Create Session Router

If you wish to know more about Batched Session Validation Module gohere .

info One crucial aspect to grasp about the session module is its limitation on sending batches of various user operations. Specifically, it does not support sending multiple ERC20 Transfers with different recipients within the same batch, nor does it allow combining an ERC20 transfer with an NFT Mint operation. To overcome this constraint, we rely on the Batched Session Router Module.

Imports

Let's add a few more imports.

import

React ,

{ useEffect , useState }

from

"react" ; import

{ ethers }

from

"ethers" ; import

{

BiconomySmartAccountV2 , createSmartAccountClient ,

SessionKeyManagerModule ,

DEFAULT_SESSION_KEY_MANAGER_MODULE , createSessionKeyManagerModule , createBatchedSessionRouterModule ,

DEFAULT_BATCHED_SESSION_ROUTER_MODULE

}

from

"@biconomy/account" import

{ defaultAbiCoder }

from

"ethers/lib/utils" ; import

{ toast ,

ToastContainer

}

from

'react-toastify' ; import

'react-toastify/dist/ReactToastify.css' ; We addedcreateBatchedSessionRouterModule andDEFAULT_BATCHED_SESSION_ROUTER_MODULE .

Add one moreuseState for batched sesion module.

Check if both Session Validation and Batched Session Validation modules are enabled.

const

[isSessionKeyModuleEnabled , setIsSessionKeyModuleEnabled]

=

useState (false) ; const

[isSessionActive , setIsSessionActive]

=

useState (false) ; const

[isBRMenabled , setIsBRMenabled]

=

useState (false) ; Now we also track if batched session module is enabled or not.

YouruseEffect should now look like this:

useEffect (( )

=>

{ let

checkSessionModuleEnabled

=

async

( )

=>

{ if (! address ||

! smartAccount ||

! provider)

{ setIsSessionKeyModuleEnabled (false) ; return } try

{ const isEnabled =

await smartAccount . isModuleEnabled (DEFAULT_SESSION_KEY_MANAGER_MODULE) console . log ("isSessionKeyModuleEnabled" , isEnabled) ; setIsSessionKeyModuleEnabled (isEnabled) ; }

catch (err : any)

{ console . error (err) setIsSessionKeyModuleEnabled (false) ; return ; } try

{ const isEnabled =

await smartAccount . isModuleEnabled (DEFAULT_BATCHED_SESSION_ROUTER_MODULE) console . log ("isSessionKeyRouterModuleEnabled" , isEnabled) ; setIsBRMenabled (isEnabled) ; }

catch (err : any)

{ console . error (err) setIsBRMenabled (false) ; return ; } return ; } checkSessionModuleEnabled ( ) } , [isSessionKeyModuleEnabled , isBRMenabled , address , smartAccount , provider]) This will check if a session module AND a batched session module is enabled - it will return if we do not have an address, smart Account, or provider and will then enable the module for our smartAccount if needed.

Create and add the Batched Session Module

Inside ourcreateSession function we will add a few extra lines of code to create and enable theBatched Session Router module

