[

image

1920×1440 168 KB

](https://collective.flashbots.net/uploads/default/original/2X/8/83ab4a7b6d4bb62fbd9acc4c1c22a7e58f3e98b8.jpeg)

MEV-Share: Where transactions and bundles find their perfect match

The following document outlines a design for MEV-Share, a permissionless and private matchmaking protocol between users and searchers. It has the following benefits:

- Users / wallets / applications: receive the MEV their transactions create
- Searchers: extract MEV from transactions they otherwise wouldn't have access to
- Builders: build blocks with additional orderflow

MEV-Share is designed to hand power back to users - they choose exactly what transaction data to hide and reveal through programmable privacy

. Simultaneously, searchers bid in a competitiveorderflow auction ("OFA") for the right to execute against users' private transactions. Selectively revealing transaction data can help searchers better optimize their bids, which means higher payments back to the user.

To achieve this, we introduce a new entity called the matchmaker

. The matchmaker inserts users' private transactions into searchers' partially constructed bundles and simulates them, looking for matches. Matched bundles are sent to builders with a condition ("validity condition") that MEV payments are made to users. Importantly, MEV-Share does not enshrine any individual builder.

MEV-Share builds on MEV-Boost by further unbundling the transaction supply chain. Whereas MEV-Boost enabled collaboration between validators and builders, MEV-Share does the same for searchers and users. By keeping their transactions private, users can now bargain for the MEV they create without permissioning searcher competition. Programmable information sharing enables collaboration and optimization.

Finally, MEV-Share is an early exploration of how programmable privacy, preference expression, and credible commitments can enable decentralized block building. MEV-Share facilitates collaboration between users and searchers to address the need for MEV redistribution. However, the general concepts and infrastructure can be applied to all relationships in the transaction supply chain: searcher to searcher, searcher to builder, and even builder to builder. Hence, MEV-Share can be seen as a "proto-SUAVE" - an early implementation of SUAVE's primitives that can progressively evolve into a fully decentralized network.

Design Goals

MEV-Share was designed with the following goals:

- MEV Redistribution

: As much MEV generated by a user's transaction is returned to them as possible.

- Permissionless for Searchers

: Any searcher should be able to participate and extract MEV. There should be no exclusivity deals or allow lists, and barriers to entry should be minimized. This maximizes competition between searchers and improves user outcomes.

- Backwards Compatible

: The system should be able to support MEV redistribution for users who are making regular transactions. Alternative transaction types (e.g. EIP-712 orders) should not be needed, although they can ideally be supported in the future. This lowers the adoption cost for the system.

- Programmably Private

: User transactions should be hidden from searchers to prevent exploitation and improve user's bargaining power. The system should support selective sharing of information to drive better outcomes for users and efficiency in MEV redistribution.

- Credibility

: The MEV-Share mechanism must be credible, i.e. all sides of the market (users, searchers, and block builders) must be able to rely on the rules of the game and no party should have an incentive to cheat. This includes the matchmaker of the auction.

- Doesn’t Enshrine a Single Block Builder

: bundles from MEV-Share can be shared with several block builders to maximize competition in the block builder market and inclusion guarantees for users. Notably, this goal must be balanced with maximizing the credibility of the mechanism for users and searchers, hence builders must have a way to make credible commitments to users and searchers or be trusted by them.

Glossary

- User

: a normal Ethereum user, or wallet, who sends transactions

- Searcher

: advanced ethereum user specialized in finding MEV opportunities and sending advanced transaction types like bundles

- Builder

: party specialized in the construction of blocks

- Matchmaker

: party specialized in matching searchers’ bundles against users’ private transactions

- Privacy preferences

: user preferences for selective data sharing about their transactions that help maximize their MEV payments and optimize searcher bundles

- Validity conditions

: conditions under which a bundle is considered valid and should be included in a block

- Hints

: information provided by a searcher that helps matchmakers match bundles against user’s transactions

Architecture

[

1510×860 53.3 KB

](https://collective.flashbots.net/uploads/default/original/2X/d/d973c7d3ac03093c609955039571e6c6a8db35e7.jpeg)

The process of matching a user’s transactions against searcher’s bundles works as follows:

1. Users send transactions to the matchmaker. They, or a service operating on their behalf such as a wallet, may configure privacy preferences

to program what data about their transactions to share, if any.

