Solana Provider

@web3auth/solana-provider

â

The Solana Blockchain Provider is basically a wrapper around theSolana JSON RPC API making it easier to interact with the Solana Blockchain.

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

Installationâ

@web3auth/solana-provider

â

- npm
- Yarn
- pnpm

npm install --save @web3auth/solana-provider yarn add @web3auth/solana-provider pnpm add @web3auth/solana-provider

Initialisationâ

Import theSolanaPrivateKeyProvider class from@web3auth/solana-provider .

import

{

SolanaPrivateKeyProvider

}

from

"@web3auth/solana-provider" ;

Assign theSolanaPrivateKeyProvider

class to a variableâ

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

SolanaPrivateKeyProvider ({

config :

SolanaPrivKeyProviderConfig

}) ; This constructor takes an object with a config ofSolanaPrivKeyProviderConfig as input.

Argumentsâ

SolanaPrivKeyProviderConfig

export

interface

SolanaPrivKeyProviderConfig

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 Configâ

While connecting your preferred chain, you need to pass the chainConfig as a parameter. For solana you can use one of the following chain configs according to your use case.

- Solana Mainnet
- Solana Testnet
- Solana Devnet

const chainConfig =

{ chainNamespace :

CHAIN_NAMESPACES . SOLANA , chainId :

"0x1" ,

// Please use 0x1 for Mainnet rpcTarget :

""

// Please use some Production RPC Target for Solana Mainnet displayName :

"Solana Mainnet" , blockExplorer :

"https://explorer.solana.com" , ticker :

"SOL" , tickerName :

"Solana" , } ; const chainConfig =

{ chainNamespace :

CHAIN_NAMESPACES . SOLANA , chainId :

"0x2" ,

// Please use 0x2 for Testnet rpcTarget :

""

// Please use some Testing/ Public RPC Target for Solana Tesnet displayName :

"Solana Testnet" , blockExplorer :

"https://explorer.solana.com" , ticker :

"SOL" , tickerName :

"Solana" , } ; const chainConfig =

{ chainNamespace :

CHAIN_NAMESPACES . SOLANA , chainId :

"0x3" ,

// Please use 0x3 for Devnet rpcTarget :

""

// Please use some Testing/ Public RPC Target for Solana Devnet displayName :

"Solana Devnet" , blockExplorer :

"https://explorer.solana.com" , ticker :

"SOL" , tickerName :

"Solana" , } ;

Exampleâ

const chainConfig =

{ chainNamespace :

CHAIN_NAMESPACES . SOLANA , chainId :

"0x3" ,

// Please use 0x1 for Mainnet, 0x2 for Testnet, 0x3 for Devnet rpcTarget :

"https://api.devnet.solana.com" , displayName :

"Solana Devnet" , blockExplorer :

"https://explorer.solana.com" , ticker :

"SOL" , tickerName :

"Solana" , } ;

const privateKeyProvider =

new

SolanaPrivateKeyProvider ({ config :

{ chainConfig } , }) ;

Setting up the providerâ

For Web3Auth PnP Web SDKsâ

If you are usingchainNamespace: "solana" while initializingWeb3Auth orWeb3AuthNoModal with theOpenloginAdapter , you need to add theprivateKeyProvider to the OpenLogin instance.

const chainConfig =

{ chainNamespace :

CHAIN_NAMESPACES . SOLANA , chainId :

"0x3" ,

// Please use 0x1 for Mainnet, 0x2 for Testnet, 0x3 for Devnet rpcTarget :

"https://api.devnet.solana.com" , displayName :

"Solana Devnet" , blockExplorer :

"https://explorer.solana.com" , ticker :

"SOL" , tickerName :

"Solana" , } ;

const web3auth =

new

Web3AuthNoModal ({ clientId , chainConfig , web3AuthNetwork :

"sapphire_mainnet" , }) ;

const privateKeyProvider =

new

SolanaPrivateKeyProvider ({ 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 SDKâ

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 . SOLANA , chainId :

