Create Session

info In this instance we are not sponsoring the transaction please ensure that you send mumbai matic to your smart account to pay for gas. You can get matic at any polygon faucet. You will not be able to execute this code unless your smart account has some mumbai matic. Let's set up the component we need to create our session!

Setup Component

In thesrc folder create a new folder calledcomponents .

Create a new file calledCreateSession.tsx

Imports and Prop types

Let's add our imports to this file and create an interface for our props.

import React ,

{ useEffect , useState }

from

"react" ; import

{ ethers }

from

"ethers" ; import

{ BiconomySmartAccountV2 ,

DEFAULT_SESSION_KEY_MANAGER_MODULE , createSessionKeyManagerModule }

from

"@biconomy/account" import

{ hexDataSlice , id , parseEther }

from

"ethers/lib/utils" ; import

{ toast , ToastContainer }

from

'react-toastify' ; import

'react-toastify/dist/ReactToastify.css' ;

interface

props

{ smartAccount : BiconomySmartAccountV2 ; address :

string ; provider : ethers . providers . Provider ; } Our component props will take the smart account instancs, the address, and provider.

The initial component should look like this:

const CreateSession : React . FC < props

=

({ smartAccount , address , provider })

=>

{

return

(< div

< / div

) }

export

default CreateSession ; You can go ahead an import it into your index.tsx at this point as well and add it before the closing main tag like this:

{ smartAccount && provider &&

(< CreateSession smartAccount = { smartAccount } address = { address } provider = { provider } /

) ; } We only want it to render when the smartAccount and provider are both available.

Continuing with the session key component let's create our state variables:

const

[isSessionActive , setIsSessionActive]

=

useState (false) ; const

[isSessionKeyModuleEnabled , setIsSessionKeyModuleEnabled]

= useState (false) ;

Check if Session is enabled

We're going to be tracking if the session key module is enabled and if there is an active session.

In auseEffect hook let's write the following code:

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) ; return ; }

catch (err :

any)

{ console . error (err) setIsSessionKeyModuleEnabled (false) ; return ; } } checkSessionModuleEnabled ( ) } , [isSessionKeyModuleEnabled , address , smartAccount , provider]) This will check if a 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.

Add Utility Method

Add getABISVMSessionKeyData utility method in your project and declare interfaces

// Make sure you have these imports import

{ BigNumber }

from

"ethers" ; import

{ BytesLike , hexConcat , hexZeroPad , hexlify }

from

"ethers/lib/utils" ;

export

interface

Rule

{ offset :

number ; condition :

number ; referenceValue :

string

| BytesLike ; }

export

interface

Permission

{ destContract :

string ; functionSelector :

string ; valueLimit : BigNumber ; rules : Rule [ ] ; }

async

function

getABISVMSessionKeyData (sessionKey :

string , permission : Permission ,) :

Promise < string

{ let sessionKeyData =

hexConcat ([sessionKey , permission . destContract , permission . functionSelector , hexZeroPad (permission . valueLimit . toHexString ( ) ,

16) , hexZeroPad (hexlify (permission . rules . length) ,

2) ,

// this can't be more 2**11 (see below), so uint16 (2 bytes) is enough]) ;

for

(let i =

0 ; i < permission . rules . length ; i ++)

{ sessionKeyData =

hexConcat ([sessionKeyData , hexZeroPad (hexlify (permission . rules [i] . offset) ,

2) ,

// offset is uint16, so there can't be more than 2**16/32 args = 211 hexZeroPad (hexlify (permission . rules [i] . condition) ,

1) ,

// uint8 permission . rules [i] . referenceValue ,]) ; } return sessionKeyData ; }

Creating Session

note Replace address : Make sure to replace the recipient address value (current 0xAddress). Now let's set up the function for creating the session:

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

{ // Address of ABI Session Validation Module const abiSVMAddress =

"0x000006bC2eCdAe38113929293d241Cf252D91861" // -----> 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 , } ) ;
const nftAddress =
"0xdd526eba63ef200ed95f0f0fb8993fe3e20a23d0" ; const recipient =
"0xAddress" ; // get only first 4 bytes for function selector const functionSelector =
hexDataSlice ( id ( "safeMint(address)" ) ,
0 ,
4 ) ;
// create session key data const sessionKeyData =
await
getABISVMSessionKeyData ( sessionKeyEOA ,
{ destContract : nftAddress ,
// destination contract to call functionSelector : functionSelector ,
// function selector allowed valueLimit :
parseEther ( "0" ) ,
// no native value is sent // In rules, we make sure that referenceValue is equal to recipient rules :
[ { offset :
0 ,
// defines the position of a 32bytes word in the calldata (our recipient address) condition :
0 ,
// 0 = Condition.EQUAL referenceValue : ethers . utils . hexZeroPad ( recipient ,
32 )
// recipient address } , ] , } ) ;
const sessionTxData =
await sessionModule . createSessionData ( [ { validUntil :
0 , validAfter :
0 , sessionValidationModule : abiSVMAddress , sessionPublicKey : sessionKeyEOA as
0x { string } , sessionKeyData : sessionKeyData as
0x { string } , } , ] ) ; console . log ( "sessionTxData" , sessionTxData ) ;
// 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) ; }

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) } } Let's break down this function:

We're using the react-toastify package for a little bit of UX feedback to the user letting them know that a session is being created and doing our checks to make sure that address, smart account, and provider are defined.

