Using of PnP React Native SDK

Once you've installed and successfully initialized Web3Auth, you can use it to authenticate your users.

Web3Auth returns the following functions:

- login()
  - Logs in the User using the particular configuration you require for authentication.
- logout()
  - Logs out the User.
- userInfo()
  - After logging in, the Web3Auth instance will provide you with information regarding the user that is logged in.
- privKey()
  - After login, gets the private key of the user (default issecp256k1
- key).
- ed25519Key()
  - After login, gets theed25519
- key.

Logging in a Userâ

login()

â

Trigger the login flow that navigates the user to a browser model that allows them to log in with the selected login provider.

Argumentsâ

- Table
- Interface

Parameter Description loginProvider It sets the OAuth login method to be used. You can use any of the supported values areGOOGLE ,FACEBOOK ,REDDIT ,DISCORD ,TWITCH ,APPLE ,LINE ,GITHUB ,KAKAO ,LINKEDIN ,TWITTER ,WEIBO ,WECHAT ,EMAIL_PASSWORDLESS . extraLoginOptions? It can be used to set the OAuth login options for correspondingloginProvider . For instance, you'll need to pass user's email address as. Default value for the field isnull , and it acceptsExtraLoginOptions as a value. redirectUrl? Url where user will be redirected after successfull login. By default user will be redirected to same page where login will be initiated. Default value for the field isnull , and acceptsstring as a value. appState? It can be used to keep track of the app state when user will be redirected to app after login. Default isnull , and acceptsstring as a value. mfaLevel? Customize the MFA screen shown to the user during OAuth authentication. Default value for field isMfaLevelType.default , which shows MFA screen every 3rd login. It acceptsMfaLevelType as a value. dappShare? Custom verifier logins can get a dapp share returned to them post successful login. This is useful if the dapps want to use this share to allow users to login seamlessly. It acceptsstring as a value. curve? It will be used to determine the public key encoded in the jwt token which returned ingetUserInfo function after user login. This parameter won't change format of private key returned by We3Auth. Private key returned bygetPrivKey is always secp256k1. To get the ed25519 key you can useed25519Key method. The default value isSUPPORTED_KEY_CURVES_TYPE.secp256k1 . async

login (options : SdkLoginParams) :

Promise < void

; export

type

SdkLoginParams

= Omit < LoginParams ,

"fastLogin"

|

"skipTKey"

|

"getWalletKey"

& Required < Pick < LoginParams ,

"loginProvider"

;

export

type

BaseRedirectParams

=

{ /* * redirectUrl is the dapp's url where user will be redirected after login. * * @remarks * Register this url at { @link "https://dashboard.web3auth.io"| developer dashboard } * else initialization will give error. */ redirectUrl ? :

string ; /* * Any custom state you wish to pass along. This will be returned to you post redirect. * Use this to store data that you want to be available to the dapp after login. */ appState ? :

string ; } ;

export

type

LoginParams

= BaseRedirectParams &

{ /* * loginProvider sets the oauth login method to be used. * You can use any of the valid loginProvider from the supported list. * * If this param is not passed then it will show all the available * login methods to user in a modal. * */ loginProvider :

LOGIN_PROVIDER_TYPE

|

CUSTOM_LOGIN_PROVIDER_TYPE ;

/* * You can set the `mfaLevel` to customize when mfa screen should be shown to user. * It currently accepts 4 values:- * - *'default'*: Setting mfa level to `default` will present mfa screen to user on every third login. * - *'optional'*: Setting mfa level to `default` will present mfa screen to user on every login but user can skip it. * - *'mandatory'*: Setting mfa level to `mandatory` will make it mandatory for user to setup mfa after login. * - *'none'*: Setting mfa level to `none` will make the user skip the mfa setup screen * * Defaults to `default` * @defaultValue `default` */ mfaLevel ? : MfaLevelType ;

/* * extraLoginOptions can be used to pass standard oauth login options to * loginProvider. * * For ex: you will have to pass `login_hint` as user's email and `domain` * as your app domain in `extraLoginOptions` while using `email_passwordless` * loginProvider */ extraLoginOptions ? : ExtraLoginOptions ;

/* * Custom Logins can get a dapp share returned to them post successful login. * This is useful if the dapps want to use this share to allow users to login seamlessly * dappShare is a 24 word seed phrase */ dappShare ? :

string ;

/* * How long should a login session last at a minimum in seconds * * @defaultValue 86400 seconds * @remarks Max value of sessionTime can be 7 * 86400 (7 days) */ sessionTime ? :

number ;

/* * This curve will be used to determine the public key encoded in the jwt token which returned in `getUserInfo` function after user login. * You can use that public key from jwt token as a unique user identifier in your backend. * * - *'secp256k1'*: secp256k1 based pub key is added as a wallet public key in jwt token to use. * - *'ed25519'*: ed25519 based pub key is added as a wallet public key in jwt token to use. * * Note: This parameter won't change format of private key returned by openlogin.

*Private key returned * by openlogin is always secp256k1. As of now you have to convert it to'ed25519' if you want. * You can use @toruslabs/openlogin-ed25519 npm package for this purpose. * * * @defaultValue secp256k1/* curve ? :

SUPPORTED_KEY_CURVES_TYPE ; } ;

Exampleâ

example.tsx await web3auth . login ({ loginProvider : LoginProvider . GOOGLE , redirectUrl : resolvedRedirectUrl , }) ;

Selecting Curveâ

Theweb3auth.login() method accepts acurve parameter. This parameter can be used to select the elliptic curve to use for the signature.

await web3auth . login ({ loginProvider : LoginProvider . GOOGLE , redirectUrl : resolvedRedirectUrl , curve :

"secp256k1" ,

// secp256k1 and ed25519 are supported }) ;

Exampleâ

- SECP256K1
- ED25519

await web3auth . login ({ loginProvider : LoginProvider . GOOGLE , redirectUrl : resolvedRedirectUrl , curve :

"secp256k1" , }) ; await web3auth . login ({ loginProvider : LoginProvider . GOOGLE , redirectUrl : resolvedRedirectUrl , curve :

"ed25519" , }) ;

Logging out a userâ

logout()

â

info Trigger the logout flow. It is not advised to be used in iOS since it will trigger a system dialog that asks if users want to allow a Login operation. This method has no input and returns aPromise that resolves if the operation is successful, and rejects if the operation failed.

Exampleâ

await web3auth . logout ( ) ;

Getting the User Informationâ

To get the information of the user, extracted from the id token, you can call theuserInfo method in the web3auth instance.

const userInfo = web3auth . userInfo ( ) ; { "aggregateVerifier": "tkey-google", "email": "john@gmail.com", "name": "John Dash", "profileImage": "https://lh3.googleusercontent.com/a/Ajjjsdsmdjmnm...", "typeOfLogin": "google", "verifier": "torus", "verifierId": "john@gmail.com", "dappShare": "<24 words seed phrase>", // will be sent only incase of custom verifiers "idToken": "", "oAuthIdToken": "", // will be sent only incase of custom verifiers "oAuthAccessToken": " // will be sent only incase of custom verifiers }

Getting thesecp256k1

private keyâ

Using theprivKey() method in the web3auth instance you can get thesecp256k1 private key.

const privateKey = web3auth . privKey ( ) ; "0ajjsdsd...."

Getting theed25519

private keyâ

Using theed25519Key() method in the web3auth instance you can get theed25519 private key.

const ed25519Key = web3auth . ed25519Key ( ) ; "666523652352635...."Edit this page Previous Initialize Next Whitelabel