share authors: [brock, shea] tags: [mev-share] image: /img/simple-blind-arbitrage-screenshot.png hide_table_of_contents: false description: MEV-Share is a new paradigm for searchers. It unlocks new orderflow, features, and searching strategies. This short guide will get you up to speed on new ways of searching and explain how to upgrade your strategies to leverage MEV-Share. forum_link: https://collective.flashbots.net/t/its-time-to-search-on-mev-share/1722

MEV-Share is a new paradigm for searchers. It unlocks new orderflow, features, and searching strategies. This short guide will get you up to speed on new ways of searching and explain how to upgrade your strategies to leverage MEV-Share.

TL;DR

- *Searching on MEV-Share is a brief introductory guide to get searchers up to speed with MEV-Share. It explains the new features of MEV-Share and gives instructions and examples of searching strategies.*
- *Searchers can access new orderflow from the Flashbots MEV-Share Matchmaker at https://mev-share.flashbots.net (open the link in your browser to see active transactions)! This includes all transactions that are sent to Flashbots Protect, which is approximately 3 percent of all Ethereum transactions.*
- *To search on MEV-Share, you can use Flashbots' example arbitrage bot or create your own strategies using a client library like matchmaker-ts.*
- *Share your ideas and learn about future updates by joining the #mev-share channel in the Flashbots discord!*

Last month Flashbots announced the beta release of MEV-Share. The rollout was a first step towards introducing programmable privacy to the MEV supply chain — enabling users to decide how, when, and by who their transactions are executed. Today, searchers can subscribe to the Flashbots Matchmaker to search on programmably private orderflow. This includes all transactions sent to Flashbots Protect. But what does searching on programmably private orderflow really mean for searchers? And how can searchers bootstrap strategies in this new paradigm?

Searching on MEV-Share is a brief introductory guide to get searchers up to speed with MEV-Share. **We'll explain the new ideas behind MEV-Share, how they differ from MEV-Boost and MEV-Geth, and how to think about designing searching strategies. We'll also touch on some new features of MEV-Share and give examples of how to use them.**

What is MEV-Share?

MEV-Share is a protocol that allows users to selectively share information about their transactions with searchers. This enables users to keep sensitive information private while still allowing searchers to execute strategies such as blind backruns, Uniswap v2 and v3 arbitrage, and more. Flashbots runs a service called the Flashbots Matchmaker that implements the MEV-Share protocol. All transactions sent to Flashbots Protect are automatically routed through the Flashbots Matchmaker.

Today, **around 3 percent of all Ethereum transactions flow through Flashbots Protect**, *giving searchers the ability to search on increasingly valuable source of private orderflow.** Furthermore, MEV-Share is a proto-SUAVE implementation which will give searchers early to MEV-Share a better understanding of how MEV market dynamics will evolve as the MEV supply chain decentralizes.

> **On April 30th, 32,011 / 974,972 (3.28%) of transactions that were included on Ethereum Mainnet originated from Flashbots Protect (source: Flashbots Protect and https://etherscan.io/chart/tx)*

How to use MEV-Share

Ethereum users **connect to the Flashbots Matchmaker** via **Flashbots Protect.** - Users can specify what information to share about their transactions. By default, only the transaction hash and supported pool/pair addresses are shared. Currently, the Flashbots Matchmaker supports sharing the pool/pair addresses on **Uniswap v2/v3, Balancer, and Curve**. All other data (calldata, logs, etc.) are hidden from searchers unless the user specifies otherwise.

Searchers **listen for information about pending transactions** via the Matchmaker's SSE endpoint. Searchers use this information to craft partial bundles, which they send via the `mev_sendBundle` API to `https://relay.flashbots.net` where it is routed to the Flashbots Matchmaker. - Searchers include partial transactions in their bundles because they only have access to the transaction *hash*, not the full *signature*. Searchers simply place the transaction hash in their bundle, rather than a signed transaction.

