

# XRPL Provider

## [@web3auth/xrpl-provider](#)

â

The [@web3auth/xrpl-provider](#) is a Web3Auth provider that simplifies interaction with the XRPL Blockchain by serving as a wrapper around the [XRPL JSON RPC API](#) . It is used to interact with the XRPL blockchain and perform various operations like getting user's account, balance, signing a transaction, sending a transaction etc.

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

## Installationâ

### [@web3auth/xrpl-provider](#)

â

- npm
- Yarn
- pnpm

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

## Initialisationâ

Import the `XrplPrivateKeyProvider` class from [@web3auth/xrpl-provider](#) .

```
import
{
  XrplPrivateKeyProvider
}
from
"@web3auth/xrpl-provider" ;
```

## Assign the `XrplPrivateKeyProvider`

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
XrplPrivateKeyProvider ( {
  config :
  XrplPrivKeyProviderConfig
} ) ; This constructor takes an object with a config of XrplPrivKeyProviderConfig as input.
```

## Argumentsâ

```
XrplPrivKeyProviderConfig
export
interface
XrplPrivKeyProviderConfig
```

extends

BaseProviderConfig

{ chainConfig :

Omit < CustomChainConfig ,

"chainNamespace"

&

Pick < CustomChainConfig ,

"wsTarget"

; } 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

For getting the Chain Config for XRPL Chain, you need to use the `getXRPLChainConfig` function from `@web3auth/xrpl-provider`.

```
export
```

```
declare
```

```
const getXRPLChainConfig :
```

```
( network :
```

```
XRPLNetworkType , customChainConfig ? :
```

```
Partial < Omit < CustomChainConfig ,
```

```
"chainNamespace"
```

```
)
```

```
=>
```

```
CustomChainConfig
```

```
&
```

```
Pick < CustomChainConfig ,
```

```
"wsTarget"
```

; In this function, you can pass over your custom config as well as just the network type for web3auth to generate a standard config according to your requirement.

## Example

```
const xrplProvider :
```

```
any
```

```
=
```

```
new
```

```
XrplPrivateKeyProvider ( { config :
```

```
{ chainConfig :
```

```
getXRPLChainConfig ( "testnet" )
```

```
} , } ) ;
```

```
// devnet, testnet, mainnet
```

## Setting up the provider

### For Web3Auth PnP Web SDKs

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

```
import
```

```
{
```

```
Web3Auth
```

```
}
```

```
from
```

```
"@web3auth/modal" ; import
```

```

{
CHAIN_NAMESPACES
}

from
"@web3auth/base" ; import
{
OpenloginAdapter
}

from
"@web3auth/openlogin-adapter" ; import
{
XrplPrivateKeyProvider , getXRPLChainConfig }

from
"@web3auth/xrpl-provider" ; import
{ convertStringToHex ,
Payment , xrpToDrops }

from
"xrpl" ;

const web3auth =
new
Web3Auth ( { chainConfig :
{ chainNamespace :
CHAIN_NAMESPACES . OTHER , } , clientId =
"YOUR_WEB3AUTH_CLIENT_ID" ,
// get from https://dashboard.web3auth.io web3AuthNetwork =
"sapphire_mainnet" , } ) ;

const xrplProvider =
new
XrplPrivateKeyProvider ( { config :
{ chainConfig :
getXRPLChainConfig ( "testnet" ) ,
// devnet, testnet, mainnet } , } ) ;

const adapter =
new
OpenloginAdapter ( { privateKeyProvider : xrplProvider ,
// <-- Injecting the XRPL provider } ) ; web3AuthInstance . configureAdapter ( adapter ) ;

await web3auth . initModal ( ) ;

```

```
const web3authProvider =
await web3auth . connect ( ) ;
```

## Usage

After configuring the provider, you may utilize various functions from the `@web3auth/xrpl-provider` library for tasks such as obtaining the user's account, executing transactions, and signing messages. Below are a few sample use cases to assist you in getting started:

### Get User Account and Balance

#### `xrpl_getAccounts`

The purpose of this function is to retrieve the details of the connected account.

#### Example