1. Searchers listen to the matchmaker, which shares selective data about users’ transactions according to their privacy preferences.

2. Searchers send bundles to be matched to user transactions by the matchmaker. These bundles contain hints

, which are information provided by searchers that the matchmaker uses to help it match bundles against user transactions.

1. Searcher bundles also indicate where the matchmaker inserts private transactions

.

1. The matchmaker inserts its private transactions into bundles and simulates them, looking for bundles that successfully extract MEV.

2. The matchmaker sends any matched bundles to builders along with a validity condition

requiring that the user receives a payment for the MEV their transaction creates. How this payment is determined is discussed in the “further discussion” section of this document.

1. Builders create full blocks which include the bundle and pay MEV extracted to the user per the validity condition.

In effect, users send transactions and the matchmaker matches them against searchers who pay users for the usage of their transactions. User transactions are by default hidden from searchers, who must create partially constructed bundles optimistically, e.g. using non-atomic MEV strategies. Programmable privacy preferences allow for selective data sharing that helps searchers optimize their bundles. Searchers pay ETH to the builder as they do currently, and validity conditions require that the builder pay some of that MEV to users.

Note that MEV-Share bundles are more expressive than traditional bundles. They require new features to aid in matching bundles against private transactions and enforcing payments of MEV to users. In particular, MEV-Share introduces privacy preferences

, hints

, private transaction insertion

, and validity conditions

. These are early use cases and implementations of SUAVE’s primitives, and are discussed below.

NOTE

: The pseudo-code APIs below are given solely as a demonstration and are subject to change after testing and community feedback is gathered.

Privacy Preferences

Users can specify how their transactions’ privacy is managed. Specifically, they can opt to share certain information about their transactions in order to help searchers optimize their bundles for MEV extraction. There are two types of data that can potentially be shared:

1. Upfront and static that can be broadcast publicly. Examples would be the to or from address of a transaction or a pool being swapped on.
2. Dynamic feedback based off of a simulation with searcher’s bundles and returned as feedback to searchers. An example could be whether the user’s transaction reverts or not.

A pseudo-code example of upfront and static data sharing could be:

```
{

 tx_hash_1: {
     from: 0xtrader,
     to: uniswap_v3_router,
     tokens_traded: [ETH, USDC]
 },
 tx_hash_2: {
     data: oracle_update_parameters
 }

}
```

Sharing selective data about a transaction becomes especially powerful when combined with the private transaction insertion API detailed below.

It is not clear yet whether, or how, to provide dynamic feedback to searchers. Dynamic data feedback enables more efficient optimization, but can allow adversarial searchers to extract information about private transactions. How to make them safe is an area for further exploration.

Private Transaction Insertion

The bundle API is extended to allow searchers to specify positions in their bundles for which they would like private transactions to be inserted. There are two cases to be aware of. First, when the searcher is aware of a specific transaction hash they would like inserted from the private data sharing API detailed above. In this case they can use a transaction hash to indicate to the matchmaker where to insert the private transaction corresponding to that hash. Here is a pseudo-code example:

```
{ txs: [0xhash, 0xf145hb0t5] blockNumber: 1000000, }
```

The second case is where the searcher allows the matchmaker to insert any private transactions instead of indicating a specific transaction hash. Here is a pseudo-code example:

{ txs: [__ , 0xf145hb0t5] blockNumber: 1000000, }

In this example a searcher is attempting to backrun some private transaction with their transaction 0xf145hb0t5

. They omitted the first entry in their list, indicating to the matcher where to insert private transactions it has access to and that the searcher is not aware of. By allowing for this, searchers can attempt to match against transactions that the matchmaker has but which haven't shared enough information for the searcher to know to insert it into their bundles.

Hints

Hints are information provided by searchers to help in matching their bundles against private transactions. Here is a pseudo-code example of a bundle with a hint:

{ txs: [__ , 0xf145hb0t5] blockNumber: 1000000, hints: { addresses_touched: [0x7a25, 0xd14b], Logs_emitted: 0x41000, } }

txs: A list of signed transactions to execute in an atomic bundle

blockNumber: A block number for which this bundle is valid on

hints: an object containing hints used to match bundles against private transactions

addresses_touched: A list of addresses which this bundle is interested in

logs_emitted: A list of logs which this bundle is interested in.

