

# How to create and use a Biconomy account with permissionless.js

[Biconomy Smart Account](#) is a smart account building on the core concepts of Gnosis and Argent safes. You can use Biconomy with plugins such as session keys, and even write your own plugins.

## Steps

### Import the required packages

...

```
import{ ENTRYPOINT_ADDRESS_V06, createSmartAccountClient }from"permissionless" import{
signerToBiconomySmartAccount }from"permissionless/accounts" import{ createPimlicoBundlerClient,
createPimlicoPaymasterClient, }from"permissionless/clients/pimlico" import{ createPublicClient, getContract, http,
parseEther }from"viem" import{ sepolia }from"viem/chains"
```

...

### Create the clients

First we must create the public, bundler, and (optionally) paymaster clients that will be used to interact with the Biconomy account.

...

```
exportconstpublicClient=createPublicClient({ transport:http("https://rpc.ankr.com/eth_sepolia"), })

exportconstpaymasterClient=createPimlicoPaymasterClient({ transport:http("https://api.pimlico.io/v1/sepolia/rpc?
apikey=API_KEY"), entryPoint:ENTRYPOINT_ADDRESS_V06, })
```

...

### Create the signer

Biconomy accounts can work with a variety of signing algorithms such as ECDSA, passkeys, and multisig. In permissionless.js, the default Biconomy account validates ECDSA signatures. [Any signer](#) can be used as a signer for the Biconomy account.

For example, to create a signer based on a private key:

...

```
import{ privateKeyToAccount }from"viem/accounts"

constsigner=privateKeyToAccount("0xPRIVATE_KEY")
```

...

### Create the Biconomy account

With a signer, you can create a Biconomy account as such:

...

```
constbiconomyAccount=awaitsignerToBiconomySmartAccount(publicClient, { entryPoint:ENTRYPOINT_ADDRESS_V06,
signer: signer, index:0n,// optional address:"0x..."// optional, only if you are using an already created account })
```

...

The Biconomy account address is computed deterministically from the signer, but you can optionally pass an `index` to create any number of different accounts using the same signer. You can also pass an `address` to use an already created Biconomy account.

### Create the smart account client

The smart account client is a permissionless.js client that is meant to serve as an almost drop-in replacement for viem's [walletClient](#).

...

```
const smartAccountClient = createSmartAccountClient({ account: biconomyAccount,
entryPoint: ENTRYPOINT_ADDRESS_V06, chain: sepolia, bundlerTransport: http("https://api.pimlico.io/v1/sepolia/rpc?
apikey=API_KEY"), middleware: { sponsorUserOperation: paymasterClient.sponsorUserOperation, // optional }, })
```

...

## Fetch the gas prices (optional)

If you're using Pimlico as your bundler, fetch the required gas price to use beforehand and pass it in as `maxFeePerGas` and `maxPriorityFeePerGas` parameters. Other providers might have different requirements for fetching the gas price.

...

```
export const bundlerClient = createPimlicoBundlerClient({ transport: http("https://api.pimlico.io/v2/sepolia/rpc?
apikey=API_KEY"), entryPoint: ENTRYPOINT_ADDRESS_V06, })
```

```
const gasPrices = await bundlerClient.getUserOperationGasPrice()
```

...

## Send a transaction

Transactions using `permissionless.js` simply wrap around user operations. This means you can switch to `permissionless.js` from your existing viem EOA codebase with minimal-to-no changes.

...

```
const txHash = await smartAccountClient.sendTransaction({ to: "0xd8da6bf26964af9d7eed9e03e53415d37aa96045",
value: parseEther("0.1"), maxFeePerGas: gasPrices.fast.maxFeePerGas, // if using Pimlico maxPriorityFeePerGas:
gasPrices.fast.maxPriorityFeePerGas, // if using Pimlico })
```

...

This also means you can also use viem Contract instances to transact without any modifications.

...

```
const nftContract = getContract({ address: "0xFBA3912Ca04dd458c843e2EE08967fC04f3579c2", abi: nftAbi, client: { public:
publicClient, wallet: smartAccountClient, }, })
```

```
const txHash = await nftContract.write.mint()
```

...

You can also send an array of transactions in a single batch.

...

```
const txHash = await smartAccountClient.sendTransactions({ transactions: [ {
to: "0xd8da6bf26964af9d7eed9e03e53415d37aa96045", value: parseEther("0.1"), data: "0x", }, {
to: "0x1440ec793aE50fA046B95bFeCa5aF475b6003f9e", value: parseEther("0.1"), data: "0x1234", }, ], maxFeePerGas:
gasPrices.fast.maxFeePerGas, // if using Pimlico maxPriorityFeePerGas: gasPrices.fast.maxPriorityFeePerGas, // if using
Pimlico })
```

...