EIP1193 (EVM) MPC Provider

@web3auth/ethereum-mpc-provider

â

TheEIP1193 Provider can be used to interact with any EVM compatible blockchain. This is a wrapper around theEthereum JavaScript Provider API with support for the Web3Auth MPC SDKs directly, where the private key is never reconstructed. The provider is used to sign the message using the partial keys stored in the Auth Network.

In this section we'll explore more about how you can use this provider with our SDKs.

Installationâ

@web3auth/ethereum-mpc-provider

â

- npm
- Yarn
- pnpm

npm install --save @web3auth/ethereum-mpc-provider yarn add @web3auth/ethereum-mpc-provider pnpm add @web3auth/ethereum-mpc-provider

Initialisationâ

ImportEthereumSigningProvider from@web3auth/ethereum-mpc-provider .

import

{

EthereumSigningProvider

}

from

"@web3auth/ethereum-mpc-provider" ;

const evmProvider =

new

EthereumSigningProvider ({ config :

EthereumSigningProviderConfig , }) ; This constructor takes an object with a config ofEthereumSigningProviderConfig as input.

Argumentsâ

EthereumSigningProviderConfig

export

interface

EthereumSigningProviderConfig

extends

BaseProviderConfig

{ chainConfig :

CustomChainConfig ; }

export

```
type

CustomChainConfig

=

{ chainNamespace :

ChainNamespaceType ; /* * The chain id of the chain/ chainId :

string ; /* * RPC target Url for the chain/ rpcTarget :

string ; /* * web socket target Url for the chain/ wsTarget ? :

string ; /* * Display Name for the chain/ displayName ? :

string ; /* * Url of the block explorer/ blockExplorerUrl ? :

string ; /* * Default currency ticker of the network (e.g: ETH)/ ticker ? :

string ; /* * Name for currency ticker (e.g:Ethereum) / tickerName ? :

string ; /* * Number of decimals for the currency ticker (e.g: 18)/ decimals ? :

number ; /* * Logo for the token/ logo ? :

string ; /* * Whether the network is testnet or not/ isTestnet ? :

boolean ; } ;

export

interface

BaseProviderConfig

extends

BaseConfig

{ chainConfig :

Partial < CustomChainConfig

      ; networks ? :

Record < string ,

CustomChainConfig

      ; skipLookupNetwork ? :

boolean ; } export

interface

BaseConfig

{ /* * Determines if this controller is enabled / disabled ? :

boolean ; }
```

Chain Configâ

While connecting your preferred chain, you need to pass the chainConfig as a parameter. The Chain IDs for the supported chains can be found onChainList . Please note that you need to pass over the hex value of the chain id in the provider config.

Some of the commonly used L2s and the Ethereum chain ids are listed below.

Hex Decimal Network 0x1 1 Ethereum Mainnet 0xAA36A7 11155111 Sepolia Testnet 0x38 56 Binance Smart Chain Mainnet

0x89 137 Polygon Mainnet 0xA86A 43114 Avalanche C-Chain 0xA 10 Optimism 0xE 14 Flare 0x13 19 Songbird

Exampleâ

const chainConfig =

{ chainNamespace :

CHAIN_NAMESPACES . EIP155 , chainId :

"0xaa36a7" ,

// Please use 0x1 for Mainnet rpcTarget :

"https://rpc.ankr.com/eth_sepolia" , displayName :

"Ethereum Sepolia Testnet" , blockExplorerUrl :

"https://sepolia.etherscan.io/" , ticker :

"ETH" , tickerName :

"Ethereum" , } ;

Setting up the providerâ

Once you have initialised the provider, you can set it up by passing in the MPC SDK instance into it.

const coreKitInstance =

new

Web3AuthMPCCoreKit ({ web3AuthClientId , web3AuthNetwork :

WEB3AUTH_NETWORK . MAINNET , chainConfig , manualSync :

true ,

// This is the recommended approach }) ;

const provider =

new

EthereumSigningProvider ({ config :

{ chainConfig }

}) ; provider . setupProvider (coreKitInstance) ;

Using the providerâ

On connection, you can use thisprovider as anEIP1193 provider withweb3.js orethers library.

