

Usage of MPC Core Kit Web SDK

Once you've installed and successfully initialized `Web3AuthMPCCoreKit`, 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 of `Web3AuthMPCCoreKit` (referred to as `scoreKitInstance` in our examples) returns the following functions:

Authentication and Handling Functions:

- `loginWithOAuth()`
 - - Logs in the user in OAuth Implicit Flow.
- `loginWithJWT()`
 - - Logs in the user with a JWT-based ID Token.
- `logout()`
 - - Logs out the User.
- `handleRedirectResult()`
 - - Handles the redirect result from the OAuth Implicit Flow.

tip Checkout the [Authentication section](#) of the SDK Reference to learn more about these functions. Handling Factors:

- `setTssWalletIndex()`
 - - Sets the TSS Wallet Index for the User's account.
- `getPublicSync()`
 - - Returns the Public Sync of the User's account.
- `enableMFA()`
 - - Enable MFA for the user. Deletes the Hashed Factor and generates a new Factor Key.
- `inputFactorKey()`
 - - Inputs the Factor Key to recover the User's account.
- `getCurrentFactorKey()`
 - - Returns the current Factor Key of the User's account, present in the state of the SDK.
- `createFactor()`
 - - Creates a new Factor for the User's account.
- `deleteFactor()`
 - - Deletes a Factor from the User's account.

User Functions:

- `getUserInfo()`
 - - Returns the User's information received from the ID Token.
- `getKeyDetails()`
 - - Returns the User's Key Details.
- `commitChanges()`
 - - Commits the changes made to the User's account while in manual sync mode.

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.

Set TSS Wallet Index^â

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.

This feature is still in [pre-release](#) , so there can be some breaking changes in upcoming versions.

Usage^â

```
await coreKitInstance . setTssWalletIndex ( index ) ;
```

Get Public Sync^â

getPublicSync()

^â

This function returns the final public key of the user.

NOTE The public key returned will depend on the account index set using the `setTssWalletIndex()` function. A different account indexes will return a different public key.

Usage^â

```
await coreKitInstance . getPublicSync ( ) ;
```

Enable MFA

enableMFA(enableMFAParams: EnableMFAParams): Promise;

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.

Parameters

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" , }

Usage

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 ) ;
```

Input Factor Key[^]

inputFactorKey(factorKey: BN): Promise;

[^]

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.

Usage[^]

```
const factorKey =
```

```
new
```

```
BN ( backupFactorKey ,
```

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

Get Current Factor Key[^]

getCurrentFactorKey(): IFactorKey;

[^]

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

Usage[^]

```
const factorKey = coreKitInstance . getCurrentFactorKey ( ) ;
```

Create Factor[^]

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.

Parameters[^]

```
export
```

```
interface
```

```
CreateFactorParams
```

```
extends
```

```
EnableMFAParams
```

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

```
TssShareType ; }
```

```
export
```

```
enum
```

```
TssShareType
```

```
{ DEVICE
```

```

=
2 , RECOVERY

=
3 , }

export

interface

EnableMFAPParams

{ /* * 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" , }

```

Usage

```

const factorKey =

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

```

Delete Factor

deleteFactor(factorPub: TKeyPoint): Promise;

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.

Usage

```
const pubBuffer =
Buffer . from ( factorPubToDelete ,
"hex" ) ; const pub =
Point . fromBufferSEC1 ( pubBuffer ) ; await coreKitInstance . deleteFactor ( pub . toTkeyPoint ( ) ) ;
```

Get User Info

getUserInfo(): UserInfo;

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

Returns

```
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 ; }
```

Usage[â](#)

```
const user = coreKitInstance ?. getUserInfo ( ) ;
```

Get Key Details[â](#)

getKeyDetails(): MPCKKeyDetails;

[â](#)

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

Returns[â](#)

```
export
```

```
type
```

```
MPCKKeyDetails
```

```
=
```

```
{ metadataPubKey :
```

```
TkeyPoint ; threshold :
```

```
number ; requiredFactors :
```

```
number ; totalFactors :
```

```
number ; shareDescriptions :
```

```
ShareDescriptionMap ; tssPubKey ? :
```

```
TkeyPoint ; } ; export
```

```
type
```

```
ShareDescriptionMap
```

```
=
```

```
{ [ shareIndexStr :
```

```
string ] :
```

```
string [ ] ; } ;
```

Usage[â](#)

```
const keyDetails = coreKitInstance ?. getKeyDetails ( ) ;
```

Commit Changes[â](#)

commitChanges(): Promise;

[â](#)

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

Usage[â](#)

await coreKitInstance?.commitChanges(); * Web StoragestoreWebBrowserFactor() ** - 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.

Web Storage Helper Functions[â](#)

Store Web Browser Factor[â](#)

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

[â](#)

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

Usage[â](#)

```
import
browser
from
"browser" ; import
{ storeWebBrowserFactor }
from
"@web3auth/mpc-core-kit" ;
const browserInfo = browser . 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 Factor[â](#)

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

[â](#)

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

Usage[â](#)


```
import
{ getWebBrowserFactor }
from
"@web3auth/mpc-core-kit" ;
const factorKey =
await
getWebBrowserFactor ( coreKitInstance ! ) ;
```

Security Question[^](#)

Set Security Question[^](#)

setSecurityQuestion(params: setSecurityQuestionParams): Promise;

[^](#)

Parameters[^](#)

```
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.
```

Usage[^](#)

```
import
{
TssSecurityQuestion
}
```

```

from
"@web3auth/mpc-core-kit" ;

const securityQuestion :
TssSecurityQuestion
=
new
TssSecurityQuestion ( ) ;

await securityQuestion . setSecurityQuestion ( { mpcCoreKit : coreKitInstance , question , answer , shareType :
TssShareType . RECOVERY
} ) ;

```

Change Security Question[^](#)

changeSecurityQuestion(params: changeSecurityQuestionParams): Promise;

[^](#)

Parameters[^](#)

```

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.

Usage[^](#)

```

import
{
TssSecurityQuestion
}
from
"@web3auth/mpc-core-kit" ;

const securityQuestion :
TssSecurityQuestion
=
new
TssSecurityQuestion ( ) ;

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

```

Delete Security Question Share^â

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.

Usage^â

```
import
{
  TssSecurityQuestion
}
from
"@web3auth/mpc-core-kit" ;
const securityQuestion :
  TssSecurityQuestion
=
new
  TssSecurityQuestion ( ) ;
await securityQuestion . deleteSecurityQuestion ( coreKitInstance ) ;
```

Get Factor Key from Security Question^â

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

^â

Returns the Factor Key stored against the security question share.

Usage^â

```
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 Account[^](#)

getQuestion(mpcCoreKit: Web3AuthMPCCoreKit): string;

[^](#)

Get the security question for the user's account.

Usage[^](#)

```
const question = securityQuestion . getQuestion ( coreKitInstance ! ) ;
```

Mnemonic Conversations[^](#)

Mnemonic to Key[^](#)

mnemonicToKey(shareMnemonic: string): string

[^](#)

Converts a Mnemonic to a BN.

```
const factorKeyBN =  
keyToMnemonic ( factorKeyMnemonic ) ;
```

Key to Mnemonic[^](#)

keyToMnemonic(shareHex: string): string

[^](#)

Converts a factor key hex string to a Mnemonic.

Usage[^](#)

```
const factorKeyMnemonic =  
keyToMnemonic ( factorKeyHex ) ;Edit this page Previous Authentication Next Web3Auth tKey JS SDK
```