

Using tKey iOS SDK

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

ThresholdKey

[^](#)

Natively, the instance of tKey, (i.e., 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 check out the complete list of functions.

Functions	Function Description	Arguments	Async	return
initialize	Generates a Threshold Key object corresponding to your login provider.	import_share: String, input: ShareStore?, never_initialize_new_key: Bool?		
reconstruct	Reconstructs the user private key (minimum threshold no. of shares required)	include_local_metadata_transitions: Bool? Yes		KeyDetails
reconstruct_latest_poly	Returns the latest polynomial from all the available shares (for this pub-poly). We using Lagrange's interpolation to derive the polynomial	void Yes		KeyReconstructionDetails
get_all_share_stores_for_latest_polynomial	Get all available ShareStores from latest polynomial	void No		ShareStoreArray
generate_new_share	Generate a new share for the reconstructed private key.	void Yes		GenerateShareStoreResult
delete_share	Delete a share from private key.	share_index: String	Yes	void
delete_tkey	Deletes a threshold key, all shares will be removed, use with caution	void	Yes	void
get_key_details	Get the details of the keys.	void No		KeyDetails
output_share	Output a share from the tKey	shareIndex: String, shareType: String?	No	String
share_to_share_store	Convert Share to ShareStore	share: String	No	ShareStore
input_share	Adds an existing share to tkey.	share: String, shareType: String?	Yes	void
output_share_store	Output a share store from the tKey	shareIndex: String, polyId: String?	No	ShareStore
input_share_store	Input a share store into the tKey	ShareStore	Yes	void
get_shares_indexes	Returns an array of all the share indexes from latest polynomial	void No		[String]
encrypt	Encrypt a message/data with the provided publicKey.	msg: String	No	String
decrypt	Decrypt a message/data with the provided publicKey.	msg: String	No	String
get_tkey_store	Returns data from tkey store given a module name	moduleName: String	No	[[String:Any]]
get_tkey_store_item	Returns data from tkey store given id and a module name	moduleName: String, id: String	No	String
get_shares	get shares from tKey	void No		ShareStorePolyIdIndexMap
get_share_descriptions	Get a description to a share	void No		[String: [String]]
add_share_description	Add a description to a share key:	String, description: String, update_metadata: Bool	Yes	void
update_share_description	Update a description to a share key:	String, oldDescription: String, newDescription: String, update_metadata: Bool	Yes	void
delete_share_description	Delete a description to a share key:	String, description: String, update_metadata: Bool	Yes	void

Log In

The login with the tKey SDK is a two-step process. First, you need to trigger the login process by calling the triggerLogin() function of the CustomAuth SDK. Using the returned information, use the initialize() function of the tKey to generate the Threshold Key corresponding to the user.

However, before starting this process, you must set up Custom Authentication on your Web3Auth Dashboard. You must [Create a Verifier](#) from the Custom Auth section of the [Web3Auth Developer Dashboard](#) with your desired configuration.

tip For further information on how to set up and use a verifier, please visit the [Custom Authentication Documentation](#).

Triggering Login and Initializing Service Provider

```
import
CustomAuth

let sub =
SubVerifierDetails ( loginType :
. web ,

// default .web loginProvider :

< typeOfLogin

,

// .google, clientId :

"" , verifierName :
```

```

"" , redirectURL :

"" , browserRedirectURL :

"" )

let tdsdk =

CustomAuth ( aggregateVerifierType :

"" ,

// singleLogin, singleIdVerifier supported aggregateVerifierName :

"" ,

// Web3Auth Custom verifier name subVerifierDetails :

[ sub ] , network :

. TESTNET )

tdsdk . triggerLogin ( controller :

< UIViewController

    ? , browserType :

< method - of - opening - browser

    , modalPresentationStyle :

< style - of - modal

    ) . done { data in print ( "user data" , data ) let key = data [ "privateKey" ] service_provider =

try !

ServiceProvider ( enable_logging :

true , postbox_key : key ) } . catch { err in print ( err ) } Generating a private key is an essential step for the tKey to create its
share. The triggerLogin() function of the CustomAuth is called to accomplish this.

