ERC20 transfer

We will need the bundler however the paymaster is optional in this case. If you want to sponsor transactions later or allow users to pay for gas in erc20 tokens make sure to set this up but for the purposes of this tutorial we will not be sponsoring any transactions. The paymaster URL here will only work for tutorial contracts.

First we will get the smartAccount. This is similar to the initialize account. We specify the erc20 module address and get the private key we stored in our storage and create a new session signer from it.

const erc20ModuleAddr =

"0x000000D50C68705bd6897B2d17c7de32FB519fDA" ; const sessionKeyPrivKey = sessionFileStorage . getSignerBySession ({ }) ; if

(! sessionKeyPrivKey)

{ console . log ("Session key not found please create session") ; return ; } const sessionSigner =

new

ethers . Wallet (sessionKeyPrivKey) ; console . log ("sessionSigner" , sessionSigner) ; Now we'll generate a session module using the Session Key Manager Module and then set the active validation module to be the session module. This updates the original configureation on the smart account.

// generate sessionModule const sessionModule =

await

createSessionKeyManagerModule ({ moduleAddress :

DEFAULT_SESSION_KEY_MANAGER_MODULE , smartAccountAddress : address , }) ;

// set active module to sessionModule smartAccount = smartAccount . setActiveValidationModule (sessionModule) ; We now create an instance of the contract. Note that USDC does not have 18 decimals so we update the decimals based on the USDC contract.

const tokenContract =

new

ethers . Contract (// polygon mumbai usdc address "0xdA5289fCAAF71d52a80A254da614a192b693e977" , usdcAbi , provider) ; let decimals =

18 ;

try

{ decimals =

await tokenContract . decimals ( ) ; }

catch

(error)

{ throw

new

Error ("invalid token address supplied") ; } Now we will get raw transaction data for a transfer of 1 usdc to the receiver address we specified. Using any other reciever other than the one registered on the session key will result in an error. We are sending 1 USDC in this case but can send up to 50 with this transaction as that is the maximum amount that we specified.

const

{ data }

=

await tokenContract . populateTransaction . transfer ("0x322Af0da66D00be980C7aa006377FCaaEee3BDFD" ,

// receiver address ethers . utils . parseUnits ("1" . toString ( ) , decimals)) ;

const tx1 =

{ to :

"0xdA5289fCAAF71d52a80A254da614a192b693e977" ,

//erc20 token address data : data , value :

"0" , } ; Now we send the transaction to be wrapped in a user op and then execute it. Note the additional arguments you can add in thesendTransaction method such as overrides if needed, and most importantly params object that will contain information about the session signer and session validation module.

let userOp =

await smartAccount . sendTransaction (tx1 ,

{ overrides :

{ } , params :

{ sessionSigner : sessionSigner , sessionValidationModule : erc20ModuleAddr , } , }) ; Finally to give the user a succesful feedback we provide them with a link to the transaction once it has been executed.

Running this code should now allow you to sign in using your EOA, create a session, and then send USDC without the need to sign any further transactions!

Checkout below for entire code snippet

Expand for Code import usdcAbi from

"./usdcabi.json" ; import

{ config }

from

"dotenv" ; import

{ createSmartAccountClient , DEFAULT_SESSION_KEY_MANAGER_MODULE , createSessionKeyManagerModule , }

from

"@biconomy/account" ; import

{ Wallet , providers , ethers }

from

"ethers" ;

import

{ SessionFileStorage }

from

"./customSession" ; config ( ) ;

let smartAccount : BiconomySmartAccountV2 ; let address :

string ;

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
erc20Transfer
=
async
( sessionFileStorage : SessionFileStorage , amount :
string )
=>
{ if
( ! address ||
! smartAccount )
{ console . log ( "Please connect wallet first" ) ; return ; } try
{ const erc20ModuleAddr =
"0x000000D50C68705bd6897B2d17c7de32FB519fDA" ; // get session key from file storage const sessionKeyPrivKey =
await sessionFileStorage . getSignerBySession ( { } ) ; console . log ( "sessionKeyPrivKey" , sessionKeyPrivKey ) ; if
( ! sessionKeyPrivKey )
{ console . log ( "Session key not found please create session" ) ; return ; } //@ts-ignore const sessionSigner =
new
ethers . Wallet ( sessionKeyPrivKey ) ; console . log ( "sessionSigner" , sessionSigner ) ; // generate sessionModule const
sessionModule =
await SessionKeyManagerModule . create ( { moduleAddress :
DEFAULT_SESSION_KEY_MANAGER_MODULE , smartAccountAddress : address , sessionStorageClient :
sessionFileStorage , } ) ; // set active module to sessionModule smartAccount = smartAccount . setActiveValidationModule (
sessionModule ) ;
const tokenContract =
new
ethers . Contract ( // polygon mumbai usdc address "0xdA5289fCAAF71d52a80A254da614a192b693e977" , usdcAbi ,
provider ) ; let decimals =
18 ;
try
{ decimals =

await tokenContract . decimals ( ) ; }

catch

( error )

{ throw

new

Error ( "invalid token address supplied" ) ; } const

{ data }

=

await tokenContract . populateTransaction . transfer ( "0x322Af0da66D00be980C7aa006377FCaaEee3BDFD" ,

// receiver address ethers . utils . parseUnits ( amount , decimals ) ) ;

// generate tx data to erc20 transfer const tx1 =

{ to :

"0xdA5289fCAAF71d52a80A254da614a192b693e977" ,

//erc20 token address data : data , value :

"0" , } ;

// build user op // const ifModuleEnabled = await smartAccount.isModuleEnabled("") let userOpResponse =

await smartAccount . sendTransaction ( tx1 ,

{ params :

{ sessionSigner : sessionSigner , sessionValidationModule : erc20ModuleAddr , } , } ) ; console . log ( "userOpHash" ,
userOpResponse ) ; const

{ receipt }

=

await userOpResponse . wait ( 1 ) ; console . log ( "txHash" , receipt . transactionHash ) ; const polygonScanlink =

https://mumbai.polygonscan.com/tx/ { receipt . transactionHash }  ; }

catch

( err :

any )

{ console . error ( err ) ; } } ;

async

function

executeTransaction ( )

{ await

createAccount ( ) ; const sessionFileStorage : SessionFileStorage =

new

SessionFileStorage ( address ) ; await

erc20Transfer ( sessionFileStorage ,

"0.019" ) ; await
```

erc20Transfer (sessionFileStorage ,

"0.018") ; }

executeTransaction ( ) ; If you would like to see the completed project on custom session storage, do checkout:https://github.com/bcnmy/custom-session-storage-tutorial Previous Session Module Next Custom Validation Module