The above bundle is a backrun that is interested in backrunning private transactions and indicates so with an empty first position in their bundle. To help match this bundle against private transactions, the bundle provides a hint that it is interested in any transaction which touches the 0x7a25

address (the Uniswap router) and which emits a log with a hash of 0x41000

(the Sync log emitted by a pool).

Hints are useful when the matchmaker needs to match their private transactions against searcher bundles. In other words, they are useful when searchers have an empty entry in their bundle where the matchmaker can insert transactions and needs to find a match. In this case, they help the matchmaker narrow down the amount of possible transactions they would attempt to match. Hints are not useful when bundles indicate specific transactions or transaction hashes for inclusion. In that case, there isn't a need to search through many transactions for matches.

Validity Conditions

Validity conditions are conditions under which a bundle can be considered valid. For example, a validity condition could be that an address needs to have an ETH balance greater than or equal to 1 for the bundle to be considered valid. Here is a pseudo-code example:

{ txs: [0xpr1v4t3 , 0xf145hb0t5] blockNumber: 1000000, validity_conditions: { eth: {0xprivsender, 1.1} } }

Validity conditions enable matchmaker to encode a requirement that builders make MEV payments to users for their bundles to be considered valid. They can also be used to pay MEV to another address, such as a wallet's address for later disbursement. Further, validity conditions can be extended to accommodate other user cases as well, such as erc20 validity conditions.

Optionally users may set their own validity conditions for the matchmaker. As an example, a user could enforce that they receive a certain payment or else their bundle is not valid.

Summary

- Users, or wallets acting on their behalf, can program their transaction privacy preferences

to decide what data to selectively disclose to searchers

- Searchers indicate in their bundles where to insert private transactions
- Searchers provide hints

, which are information that helps to match their bundles against private transactions

- The matchmaker

attempts to match searcher bundles against private transactions and returns feedback to searchers to help them optimize

their bundles.

- Any successfully matched bundles are sent on to builders along with a validity condition

that requires the user be paid ETH.

Further Discussion

Trustless Enforcement of Validity Conditions

Initially validity conditions are enforced only by social convention, trusting that builders will respect them and pay MEV to users. In practice this limits matchmakers to sending bundles to only trusted builders. Two avenues that should be considered for improving trust guarantees are the usage of trusted hardware and cryptoeconomics.

First, if builders are leveraging trusted hardware then the matchmaker can verify that certain code is running inside of the trusted hardware which cannot be tampered with. As a result, the matchmaker has a guarantee that its bundles with validity conditions will be handled correctly.

Second, builders could post security bonds to make commitments to the matchmaker about users' outcomes and inclusion. However, it is unclear how to make these commitments falsifiable if bundles are shared with multiple builders. More work is needed to find a suitable cryptoeconomic solution, although this is a promising direction.

It should be noted that both stronger forms of privacy and the ability to make credible commitments are core building blocks of SUAVE; hence we expect MEV-Share to fold into SUAVE eventually and use it as infrastructure to decentralize over time

.

Privacy and Efficiency

There is a tradeoff between the amount of data that a user shares and the efficiency of the matchmaker. To make this clear imagine the extremes:

1. Searcher has complete visibility over a user's transaction

: searchers extract MEV as they do currently in the mempool. Optimization is efficient and bundles are submitted crafted specifically for individual transactions, such as atomic arbitrage backruns or sandwiches.

1. Searcher has no visibility over a user's transaction

: searchers are unable to optimize their extraction, so they must submit many bundles without knowing which will be valid or what parameters to use. Atomic MEV extraction becomes very difficult to optimize for.

Sharing some information, but not all, is a compromise. For example, a user could share the pair and protocol that they want to trade on, but not the direction or amount they want to trade. That lets searchers drastically narrow down the space of possible pairs they should submit bundles for without posing a risk to the users' execution. However, users risk being frontrun if they share too much information. The question is how much information is optimal for users to share to optimize their MEV payments without getting worse execution.