The Flashbots Matchmaker then **replaces the transaction hashes** in searchers' partial bundles with the original signed transactions, and sends the bundles (now fully comprised of signed transactions) to block builders.

How is MEV-Share different from MEV-Boost or MEV-Geth?

The most obvious difference is that pending transactions are not shown in the clear. In other words, searchers may not have access to all the data they're used to seeing from pending transactions, like calldata and logs.

You can access orderflow from the Flashbots Matchmaker right now. Just go to https://mev-share.flashbots.net (open the link in your browser, or make a GET request with curl).

> For example, a pending Uniswap trade will (by default) only expose the transaction hash and the token pair being traded. The amount being traded, and the direction of the trade (ETH → TKN or TKN → ETH, for example) is hidden.

New MEV-Share Features

MEV-Share introduces several exciting features that give searchers more control over how their bundles are handled.

The bulk of the excitement is in the mev_sendBundle endpoint. We'll describe its features in the following subsections. See the mev-share protocol repository for the canonical definition of the endpoint, and check out the Flashbots MEV-Share docs for implementation-specific details from Flashbots.

For reference, this is the current mev_sendBundle definition. Note that fields with a "?" are optional.

```
json { jsonrpc: "2.0", id: string | number, method: "mev_sendBundle", params: [{ /* MevSendBundleParams */ version: "0" | "beta-1.0", inclusion: { block: string, // hex-encoded number maxBlock?: string, // hex-encoded number }, body: Array< { hash: string } | { tx: string, canRevert: boolean } | { bundle: MevSendBundleParams } >, validity: { refund: Array<{ bodyIdx: number, percent: number, }>, refundConfig: Array<{ address: string, percent: number, }>, }, privacy?: { hints?: Array< "calldata" | "contract_address" | "logs" | "function_selector" | "hash" >, builders?: Array<string>, }, }] }
```

Pre-Inclusion Predicates

Pre-inclusion predicates (specified in the inclusion parameter of mev_sendBundle) define requirements for a bundle to be placed in a block. They are evaluated *before* the bundle is considered for placement in a block. These parameters are specifically designed to be low-compute, allowing builders to efficiently filter for eligible bundles.

Searchers can specify block and maxBlock to indicate the block(s) in which their bundle can land. maxBlock is a convenient way to specify a *range* of blocks with only one RPC request, as opposed to sending one for each block, which is the case with MEV-Boost.

This example specifies that the bundle should target the next 5 blocks, starting at block number 0x105f2**7a** and finishing at block number 0x105f27**e**:

```
json { jsonrpc: "2.0", id: 1, method: "mev_sendBundle", params: [{ version: "beta-1", inclusion: { block: "0x105f27a", maxBlock?: "0x105f27e", }, body: [...], ... }] }
```

Post-Inclusion Predicates

Post-inclusion predicates (specified in the validity parameter of mev_sendBundle) define requirements for a bundle to be included in a block on-chain. They are evaluated *after* a bundle is placed in a prospective block by a block builder. These parameters define restrictions that can only be evaluated in the context of a block, where the transactions of a bundle have been assigned a specific block position, resulting in an EVM state unique to that bundle.

Post-inclusion predicates give searchers (and matchmakers) hard guarantees about the conditions under which their bundles are allowed to be included in a block — such as the minimum MEV **refund** a bundle's transactions should earn, and where the refund is sent. We expect more parameters to be added in the future, to further enhance the granularity of these preferences.

Bundle Composition & Searcher Kickbacks

One of the most exciting features in MEV-Share is bundle composition, which allows searchers to share data about their own bundles and earn more MEV. **Bundles that share data can be included in other bundles**, enabling searchers to collaborate and build more complex strategies. For example, you could backrun another searcher's backrun to clean up an inefficient trade, or take advantage of the new price incurred by the first backrun to execute another strategy. In this sense, bundle composition enables searchers to participate in collaborative block-building.

To make a bundle composable, just specify hints in the privacy parameter of mev_sendBundle. Doing so will share data about your bundle's transactions with other searchers, allowing them to include your bundle in theirs.

