Quick Start

This quickstart guide contains all the information necessary to get up and running as a searcher on Flashbots. If you have any questions, do not hesitate to ask in the'# newcomers' or '# searchers' discord channels , or in theSearcher Self-Support Forum .

See you on-chain! ⚡

Bundle Relay URLS

Network URL Mainnet https://relay.flashbots.net Goerli https://relay-goerli.flashbots.net Sepolia https://relay-sepolia.flashbots.net

Who should use Flashbots Auction?

1. Ethereum bot operators (we call them "searchers") looking for fast, and risk free access to blockspace (for example, arbitrage and liquidation bots)
2. Ethereum users looking for frontrunning protection on their transactions (for example, Uniswap traders)
3. Ethereum Dapps with advanced use cases like account abstraction or gasless transactions

How does Flashbots work for searchers?

Flashbots provides a platform that connects searchers with validators, bypassing the public transaction pool. Searchers create 'bundles' of transactions they wish to send and forward these to block builders, such as Flashbots. The builder then simulates these bundles to ensure they won't revert and constructs a full block using the available bundles and transactions. Through the use of mev-boost and a network of relays and builders, these blocks are delivered to validators while preserving pre-trade privacy.

Getting onboarded to Flashbots is easy for searchers; you simply need to update how you send transactions.

How to send your first Flashbots bundle

To access the Flashbots network you will need three things:

1. A unique ECDSA-secp256k1 key pair for Flashbots to identify you
2. A method to communicate with the Flashbots network*Alchemy
3.
    - provides a convenient way to dispatch individual transactions to Flashbots.
4. A "bundle" comprising your transactions

When you send bundles to Flashbots, they are signed with your key, which allows us to confirm your identity and accumulate yourreputation over time. Reputation system is set up to protect the infrastructure from attacks like DDoS. Searcheres with higher reputation will have better access to the network especially during times of high congestion.

It's crucial to understand that this keydoes not manage any funds and doesnot have to be the main Ethereum key used for authenticating transactions. Its only function is to establish your identity with Flashbots. You can use any ECDSA-secp256k1 key for this purpose.

Next, you need a means to communicate with the Flashbots network. The Flashbots builder accepts bundles atrelay.flashbots.net , and there are specific RPC endpoints that you must use to transmit transactions to us. To simplify this process, we've integrated with several widely-used developer tools such as Ethers.js and web3.py. Below are some examples of how to configure a Flashbots provider:

- ethers.js
- web3.py
- go
- rust

const ethers =

require ("ethers.js") ; const

{ FlashbotsBundleProvider , }

=

require ("@flashbots/ethers-provider-bundle") ;

// Standard json rpc provider directly from ethers.js. You can use Infura, Alchemy, or your own node. const provider =

new

ethers . providers . JsonRpcProvider ({ url :

ETHEREUM_RPC_URL , }) ;

// authSigner is an Ethereum private key that does NOT store funds and is NOT your bot's primary key. // This is an identifying key for signing payloads to establish reputation and whitelisting const authSigner =

new

ethers . Wallet ("0x0000000000000000000000000000000000000000000000000000000000000000") ;

// Flashbots provider requires passing in a standard provider and an auth signer const flashbotsProvider =

await FlashbotsBundleProvider . create (provider , authSigner) ; import os

from eth_account . account import Account from eth_account . signers . local import LocalAccount from flashbots import flashbot from web3 import Web3 , HTTPProvider

Create a web3 object with a standard json rpc provider, such as Infura, Alchemy, or your own node.

w3

Web3 (HTTPProvider ("http://localhost:8545"))

ETH_ACCOUNT_SIGNATURE is an Ethereum private key that does NOT store funds and is NOT your bot's primary key.

This is an identifying key for signing payloads to establish reputation and whitelisting

ETH_ACCOUNT_SIGNATURE : LocalAccount = Account . from_key (os . environ . get ("ETH_SIGNATURE_KEY"))

Flashbots providers require both a standard provider and ETH_ACCOUNT_SIGNATURE (to establish reputation)

flashbot (w3 , ETH_ACCOUNT_SIGNATURE) package main

import

("bytes" "crypto/ecdsa" "encoding/json" "fmt" "io/ioutil" "math/big" "net/http" "time"

