Account Abstraction Provider

Installationâ

To use native account abstraction, you'll need to install the@web3auth/account-abstraction-provider , which allows you to create and interact with Smart Contract Wallets (SCWs). This provider simplifies the entire process by managing the complex logic behind configuring the account abstraction provider, bundler, and preparing user operations.

```
npm install --save @web3auth/account-abstraction-provider
```

Configureâ

When instantiating the Account Abstraction Provider, you can pass configuration objects to the constructor. These configuration options allow you to select the desired Account Abstraction (AA) provider, as well as configure the bundler and paymaster, giving you flexibility and control over the provider.

```
import

{ AccountAbstractionProvider , SafeSmartAccount , }

from

"@web3auth/account-abstraction-provider" ;

const chainConfig =

{ chainNamespace :

CHAIN_NAMESPACES . EIP155 , chainId :

"0xaa36a7" , rpcTarget :

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

"Ethereum Sepolia Testnet" , blockExplorerUrl :

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

"ETH" , tickerName :

"Ethereum" , logo :

"https://cryptologos.cc/logos/ethereum-eth-logo.png" , } ;

const accountAbstractionProvider =

new

AccountAbstractionProvider ( { config :

{ chainConfig , smartAccountInit :

new

SafeSmartAccount ( ) , bundlerConfig :

{ // Get the pimlico API Key from dashboard.pimlico.io url :

https://api.pimlico.io/v2/11155111/rpc?apikey= { pimlicoAPIKey } , } , } , } ) ;
```

Please note this is the important step for setting the Web3Auth account abstraction provider.

- Configure Smart Account provider
- Configure Bundler
- Configure Sponsored Paymaster
- Configure ERC-20 Paymaster

Configure Smart Account Providerâ

Web3Auth offers the flexibility to choose your preferred Account Abstraction (AA) provider. Currently, we support Safe, Kernel, Biconomy, and Trust.

- Safe
- Kernel
- Biconomy
- Trust

```
import

{ AccountAbstractionProvider , SafeSmartAccount , }

from

"@web3auth/account-abstraction-provider" ;

const chainConfig =

{ chainNamespace :

CHAIN_NAMESPACES . EIP155 , chainId :

"0xaa36a7" , rpcTarget :

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

"Ethereum Sepolia Testnet" , blockExplorerUrl :

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

"ETH" , tickerName :

"Ethereum" , logo :

"https://cryptologos.cc/logos/ethereum-eth-logo.png" , } ;

const accountAbstractionProvider =

new

AccountAbstractionProvider ( { config :

{ chainConfig , smartAccountInit :

new

SafeSmartAccount ( ) , bundlerConfig :

{ // Get the pimlico API Key from dashboard.pimlico.io url :

https://api.pimlico.io/v2/11155111/rpc?apikey= { pimlicoAPIKey } , } , } , } ) ; import

{ AccountAbstractionProvider , KernelSmartAccount , }

from

"@web3auth/account-abstraction-provider" ; const chainConfig =

{ chainNamespace :

CHAIN_NAMESPACES . EIP155 , chainId :

"0xaa36a7" , rpcTarget :

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

"Ethereum Sepolia Testnet" , blockExplorerUrl :

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

"ETH" , tickerName :

"Ethereum" , logo :

"https://cryptologos.cc/logos/ethereum-eth-logo.png" , } ;
```

```
const accountAbstractionProvider =
new
AccountAbstractionProvider ( { config :
{ chainConfig , smartAccountInit :
new
KernelSmartAccount ( ) , bundlerConfig :
{ // Get the pimlico API Key from dashboard.pimlico.io url :
https://api.pimlico.io/v2/11155111/rpc?apikey= { pimlicoAPIKey } , } , } , } ) ; import
{ AccountAbstractionProvider , BiconomySmartAccount , }
from
"@web3auth/account-abstraction-provider" ;
const chainConfig =
{ chainNamespace :
CHAIN_NAMESPACES . EIP155 , chainId :
"0xaa36a7" , rpcTarget :
"https://rpc.ankr.com/eth_sepolia" , displayName :
"Ethereum Sepolia Testnet" , blockExplorerUrl :
"https://sepolia.etherscan.io" , ticker :
"ETH" , tickerName :
"Ethereum" , logo :
"https://cryptologos.cc/logos/ethereum-eth-logo.png" , } ;
const accountAbstractionProvider =
new
AccountAbstractionProvider ( { config :
{ chainConfig , smartAccountInit :
new
BiconomySmartAccount ( ) , bundlerConfig :
{ // Get the pimlico API Key from dashboard.pimlico.io url :
https://api.pimlico.io/v2/11155111/rpc?apikey= { pimlicoAPIKey } , } , } , } ) ; import
{ AccountAbstractionProvider , TrustSmartAccount , }
from
"@web3auth/account-abstraction-provider" ; const chainConfig =
{ chainNamespace :
CHAIN_NAMESPACES . EIP155 , chainId :
"0xaa36a7" , rpcTarget :
"https://rpc.ankr.com/eth_sepolia" , displayName :
"Ethereum Sepolia Testnet" , blockExplorerUrl :
```

```
"https://sepolia.etherscan.io" , ticker :
"ETH" , tickerName :
"Ethereum" , logo :
"https://cryptologos.cc/logos/ethereum-eth-logo.png" , } ;
const accountAbstractionProvider =
new
AccountAbstractionProvider ( { config :
{ chainConfig , smartAccountInit :
new
TrustSmartAccount ( ) , bundlerConfig :
{ // Get the pimlico API Key from dashboard.pimlico.io url :
https://api.pimlico.io/v2/11155111/rpc?apikey= { pimlicoAPIKey } , } , } , } ) ;
```