```

SubVerifierDetails

[â](#)

Parameter	Type	Mandatory	Description
loginType	SubVerifierType	No	loginType to be used. [web : default, installed]
loginProvider	LoginProviders	Yes	loginProvider to be used. [google , facebook , twitch , reddit , discord , apple , github , linkedin , kakao , twitter , weibo , line , wechat , email_password , and jwt]
clientId	String	Yes	login provider's client Id.
verifier	String	Yes	Web3Auth verifier name
redirectURL	String	Yes	It refers to a url for the login flow to redirect into your app, it should have a scheme that is registered by your app, for example com.mycompany.myapp://redirect
browserRedirectURL	String	No	It refers to a page that the browser should use in the login flow, it should have a http or https scheme. e.g. https://scripts.toruswallet.io/redirect.html
jwtParams	String	No	Additional JWT parameters to be passed.
urlSession	URLSession	No	Custom URLSession to be used.

CustomAuth

[â](#)

Parameter	Type	Mandatory	Description
aggregateVerifierType	String	Yes	Type of the aggregate verifier.
aggregateVerifier	String	Yes	Name of the aggregate verifier.
subVerifierDetails	[SubVerifierDetails]	Yes	Array of SubVerifierDetails.
network	Network	Yes	Network to be used. [MAINNET , TESTNET , CYAN , AQUA]

Instantiate tKey

```

let thresholdKey =

try ?

ThresholdKey ( storage_layer : storage_layer , service_provider : service_provider , enable_logging :

```

true , manual_sync :

false)

Parameters

Parameter Type Description Mandatory metadata Metadata Metadata object containing the metadata details of tKey. No shares ShareStorePolyIdIndexMap Array of ShareStore with PolyId. No storage_layer StorageLayer Takes in the Storage Provider Instance No service_provider ServiceProvider Takes in the Service Provider Instance No local_metadata_transitions Metadata Local metadata transitions No enable_logging Bool This option is used to specify whether to enable logging or not. No manual_sync Bool manual sync provides atomicity to your tkey share. If manual_sync is true, you should sync your local metadata transitions manually to your storage_layer, which means your storage layer doesn't know the local changes of your tkey unless you manually sync, gives atomicity. Otherwise, If manual_sync is false, then your local metadata changes will be synced automatically to your storage layer. If manual_sync = true and want to synchronize manually, you need to call sync_local_metadata_transitions() manually. No

Example

```
guard

let postboxkey = userData [ "privateKey" ]

as ?

String

else

{ alertContent =

"Failed to get postboxkey" return }

guard

let storage_layer =

try ?

StorageLayer ( enable_logging :

true , host_url :

"https://metadata.tor.us" , server_time_offset :

2 )

else

{ alertContent =

"Failed to create storage layer" return }

guard

let service_provider =

try ?

ServiceProvider ( enable_logging :

true , postbox_key : postboxkey )

else

{ alertContent =

"Failed to create service provider" return }

guard

let thresholdKey =

try ?
```

```

ThresholdKey ( storage_layer : storage_layer , service_provider : service_provider , enable_logging :
true , manual_sync :
false )
else
{ alertContent =
"Failed to create threshold key" return }

```

threshold_key

thresholdKey

Initialize tKey

threshold_key.initialize(params?)

[^](#)

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.

Parameters

Parameter Type Description Mandatory import_share String An optional string representing the import share. No input ShareStore An optional ShareStore object representing the input. No never_initialize_new_key Bool A boolean value indicating whether or not to initialize a new key. No include_local_metadata_transitions Bool A boolean value indicating whether or not to include local metadata transitions. No

Example

```

guard
let key_details =
try ?
await threshold_key . initialize ( )
else
{ alertContent =
"Failed to get key details" return }

```

Get tKey Details

```
let key_details = try! threshold_key.get_key_details()
```

The `get_key_details()` function provides information about the keys created for a particular user. It includes the user's public key X and Y, as well as the share descriptions, number of required shares, total shares, and threshold.

Usage Sample

```

let key_details =
try ! threshold_key . get_key_details ( )
// Returns a KeyDetails object.
// To get the public key let pub_key = key_details . pub_key // Returns a KeyPoint object
// For key x and y, or serialized representation let x_coord =
try ! pub_key . getX ( ) let y_coord =

```

```

try ! pub_key . getY ( ) let serialized =
try ! pub_key . getAsCompressedPublicKey ( format :
"elliptic-compressed" )
//Required shares key_details . required_shares
// Threshold key_details . threshold
// Total Shares key_details . total_shares

// Share Descriptions key_details . share_descriptions // This is a json object in string format From here, you can know
whether the user key can be reconstructed or not.

• If the value ofrequired_shares
• is more than zero, it implies that the threshold hasn't been met yet, and as a result, the key can't be
• reconstructed since the user hasn't generated enough shares.
• When the value ofrequired_shares
• is 0 or less, the user can reconstruct the key. They can then use the shares to generate their private key and
• carry out additional operations on the tKey to manage their keys.

