Usage of MPC Core Kit Web SDK

Once you've installed and successfully initializedWeb3AuthMPCCoreKit , you can use it to authenticate your users. Further, you can use the native provider given by Web3Auth to sign transactions and interact with the blockchain.

Natively, the instance ofWeb3AuthMPCCoreKit (referred to ascoreKitInstance in our examples) returns the following functions:

Authentication Functions:

- loginWithOAuth()
- Logs in the user using OAuth Flow.
- loginWithJWT()
- Logs in the user with a JWT-based ID
- Token.
- handleRedirectResult()
- Handles the redirect result from the OAuth Flow.
- inputFactorKey()
- Inputs the Factor Key to recover the User's account in
- case they don't have enough factors.
- logout()
- Logs out the User.

Get help with Authentication Check out theAuthentication section of the SDK Reference to learn more about these functions.
Manual Sync Function:

- commitChanges()
- Commits the changes made to the User's account while in
- manual sync mode.
- To use this function, you'll need to configure the manual sync in theinitialization
- step.

Factor Handling Functions:

account, present in the state of the SDK.

- enableMFA()
- Enable MFA for the user. Deletes the Hashed Factor and generates a
- new Factor Key.
- createFactor()
- Creates a new Factor for the User's account to increase the
- factors of your account. It can be of type Device(index:2
-) or Recovery(index:3
-).
- deleteFactor()
- Deletes a Factor from the User's account.
- getCurrentFactorKey()
- Returns the current Factor Key of the User's

Note You shouldn't create too many factors, we recommend amaximum of 10 factors User & Wallet Account Functions:

- setTssWalletIndex()
- Sets the TSS Wallet Index for the User's account.
- getPublicSync()
- Returns the Public Address of the user in a Synchronous
- call.
- getUserInfo()

-
    - Returns the User's information received from the ID Token.
- getKeyDetails()
-
    - Returns the User's Key Details.

Additional Helper Functions:

- Web Storage
-
    - storeWebBrowserFactor()
-
        - Stores a Factor in the Web Local/
-
    - Session Storage.
-
    - getWebBrowserFactor()
-
        - Returns the Factor from the Web Local/
-
    - Session Storage.
- Security Questions
-
    - setSecurityQuestion()
-
        - Sets the Security Question Factor for the
-
    - User's account.
-
    - changeSecurityQuestion()
-
        - Changes the Security Question for the
-
    - User's account.
-
    - deleteSecurityQuestion()
-
        - Deletes the Security Question
-
    - for the User's account.
-
    - recoverFactor()
-
        - Recovers the Security Question
-
    - Factor for the User's account using the given password.
-
    - getQuestion()
-
        - Returns the Security Question for
-
    - the User's account.
- Conversion from Mnemonic
-
    - mnemonicToKey()
-
        - Converts a Mnemonic to a BN.
-
    - keyToMnemonic()
-
    -

- Converts a hex string to a Mnemonic.

Authentication Functionsâ

Login with OAuth

â

- loginWithOAuth()
- - Logs in the user using OAuth flow.

Login with JWT

â

- loginWithJWT()
- - Logs in the user with a JWT-based ID
- Token.

Handle Redirect Resultâ

- handleRedirectResult()
- - Handles the redirect result from the OAuth Flow.

Input Factor Keyâ

inputFactorKey(factorKey: BN): Promise;

â

Inputs the Factor Key into the SDK. This function is used to recover the User's account using the Factor Key. If the factor key is correct, the SDK initializes the User's account and logs them in.

If you want to change the factor key in the current state of the SDK, you can use this function.

Usageâ

const factorKey =

new

BN (backupFactorKey ,

"hex") ; await coreKitInstance . inputFactorKey (factorKey) ;

Logout

â

logout(): Promise;

â

Logs out the User from the application and clears the local storage.

- logout()
- - Logs out the User.

Manual Sync Functionâ

Commit Changesâ

commitChanges(): Promise;

â

Syncs the local metadata with the web3auth metadata server. This function is only to be used while in manual sync mode.

Usageâ

await coreKitInstance ?. commitChanges ( ) ;

Factor Handling Functionsâ

Enable MFAâ

enableMFA(enableMFAParams: EnableMFAParams): Promise;

â

Enables MFA for the user. It creates a device factor and stores it in the local storage. It also creates a backup factor and returns it to the user. You can also pass a factor key that can be used for the backup factor. If you don't pass a factor key, a new factor key will be generated.

Most importantly, this function deletes the Hashed Factor Key enabling a non-custodial flow.