Next we specify the erc20ModuleAddr - this is the deployed contract we went over in the second part of this tutorial. Next we use the ethers npm package to generate a random wallet and get its address - this is the session key EOA which will have permission to sign on our behalf so we won't have to in the future.

// BREWARE JUST FOR DEMO: update local storage with session key window . localStorage . setItem ("sessionPKey" , sessionSigner . privateKey) ; Now just for the sake of this demo we will save the private key of the session to local storage. You will need to consider the trade offs of doing this in different ways. Saving to local storage leaves the responsibility to the user to not install any malicious packages or extensions that could read the private key from storage. If you encrypt and save it to a database you will have access to the private key. You will need to consider what the best course of action is for your use case.

// generate sessionModule const sessionModule =

await

createSessionKeyManagerModule ({ moduleAddress :

DEFAULT_SESSION_KEY_MANAGER_MODULE , smartAccountAddress : address , }) ; We will now use the Session Key Manager Module to create a session module using the module address and smart account address. This is an important relationship to establish - the module provided by the SDK gives you an easy way to interact with modules you write on a smart contract with whatever arbitrary validation logic you need.

// create session key data const sessionKeyData =

await

getABISVMSessionKeyData (sessionKeyEOA ,

{ destContract : nftAddress ,

// destination contract to call functionSelector : functionSelector ,

// function selector allowed valueLimit :

parseEther ("0") ,

// no native value is sent // This rule ensures that the address we are going to mint NFT to should be equal the reference address (receiver var). rules :

[{ offset :

0 ,

// defines the position of a 32bytes word in the calldata (our recipient address) condition :

0 ,

// 0 = Condition.EQUAL referenceValue : ethers . utils . hexZeroPad (recipient ,

32)

// recipient address } ,] , }) ; Next we create the session key data - if you recall from the smart contract it needs four pieces of data - the session key address, the destination contract address (in our case, the NFT contract address),a function selector, the value limit and the rules.

Session Key - address, returned to thevalidateSessionUserOp or to the Session Key Manager to validate that the signer of the userOp is one of the enabled Session Keys.

Permitted destination contract - address of the contract that Smart Account is allowed to call within theexecute orexecuteBatch execution.

Permitted selector - the selector of the method that is allowed to be called on the Permitted destination contract

Permitted value limit - maximum value in chain native tokens that is allowed to be transferred along with the call

const sessionTxData =

await sessionModule . createSessionData ([{ validUntil :

0 , validAfter :

0 , sessionValidationModule : abiSVMAddress , sessionPublicKey : sessionKeyEOA as

0x { string } , sessionKeyData : sessionKeyData as

0x { string } , } ,]) ; Next, we create the data for the transaction that enables the sesson. For every session we need to specify how long it should be valid until and/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 that is going to validate userOps that leverage this session
- Session key address, that should sign the userOp
- And the session key data we just created.

// tx to set session key const setSessiontrx =

{ to :

DEFAULT_SESSION_KEY_MANAGER_MODULE ,

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

const transactionArray =

[ ] ; Now we construct the transaction to actually set the session key and create an array where our transactions will be placed.

if

(enableSessionKeyModule)

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

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

transactionArray . push (setSessiontrx) ;

const 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" , }) ; Finally if we need to enable session key module we create a transaction here as well using the getEnableModuleData and pass the session key manager module address and push this to the array. Additionally we push the session transaction to the array as well, we will be batching these transactions together.

Next we will call sendTransaction which will build the user op and send it to Bundler. Don't forget to check the Info box at the top of this page before executing this code!

On success we'll show a nice toast to the user letting them know all is well.

Update your jsx to now look like this:

< div

< ToastContainer position = "top-right" autoClose = { 5000 } hideProgressBar = { false } newestOnTop = { false } closeOnClick rtl = { false } pauseOnFocusLoss draggable pauseOnHover theme = "dark" /

{ isSessionKeyModuleEnabled ?