Builder Selection

Users and searchers can also specify which builders have permission to receive their orderflow (bundles/transactions). By specifying builders, users indicate that they wish for the matchmaker to share their orderflow with the given builder(s). Doing so can improve the speed at which transactions are included on-chain, because it will be more likely that one of the specified builders has an opportunity to build a block for a given slot. See the Bidding for Inclusion section of Searching Post-Merge for a more detailed explanation of why this works.

Searchers can also specify their own set of builders if they want to further restrict which builders receive their bundles. The builders parameter on a bundle will be the **intersection** of the lists specified by all the user(s) and searcher(s) who contributed transactions to the bundle.

Wallet users specify their preferred builders in the Protect RPC URL as query params. Searchers specify builders in the privacy parameter of their mev_sendBundle call.

Searching Strategies

There are two primary methods to consider for searching on MEV-Share: **on-chain searching** and **probabilistic searching**.

On-Chain Searching

On-chain searching implies that **computation which was formerly done off-chain by searchers will need to be done on-chain**, because the data required is not shared outside of the execution context (i.e. the smart contract). For example, rather than using calldata/logs to determine the amount of tokens being traded, one would calculate this amount using reserve deltas on-chain.

Concretely, searchers will have to write smart contracts which read the data required to calculate parameters for their strategy (e.g. amount of tokens to swap for an arbitrage), make the calculation, and build the transaction to execute their strategy, all at execution-time. Doing these calculations on-chain provides the benefit of accurate intra-block information, at the cost of gas usage.

See simple-blind-arbitrage for an example of an on-chain searcher.

Probabilistic Searching

Because on-chain information can be limited, searchers may have to **make several attempts in parallel for a given opportunity**, each with slightly different parameters. For example, if we don't know the trade direction, we can simply send two bundles, each of which assumes one of two trade directions. In this case, one bundle will revert, and the other (if done correctly) will not. Using a probabilistic strategy like this saves gas which makes you more competitive as a searcher. In lieu of probabilistic strategies, searchers must calculate their strategy parameters on-chain, which can be costly in terms of gas usage.

> Note: The beta release of MEV-Share only supports backruns, meaning that sandwich strategies are not currently viable.

Getting Started

To access orderflow from MEV-Share, query https://mev-share.flashbots.net. (You can even open this in your web browser!)

To get started searching on MEV-Share, take a look at our example arbitrage bot for an introductory example of on-chain searching. You can also build your own bot using a client library like matchmaker-ts. **Matchmaker-ts** is Flashbots' reference implementation of the MEV-Share Matchmaker client, written in Typescript. We expect clients for more languages to be available soon, and you can also check out simple-blind-arbitrage for an example in js. If you prefer, you can also interact with the RPC endpoint directly.

See the Getting Started Docs for more detailed instructions on connecting to, and interacting with MEV-Share.

Further Reading & Resources

In this short guide, we covered the basics of MEV-Share, and how to design strategies for searching on MEV-Share orderflow. We touched on new features like inclusion predicates and bundle composition — which give users more granular execution guarantees and allow searchers to execute more advanced strategies.

Today, you can access orderflow from MEV-Share at https://mev-share.flashbots.net. You can search on this orderflow using our example arbitrage bot or create your own bot with a client library like matchmaker-ts.

We hope this post kickstarts your curiosity about searching on MEV-Share. While the system is still in beta, we're excited to see what you do with it, and are eager to get your feedback! You can help us develop more exciting and innovative features by **getting in touch on our Forum or Discord** (say hi in the new **#mev-share** channel). Check out the links below to learn

more and start searching!

- MEV-Share Design Document
- MEV-Share Beta Release
- Flashbots MEV-Share Docs
- MEV-Share Protocol Specification
- simple-blind-arbitrage bot
- matchmaker-ts client
- Flashbots Forum
- Flashbots Discord

Thank you to Robert Miller, Elaine Hu, Vitaliy Drogan, and Reid Yager for their review and contributions to this post.