Using tKey JS SDK

Once you've installed all packages, instantiated tKey and initialized the service provider instance in your constructor, you can use it to authenticate your users and generate their tKey shares. Further, you can use a variety of functions exposed by the tKey SDK and its modules to manage different aspects of your users' authentication needs.

We'll be talking in-depth about the following operations:

- Logging in the User
- Getting tKey Details
- Reconstructing User's Private Key
- Generating a new Share
- Deleting a Share
- Using Modules for Further Operations
- Making Blockchain Calls

ThresholdKey

â

Natively, the instance oftKey , (ie.ThresholdKey) returns many functions, however, we have documented a few relevant ones here. You can check the table below for a list of all relevant functions, or the class reference to checkout the full list of functions.

- Table
- Class Reference

Function Description Arguments Return initialize Generates a Threshold Key corresponding to your login provider. params KeyDetails reconstructKey Reconstructs the user private key. Can only work if the minimum threshold of keys has reached _reconstructKeyMiddleware?: boolean ReconstructedKeyResult deleteShare Delete a share details from metadata. The corresponding share won't be valid anymore. shareIndex: BNString DeleteShareResult getKeyDetails Get the details of the keys present for the particular user. void KeyDetails generateNewShare Generate a new share for the reconstructed private key. void GenerateNewShareResult inputShareStoreSafe Input a share store into the tKey shareStore: ShareStore, autoUpdateMetadata?: boolean void outputShareStore Output a share store from the tKey shareIndex: BNString, polyID?: string ShareStore addShareDescription Add a description to a share shareIndex: string, description: string, updateMetadata?: boolean void deleteShareDescription Delete a description from a share shareIndex: string, description: string, updateMetadata?: boolean void updateShareDescription Update a description from a share shareIndex: string, oldDescription: string, newDescription: string, updateMetadata?: boolean void outputShare Output a share shareIndex: BNString, type?: string unknown inputShare Input a share share: unknown, type?: string void toJSON Convert the tKey to JSON void StringifiedType class

ThresholdKey

implements

ITKey

{ modules :

ModuleMap ; enableLogging :

boolean ; serviceProvider :

IServiceProvider ; storageLayer :

IStorageLayer ; shares :

ShareStorePolyIDShareIndexMap ; privKey :

BN ; lastFetchedCloudMetadata :

Metadata ; metadata :

Metadata ; manualSync :

boolean ; private setModuleReferences ; private initializeModules ; storeDeviceShare :

(deviceShareStore :

ShareStore , customDeviceInfo ? :

StringifiedType)

=>

Promise < void

;

// Call to trigger module to store shares constructor (args ? :

TKeyArgs) ; static

fromJSON (value :

StringifiedType , args :

TKeyArgs) :

Promise < ThresholdKey

; getStorageLayer ( ) :

IStorageLayer ; initialize (params ? :

{ withShare ? :

ShareStore ; importKey ? :

BN ; neverInitializeNewKey ? :

boolean ; transitionMetadata ? :

Metadata ; previouslyFetchedCloudMetadata ? :

Metadata ; previousLocalMetadataTransitions ? :

LocalMetadataTransitions ; delete1OutOf1 ? :

boolean ; }) :

Promise < KeyDetails

;

reconstructKey (_reconstructKeyMiddleware ? :

boolean) :

Promise < ReconstructedKeyResult

; deleteShare (shareIndex :

BNString) :

Promise < DeleteShareResult

; generateNewShare ( ) :

Promise < GenerateNewShareResult

; syncLocalMetadataTransitions ( ) :

Promise < void

; inputShareStoreSafe (shareStore :

ShareStore , autoUpdateMetadata ? :

boolean) :

Promise < void

; outputShareStore (shareIndex :

BNString , polyID ? :

string) :

ShareStore ; getKeyDetails ( ) :

KeyDetails ; addShareDescription (shareIndex :

string , description :

string , updateMetadata ? :

boolean) :

Promise < void

; deleteShareDescription (shareIndex :

string , description :

string , updateMetadata ? :

boolean) :

Promise < void

; updateShareDescription (shareIndex :

string , oldDescription :

string , newDescription :

string , updateMetadata ? :

boolean) :

Promise < void

; getTKeyStore (moduleName :

string) :

Promise < TkeyStoreItemType [ ]

; getTKeyStoreItem (moduleName :

string , id :

string) :

Promise < TkeyStoreItemType

; outputShare (shareIndex :

BNString , type ? :

string) :

Promise < unknown

; inputShare (share :

unknown , type ? :

string) :

Promise < void

; toJSON ( ) :

StringifiedType ; getApi ( ) :