Finally, some users may not want privacy at all, such as some oracle updates. MEV-Share is flexible enough to accommodate this use case and can still enforce MEV payments to users.

Types of MEV Allowed

MEV-Share allows for any type of MEV, however MEV-Share attempts to maximize value for users and some types of MEV will not make sense, such as allowing a user to get sandwiched only to pay most of the MEV extracted from that sandwich to the user. In this case the user would be better off being backrun instead of sandwiched. Given this, matchmakers may attempt to restrict sandwiches and only allow for backruns or forms of frontrunning that are strictly positive for users, like just-in-time liquidity as a placeholder solution. Finding a robust method for assessing the outcome of the matchmaking process for the user is an open question necessary for future iterations of the design. In particular, combining the result of the execution of a user's transaction and the payment they receive is an important subproblem.

Alternatives to Enforcing Payments With Validity Conditions

MEV-Share uses the notion of validity conditions to enforce that builders make MEV payments back to users. An alternative way to achieve the same effect is for the matchmaker to require that searchers make payments to a contract instead of the builder. Then the matchmaker adds a transaction to each matched bundle that makes payments to the user and the builder (through gas fees or block.coinbase transfers) and sends this bundle on to builders. The benefit of this method is that it doesn't require trusting builders to enforce validity conditions. However, not all trust requirements can be lifted: builders still need to be trusted not to unbundle matched bundles sent to it. Moreover, the primary drawback is that requiring payments to be made to a contract means that searchers will need to change their execution to participate.

Preventing DDOS

The matchmaker is susceptible to DDOS attacks by searchers as it is both intended to be permissionless and requires a large amount of simulation to run. Some mechanism for preventing DDOS attacks is needed. Potential options could be reputation (as the Flashbots Builder uses today), requiring payment from searchers per bundle, or requiring searchers to post bonds that are slashable if they DDOS the matchmaker.

Decentralization and Relationship to Block Building

MEV-Share is a protocol for how to merge bundles with special rules. In this way, decentralizing MEV-Share can be seen as a similar problem as decentralizing block building more generally. The matchmaker can leverage similar methods for improving its trust guarantees as builders, such as trusted hardware or cryptoeconomics. However, this is most impactful when paired with similar improvements for builders, as the matchmaker relies on builders for inclusion of its bundles. Finally, MEV-Share is an early implementation of SUAVE's primitives, and we expect will be natively integrated into SUAVE's decentralized network in the future.

MEV-Share on non-PBS Chains

MEV-Share can be implemented on non-PBS chains, in particular on chains with "first-come-first-serve" ordering. On these chains users MEV-Share would work the same except instead of sending an atomic bundle to a builder, MEV-Share would send the desired ordering of transactions in rapid succession to the sequencer (or network). This, in effect, replicates an atomic bundle with some new risk. Furthermore, MEV-Share would need to enforce payments in another way, potentially using a contract as discussed above.

Incentivizing the Matchmaker

MEV-Share offers a valuable service to several parties, as outlined in the introduction. To ensure incentive compatibility and sustainability the matchmaker could charge a fee for these services, although where that fee is taken is not specified here.

How Much MEV to Pay to Users

The matchmaker is trusted to set the validity condition of a matched bundle but there is no defined method for how to set that validity condition. As an example, a simple approach would be to set it equal to a proportion of the payment a searcher makes to the fee recipient when simulating a matched bundle. More work is needed to find an optimal method for determining how much MEV to pay to users, as reducing payment to the validator changes block building calculus, impacting the execution of the user order. See this thread and this thread for further discussion and resources on MEV redistribution.

Batching User Transactions

It is desirable to batch user transactions because a group of transactions may generate MEV that no single transaction does, enabling users to receive greater amounts of MEV. However, there are a two key challenges with doing so:

1. There is no way to distinguish between a regular user's transaction and a searcher posing as a user. Searchers may submit their transactions as regular users attempting to get their transactions placed in a batch behind MEV generating user transactions. The matchmaker can detect and attempt to filter this, but it adds a significant amount of additional computational overhead.

2. It is not clear how to redistribute MEV to users when it is generated in a batch.

One area of future exploration would be to allow searchers to batch user transactions themselves instead of having the matchmaker batch user transactions. But again, the challenge of how to redistribute MEV appears here again.