"github.com/ethereum/go-ethereum/accounts" "github.com/ethereum/go-ethereum/common/hexutil" "github.com/ethereum/go-ethereum/crypto")

const

(j =

"application/json" flashbotURL =

"https://relay.flashbots.net" stats =

"flashbots_getUserStats" flashbotXHeader =

"X-Flashbots-Signature" p =

"POST")

var

(privateKey ,

_

= crypto . HexToECDSA ("2e19800fcbbf0abb7cf6d72ee7171f08943bc8e5c3568d1d7420e52136898154" ,))

func

flashbotHeader (signature [ ] byte , privateKey * ecdsa . PrivateKey)

string

{ return crypto . PubkeyToAddress (privateKey . PublicKey) . Hex ( )

+ ":"

+ hexutil . Encode (signature) }

func

main ( )

{ mevHTTPClient :=

& http . Client { Timeout : time . Second *

3 , } currentBlock := big . NewInt (12_900_000) params :=

map [string] interface { } { "jsonrpc" :

"2.0" , "id" :

1 , "method" : stats , "params" :

[ ] interface { } { fmt . Sprintf ("0x%x" , currentBlock . Uint64 ( )) , } , } payload ,

_

:= json . Marshal (params) req ,

_

:= http . NewRequest (p , flashbotURL , bytes . NewBuffer (payload)) headerReady ,

_

:= crypto . Sign (accounts . TextHash ([ ] byte (hexutil . Encode (crypto . Keccak256 (payload)))) , privateKey ,) req . Header . Add ("content-type" , j) req . Header . Add ("Accept" , j) req . Header . Add (flashbotXHeader ,

flashbotHeader (headerReady , privateKey)) resp ,

_

:= mevHTTPClient . Do (req) res ,

_

:= ioutil . ReadAll (resp . Body) fmt . Println (string (res)) } use

ethers :: core :: rand :: thread_rng ; use

ethers :: prelude :: * ; use

ethers_flashbots :: * ; use

std :: convert :: TryFrom ; use

url :: Url ;

[tokio::main]

async

fn

main ( )

{ // Connect to the network let provider =

Provider :: < Http

:: try_from ("https://www.ankr.com/rpc/eth/") . unwrap ( ) ;

// This is your searcher identity let bundle_signer =

LocalWallet :: new (& mut

thread_rng ( )) ;

// This signs transactions and is randomly generated in this example let wallet =

LocalWallet :: new (& mut

thread_rng ( )) ;

// Add signer and Flashbots middleware let flashbots_client =

SignerMiddleware :: new (FlashbotsMiddleware :: new (provider , Url :: parse ("https://relay.flashbots.net") . unwrap ( ) , bundle_signer ,) , wallet ,) ; } Now that we have a private key to identify ourselves with and a Flashbots provider we can create and send a bundle. Here's how:

- ethers.js
- rust

const ethers =

require ("ethers.js") ; const

{ FlashbotsBundleProvider , }

=

require ("@flashbots/ethers-provider-bundle") ; const provider =

new

ethers . providers . JsonRpcProvider ({ url :

ETHEREUM_RPC_URL , }) ;

const authSigner =

new

ethers . Wallet ("0x2000000000000000000000000000000000000000000000000000000000000000") ; const flashbotsProvider =

await

FlashbotsBundleProvider . create (provider , authSigner) ;

const signedBundle =

await flashbotsProvider . signBundle ([{ signer :

SOME_SIGNER_TO_SEND_FROM , transaction :

SOME_TRANSACTION_TO_SEND , } ,]) ;

const bundleReceipt =

await flashbotsProvider . sendRawBundle (signedBundle , TARGET_BLOCK_NUMBER) ;

Seehttps://github.com/onbjerg/ethers-flashbots

let tx =

TransactionRequest :: pay ("vitalik.eth" ,

100) ; let pending_tx = client . send_transaction (tx ,

None) . await ? ;

// Get the receipt let receipt = pending_tx . await ? . ok_or_else (| |

eyre :: format_err! ("tx not included")) ? ; let tx = client . get_transaction (receipt . transaction_hash) . await ? ;

println! ("Sent transaction: {}\n" ,

serde_json :: to_string (& tx) ?) ; println! ("Receipt: {}\n" ,

serde_json :: to_string (& receipt) ?) ; That's it!

Next steps

Congrats! You should now have everything you need to start sending transactions to the Flashbots network.

- If you are looking to interact with Flashbots without using one of the libraries, check out the RPC endpoint documentation
- and other advanced concepts.
- For examples of advanced usage of Flashbots, check out the example searchers
- .
- For additional tools, check out the searcher libraries
- .
- For potential MEV opportunities, check out the MEV job board
- Edit this page Last updated on Feb 13, 2024 Previous Overview Next Simple Arbitrage Bot