ITKeyApi ;

```
_refreshShares ( threshold :
number , newShareIndexes :
Array < string
        , previousPolyID :
PolynomialID ) :
Promise < RefreshSharesResult
        ; _initializeNewKey ( { determinedShare , initializeModules , importedKey , delete1OutOf1 , } ? :
{ determinedShare ? :
BN ; initializeModules ? :
boolean ; importedKey ? :
BN ; delete1OutOf1 ? :
boolean ; } ) :
Promise < InitializeNewKeyResult
        ; _setKey ( privKey :
BN ) :
void ;
*
reconstructLatestPoly ( ) :
Polynomial ;
* *
updateSDK ( params ? :
{ withShare ? :
ShareStore
} ) :
Promise < ThresholdKey
        ;
* *
inputShareStore ( shareStore :
ShareStore ) :
void ;
* *
addLocalMetadataTransitions ( params :
{ input :
LocalTransitionData ; serviceProvider ? :
IServiceProvider ; privKey ? :
Array < BN
        ; acquireLock ? :
```

boolean ; }) :

Promise < void

;

* /* * *catchupToLatestShare recursively loops fetches metadata of the provided share and checks if there is an encrypted share for it. * @param shareStore - share to start of with * @param polyID - if specified, polyID to refresh to if it exists* /*

catchupToLatestShare (params :

{ shareStore :

ShareStore ; polyID ? :

PolynomialID ; includeLocalMetadataTransitions ? :

boolean ; }) :

Promise < CatchupToLatestShareResult

;

* *

getMetadata ( ) :

IMetadata ;

*

*

getCurrentShareIndexes ( ) :

string [ ] ;

* * haveWriteMetadataLock :

string ;

*

*

generateAuthMetadata (params :

{ input :

Metadata [ ]

}) :

AuthMetadata [ ] ;

* *

setAuthMetadata (params :

{ input :

Metadata ; serviceProvider ? :

IServiceProvider ; privKey ? :

BN

}) :

Promise < { message :

string ; }

;

* *

setAuthMetadataBulk (params :

{ input :

Metadata [ ] ; serviceProvider ? :

IServiceProvider ; privKey ? :

BN [ ]

}) :

Promise < void

; getAuthMetadata (params :

{ serviceProvider ? :

IServiceProvider ; privKey ? :

BN ; includeLocalMetadataTransitions ? :

boolean

}) :

Promise < Metadata

;

* *

getGenericMetadataWithTransitionStates (params :

{ fromJSONConstructor :

FromJSONConstructor ; serviceProvider ? :

IServiceProvider ; privKey ? :

BN ; includeLocalMetadataTransitions ? :

boolean ; _localMetadataTransitions ? :

LocalMetadataTransitions ; }) :

Promise < unknown

;

* *

acquireWriteMetadataLock ( ) :

Promise < number

; releaseWriteMetadataLock ( ) :

Promise < void

; _syncShareMetadata (adjustScopedStore ? :

(ss :

unknown)

=>

unknown) :

Promise < void

```
        ; syncMultipleShareMetadata ( shares :
Array < BN
        , adjustScopedStore ? :
( ss :
unknown )
=>
unknown ) :
Promise < void
        ; _addRefreshMiddleware ( moduleName :
string , middleware :
( generalStore :
unknown , oldShareStores :
ShareStoreMap , newShareStores :
ShareStoreMap )
=>
unknown ) :
void ; _addReconstructKeyMiddleware ( moduleName :
string ,
middleware :
( )
=>
Promise < Array < BN
                ) :
void ; _addShareSerializationMiddleware ( serialize :
( share :
BN , type :
string )
=>
Promise < unknown
        , deserialize :
( serializedShare :
unknown , type :
string )
=>
Promise < BN
        ) :
void ; _setDeviceStorage ( storeDeviceStorage :
```

(deviceShareStore :

ShareStore)

=>

Promise < void

) :

void ;

* *

encrypt (data :

Buffer) :

Promise < EncryptedMessage

; decrypt (encryptedMessage :

EncryptedMessage) :

Promise < Buffer

; _setTKeyStoreItem (moduleName :

string , data :

TkeyStoreItemType) :

Promise < void

; _deleteTKeyStoreItem (moduleName :

string , id :

string) :

Promise < void

;

* _localMetadataTransitions :

LocalMetadataTransitions ; _refreshMiddleware :

RefreshMiddlewareMap ; _reconstructKeyMiddleware :

ReconstructKeyMiddlewareMap ; _shareSerializationMiddleware :

ShareSerializationMiddleware ; }

Log Inâ

The login with the tKey SDK is a 3-step process.

