Session Module

As described in the last section, an off-chain storage solution is essential for the session leafs. In this context, we will employ Files for storing this data. Any customized storage layer must adhere to a specific interface, ensuring the correct implementation of methods for saving or retrieving information. Alternatively, one can develop their own implementation using a Database server layer. This guide will lead you through the process of creating a fundamental Node.js script using TypeScript that enables users to initiate a session..

This tutorial has a previous other steps in the previous sections:Environment Setup andInitializing Account This tutorial will be done on the Polygon Mumbai Network. We will be using session Module for this.

We will import sessionKeyManagerModule and DEFAULT_SESSION_KEY_MANAGER_MODULE from Biconomy Modules package.

First of all we will initialise the sessionFileStorage using our custom File storage

const sessionFileStorage : SessionFileStorage =

new

SessionFileStorage (address) ; We store the private key in the file using following method. This saves the signer in the file.

const sessionSigner = ethers . Wallet . createRandom ( ) ; const sessionKeyEOA =

await sessionSigner . getAddress ( ) ; await sessionFileStorage . addSigner (sessionSigner) ; The Session Key Manager module is responsible for overseeing the storage of session leaf data generated by any type of session key. It utilizes a storage client to keep track of all leaves, enabling the generation of a Merkle proof for a specific leaf. Module provided by the SDK gives you easy way to

- create session key generation data
- manage leaf information for merkle proof generation
- gain dummy signature for userop estimation
- generate userop.signature for transaction that utilises session key

const sessionModule =

await

createSessionKeyManagerModule ({ moduleAddress :

DEFAULT_SESSION_KEY_MANAGER_MODULE , smartAccountAddress : address , sessionStorageClient : sessionFileStorage }) ; now, we will use a deployed contract that validates specific permissions to execute ERC20 token transfers. ERC20 Session Validation Module is one kind of Session Validation Module. Using this ERC20 Validation module you will be able to create a dApp that allows user to send a limited amount of funds to a specific address without needing to sign a transaction every single time. In the following example user can only transfer 50 tokens at once. Individuals have the flexibility to create their own Session Validation Module (SVM), deploy it, utilize its address as the sessionValidationModule, and generate sessionKeyData based on how permissions are configured.

const sessionKeyData = defaultAbiCoder . encode (["address" ,

"address" ,

"address" ,

"uint256"] , [sessionKeyEOA , "0xdA5289fCAAF71d52a80A254da614a192b693e977" ,

// erc20 token address "0x322Af0da66D00be980C7aa006377FCaaEee3BDFD" ,

// receiver address ethers . utils . parseUnits ("50" . toString ( ) ,

6) . toHexString ( ) ,

// 50 usdc amount] ,) ; next, we create the session data. We specify how long this should be valid until or how long it is valid after. This should be a unix timestamp to represent the time. Passing 0 on both makes this session never expire, do not do this in production. Next we pass the session validation module address, session public key, and the session key data we just created.

const sessionTxData =

await sessionModule . createSessionData ([{ validUntil :

0 , validAfter :

0 , sessionValidationModule : erc20ModuleAddr , sessionPublicKey : sessionKeyEOA , sessionKeyData : sessionKeyData , } ,]) ; We're going to be tracking if the session key module is already enabled. If we need to enable session key module, we create a transaction using the getEnableModuleData and pass the session key manager module address and push this to the array.

const isEnabled =

await smartAccount . isModuleEnabled (DEFAULT_SESSION_KEY_MANAGER_MODULE ,) ; if

(! isEnabled)

{ const enableModuleTrx =

await smartAccount . getEnableModuleData (DEFAULT_SESSION_KEY_MANAGER_MODULE ,) ; transactionArray . push (enableModuleTrx) ; } We initialise a transaction array and push the create session Transaction. These give (to, value, data) as traditional transaction. Subsequently, we invoke the buildUserOp function to create a user operation for either an array of transactions or a single transaction.

const transactionArray =

[ ] ; const setSessiontrx =

{ to :

DEFAULT_SESSION_KEY_MANAGER_MODULE , data : sessionTxData . data , } ; transactionArray . push (setSessiontrx) ; Next we will send the transactions which will get wrapped in a user op and sent to the Bundler. Ensure that a paymaster is setup and the corresponding gas tank has sufficient funds to sponsor the transactions. Also enable the session validation module address in the policy section for the paymaster.

let userOpResponse =

await smartAccount . sendTransaction (transactionArray ,

{ paymasterServiceData :

{ mode : PaymasterMode . SPONSORED , } , }) ;

console . log (userOp Hash: { userOpResponse . userOpHash }) ;

const transactionDetails =

await userOpResponse . wait ( ) ; console . log ("txHash" , transactionDetails . receipt . transactionHash) ; Once this transaction is successful, a session gets enabled on chain. Anyone holding the session private key can make use of the session (till te time session is valid). Now with this implemented, let's take a look at executing the ERC20 token transfer with this session in the next section.