Parametersâ

export

interface

EnableMFAParams

{ /* * *A BN used for encrypting your Device/ Recovery TSS Key Share. You can generate it using* *generateFactorKey()* *function or use an existing one.* / factorKey ? :

BN ; /* * *Setting the Description of Share - Security Questions, Device Share, Seed Phrase, Password Share, Social Share, Other. Default is Other.* / shareDescription ? :

FactorKeyTypeShareDescription ; /* * *Additional metadata information you want to be stored alongside this factor for easy identification.* / additionalMetadata ? :

Record < string , string

; } export

enum

FactorKeyTypeShareDescription

{ HashedShare

=

"hashedShare" , SecurityQuestions

=

"tssSecurityQuestions" , DeviceShare

=

"deviceShare" , SeedPhrase

=

"seedPhrase" , PasswordShare

=

"passwordShare" , SocialShare

=

"socialShare" , Other

=

"Other" , }

Usageâ

const factorKey =

await coreKitInstance . enableMFA ({ }) ; const factorKeyMnemonic =

keyToMnemonic (factorKey) ;

uiConsole ("MFA enabled, device factor stored in local store, deleted hashed cloud key, your backup factor key: " , factorKeyMnemonic ,) ;

Create Factorâ

A low-level function, helps you to create a backup factor key based on the type of TSS Share you want to create. You can pass your own factor key or let the SDK generate one for you.

Parametersâ

export

interface

CreateFactorParams

extends

EnableMFAParams

{ / *** Setting the Type of Share - Device or Recovery.**/ shareType :

TssShareType ; }

export

enum

TssShareType

{ DEVICE

=

2 , RECOVERY

=

3 , }

export

interface

EnableMFAParams

{ /* ** A BN used for encrypting your Device/ Recovery TSS Key Share. You can generate it using generateFactorKey() function or use an existing one.* / factorKey ? :

BN ; /* ** Setting the Description of Share - Security Questions, Device Share, Seed Phrase, Password Share, Social Share, Other. Default is Other.* / shareDescription ? :

FactorKeyTypeShareDescription ; /* ** Additional metadata information you want to be stored alongside this factor for easy identification.* / additionalMetadata ? :

Record < string ,

string

; }

export

enum

FactorKeyTypeShareDescription

{ HashedShare

=

"hashedShare" , SecurityQuestions

=

"tssSecurityQuestions" , DeviceShare

=

"deviceShare" , SeedPhrase

=

"seedPhrase" , PasswordShare

=

"passwordShare" , SocialShare

=

"socialShare" , Other

=

"Other" , }

Usageâ

const factorKey =

generateFactorKey ( ) ; await coreKitInstance . createFactor ({ shareType : exportTssShareType , factorKey : factorKey . private , }) ;

Delete Factorâ

deleteFactor(factorPub: TkeyPoint): Promise;

â

Deletes the Factor, respective to the factorPub provided for the User's account. You can get the factor Pubs by using thegetKeyDetails() function.

It will throw an error if you try to delete the factor that is currently active within the state of the SDK. Use theinputFactorKey() function to change the factor key in the current state of the SDK to be able to delete that factor.

Usageâ

const pubBuffer =

Buffer . from (factorPubToDelete ,

"hex") ; const pub =

Point . fromBufferSEC1 (pubBuffer) ; await coreKitInstance . deleteFactor (pub . toTkeyPoint ( )) ;

Get Current Factor Keyâ

getCurrentFactorKey(): IFactorKey;

â

Returns the current factor key in usage within the state of the SDK.

Usageâ

const factorKey = coreKitInstance . getCurrentFactorKey ( ) ;

User & Wallet Account Functionsâ

Set TSS Wallet Indexâ

setTssWalletIndex(accountIndex: number): void;

â

Sets the TSS Wallet Index for the User's account. This function can be used to deterministically derive different wallet addresses by using account indexes. Using this function you can add multi-address account support to your application.

note As an application, you need to persist this account index in the application state.

You can either choose to persist the account index temporarily using the state variable or use a more permanent storage likelocalStorage . Whenever you add, delete or update a share, you need to set the account index again.

Usageâ

await coreKitInstance . setTssWalletIndex (index) ;

Get Public Syncâ

getPublicSync()

â

This function returns the final public key of the user in a synchronous call.

NOTE When you use thesetTssWalletIndex() function to establish your account index, a unique public key is generated!Remember that each account index you set will result in a different public key.

Usageâ

await coreKitInstance . getPublicSync ( ) ;

Get User Infoâ

getUserInfo(): UserInfo;

â

Gives you the User's information received from the ID Token.

Returnsâ

export

type

UserInfo

=

TorusVerifierResponse

&

LoginWindowResponse ; export

interface

TorusVerifierResponse

{ email :

string ; name :

string ; profileImage :

string ; aggregateVerifier ? :

string ; verifier :

string ; verifierId :

string ; typeOfLogin :

LOGIN_TYPE ; ref ? :

string ; registerOnly ? :

boolean ; extraVerifierParams ? :

WebAuthnExtraParams ; } export

interface

LoginWindowResponse

{ accessToken :

string ; idToken ? :

string ; ref ? :

string ; extraParams ? :

string ; extraParamsPassed ? :

string ; state :

TorusGenericObject ; }

Usageâ

const user = coreKitInstance ?. getUserInfo ( ) ;

Get Key Detailsâ

getKeyDetails(): MPCKeyDetails;

â

Gives you the details of the keys and factors held by the User's account.