Configure Bundlerâ

Web3Auth enables you to configure your bundler and define the paymaster context. The bundler aggregates the UserOperations and submits them on-chain via a global entry point contract.

Bundler support is not limited to the examples belowâyou can use any bundler of your choice.

- Pimlico
- ZeroDev

```
import
{ AccountAbstractionProvider , SafeSmartAccount , }
from
"@web3auth/account-abstraction-provider" ;
const chainConfig =
{ chainNamespace :
CHAIN_NAMESPACES . EIP155 , chainId :
"0xaa36a7" , rpcTarget :
"https://rpc.ankr.com/eth_sepolia" , displayName :
"Ethereum Sepolia Testnet" , blockExplorerUrl :
"https://sepolia.etherscan.io" , ticker :
"ETH" , tickerName :
"Ethereum" , logo :
"https://cryptologos.cc/logos/ethereum-eth-logo.png" , } ;
const accountAbstractionProvider =
new
AccountAbstractionProvider ( { config :
{ chainConfig , smartAccountInit :
new
SafeSmartAccount ( ) , bundlerConfig :
```

{ // Get the pimlico API Key from dashboard.pimlico.io url :

https://api.pimlico.io/v2/ { chainId } /rpc?apikey= { pimlicoAPIKey } , } , } , }) ; import

{ AccountAbstractionProvider , KernelSmartAccount , }

from

"@web3auth/account-abstraction-provider" ;

const chainConfig =

{ chainNamespace :

CHAIN_NAMESPACES . EIP155 , chainId :

"0xaa36a7" , rpcTarget :

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

"Ethereum Sepolia Testnet" , blockExplorerUrl :

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

"ETH" , tickerName :

"Ethereum" , logo :

"https://cryptologos.cc/logos/ethereum-eth-logo.png" , } ;

const accountAbstractionProvider =

new

AccountAbstractionProvider ({ config :

{ chainConfig , smartAccountInit :

new

KernelSmartAccount ( ) , bundlerConfig :

{ // Get the ZeroDev API from https://dashboard.zerodev.app/ url :

https://rpc.zerodev.app/api/v2/bundler/ { projectId } , } , } , }) ;

Configure Paymasterâ

You can configure the paymaster of your choice to sponsor gas fees for your users, along with the paymaster context. The paymaster context lets you set additional parameters, such as choosing the token for ERC-20 paymasters, defining gas policies, and more.