Checkout below for entire code snippet

Expand for Code import

{ defaultAbiCoder }

from

"ethers/lib/utils" ; import

{ config }

from

"dotenv" ; import

{ createSmartAccountClient , DEFAULT_SESSION_KEY_MANAGER_MODULE , createSessionKeyManagerModule , BiconomySmartAccountV2 }

from

"@biconomy/account" ; import

{ Wallet , providers , ethers }

```
from

"ethers" ; import

{ SessionFileStorage }

from

"./customSession" ;

let smartAccount : BiconomySmartAccountV2 ; let address :

string ;

config ( ) ;

const provider =

new

providers . JsonRpcProvider ( "https://rpc.ankr.com/polygon_mumbai" , ) ; const wallet =

new

Wallet ( process . env . PRIVATE_KEY

||

"" , provider ) ;

async

function

createAccount ( )

{ let biconomySmartAccount =

await

createSmartAccountClient ( { signer : wallet , bundlerUrl , biconomyPaymasterApiKey : paymasterApiKey , } ) ; address =

await biconomySmartAccount . getAccountAddress ( ) ; console . log ( address ) ; smartAccount = biconomySmartAccount ;

return biconomySmartAccount ; }

const

createSession

=

async

( )

=>

{ await

createAccount ( ) ; try

{ const erc20ModuleAddr =

"0x000000D50C68705bd6897B2d17c7de32FB519fDA" ; // -----> setMerkle tree tx flow // create dapp side session key const sessionSigner = ethers . Wallet . createRandom ( ) ; const sessionKeyEOA =

await sessionSigner . getAddress ( ) ; console . log ( "sessionKeyEOA" , sessionKeyEOA ) ; const sessionFileStorage : SessionFileStorage =

new

SessionFileStorage ( address , ) ;
```

```
// generate sessionModule console . log ( "Adding session signer" , sessionSigner . publicKey , sessionSigner , ) ;

await sessionFileStorage . addSigner ( sessionSigner ) ; const sessionModule =

await SessionKeyManagerModule . create ( { moduleAddress :

DEFAULT_SESSION_KEY_MANAGER_MODULE , smartAccountAddress : address , sessionStorageClient :
sessionFileStorage , } ) ;

// cretae session key data const sessionKeyData = defaultAbiCoder . encode ( [ "address" ,

"address" ,

"address" ,

"uint256" ] , [ sessionKeyEOA , "0xdA5289fCAAF71d52a80A254da614a192b693e977" ,

// erc20 token address "0x322Af0da66D00be980C7aa006377FCaaEee3BDFD" ,

// receiver address ethers . utils . parseUnits ( "50" . toString ( ) ,

6 ) . toHexString ( ) ,

// 50 usdc amount ] , ) ; const sessionTxData =

await sessionModule . createSessionData ( [ { validUntil :

0 , validAfter :

0 , sessionValidationModule : erc20ModuleAddr , sessionPublicKey : sessionKeyEOA , sessionKeyData : sessionKeyData ,
} , ] ) ;

// tx to set session key const setSessiontrx =

{ to :

DEFAULT_SESSION_KEY_MANAGER_MODULE ,

// session manager module address data : sessionTxData . data , } ;

const transactionArray =

[ ] ;

const isEnabled =

await smartAccount . isModuleEnabled ( DEFAULT_SESSION_KEY_MANAGER_MODULE , ) ; if

( ! isEnabled )

{ const enableModuleTrx =

await smartAccount . getEnableModuleData ( DEFAULT_SESSION_KEY_MANAGER_MODULE , ) ; transactionArray .
push ( enableModuleTrx ) ; }

transactionArray . push ( setSessiontrx ) ; let userOpResponse =

await smartAccount . sendTransaction ( transactionArray ,

{ paymasterServiceData :

{ mode : PaymasterMode . SPONSORED , } , } ) ; console . log ( `userOp Hash: ${ userOpResponse . userOpHash } ` ) ; const
transactionDetails =

await userOpResponse . wait ( ) ; console . log ( "txHash" , transactionDetails . receipt . transactionHash ) ; }

catch

( err :

any )

{ console . error ( err ) ; } } ;
```

createSession ( ) ;Previous Create Custom session Next ERC20 transfer