# Create a session

## Overview

This tutorial demonstrates how a dapp can create a simple session using viem and the Biconomy Smart Account with the@biconomy/account SDK. The provided code assumes you have a Biconomy Paymaster API key and a connected user. The following is appropriately viewed from the perspective of a dapp, looking to make txs on a users behalf.

You can get your Biconomy Paymaster API key from the dashboard[here](#) .

## Prerequisites

- Biconomy Paymaster API key
- A Bundler url if you don't want to use the testnet one (for Amoy you can usehttps://bundler.biconomy.io/api/v2/80002/nJPK7B3ru.dd7f7861-190d-41bd-af80-6877f74b8f44
- )
- A user with a connected signer (viem WalletClient or ethers.Wallet for example)

## Step 1: Create the SmartAccountClient for the user

import

{ polygonAmoy as chain }

from

"viem/chains" ; import

{ PaymasterMode , createSmartAccountClient , createSession , Rule , Policy , }

from

"@biconomy/account" ;

const nftAddress =

"0x1758f42Af7026fBbB559Dc60EcE0De3ef81f665e" ; const withSponsorship =

{ paymasterServiceData :

{ mode : PaymasterMode . SPONSORED

} , } ;

// Create Biconomy Smart Account instance const usersSmartAccount =

await

createSmartAccountClient ( { signer : usersWalletClient ,

// assumes that a user has connected his walletClient (or an ethers Wallet) to your dapp biconomyPaymasterApiKey : config . biconomyPaymasterApiKey , bundlerUrl : config . bundlerUrl , } ) ;

## Step 2: Generate a store for your dapp's session keys

This function is used to create a new session key and store it in the sessionStorageClient. You can feed the sessionStorageClient into thecreateSessionKeyEOA(...args) as the third argument. If you do not provide a sessionStorageClient then one will get generated for you based on the environment. When localStorage is supported, it will return aSessionLocalStorage store, otherwise it will assume you are in a backend and useSessionFileStorage store.

const

{ sessionKeyAddress , sessionStorageClient }

=

await

createSessionKeyEOA ( usersSmartAccount , chain ) ;

## Step 3: Create the Policy

Next we need to generate a policy that we can use to request permission (granted via a users signature). The policy is comprised of a list of rules applied to a single contract method, along with an interval over which the session remains valid.

const rules : Rule [ ]

=

[ { /* *The index of the param from the selected contract function upon which the condition will be applied* offset :

0 , /* * *Conditions: * * 0 - Equal * 1 - Less than or equal * 2 - Less than * 3 - Greater than or equal * 4 - Greater than * 5 - Not equal* / condition :

0 , /* *The value to compare against*/ referenceValue : smartAccountAddress , } , ] ;

/* *The policy is made up of a list of rules applied to the contract method with and interval* const policy : Policy [ ]

=

[ { /**The address of the sessionKey upon which the policy is to be imparted */ sessionKeyAddress ,** /The address of the contract to be included in the policy */ contractAddress : nftAddress , /The specific function selector from the contract to be included in the policy* / functionSelector :

"safeMint(address)" , /**The list of rules which make up the policy */ rules ,** /The time interval within which the session is valid. Setting both to 0 will keep a session alive indefinitely */ interval :

{ validUntil :

0 , validAfter :

0 , } , /* *The maximum value that can be transferred in a single transaction*/ valueLimit :

0n , } , ] ;

## Step 4: Request Permission from the user with the policy

The session keys are imbibed with the relevant permissions when the user signs over the policy. The session can then be accessed from the sessionStorageClient and later used, even after the usersSmartAccount signer has left the dapp.

const

{ wait , session }

=

await

createSession ( usersSmartAccount , policy , sessionKeyAddress , sessionStorageClient , withSponsorship ) ;

const

{ receipt :

{ transactionHash } , success , }

=

await

wait ( ) ; Send the transaction using the Biconomy Smart Account and get the transaction hash. The transaction will be built into a User Operation and then send to the Bundler.

That's it! You've successfully requested permissions from your user, and stored session keys which can later be used to make txs on their behalf. Previous Sessions Next Use a session