Sponsored Paymasterâ

- Pimlico
- ZeroDev

import

{ AccountAbstractionProvider , SafeSmartAccount , }

from

"@web3auth/account-abstraction-provider" ;

const chainConfig =

{ chainNamespace :

CHAIN_NAMESPACES . EIP155 , chainId :

"0xaa36a7" , rpcTarget :

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

"Ethereum Sepolia Testnet" , blockExplorerUrl :

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

"ETH" , tickerName :

"Ethereum" , logo :

"https://cryptologos.cc/logos/ethereum-eth-logo.png" , } ;

const accountAbstractionProvider =

new

AccountAbstractionProvider ({ config :

{ chainConfig , smartAccountInit :

new

SafeSmartAccount ( ) , bundlerConfig :

{ // Get the pimlico API Key from dashboard.pimlico.io url :

https://api.pimlico.io/v2/ { chainId } /rpc?apikey= { pimlicoAPIKey } , } , paymasterConfig :

{ // Get the pimlico API Key from dashboard.pimlico.io url :

https://api.pimlico.io/v2/ { chainId } /rpc?apikey= { pimlicoAPIKey } , } , } , }) ; import

{ AccountAbstractionProvider , SafeSmartAccount , }

from

"@web3auth/account-abstraction-provider" ;

const chainConfig =

{ chainNamespace :

CHAIN_NAMESPACES . EIP155 , chainId :

"0xaa36a7" , rpcTarget :

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

"Ethereum Sepolia Testnet" , blockExplorerUrl :

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

"ETH" , tickerName :

"Ethereum" , logo :

"https://cryptologos.cc/logos/ethereum-eth-logo.png" , } ;

const accountAbstractionProvider =

new

AccountAbstractionProvider ({ config :

{ chainConfig , smartAccountInit :

new

SafeSmartAccount ( ) , bundlerConfig :

{ // Get the ZeroDev API from https://dashboard.zerodev.app/ url :

https://rpc.zerodev.app/api/v2/bundler/ { projectId } , } , paymasterConfig :

{ // Get the ZeroDev API from https://dashboard.zerodev.app/ url :

https://rpc.zerodev.app/api/v2/paymaster/ { projectId } , } , } , }) ;

ERC-20 Paymasterâ

When using an ERC-20 paymaster, ensure you include the approval transaction, as Web3Auth does not handle the approval internall.

For Pimlico, you can specify the token you want to use in the paymasterContext. If you want to set up sponsorship policies, you can define those in the paymasterContext as well.Checkout the supported tokens for ERC-20 Paymaster on Pimlico.

import

{ AccountAbstractionProvider , SafeSmartAccount , }

from

"@web3auth/account-abstraction-provider" ;

const chainConfig =

{ chainNamespace :

CHAIN_NAMESPACES . EIP155 , chainId :

"0xaa36a7" , rpcTarget :

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

"Ethereum Sepolia Testnet" , blockExplorerUrl :

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

"ETH" , tickerName :

"Ethereum" , logo :

"https://cryptologos.cc/logos/ethereum-eth-logo.png" , } ;

const accountAbstractionProvider =

new

AccountAbstractionProvider ({ config :

{ chainConfig , bundlerConfig :

{ // Get the pimlico API Key from dashboard.pimlico.io url :

https://api.pimlico.io/v2/ { chainId } /rpc?apikey= { pimlicoAPIKey } , paymasterContext :

{ token :

"SUPPORTED_TOKEN_CONTRACT_ADDRESS" , } , } , smartAccountInit :

new

SafeSmartAccount ( ) , paymasterConfig :

{ // Get the pimlico API Key from dashboard.pimlico.io url :

https://api.pimlico.io/v2/ { chainId } /rpc?apikey= { pimlicoAPIKey } , } , } , }) ;

Set upâ

Once you have configured your AccountAbstractionProvider, you can now plug it in your Web3Auth Modal instance. If you are using the external wallets like MetaMask, Coinbase, etc, you can define whether you want to use the AccountAbstractionProvider, or EthereumPrivateKeyProvider by setting theuseAAWithExternalWallet inIWeb3AuthCoreOptions .

If you are settinguseAAWithExternalWallet totrue , it'll create a new Smart Account for your user, where the signer/creator of the Smart Account would be the external wallet.

If you are settinguseAAWithExternalWallet tofalse , it'll skip creating a new Smart Account, and directly use the external wallet to sign the transactions.