1. Login using your desired login provider and get their OAuth Id Token
2. Use the OAuth ID Token to connect to the Web3Auth Service Provider to generate the OAuth Key
3. Once the OAuth Key is generated, initialize the tKey Instance

However, before starting this process, you need to set up Custom Authentication on your Web3Auth Dashboard. For this, you need toCreate a Verifier within theWeb3Auth Developer Dashboard with your desired configuration.

tip If you want to know more about setting up a verifier and how to use it, please refer to theAuth Provider Setup Documentation .

Login using OAuthâ

You can choose any OAuth Provider of your choice and login using their standard process. You can check out our examples where we have used the most common OAuth providers like Google, Auth0, Firebase etc. Additionally, you can checkout

our documentation around OAuth Provider setuphere .

Implicit Auth Flow If you prefer using an implicit login method with your OAuth Provider, you can do so by using theTorusServiceProvider . You can check out theadditional reading section for more information. This provider doesn't work in a React Native environment.

Generating OAuth Keyâ

This setup helps you generate a private key which will be needed by the tKey Instance to generate the OAuth Share. This is done by calling theconnect() function within thetKey instance'sserviceProvider .

tKeyInstance . serviceProvider . connect (params :

LoginParams) :

Promise < BN

;

LoginParams

â

- Table
- Type Declarations

Parameter Type Description Mandatory verifier string Details of the verifier (verifier type, ie.torus ,metamask ,openlogin etc.) Yes verifierId string Verifier ID's value,sub oremail value present in the idToken. Yes idToken string A newly createdJWT Token that has not already been sent to Web3Auth or aDuplicate Token error will be thrown. Yes subVerifierInfoArray? TorusSubVerifierInfo[] Sub verifier info No serverTimeOffset? number Server time offset No export

type

LoginParams

=

{ verifier :

string ; verifierId :

string ; idToken :

string ; subVerifierInfoArray ? : TorusSubVerifierInfo [ ] ; serverTimeOffset ? :

number ; } ;

Usageâ

const

OAuthShareKey

=

await (tKeyInstance . serviceProvider

as

SfaServiceProvider) . connect ({ verifier , verifierId , idToken , }) ;

Initializing tKeyâ

tKey.initialize(params?)

â

Once you have triggered the login process, you're ready to initialize the tKey. This will generate a Threshold Key corresponding to your login provider.

Parametersâ

params

â

Theinitialize function accepts the following optional parameters:

- Table
- Type Declaration

Parameter Type Description Mandatory withShare? ShareStore Initialize tkey with an existing share store. This allows you to directly initialize tKey without using the service provider login. No importKey? BN Import a key into tkey for initialisation. No neverInitializeNewKey? boolean Never initialize using a new key if the shares are already formed No transitionMetadata? Metadata If we've been provided with transition metadata we use that as the current metadata instead as we want to maintain state before and after serialization. No previouslyFetchedCloudMetadata? Metadata Pass the previous cloud metadata No previousLocalMetadataTransitions? LocalMetadataTransitions Pass the previous transition metadata No initialize (params ? :

{ withShare ? :

ShareStore ; importKey ? :

BN ; neverInitializeNewKey ? :

boolean ; transitionMetadata ? :

Metadata ; previouslyFetchedCloudMetadata ? :

Metadata ; previousLocalMetadataTransitions ? :

LocalMetadataTransitions ; delete1OutOf1 ? :

boolean ; }) :

Promise < KeyDetails

;

Exampleâ

import

{ decode as atob }

from

"base-64" ;

const loginRes =

await

signInWithEmailPassword ( ) ;

// Logging in via email and password const idToken =

await loginRes ! . user . getIdToken (true) ;

// Getting ID Token from the Login Provider const parsedToken =

parseToken (idToken) ;

// Parsing the ID Token

// function to Parse Id Token const

parseToken

=

(token :

any)

=>

{ try

{ const base64Url = token . split (".") [1] ; const base64 = base64Url . replace ("-" ,

"+") . replace ("_" ,

"/") ; return

JSON . parse (atob (base64 ||

"")) ; }

catch

(err)

{ console . log (err) ; return

null ; } } ;

console . log ("User Information:" , parsedToken) ;

// Connecting to the service provider const verifier =

"web3auth-firebase-examples" ; const verifierId = parsedToken . sub ; const

OAuthShareKey

=

await

(tKeyInstance . serviceProvider

as

SfaServiceProvider) . connect ({ verifier , verifierId , idToken , }) ;

console . log ("OAuth Share:" ,

OAuthShareKey) ;

await tKeyInstance . initialize ( ) ;

Get tKey Detailsâ

tKey.getKeyDetails()

â

The functiongetKeyDetails() returns the details of the keys present generated for the specific user. This includes the public key X & Y of the user, alongside the shares details and the threshold.