```
try
{ // web3authProvider is from above const accounts =
await web3authProvider . request < string [ ]
    ( { method :
"xrpl_getAccounts" , } ) ;
if
( accounts )
{ const accInfo =
( await web3authProvider . request ( { method :
"account_info" , params :
[ { account : accounts [ 0 ] , strict :
true , ledger_index :
"current" , queue :
true , } , ] , } ) )
as
Record < string ,
Record < string ,
string
    ; console . log ( "XRPL account info" , accInfo ) ; // xrpl Account const account = accInfo ?.
account_data ?. Account ; // Balance const balance = accInfo ?. account_data ?. Balance ; }
else
{ console . log ( "No accounts found, please report this issue." ) ; } }
catch
( error )
{ console . error ( "Error" , error ) ; }
```

### Sign a Transaction

## **xrpl\_signTransaction**

[^](#)

The function is utilized for signing transactions.

### **Example**[^](#)

```
try
{
  const accounts =
    await web3authProvider . request < string [ ]
      ( { method :
        "xrpl_getAccounts" , } ) ;
  if
    ( accounts && accounts . length
      0 )
  {
    const tx :
      Payment
      =
      {
        TransactionType :
          "Payment" , Account : accounts [ 0 ]
        as
          string , Amount :
            xrpToDrops ( 2 ) , Destination :
              "rJAHPYmy4g3h7kzfj2Mzm2nHwpKuVdEvX" ,
              // Destination address } ;
    const txSign =
      await web3authProvider . request ( { method :
        "xrpl_signTransaction" , params :
          { transaction : tx , } , } ) ;
    console . log ( "txRes" , txSign ) ;
  }
  else
  {
    console . log ( "failed to fetch accounts" ) ;
  }
}
catch
( error )
{
  console . log ( "error" , error ) ;
}
```

### **Sign and Send a Transaction**[^](#)

## **xrpl\_submitTransaction**

[^](#)

The purpose of this function is to sign and broadcast a transaction onto the blockchain. Upon successful execution, it will return the signature of the broadcasted transaction.

### **Example:**[^](#)

```
try
```

```

{ const accounts =
await web3authProvider . request < string [ ]
    ( { method :
"xrpl_getAccounts" , } ) ;
if
( accounts && accounts . length
0 )
{ const tx :
Payment
=
{ TransactionType :
"Payment" , Account : accounts [ 0 ]
as
string , Amount :
xrpToDrops ( 2 ) , Destination :
"rJAHPYmy4g3h7kzfj2Mzm2nHwpKuVdEvX" , } ; const txSign =
await provider . request ( { method :
"xrpl_submitTransaction" , params :
{ transaction : tx , } , } ) ; console . log ( "txRes" , txSign ) ; }
else
{ console . log ( "failed to fetch accounts" ) ; } }
catch
( error )
{ console . log ( "error" , error ) ; }

```

## Sign a Message

### xrpl\_signMessage

This method is utilized for signing any message using the currently connected wallet.

### Example

```

try
{ const msg =
"Hello world" ; const hexMsg =
convertStringToHex ( msg ) ; const txSign =
await web3authProvider . request < { signature :
string
}
    ( { method :

```

```
"xrpl_signMessage" , params :  
{ message : hexMsg , } , } ) ; console . log ( "txRes" , txSign ) ; }  
catch  
( error )  
{ console . log ( "error" , error ) ; }
```

## Fetch User's Private Key

**private\_key**

The purpose of this method is to retrieve the private key of the currently logged-in user.

### Example

```
//Assuming user is already logged in. async  
getPrivateKey ( )  
{ const privateKey =  
await web3authProvider . request ( { method :  
"private_key" } ) ; //Do something with privateKey }Edit this page Previous Solana Provider Next Common Provider
```