Configure Web3Auth Instanceâ

- PnP Modal SDK
- PnP No Modal SDK

```
import

{ EthereumPrivateKeyProvider }

from

"@web3auth/ethereum-provider" ; import

{ AccountAbstractionProvider , SafeSmartAccount }

from

"@web3auth/account-abstraction-provider" ; import

{ Web3Auth }

from

"@web3auth/modal" ;

const chainConfig =

{ chainNamespace :

CHAIN_NAMESPACES . EIP155 , chainId :

"0xaa36a7" , rpcTarget :

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

"Ethereum Sepolia Testnet" , blockExplorerUrl :

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

"ETH" , tickerName :

"Ethereum" , logo :

"https://cryptologos.cc/logos/ethereum-eth-logo.png" , } ;

const accountAbstractionProvider =

new

AccountAbstractionProvider ( { config :

{ chainConfig , bundlerConfig :

{ // Get the pimlico API Key from dashboard.pimlico.io url :

https://api.pimlico.io/v2/11155111/rpc?apikey= { pimlicoAPIKey } , } , smartAccountInit :

new

SafeSmartAccount ( ) , } } ) ;

const privateKeyProvider =

new

EthereumPrivateKeyProvider ( { config :

{ chainConfig } , } ) ;
```

```
const web3auth =

new

Web3Auth ( { clientId :

"YOUR_WEB3AUTH_CLIENT_ID" , web3AuthNetwork :

WEB3AUTH_NETWORK . SAPPHIRE_MAINNET , privateKeyProvider , accountAbstractionProvider // This will allow you to
use EthereumPrivateKeyProvider for // external wallets, while use the AccountAbstractionProvider // for Web3Auth
embedded wallets. useAAWithExternalWallet :

false , } ) ; import

{ EthereumPrivateKeyProvider }

from

"@web3auth/ethereum-provider" ; import

{ AccountAbstractionProvider , SafeSmartAccount }

from

"@web3auth/account-abstraction-provider" ; import

{ Web3AuthNoModal }

from

"@web3auth/no-modal" ;

const chainConfig =

{ chainNamespace :

CHAIN_NAMESPACES . EIP155 , chainId :

"0xaa36a7" , rpcTarget :

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

"Ethereum Sepolia Testnet" , blockExplorerUrl :

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

"ETH" , tickerName :

"Ethereum" , logo :

"https://cryptologos.cc/logos/ethereum-eth-logo.png" , } ;

const accountAbstractionProvider =

new

AccountAbstractionProvider ( { config :

{ chainConfig , bundlerConfig :

{ // Get the pimlico API Key from dashboard.pimlico.io url :

https://api.pimlico.io/v2/11155111/rpc?apikey= { pimlicoAPIKey } , } , smartAccountInit :

new

SafeSmartAccount ( ) , } } ) ;

const privateKeyProvider =

new

EthereumPrivateKeyProvider ( { config :
```

{ chainConfig } , }) ;

const web3auth =

new

Web3AuthNoModal ({ clientId :

"YOUR_WEB3AUTH_CLIENT_ID" , web3AuthNetwork :

WEB3AUTH_NETWORK . SAPPHIRE_MAINNET , privateKeyProvider , accountAbstractionProvider // This will allow you to use EthereumPrivateKeyProvider for // external wallets, while use the AccountAbstractionProvider // for Web3Auth embedded wallets. useAAWithExternalWallet :

false , }) ;

const authadapter =

new

AuthAdapter ( ) ; web3auth . configureAdapter (authadapter) ;

Configure Signerâ

The Web3Auth AccountAbstractionProvider is compatible with popular signer SDKs, including ethers, web3, and viem. You can choose your preferred package to configure the signer.

You can retreive the provider to configure the signer from Web3Auth instance.