"0x3" ,

// Please use 0x1 for Mainnet, 0x2 for Testnet, 0x3 for Devnet rpcTarget :

"https://api.devnet.solana.com" , displayName :

"Solana Devnet" , blockExplorer :

"https://explorer.solana.com" , ticker :

"SOL" , tickerName :

"Solana" , } ;

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

SolanaPrivateKeyProvider ({ 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 SDKâ

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 SolanaPrivateKeyProvider.

setupProvider (privKey :

string) :

Promise < void

; import

{ getED25519Key }

from

"@toruslabs/openlogin-ed25519" ;

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

{ chainNamespace :

CHAIN_NAMESPACES . SOLANA , chainId :

"0x3" ,

// Please use 0x1 for Mainnet, 0x2 for Testnet, 0x3 for Devnet rpcTarget :

"https://api.devnet.solana.com" , displayName :

"Solana Devnet" , blockExplorer :

"https://explorer.solana.com" , ticker :

"SOL" , tickerName :

"Solana" , } ;

const reconstructedKey =

await tKey . reconstructKey ( ) ; secp256k1privateKey = reconstructedKey ?. privKey . toString ("hex") ;

const privateKey =

getED25519Key (secp256k1privateKey) . sk . toString ("hex") ; const privateKeyProvider =

new

SolanaPrivateKeyProvider ({ config :

{ chainConfig } , }) ;

await privateKeyProvider . setupProvider (privateKey) ; provider = privateKeyProvider . provider ;

// use this provider to interact with the blockchain

Using the providerâ

On connection, you can useweb3auth.provider as a solana provider with@web3auth/solana-provider along with@solana/web3.js library.

import

{

SolanaWallet

}

from

"@web3auth/solana-provider" ;

const solanaWallet =

new

SolanaWallet (provider) ; Once you have setup the provider, you can use the standard functions in thesolana/web3.js library to get user's account, perform transaction, sign a message etc. Here we have listed a few examples to help you get

started there:

info All the RPC methods which are available by default on Solana Blockchain are also available on the Solana Provider. Although, for the case ofphantom adapter they are not available.

You can refer to standard RPC calls for Solanahere

Get User Account and Balanceâ

solana_requestAccounts

â

This method is used fetch the address of connected account.

Exampleâ

```
import
{
Connection ,
LAMPORTS_PER_SOL ,
PublicKey ,
SystemProgram ,
Transaction
}
from
"@solana/web3.js" ; import
{
SolanaWallet
}
from
"@web3auth/solana-provider" ;
const solanaWallet =
new
SolanaWallet ( provider ) ;
// Get user's Solana public address const accounts =
await solanaWallet . requestAccounts ( ) ;
const connectionConfig =
await solanaWallet . request ( { method :
"solana_provider_config" , params :
[ ] , } ) ;
const connection =
new
Connection ( connectionConfig . rpcTarget ) ;
// Fetch the balance for the specified public key const balance =
```

await connection . getBalance (new

PublicKey (accounts [0])) ;

Sign a Transactionâ

signTransaction

â

This method is used to sign a single serialized transaction and it returns a signedTransaction object.

Exampleâ

import

{

Connection ,

LAMPORTS_PER_SOL ,

PublicKey ,

SystemProgram ,

Transaction

}

from

"@solana/web3.js" ; import

{

SolanaWallet

}

from

"@web3auth/solana-provider" ;

const solanaWallet =

new

SolanaWallet (provider) ;

const connectionConfig =

await solanaWallet . request ({ method :

"solana_provider_config" , params :

[ ] , }) ;

const connection =

new

Connection (connectionConfig . rpcTarget) ;

const accounts =

await solanaWallet . requestAccounts ( ) ; const blockhash =

(await connection . getRecentBlockhash ("finalized")) . blockhash ; const

TransactionInstruction

=

SystemProgram . transfer ({ fromPubkey :

new

PublicKey (accounts [0]) , toPubkey :

new

PublicKey (accounts [0]) , lamports :

0.01

*

LAMPORTS_PER_SOL , }) ; const transaction =

new

Transaction ({ recentBlockhash : blockhash , feePayer :

new

PublicKey (pubKey [0]) , }) . add (TransactionInstruction) ;

const signedTx =

await solanaWallet . signTransaction (transaction) ; console . log (signedTx . signature) ;

Sign all Transactionsâ

signAllTransactions

â

This method is used to sign a list of serialized transactions and it returns back list of signed ofTransaction objects.

