

EIP1193 (EVM) Provider

[@web3auth/ethereum-provider](#)

[^](#)

The [EIP1193](#) Provider can be used to interact with any EVM compatible blockchain. This is a wrapper around the [Ethereum JavaScript Provider API](#) with some additional functionalities around Web3Auth Private Key handling.

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

Installation[^]

[@web3auth/ethereum-provider](#)

[^](#)

- npm
- Yarn
- pnpm

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

Initialisation[^]

Import the `EthereumPrivateKeyProvider` class from `@web3auth/ethereum-provider` .

```
import
```

```
{
```

```
EthereumPrivateKeyProvider
```

```
}
```

```
from
```

```
"@web3auth/ethereum-provider" ;
```

Assign the `EthereumPrivateKeyProvider`

class to a variable[^]

After creating your Web3Auth instance, you need to initialize the Torus Wallet UI Plugin and add it to a class for further usage.

```
const privateKeyProvider =
```

```
new
```

```
EthereumPrivateKeyProvider ( {
```

```
config :
```

```
EthereumPrivKeyProviderConfig
```

```
} ) ; This constructor takes an object with a config of EthereumPrivKeyProviderConfig as input.
```

Arguments[^]

```
EthereumPrivKeyProviderConfig
```

```
export
```

```
interface
```

```
EthereumPrivKeyProviderConfig
```

```

extends
BaseProviderConfig

{ chainConfig :

Omit < CustomChainConfig ,

"chainNamespace"

; }

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/ blockExplorer :

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 ; } ; 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 Config^â

While connecting your preferred chain, you need to pass the chainConfig as a parameter. The Chain IDs for the supported chains can be found on [ChainList](#) . 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

Example

```
const chainConfig =
{ chainNamespace :
CHAIN_NAMESPACES . EIP155 , chainId :
"0x1" , rpcTarget :
"https://rpc.ankr.com/eth" , displayName :
"Ethereum Mainnet" , blockExplorer :
"https://etherscan.io" , ticker :
"ETH" , tickerName :
"Ethereum" , } ;
const privateKeyProvider =
new
EthereumPrivateKeyProvider ( { config :
{ chainConfig } , } ) ;
```

Setting up the provider

For Web3Auth PnP Web SDKs

If you are using chainNamespace: "eip155" while initializing Web3Auth or Web3AuthNoModal with the OpenloginAdapter , you need to add the privateKeyProvider to the OpenLogin instance.

```
const chainConfig =
{ chainNamespace :
CHAIN_NAMESPACES . EIP155 , chainId :
"0x1" , rpcTarget :
"https://rpc.ankr.com/eth" , displayName :
"Ethereum Mainnet" , blockExplorer :
"https://etherscan.io" , ticker :
"ETH" , tickerName :
"Ethereum" , } ;
const web3auth =
new
Web3AuthNoModal ( { clientId , chainConfig , web3AuthNetwork :
"sapphire_mainnet" , } ) ;
const privateKeyProvider =
```

```

new
EthereumPrivateKeyProvider ( { config :
{ chainConfig } , } ) ;
const openloginAdapter =
new
OpenloginAdapter ( { privateKeyProvider , adapterSettings :
{ ... } , mfaSettings :
{ ... } , loginSettings :
{ ... } , } ) ; web3auth . configureAdapter ( openloginAdapter ) ;
const web3authProvider =
await web3auth . connectTo ( WALLET_ADAPTERS . OPENLOGIN , { loginProvider :
"google" , } ) ;
// use this provider to interact with the blockchain

```

For Single Factor Auth Web SDK [^](#)

While using the SFA Web SDK, you need to pass the provider during the initialisation of SDK, while calling the init() function.

```

const chainConfig =
{ chainNamespace :
CHAIN_NAMESPACES . EIP155 , chainId :
"0x1" , rpcTarget :
"https://rpc.ankr.com/eth" , displayName :
"Ethereum Mainnet" , blockExplorer :
"https://etherscan.io" , ticker :
"ETH" , tickerName :
"Ethereum" , } ;
const web3authSfa =
new
Web3Auth ( { clientId ,
// Get your Client ID from the Web3Auth Dashboard chainConfig , web3AuthNetwork :
"sapphire_mainnet" , usePnPKey :
false ,
// Setting this to true returns the same key as PnP Web SDK, By default, this SDK returns CoreKitKey. } ) ;
const privateKeyProvider =
new
EthereumPrivateKeyProvider ( { config :
{ chainConfig } , } ) ;
web3authSfa . init ( privateKeyProvider ) ;
const web3authProvider =

```

```
await web3authSFAuth . connect ( { verifier , verifierId : sub , idToken , } ) ;
```

```
// use this provider to interact with the blockchain
```

For tKey JS SDK

In tKey JS SDK, there is not internal method of passing the provider directly. Hence, you need to setup the provider using the private key generated from tKey, using the setupProvider function of the EthereumPrivateKeyProvider.

```
setupProvider ( privKey :
```

```
string ) :
```

```
Promise < void
```

```
    ; // please complete the tKey setup and login before this. const chainConfig =
```

```
{ chainNamespace :
```

```
CHAIN_NAMESPACES . EIP155 , chainId :
```

```
"0x1" , rpcTarget :
```

```
"https://rpc.ankr.com/eth" , displayName :
```

```
"Ethereum Mainnet" , blockExplorer :
```

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

```
"ETH" , tickerName :
```

```
"Ethereum" , } ;
```

```
const reconstructedKey =
```

```
await tKey . reconstructKey ( ) ; privateKey = reconstructedKey ? . privKey . toString ( "hex" ) ;
```

```
const ethereumPrivateKeyProvider =
```

```
new
```

```
EthereumPrivateKeyProvider ( { config :
```

```
{ chainConfig } , } ) ;
```

```
await ethereumPrivateKeyProvider . setupProvider ( privateKey ) ; provider = ethereumPrivateKeyProvider . provider ;
```

```
// use this provider to interact with the blockchain
```

Using the provider

On connection, you can use this provider as an [EIP1193](#) provider with web3.js or ethers 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 Balance

web3.eth.getAccounts()

[^](#)

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 Transaction

web3.eth.sendTransaction(object)

[^](#)

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 message

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

Personal Sign

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

[^](#)

```
// 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 V1[^](#)

eth_signTypedData

[^](#)

```
// 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 v3[^](#)

eth_signTypedData_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_v3" ;
const signedMessage =
await web3 . currentProvider . sendAsync ( { id :
1 , method , params , fromAddress , } ) ;

```

Sign Typed Data v4[â](#)

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 , } ) ;
```

Fetch User's Private Key^â

eth_private_key

^â

This method is used to fetch the private key of logged in user. It is only available for in-app adapters like openlogin .

// Assuming user is already logged in. async

```
getPrivateKey ( )
```

```
{ const privateKey =
```

```
await web3auth . provider . request ( { method :
```

```
"eth_private_key" } ) ; // Do something with privateKey } Edit this page Previous Overview Next Solana Provider
```