- Viem
- Ethers
- Web3

import

{ createWalletClient }

from

"viem" ;

// Use your Web3Auth instance to retreive the provider. const provider = web3auth . provider ;

const walletClient =

createWalletClient ({ transport :

custom (provider) , }) ; import

{ ethers }

from

"ethers" ;

// Use your Web3Auth instance to retreive the provider. const provider = web3auth . provider ;

const ethersProvider =

new

ethers . providers . Web3Provider (provider) ; const signer =

await ethersProvider . getSigner ( ) ; import

Web3

from

"web3" ;

// Use your Web3Auth instance to retreive the provider. const provider = web3auth . provider ;

const web3 =

new

Web3 (provider) ;

Usageâ

Smart Account Addressâ

Once, you have set up the signer, you can use it to retreive the user's smart account address.

- Viem
- Ethers
- Web3

// Use walletClient instance from previous step const addresses =

await walletClient . getAddresses ( ) ;

const smartAccountAddress = addresses [0] ; const eoaAddress = addresses [1] ; // User signer from previous step const smartAccountAddress = signer . getAddress ( ) ; // Use web3 instance from previous step const addresses =

await web3 . eth . getAccounts ( ) ;

const smartAccountAddress = addresses [0] ; const eoaAddress = addresses [1] ;

Send Transactionâ

You can use your signer to submit on-chain transactions, while the Account Abstraction Provider handles the creation and submission of the UserOperation. You only need to passto ,data , andvalue ; other parameters will be discarded and overridden by the provider.

The bundler client determinesmaxFeePerGas andmaxPriorityFeePerGas to prevent transaction rejections. If you'd like to specify custom values, you can useAccountAbstractionProvider.bundlerClient to create and send the transaction.

Smart Accounts are basically smart contract deployed on chain. Therefore, the user's first transaction also deploys the smart contract for their wallet.

- Viem
- Ethers
- Web3

// Convert 1 ether to WEI format const amount = web3 . utils . toWei (1) ;

// Submits a user operation to the blockchain const receipt =

await web3 . eth . sendTransaction ({ to :

"DESTINATION_ADDRESS" , value : amount , // This will perform the transfer of ETH data :

"0x" , }) ; // Convert 1 ether to WEI format const amount = ethers . parseEther ("1.0") ;

// Submits a user operation to the blockchain const transaction =

await signer . sendTransaction ({ to :

"DESTINATION_ADDRESS" , value : amount , // This will perform the transfer of ETH data :

"0x" , }) ;

// Wait for the transaction to be mined const receipt =

await transaction . wait ( ) ; // Convert 1 ether to WEI format const amount =

parseEther ("1") ;

// Submits a user operation to the blockchain const hash =

await walletClient . sendTransaction ({ to :

"DESTINATION_ADDRESS" , value : amount , // This will perform the transfer of ETH data :

"0x" , }) ;

// Wait for the transaction to be mined const receipt =

await publicClient . waitForTransactionReceipt ({ hash }) ;

Sign transactionâ

You can choose to sign a transaction without sending it. If it's the user's first transaction, the calldata generated for theUserOperation will also include the necessary data to deploy a new smart account for the user.

- Viem
- Ethers
- Web3

// Convert 1 ether to WEI format const amount = web3 . utils . toWei (1) ;

const signature =

await web3 . eth . signTransaction ({ from :

"YOUR_ACCOUNT_ADDRESS" , to :

"DESTINATION_ADDRESS" , value : amount , }) ; // Convert 1 ether to WEI format const amount =

parseEther ("1") ;

const signature =

await signer . signTransaction ({ to :

"DESTINATION_ADDRESS" , value : amount , }) ; // Convert 1 ether to WEI format const amount =

parseEther ("1") ; const addresses =

await walletClient . getAddresses ( ) ;

const request =

await walletClient . prepareTransactionRequest ({ account : addresses [0] , to :

"DESTINATION_ADDRESS" , value : amount , }) ;

const signature =

await walletClient . signTransaction (request as

any) ;

Sign Messageâ

You can also sign messages using a smart account, though the signature and verification process differs from EOA wallets. Smart accounts follow the EIP 1271 standard for verification, using theisValidSignature method of the smart contract wallet, instead of theecrecover function used by EOAs.

Learn more about EIP 1271 .