Sample Key Details Returnâ

[{ pubKey :

{ x :

"eb83edf410ac3cb479ccaa281f96d94e6188a746442be0f727cc72b596a1d17" , y :

"12625b4139805d0ae570a923a83b8a022aa06941aca3ace0bb538ffabeaf4242" , } , requiredShares :

- 4 , threshold :

2 , totalShares :

6 , shareDescriptions :

{ e8c4eb2197e06dd34f1b33ee58b496e7f6c7f9ebe929d5b63bde4db407e8c1f0 :

['{"module":"webStorage","userAgent":"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.0.0 Safari/537.36","dateAdded":1692014296326}' ,] , e1587941cbd1fe967387794f9fa026c9229fd2945386846a66aff41f6ecfefb3 :

['{"module":"securityQuestions","questions":"whats your password?","dateAdded":1692014335216}' ,] , } , } ,] ; From here, you can know whether the user key can be reconstructed or not.

- If therequiredShares
- is greater than 0, then the key cannot be reconstructed, because this means that the user has not yet generated enough
- shares to meet the threshold.
- Once therequiredShares
- is 0 or less than that, then the key can be reconstructed, and the user can use the shares to generate the private key and
- make further operations on the tKey and manipulate their keys.

Example const

{ requiredShares }

= tKey . getKeyDetails ( ) ; if

(requiredShares <=

0)

{ console . log ("All shares are present, you can reconstruct your private key and do operations on the tKey") ; }

else

{ console . log ("You need to generate more shares to reconstruct your private key") ; }

Reconstruct Private Keyâ

tKey.reconstructKey()

â

The functionreconstructKey() reconstructs the private key of the user from the shares generated. This function returns the private key of the user once the threshold has been met.

reconstructKey (_reconstructKeyMiddleware ? :

boolean) :

Promise < ReconstructedKeyResult

;

export

type

ReconstructedKeyResult

=

{ privKey :

BN ; seedPhrase ? :

BN [ ] ; allKeys ? :

BN [ ] ; } ; Example const

{ requiredShares }

= tKey . getKeyDetails ( ) ; if

(requiredShares <=

0)

{ const reconstructedKeyResult =

await tKey . reconstructKey ( ) ; const privateKey = reconstructedKeyResult ?. privKey . toString ("hex")) ; console . log ("Private Key: " , privateKey) ; }

Generate a New Shareâ

tKey.generateNewShare()

â

The functiongenerateNewShare() generates a new share on the same threshold set by the user. This function returns the new share generated.

generateNewShare ( ) :

Promise < GenerateNewShareResult

;

export

type

GenerateNewShareResult

=

{ newShareStores :

ShareStoreMap ; newShareIndex :

BN ; } ; Example const shareStore =

await tKey . generateNewShare ( ) ; await (tKey . modules . webStorage

as

WebStorageModule) . storeDeviceShare (shareStore . newShareStores [1]) ; console . log ("New Share Stored on the Browser Local Storage: " , shareStore . newShareIndex . toString ( )) ;

Delete a Shareâ

tKey.deleteShare(shareIndex: BNString)

â

The functiondeleteShare() deletes a share from the user's shares. This function returns the updated shareStore after the share has been deleted.

deleteShare (shareIndex :

BNString) :

Promise < DeleteShareResult

;

export

type

DeleteShareResult

=

{ newShareStores :

ShareStoreMap ; } ; Example const shareStore =

await tKey . deleteShare (previousShareIndex) ; console . log ("Share has been deleted" , shareStore) ;

Using Modulesâ

For making advanced operations on tKey and to manipulate the keys, you can use the modules provided by tKey. As mentioned in theinitialisation section, you need to configure the modules beforehand to make it work with tKey. Once that is done, the instance of the respective module is available within your tKey instance and can be used for further operations.

tip Checkout theModules section where we have listed all the available modules alongside the functions that can be used within them.

Making Blockchain Callsâ

Once you have generated the private key, you can use it to make blockchain calls. The key generated by tKey is of typesecp256k1 , which is compatible with EVM-based blockchains like Ethereum, Polygon, and many others that use the same curve. However, you can also convert this key into other curves and utilize it. For example, we have a dedicated package for converting this module to theed25519 curve for usage in Solana and other blockchains that use this curve.

In addition to that, we have dedicated provider packages for EVM, Solana and XRPL libraries. You can check out their respective documentation in theProviders Section .

tip You can checkout ourConnect Blockchain documentation which has a detailed guide on how to connect to major blockchains out there. Edit this page Previous Initialize Next Modules