Example:â

import

{

Connection ,

LAMPORTS_PER_SOL ,

PublicKey ,

SystemProgram ,

Transaction

}

from

"@solana/web3.js" ; import

{

SolanaWallet

}

from

"@web3auth/solana-provider" ;

const solanaWallet =

new

SolanaWallet (provider) ;

```
const connectionConfig =
await solanaWallet . request ( { method :
"solana_provider_config" , params :
[ ] , } ) ;
const connection =
new
Connection ( connectionConfig . rpcTarget ) ;
const accounts =
await solanaWallet . requestAccounts ( ) ; const blockhash =
( await connection . getRecentBlockhash ( "finalized" ) ) . blockhash ; const
TransactionInstruction
=
SystemProgram . transfer ( { fromPubkey :
new
PublicKey ( accounts [ 0 ] ) , toPubkey :
new
PublicKey ( accounts [ 0 ] ) , lamports :
0.01
*
LAMPORTS_PER_SOL , } ) ; const transaction =
new
Transaction ( { recentBlockhash : blockhash , feePayer :
new
PublicKey ( pubKey [ 0 ] ) , } ) . add ( TransactionInstruction ) ;
const signedTx =
await solanaWallet . signAllTransactions ( transaction ) ; console . log ( signedTx . signature ) ;
```

Sign and Send a Transactionâ

signAndSendTransaction

â

This function is used to sign and broadcast the transaction on chain. It returns the signature of broadcasted transaction.

Example:â

```
import
{
Connection ,
LAMPORTS_PER_SOL ,
PublicKey ,
SystemProgram ,
```

```
Transaction
}
from
"@solana/web3.js" ; import
{
SolanaWallet
}
from
"@web3auth/solana-provider" ;
const solanaWallet =
new
SolanaWallet ( provider ) ;
const connectionConfig =
await solanaWallet . request ( { method :
"solana_provider_config" , params :
[ ] , } ) ;
const connection =
new
Connection ( connectionConfig . rpcTarget ) ;
const accounts =
await solanaWallet . requestAccounts ( ) ; const blockhash =
( await connection . getRecentBlockhash ( "finalized" ) ) . blockhash ; const
TransactionInstruction
=
SystemProgram . transfer ( { fromPubkey :
new
PublicKey ( accounts [ 0 ] ) , toPubkey :
new
PublicKey ( accounts [ 0 ] ) , lamports :
0.01
*
LAMPORTS_PER_SOL , } ) ; const transaction =
new
Transaction ( { recentBlockhash : blockhash , feePayer :
new
PublicKey ( pubKey [ 0 ] ) , } ) . add ( TransactionInstruction ) ; const
{ signature }
```

=

await solanaWallet . signAndSendTransaction (transaction) ;

Sign a Messageâ

signMessage

â

This method is used sign any message from connected wallet.

Exampleâ

import

{

Connection ,

LAMPORTS_PER_SOL ,

PublicKey ,

SystemProgram ,

Transaction

}

from

"@solana/web3.js" ; import

{

SolanaWallet

}

from

"@web3auth/solana-provider" ;

const solanaWallet =

new

SolanaWallet (provider) ;

const msg =

Buffer . from ("Test Signing Message " ,

"utf8") ; const result =

await solanaWallet . signMessage (msg) ; console . log (result . toString ( )) ;

Fetch User's Private Keyâ

solanaPrivateKey

â

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

Exampleâ

//Assuming user is already logged in. async

getPrivateKey ( )

{ const privateKey =

await web3auth . provider . request ({ method :

"solanaPrivateKey" }) ; //Do something with privateKey }Edit this page Previous EIP1193 (EVM) Provider Next XRPL Provider