```

Reconstruct Private Key

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

```

let reconstructedKeyResult =
try !
await threshold_key . reconstruct ( )
public
final
class
KeyReconstructionDetails :
Codable
{ public
var key :
String public
var seed_phrase :
[ String ] public
var all_keys :
[ String ] } Example guard
let key_details =
try ?
await threshold_key . initialize ( never_initialize_new_key :
false )
else
{ alertContent =
"Failed to get key details" return }
guard

```

```

let reconstructionDetails =
try ?
await threshold_key . reconstruct ( )
else
{ alertContent =
"Failed to reconstruct key. ( threshold ) more share(s) required" resetAccount =
true return }

```

Generate a New Share[â](#)

The function `generate_new_share()` generates a new share on the same threshold (e.g, 2/3 -> 2/4). This function returns the new share generated.

```

let newShare =
try !
await threshold_key . generate_new_share ( )

```

Delete a Share[â](#)

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

```

let shareStore =
try !
await threshold_key . delete_share ( share_index : idx )

```

Using Modules for Further Operations[â](#)

To perform advanced operations and manipulate keys, `tKey` offers modules that can be utilized. As previously stated in the [initialization](#) section, modules need to be configured beforehand to function properly with `tKey`. After configuration, the respective module's instance is accessible within your `tKey` instance and can be utilized for additional operations.

Modules Please visit the [Modules](#) section to view a comprehensive list of available modules and their respective functions.

Consider multiple device environment[â](#)

Imagine a situation where a user wants to use the same private key on multiple devices using the Tkey SDK.

Basically, you need at least 2 shares to reconstruct a tkey. If you initialized tkey on device A, you need the 2 shares (social login share, device share) obtained through initialization on device A'. This can be accomplished by transferring the device share from device A to device A'. (by using share transfer module)

You can try like this :

1. Serialize a share created on device A and import it from device A' to reconstruct it.
2. Use share Transfer Module

Here's an example of transferring a share using `shareTransfer` module.

```

// assume that threshold_key, threshold_key2 are independent tkeys initialized on each device // initialized with the same
value of service provider and storage layer // 1. request new share from second device let request_enc =

```

```

try !

```

```

await

```

```

ShareTransferModule . request_new_share ( threshold_key : threshold_key2 , user_agent :

```

```

"agent" , available_share_indexes :

```

```

"[]" ) // 2. generate new share and approve the request from existing device let lookup =
try !
await
ShareTransferModule . look_for_request ( threshold_key : threshold_key ) let encPubKey = lookup [ 0 ] // generate a new
share let newShare =
try !
await threshold_key . generate_new_share ( ) // approve the corresponding share try !
await
ShareTransferModule . approve_request_with_share_index ( threshold_key : threshold_key , enc_pub_key_x : encPubKey ,
share_index : newShare . hex ) // 3. check the request status and reconstruct when it succeeds _
=
try !
await
ShareTransferModule . add_custom_info_to_request ( threshold_key : threshold_key2 , enc_pub_key_x : request_enc ,
custom_info :
"test info" ) _
=
try !
await
ShareTransferModule . request_status_check ( threshold_key : threshold_key2 , enc_pub_key_x : request_enc ,
delete_request_on_completion :
true ) let k2 =
try !
await threshold_key2 . reconstruct ( ) Alternatively, you can create one additional share (backup share, security question
share, etc) and utilize it on device A'.

```

Below is an example guide of leveraging security question module :

1. Initialize the tkey on Device A. (2/2 shares are needed)
2. Create an extra share using the Security question module and reconstruct it. (2/3)
3. Recover the final key from Device A' with the social login share and security question share.
4. Save the security question share locally. If you set up the device share like this, you don't need to ask the security question every time you log
5. in.

Creating an additional share also makes it easier for account management, as you can recover your account if you lose your device share.

Making Blockchain Calls

After generating your private key with tKey, you can use it to make blockchain calls on EVM-based blockchains like Ethereum, Polygon and other EVM Chains. The key is of thesecp256k1 type, which is compatible with EVM blockchains. Additionally, you can convert this key into other curves if needed.

Connect Blockchain Our [Connect Blockchain](#) documentation provides a comprehensive guide on how to connect to major blockchains. Feel free to check it out. [Edit this page](#) [Previous](#) [Initialize](#) [Next](#) [Modules](#)