Account Abstraction using Biconomy

Overview

The Biconomy Modular SDK is a comprehensive software development kit designed specifically for decentralized applications (dApps). Built on top of the ERC4337 solution, it offers a wide range of solutions, from user onboarding to sustained engagement, enhancing the overall user experience within your dApp. By leveraging modularity, the SDK offers enhanced customization, security, and functionality. This SDK operates in a non-custodial manner, providing a unified solution that combines simplicity and functionality in the realm of decentralized applications.

What's Possible?

Biconomy SDK brings blockchain-agnostic, web2-like experiences to your dApp in a non-custodial manner. Here's what you can enable today: * Easy User Onboarding: Simplifies the onboarding process for new users through social login, account creation, and recovery options, seamlessly integrating web2 users into your dApp. * Fiat On Ramp: Allows your users to easily and reliably buy or sell cryptocurrencies within your dApp, facilitating the transition between traditional and blockchain-based assets * Gasless Transactions: Sponsors gas fees for user interactions, making them as simple as web2 transactions, and improving the overall user experience. * Paying Gas Fees In ERC20 Tokens: Enable users to utilize any ERC20 asset in their wallet to pay for gas fees, offering flexibility and convenience. * Custom Transaction Bundling: Empower developers to build methods for transaction batching, allowing users to execute multiple actions in a single transaction, even across multiple chains. For example, users can approve and deposit in the same transaction without altering anything in the dApp smart contracts.

How Does It Work?

In the Biconomy SDK, the Smart Contract Wallet (SCW) is the foundation of the system and is created for every new user who joins a dApp. The SCW address is generated instantly when the user logs in, without needing any on-chain deployment. The actual smart contract is deployed along with the user's first transaction. The Smart Contract Wallet, also known as Smart Account, is associated with the user's Externally Owned Account (EOA), like a traditional MetaMask account or an account generated via Social Login. It provides enhanced security compared to EOAs by allowing users to deposit and withdraw funds securely at any time. One notable feature is that the SCW address remains consistent across different blockchain chains, enabling a chainless experience for users within the dApp. By leveraging transaction bundling with Smart Contract Wallets using the Biconomy SDK, dApps can offer seamless and efficient interactions for their users.

How Smart Contract Wallets Work

* The Biconomy Modular SDK introduces Smart Contract Wallets (SCW) that serve as a fundamental component of the system. Each SC Wallet is associated with an Externally Owned Account (EOA), acting as its owner. * The SC Wallet offers extensibility through support for various modules, such as MultiSig, Social Recovery Module, Recurring Transaction Module, Transaction Automation, Session Key Module, among others. These modules enhance wallet functionality without necessitating upgrades, enabling developers to tailor the user experience based on specific requirements. * Transactions executed through the SC Wallet are initiated by the EOA using a cryptographic signature. The transaction data, along with the signature, is then relayed by a Relayer, ensuring secure and efficient transaction processing. * Notably, the user's SC Wallet address is counterfactual, meaning it can be generated without deploying smart contract code on-chain, thanks to the CREATE2 opcode. This characteristic allows the SC Wallet address to remain consistent across different EVM chains, providing a seamless and chain-agnostic user experience within the dApp. * All interactions within a dApp, both on a single chain and across multiple chains, occur through the SC Wallet. This unified approach streamlines user interactions and fosters efficient cross-chain functionality. * the SC Wallet doubles as a Gas Tank for users, allowing them to use any token held in the wallet (accepted by the Paymaster) to pay gas fees during the same transaction. This capability enables dApps to implement transaction batching, empowering users to execute multiple actions in a single transaction, simplifying processes like approving and depositing.

Quick start with Smart Account Native Transfer

Clone the Biconomy quick start repository

git clone using SSH

git clone[email protected] :bcnmy/quickstart.git

git clone using https

git clone https://github.com/bcnmy/quickstart.git Install all dependencies yarn

install yarn dev Let’s first set up a .env file in the root of our project, this will need a Private Key of any Externally Owned Account (EOA) you would like to serve as the owner of the smart account we create. This is a private key you can get from wallets like MetaMask, TrustWallet, Coinbase Wallet etc. All of these wallets will have tutorials on how to export the Private key. PRIVATE_KEY =

"enter some private key"

Creating your rollup

You can refer to the rollup launchpad section on creating a new rollup within minutes. After you have created your rollup, please contact us via our support or the Biconomy team to request Biconomy integration. Please note both AltLayer and Biconomy will assess each team's requirement on a case-by-case basis.

Retrieving your rollup chain information

Go to the rollup launchpad and select Deployments. Look for your desired rollup.

You will need the following information for the next steps 1. 1. 2. EIP4337 entry point contract address, which will be provided to you by the AltLayer team 3. 2. 4. Bundler URL, which can be obtained from the 5. Biconomy dashboard 6. 7. 3. 8. Rollup RPC endpoint 9. 4. 10. Rollup chain ID

Reference code

Next, we will reference the official Biconomy example code to get you started. For the code snippet below, you will be required to fill in the * Rollup RPC endpoint: Line 8 * Bundler URL: Line 11 * Chain ID: Lines 12 and 17 * EIP4337 entry point contract address: Line 13 1 import

{ config }

from

"dotenv" 2 import

{ IBundler , Bundler }

from

'@biconomy/bundler' 3 import

{ ChainId }

from

"@biconomy/core-types" ; 4 import

{ BiconomySmartAccount , BiconomySmartAccountConfig ,

DEFAULT_ENTRYPOINT_ADDRESS

}

from

"@biconomy/account" 5 import

{ Wallet , providers , ethers }

from

'ethers' 6 7 config () 8 const provider =

new

providers . JsonRpcProvider ("") 9 const wallet =

new

Wallet (process . env . PRIVATE_KEY

```
||
"" , provider ); const bundler : IBundler =
new
Bundler ({ bundlerUrl :
" , chainId :
< Rollup chain ID
        , entryPointAddress :
< EIP4337 entry point contract address
        , }) const biconomySmartAccountConfig : BiconomySmartAccountConfig =
{ signer : wallet , chainId :
< Rollup chain ID
        , bundler : bundler } async
function
createAccount ()
{ const biconomyAccount =
new
BiconomySmartAccount ( biconomySmartAccountConfig ) const biconomySmartAccount =
await biconomyAccount . init () console . log ( "owner: " , biconomySmartAccount . owner ) console . log ( "address: " ,
await biconomySmartAccount . getSmartAccountAddress ()) return biconomyAccount }   // userOp sample code
// Send 0.1 native token to 0x322Af0da66D00be980C7aa006377FCaaEee3BDFD async
function
createTransaction ()
{ console . log ( "creating account" ) const smartAccount =
await
createAccount (); const transaction =
{ to :
'0x322Af0da66D00be980C7aa006377FCaaEee3BDFD' , data :
'0x' , value : ethers . utils . parseEther ( '0.1' ), }   const userOp =
await smartAccount . buildUserOp ([ transaction ]) userOp . paymasterAndData =
"0x"   const userOpResponse =
await smartAccount . sendUserOp ( userOp )   const transactionDetail =
await userOpResponse . wait ()   console . log ( "transaction detail below" ) console . log ( transactionDetail ) }
  createTransaction ()
```

Onboard Biconomy onto your rollup

You can book a call with the Biconomy team to discuss the next steps in supporting Biconomy on your rollup. If you need more developer resources, you can check out Biconomy documentation .

Public Testnets -Previous MACH Alpha for inEVM Next- External Integrations Data Availability (DA) using Celestia Last modified2mo ago