(< button onClick = { ( )

=>

createSession (false) }

Create Session < / button

)

:

(< button onClick = { ( )

=>

createSession (true) }

Enable and Create Session < / button

) } < / div

View Complete Code import React ,

{ useEffect , useState }

```
from
"react" ; import
{ ethers , BigNumber }
from
"ethers" ; import
{ BiconomySmartAccountV2 ,
DEFAULT_SESSION_KEY_MANAGER_MODULE , createSessionKeyManagerModule }
from
"@biconomy/account" import
{ hexDataSlice , id , BytesLike , hexConcat , hexZeroPad , hexlify , parseEther }
from
"ethers/lib/utils" ; import
{ toast , ToastContainer }
from
'react-toastify' ; import
'react-toastify/dist/ReactToastify.css' ;
interface
props
{ smartAccount : BiconomySmartAccountV2 ; address :
string ; provider : ethers . providers . Provider ; }
export
interface
Rule
{ offset :
number ; condition :
number ; referenceValue :
string
| BytesLike ; }
export
interface
Permission
{ destContract :
string ; functionSelector :
string ; valueLimit : BigNumber ; rules : Rule [ ] ; }
const CreateSession : React . FC < props
=
( { smartAccount , address , provider } )
```

```
=>

{

const

[ isSessionActive , setIsSessionActive ]

=

useState ( false ) ; const

[ isSessionKeyModuleEnabled , setIsSessionKeyModuleEnabled ]

=

useState ( false ) ;

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 ) ; return ; }

catch ( err :

any )

{ console . error ( err ) setIsSessionKeyModuleEnabled ( false ) ; return ; } } checkSessionModuleEnabled ( ) } , [ isSessionKeyModuleEnabled , address , smartAccount , provider ] )

async

function

getABISVMSessionKeyData ( sessionKey :

string , permission : Permission , ) :

Promise < string

{ let sessionKeyData =

hexConcat ( [ sessionKey , permission . destContract , permission . functionSelector , hexZeroPad ( permission . valueLimit . toHexString ( ) ,

16 ) , hexZeroPad ( hexlify ( permission . rules . length ) ,

2 ) ,

// this can't be more 2**11 (see below), so uint16 (2 bytes) is enough ] ) ;
```

for

(let i =

0 ; i < permission . rules . length ; i ++)

{ sessionKeyData =

hexConcat ([sessionKeyData , hexZeroPad (hexlify (permission . rules [i] . offset) ,

2) ,

// offset is uint16, so there can't be more than 2**16/32 args = 211 hexZeroPad (hexlify (permission . rules [i] . condition) ,

1) ,

// uint8 permission . rules [i] . referenceValue ,]) ; } return sessionKeyData ; }

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

{ // Address of ABI Session Validation Module const abiSVMAddress =

"0x000006bC2eCdAe38113929293d241Cf252D91861" // -----> 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 , } ) ;
const nftAddress =
"0xdd526eba63ef200ed95f0f0fb8993fe3e20a23d0" ; const recipient =
"0xAddress" ; // get only first 4 bytes for function selector const functionSelector =
hexDataSlice ( id ( "safeMint(address)" ) ,
0 ,
4 ) ;
// create session key data const sessionKeyData =
await
getABISVMSessionKeyData ( sessionKeyEOA ,
{ destContract : nftAddress ,
// destination contract to call functionSelector : functionSelector ,
// function selector allowed valueLimit :
parseEther ( "0" ) ,
// no native value is sent // In rules, we make sure that referenceValue is equal to recipient rules :
[ { offset :
0 ,
// defines the position of a 32bytes word in the calldata (our recipient address) condition :
0 ,
// 0 = Condition.EQUAL referenceValue : ethers . utils . hexZeroPad ( recipient ,
32 )
// recipient address } , ] , } ) ;
const sessionTxData =
await sessionModule . createSessionData ( [ { validUntil :
0 , validAfter :
0 , sessionValidationModule : abiSVMAddress , sessionPublicKey : sessionKeyEOA as
0x { string } , sessionKeyData : sessionKeyData as
0x { string } , } , ] ) ; console . log ( "sessionTxData" , sessionTxData ) ;
// 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) ; }

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) } }

return

(< div

< ToastContainer position = "top-right" autoClose = { 5000 } hideProgressBar = { false } newestOnTop = { false } closeOnClick rtl = { false } pauseOnFocusLoss draggable pauseOnHover theme = "dark" /

{ isSessionKeyModuleEnabled ?

(< button onClick = { ( )

=>

createSession (false) }

Create Session < / button

)

:

(< button onClick = { ( )

=>

createSession (true) }

Enable and Create Session < / button

) } < / div

) }

export

default CreateSession ; Now with this implemented let's take a look at executing the NFT Mint with a session in the next

section. Previous SDK Integration Next NFT Mint with ABI SVM