Returnsâ

export

type

MPCKeyDetails

=

{ metadataPubKey :

TkeyPoint ; threshold :

number ; requiredFactors :

number ; totalFactors :

number ; shareDescriptions :

ShareDescriptionMap ; tssPubKey ? :

TkeyPoint ; } ; export

type

ShareDescriptionMap

=

{ [shareIndexStr :

string] :

string [ ] ; } ;

Usageâ

const keyDetails = coreKitInstance ?. getKeyDetails ( ) ;

Additional Helper Functionsâ

Web Storage Helper Functionsâ

Store Web Browser Factorâ

storeWebBrowserFactor(factorKey: BN, mpcCoreKit: ICoreKit, storageKey: "local" | "session" = "local"): Promise

â

Helps you store a web browser factor in the local/ session storage.

Usageâ

import

bowser

from

"bowser" ; import

{ storeWebBrowserFactor }

from

"@web3auth/mpc-core-kit" ;

const browserInfo = bowser . parse (navigator . userAgent) ; const browserName =

{ browserInfo . browser . name } ; const browserData =

{ browserName , browserVersion : browserInfo . browser . version , deviceName : browserInfo . os . name , } ; const deviceFactorKey =

new

BN (await coreKitInstance . createFactor ({ shareType :

TssShareType . DEVICE , additionalMetadata : browserData , }) , "hex" ,) ; storeWebBrowserFactor (deviceFactorKey , coreKitInstance) ;

Get Web Browser Factorâ

getWebBrowserFactor(mpcCoreKit: ICoreKit, storageKey: "local" | "session" = "local")

â

Get the web browser factor from the local/ session storage.

Usageâ

import

{ getWebBrowserFactor }

from

"@web3auth/mpc-core-kit" ;

const factorKey =

await

getWebBrowserFactor (coreKitInstance !) ;

Security Questionâ

Set Security Questionâ

setSecurityQuestion(params: setSecurityQuestionParams): Promise;

â

Parametersâ

export

interface

setSecurityQuestionParams

{ mpcCoreKit :

Web3AuthMPCCoreKit ; question :

string ; answer :

string ; shareType ? :

TssShareType ; description ? :

Record < string ,

string

; tssIndex ? :

TssShareType ; } export

enum

TssShareType

{ DEVICE

=

2 , RECOVERY

=

3 , } Creates a security question share for your to easily recover user's account.

Usageâ

import

{

TssSecurityQuestion

}

from

"@web3auth/mpc-core-kit" ;

const securityQuestion :

TssSecurityQuestion

=

new

TssSecurityQuestion ( ) ;

await securityQuestion . setSecurityQuestion ({ mpcCoreKit : coreKitInstance , question , answer , shareType :

TssShareType . RECOVERY , }) ;

Change Security Questionâ

changeSecurityQuestion(params: changeSecurityQuestionParams): Promise;

â

Parametersâ

export

interface

changeSecurityQuestionParams

{ mpcCoreKit :

Web3AuthMPCCoreKit ; newQuestion :

string ; newAnswer :

string ; answer :

string ; } Changes the Security Question Share of the User's account. This helps you change the password if the user has lost it somehow. However, this function can only be used if the user has already logged in within the application while meeting the minimum share threshold.

Usageâ

import

{

TssSecurityQuestion

}

from

"@web3auth/mpc-core-kit" ;

const securityQuestion :

TssSecurityQuestion

=

new

TssSecurityQuestion ( ) ;

await securityQuestion . changeSecurityQuestion ({ mpcCoreKit : coreKitInstance , newQuestion , newAnswer , answer , }) ;

Delete Security Question Shareâ

deleteSecurityQuestion(mpcCoreKit: Web3AuthMPCCoreKit, deleteFactorKey?: boolean): Promise;

â

Deletes the Security Question Share of the User's account. This function can only be used if the user has already logged in within the application while meeting the minimum share threshold.

Usageâ

import

{

TssSecurityQuestion

}

from

"@web3auth/mpc-core-kit" ;

const securityQuestion :

TssSecurityQuestion

=

new

TssSecurityQuestion ( ) ;

await securityQuestion . deleteSecurityQuestion (coreKitInstance) ;

Get Factor Key from Security Questionâ

recoverFactor(mpcCoreKit: Web3AuthMPCCoreKit, answer: string): Promise

â

Returns the Factor Key stored against the security question share.

Usageâ

import

{

TssSecurityQuestion

}

from

"@web3auth/mpc-core-kit" ;

const securityQuestion :

TssSecurityQuestion

=

new

TssSecurityQuestion ( ) ;

const factorKey =

await securityQuestion . recoverFactor (coreKitInstance , answer) ;

Get Security Question for User's Accountâ

getQuestion(mpcCoreKit: Web3AuthMPCCoreKit): string;

â

Get the security question for the user's account.

Usageâ

const question = securityQuestion . getQuestion (coreKitInstance !) ;

Mnemonic Conversationsâ

Mnemonic to Keyâ

mnemonicToKey(shareMnemonic: string): string

â

Converts a Mnemonic to a BN.

const factorKeyBN =

keyToMnemonic (factorKeyMnemonic) ;

Key to Mnemonicâ

keyToMnemonic(shareHex: string): string

â

Converts a factor key hex string to a Mnemonic.

Usageâ

const factorKeyMnemonic =

keyToMnemonic (factorKeyHex) ;