- web3.js
- ethers.js

import

Web3

from

"web3" ;

const web3 =

new

Web3 (provider) ; import

{ ethers }

from

"ethers" ;

const provider =

new

ethers . providers . Web3Provider (provider) ; Once you have set up the provider, you can use the standard functions in the web3 library to get user's account, perform transactions, sign a message etc. Here we have listed a few examples to help you get started.

info ** Please refer to all the updated JSON RPC Methods with the Provider on the Official Ethereum Documentation **

Get User account and Balanceâ

web3.eth.getAccounts()

â

This method is used to fetch the address of the connected account.

// Get user's Ethereum public address const address =

(await web3 . eth . getAccounts ( )) [0] ;

// Get user's balance in ether const balance = web3 . utils . fromWei (await web3 . eth . getBalance (address) ,

// Balance is in wei) ;

Send Transactionâ

web3.eth.sendTransaction(object)

â

This function is used to broadcast a transaction on chain.

// Get user's Ethereum public address const fromAddress =

(await web3 . eth . getAccounts ( )) [0] ;

const destination =

"0xE0cef4417a772512E6C95cEf366403839b0D6D6D" ; const amount = web3 . utils . toWei (1) ;

// Convert 1 ether to wei

// Submit transaction to the blockchain and wait for it to be mined const receipt =

await web3 . eth . sendTransaction ({ from : fromAddress , to : destination , value : amount , }) ;

Sign a messageâ

The following example shows how to sign different types of messages with the connected user's private key.

Personal Signâ

web3.eth.personal.sign(originalMessage, fromAddress)

â

// Get user's Ethereum public address const fromAddress =

(await web3 . eth . getAccounts ( )) [0] ;

const originalMessage =

"YOUR_MESSAGE" ;

const signedMessage =

```
await web3 . eth . personal . sign ( originalMessage , fromAddress ) ;
```

Sign Typed Data V1â

eth_signTypedData

â

```
// Get user's Ethereum public address const fromAddress =
( await web3 . eth . getAccounts ( ) ) [ 0 ] ;
const originalMessage =
[ { type :
"string" , name :
"fullName" , value :
"John Doe" , } , { type :
"uint32" , name :
"userId" , value :
"1234" , } , ] ; const params =
[ originalMessage , fromAddress ] ; const method =
"eth_signTypedData" ;
const signedMessage =
await web3 . currentProvider . sendAsync ( { id :
1 , method , params , fromAddress , } ) ;
```

Sign Typed Data v3â

eth_signTypedData_v3

â

```
// Get user's Ethereum public address const fromAddress =
( await web3 . eth . getAccounts ( ) ) [ 0 ] ;
const originalMessage =
{ types :
{ EIP712Domain :
[ { name :
"name" , type :
"string" , } , { name :
"version" , type :
"string" , } , { name :
"verifyingContract" , type :
"address" , } , ] , Greeting :
[ { name :
"contents" , type :
"string" , } , ] , } , primaryType :
```

"Greeting" , domain :

{ name :

"web3auth" , version :

"1" , verifyingContract :

"0xE0cef4417a772512E6C95cEf366403839b0D6D6D" , } , message :

{ contents :

"Hello, from web3auth!" , } , } ; const params =

[fromAddress , originalMessage] ; const method =

"eth_signTypedData_v3" ;

const signedMessage =

await web3 . currentProvider . sendAsync ({ id :

1 , method , params , fromAddress , }) ;

Sign Typed Data v4â

eth_signTypedData_v4

â

/* Sign Typed Data v4 adds support for arrays and recursive data types.

Otherwise, it works the same as Sign Typed Data v3. */

// Get user's Ethereum public address const fromAddress =

(await web3 . eth . getAccounts ( )) [0] ;

const originalMessage =

{ types :

{ EIP712Domain :

[{ name :

"name" , type :

"string" , } , { name :

"version" , type :

"string" , } , { name :

"verifyingContract" , type :

"address" , } ,] , Greeting :

[{ name :

"contents" , type :

"string" , } ,] , } , primaryType :

"Greeting" , domain :

{ name :

"web3auth" , version :

"1" , verifyingContract :

"0xE0cef4417a772512E6C95cEf366403839b0D6D6D" , } , message :

{ contents :

"Hello, from web3auth!" , } , } ; const params =

[fromAddress , originalMessage] ; const method =

"eth_signTypedData_v4" ;

const signedMessage =

await web3 . currentProvider . sendAsync ({ id :

1 , method , params , fromAddress , }) ; Edit this page Previous EIP1193 (EVM) Private Key Provider Next Solana Private Key Provider