const

```
createSession

=

async

( enableSessionKeyModule : boolean )

=>

{ toast . info ( 'Creating Session...' ,

{ position :

"top-right" , autoClose :

15000 , hideProgressBar :

false , closeOnClick :

true , pauseOnHover :

true , draggable :

true , progress :

undefined , theme :

"dark" , } ) ; if

( ! address ||

! smartAccount ||

! provider )

{ alert ( "Please connect wallet first" ) } try

{ // module that allows only ERC20 Transfer const erc20ModuleAddr =

"0x000000D50C68705bd6897B2d17c7de32FB519fDA" // module used just for testing, allows executing any operation not
just ERC20 transfers const mockSessionModuleAddr =

"0x7Ba4a7338D7A90dfA465cF975Cc6691812C3772E" ; // -----> setMerkle tree tx flow // create dapp side session key const
sessionSigner = ethers . Wallet . createRandom ( ) ; const sessionKeyEOA =

await sessionSigner . getAddress ( ) ; console . log ( "sessionKeyEOA" , sessionKeyEOA ) ; // BREWARE JUST FOR
DEMO: update local storage with session key window . localStorage . setItem ( "sessionPKey" , sessionSigner . privateKey )
;

// generate sessionModule const sessionModule =

await

createSessionKeyManagerModule ( { moduleAddress :

DEFAULT_SESSION_KEY_MANAGER_MODULE , smartAccountAddress : address , } ) ;

// genereate batched session const sessionRouterModule =

await

createBatchedSessionRouterModule ( { moduleAddress :

DEFAULT_BATCHED_SESSION_ROUTER_MODULE , sessionKeyManagerModule : sessionModule ,
smartAccountAddress : address , } ) ;

// create 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 ] ) ;

// create batched session data // This data will allow us to execute any operation on mockSessionModule as long as it
comes from sessionKeyEOA const sessionKeyData2 = defaultAbiCoder . encode ( [ "address" ] , [ sessionKeyEOA ] ) ;

const sessionTxData =

await sessionModule . createSessionData ( [ { validUntil :

0 , validAfter :

0 , sessionValidationModule : erc20ModuleAddr , sessionPublicKey : sessionKeyEOA as

Hex , , sessionKeyData : sessionKeyData as

Hex , , } , // Include the second session data which will be used on the batched session router { validUntil :

0 , validAfter :

0 , sessionValidationModule : mockSessionModuleAddr , sessionPublicKey : sessionKeyEOA as

Hex , sessionKeyData : sessionKeyData2 as

Hex , } , ] ) ;

// tx to set session key const setSessiontrx =

{ to :

DEFAULT_SESSION_KEY_MANAGER_MODULE ,

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

const transactionArray =

[ ] ;

if

( enableSessionKeyModule )

{ // -----> enableModule session manager module const enableModuleTrx =

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

if

( ! isBRMenabled )

{ // -----> enableModule batched session router module const enableModuleTrx =

await biconomySmartAccount . getEnableModuleData ( DEFAULT_BATCHED_SESSION_ROUTER_MODULE ) ;
transactionArray . push ( enableModuleTrx ) ; }

transactionArray . push ( setSessiontrx )

let userOpResponse =

await smartAccount . sendTransaction ( transactionArray ) ;

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

await userOpResponse . wait ( ) ; console . log ( "txHash" , transactionDetails . receipt . transactionHash ) ;
setIsSessionActive ( true ) toast . success ( Success! Session created succesfully  ,

{ position :
```

```
"top-right" , autoClose :

18000 , hideProgressBar :

false , closeOnClick :

true , pauseOnHover :

true , draggable :

true , progress :

undefined , theme :

"dark" , } ) ; }

catch ( err : any )

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

Mock Session Module

The mockSessionModule referenced in the code snippet above is a testing module used to demonstrate session functionality. It allows executing any operation, not just ERC20 transfers, facilitating thorough testing scenarios. However, it's essential to emphasize that this module should only be utilized during development and testing phases.

Usage of Mock Session Module

During development and testing, developers can use the mockSessionModule to simulate various transaction scenarios and ensure the robustness of their applications. By leveraging this module, developers can explore a wide range of functionalities without constraints, aiding in the debugging and refinement of their smart contract interactions.

Production Considerations

In a production environment, it's imperative to build and deploy custom session modules tailored to the specific requirements of the application. Custom modules provide greater control and security over transaction validation and execution, ensuring adherence to business logic and regulatory compliance.

By creating session modules, developers can optimize performance, enhance security, and meet the unique demands of their decentralized applications (dApps) effectively. Therefore, while the mock session module facilitates testing and experimentation, it should be replaced with custom-built modules before deploying applications to production environments
Previous Batched Session Tutorial Next Batch ERC20 Transfer with Session Router