

Precompiles

- [Overview](#)
- [Available Precompiles](#)
- - [IsConfidential](#)
 - [ConfidentialInputs](#)
 - [ConfidentialStore](#)
 - [ConfidentialRetrieve](#)
 - [NewDataRecord](#)
 - [FetchDataRecords](#)
 - [EthCall](#)
 - [SimulateBundle](#)
 - [ExtractHint](#)
 - [SubmitBundleJsonRPC](#)
 - [FillMevShareBundle](#)
 - [BuildEthBlock](#)
 - [SubmitEthBlockBidToRelay](#)
 - [SignEthTransaction](#)
- [Precompiles Governance](#)

Overview

Precompile are MEVM contracts that are implemented in native code instead of bytecode. Precompiles additionally can communicate with internal APIs. Currently the MEVM supports all existing Ethereum Precompiles up to Dencun, and introduces four new classes of precompiles:

1. offchain computation that is too expensive in solidity
2. calls to API methods to interact with the Confidential Data Store
3. calls to suavex
4. API Methods to interact with Domain-Specific Services
5. calls to retrieve context for the confidential compute requests

Available Precompiles

A list of available precompiles in Rigil are as follows:

IsConfidential

[Implementation](#)

Address: 0x0000000000000000000000000000000042010000

Determines if the current execution mode is regular (onchain) or confidential. Outputs a boolean value.

function

isConfidential ()

internal

view

(bool b)

Implementation

Provides the confidential inputs associated with a confidential computation request. Outputs are in bytes format.

confidentialInputs ()

(bytes

memory)

Implementation

Implementation

memory key)

internal
view
returns
(bytes
memory)

NewDataRecord

[Implementation](#)

Address:0x0000000000000000000000000000000042030000

Initializes data records within the ConfidentialStore.AllowedPeekers specifies which addresses can "get" data.AllowedStores specifies which addresses can "set" data. Prior to storing data, all bids should undergo initialization via this precompile.

function

newDataRecord (uint64 decryptionCondition ,

address []

memory allowedPeekers ,

address []

memory allowedStores ,

string

memory dataType)

FetchDataRecords

[Implementation](#)

Address:0x0000000000000000000000000000000042030001

Retrieves all data records correlating with a specified decryption condition.

function

fetchDataRecords (uint64 cond ,

string

memory namespace)

internal

view

returns

(DataRecord []

memory)

EthCall

[Implementation](#)

Address:0x0000000000000000000000000000000042100003

SubmitBundleJsonRPC

Implementation

Address:0x000000000000000000000000000000000043000001

Submits bytes as JSONRPC message to the specified URL with the specified method. As this call is intended for bundles, it also signs the params and adds X-Flashbots-Signature header, as usual with bundles. Regular eth bundles don't need any processing to be sent.

function

submitBundleJsonRPC (string

memory url ,

string

memory method ,

bytes

memory params)

internal

view

returns

(bytes

memory)

FillMevShareBundle

Implementation

Address:0x0000000000000000000000000000000000043200001

Joins the user's transaction and with the backrun, and returns encoded mev-share bundle. The bundle is ready to be sent via `SubmitBundleJsonRPC` .

function

fillMevShareBundle (DataId dataId)

internal

view

returns

(bytes

memory)

BuildEthBlock

Implementation

[illegible]

Constructs an Ethereum block based on the provided `bidIds`. The construction follows the order of `bidIds` as given.

function

```
buildEthBlock ( BuildBlockArgs memory blockArgs , DataId dataId ,
```

```
memory namespace )
```

Implementation

Submits a given builderBid to a mev-boost relay. Outputs any errors that arise during submission.

submitEthBlockBidToRelay (string

bytes

```
memory builderBid )
```

Implementation

Signs an Ethereum Transaction, 1559 or Legacy, and returns raw signed transaction bytes.txn is binary encoding of the transaction.signingKey is hex encoded string of the ECDSA private keywithout the 0x prefix .chainId is a hex encoded stringwith 0x prefix .

signEthTransaction (bytes

string

memory chainId ,

string

```
memory signingKey )
```

[view](#)

returns

(bytes

memory)

The governance process for adding precompiles is in it's early stages but is as follows:

- Discuss the idea in a [forum post](#)
 - Open a PR and provide implementation
 - Feedback and review
 - Possibly merge and deploy in the next network upgrade, or sooner, depending on the precompile
- [Edit this page](#)
- [Previous MEVM Next Confidential Data Store](#)