- Viem
- Ethers
- Web3

const addresses =

await web3 . eth . getAccounts ( ) ;

const originalMessage =

"YOUR_MESSAGE" ;

const signedMessage =

await web3 . eth . personal . sign (originalMessage , addresses [0]) ; const originalMessage =

"YOUR_MESSAGE" ;

// Sign the message const signedMessage =

await signer . signMessage (originalMessage) ; const originalMessage =

"YOUR_MESSAGE" ;

const addresses =

await walletClient . getAddresses ( ) ;

const signedMessage =

await walletClient . signMessage ({ account : address [0] , message : originalMessage , }) ;

Send Batch Transactionâ

One of the most exciting features of Smart Accounts is the ability to perform batch transactions. Currently, if a user wants to swap Token A for Token B, they must first approve Token A and then complete the swap. With the batch transaction functionality of Smart Accounts, both steps can be combined into a single operation.

Performing a batch transaction differs slightly from the normal flow. To execute a batch transaction, you'll need to use theBundlerClient generated by theAccountAbstractionProvider . The Web3Auth instance provider can't be used for this, as it's a proxy provider designed to ensure compatibility with your preferred signer package for basic operations.

TheBundlerClient.sendUserOperation returns the UserOperation hash instead of transaction hash. UserOperation hash is just keccak256 hash of the entire user operation. To retreive the transaction details, we need to usewaitForUserOperationReceipt . The function will wait for the UserOperation to be included in a block, and will return a full UserOperation, with the addition of entryPoint, blockNumber, blockHash and transactionHash.

// Use the same accountAbstractionProvider we created earlier. const bundlerClient = accountAbstractionProvider . bundlerClient ! ; const smartAccount = accountAbstractionProvider . smartAccount ! ;

// 0.00001 ETH in WEI format const amount =

10000000000000n ;

const userOpHash =

await bundlerClient . sendUserOperation ({ account : smartAccount , calls :

[{ to :

"DESTINATION_ADDRESS" , value : amount , data :

"0x" , } , { to :

"DESTINATION_ADDRESS" , value : amount , data :

"0x" , } ,] , }) ;

// Retrieve user operation receipt const receipt =

await bundlerClient . waitForUserOperationReceipt ({ hash : userOpHash , }) ;

const transactionHash = receipt . receipt . transactionHash ;

Send transaction using ERC-20 Paymasterâ

To submit the transaction using ERC-20 paymaster, we'll require to first need to approve ERC-20 token to be used by the paymaster. Ideally, we should first check the token allowance, and only provide approve allowance to be used by the paymaster.

To modify the token allowance, you'll need to perform a write operation on the ERC-20 contract. In the example below, we're using Pimlico, but be sure to update the paymaster and ERC-20 token addresses according to your specific case. For using ERC-20 Paymaster, refer to the configuration forPimlico's ERC-20 paymaster in the AccountAbstractionProvider .

// Use the same accountAbstractionProvider we created earlier. const bundlerClient = accountAbstractionProvider . bundlerClient ! ; const smartAccount = accountAbstractionProvider . smartAccount ! ;

// Pimlico's ERC-20 Paymaster address const pimlicoPaymasterAddress =

```
"0x0000000000000039cd5e8aE05257CE51C473ddd1" ;
// USDC address on Ethereum Sepolia const usdcAddress =
"0x1c7D4B196Cb0C7B01d743Fbc6116a902379C7238" ;
// 0.00001 ETH in WEI format const amount =
10000000000000n ;
// 10 USDC in WEI format. Since USDC has 6 decimals, 10 * 10^6 const approvalAmount =
10000000n ;
const userOpHash =
await bundlerClient . sendUserOperation ( { account : smartAccount , calls :
[ // Approve USDC on Sepolia chain for Pimlico's ERC 20 Paymaster { to : usdcAddress , abi :
parseAbi ( [ "function approve(address,uint)" ] ) , functionName :
"approve" , args :
[ pimlicoPaymasterAddress , approvalAmount ] , } , { to :
"DESTINATION_ADDRESS" , value : amount , data :
"0x" , } , { to :
"DESTINATION_ADDRESS" , value : amount , data :
"0x" , } , ] , } ) ;
// Retrieve user operation receipt const receipt =
await bundlerClient . waitForUserOperationReceipt ( { hash : userOpHash , } ) ;
const transactionHash = receipt . receipt . transactionHash ;
```
Edit this page Previous EVM Provider Next Solana Provider