Smart Accounts : V1 to V2 Upgrade

All smart accounts already deployed as account V1 can be safely upgraded to account V2 implementation (aka Modular Smart Account).

Biconomy Smart Account V1 and Biconomy Smart Account V2 use different smart account factories; hence, the account address generated for the same EOA and salt would be different for both. Account V2 does not have default owner storage and manages authorisation via validation(aka authorisation) modules.

Account V1's single EOA ownership is comparable to Account V2 with the ECDSA Ownership module and appears to be the most common upgrade path. Using one default auth module, developers can use the same address and upgrade implementation.

In theSDK release (npm version 3.1.1), below API methods have been added to V1 Account API specs (BiconomySmartAccount.ts in accounts package) to help with the upgrades

- getUpdateImplementationData( )
- : This returns transaction object which can be used in buildUserOp() method for batching with other transactions. If you do an upgrade by using this transaction payload you must also enable a validation module (get this data by getModuleSetupData()) as part of the Same transaction. These methods can be used while batching the upgrade with other transactions on your new implementation.
- updateImplementationUserOp( )
- : This returns the userOp, which can be sent as asingle
- action to do the upgrade.

Steps to upgrade

1. create a Biconomy account V1 instance.

const biconomyAccount =

new

BiconomySmartAccount (biconomySmartAccountConfig) ; const biconomySmartAccount =

await biconomyAccount . init ( ) ; 1. Create partial userOp using one of the following methods depending on the usecase.

- If user wants to batch the upgrade transaction with other transactions,getUpdateImplementationData
- method can be used along withgetModuleSetupData
- .

info Its critical to add themoduleSetupData while doing the upgrade when using getUpdateImplementationData to ensure at least one validation module is enabled. Otherwise your V2 account will be left without any authorization and unusable. const txList =

[ ] ; const updrageTransaction =

await biconomySmartAccount . getUpdateImplementationData ( ) ; const moduleSetupData =

await biconomySmartAccount . getModuleSetupData ( ) ; txList . push (updrageTransaction) txList . push (moduleSetupData) txList . push (transaction)

// any transaction user needs to combine const partialUserOp =

await biconomySmartAccount . buildUserOp (txList) ; * If user wants to only upgrade the account thenupdateImplementationUserOp * method can be used which returns the partial userOp directly.const * partialUserOp * = * await * biconomySmartAccount * . * updateImplementationUserOp * (*) * console * . * log * (* 'partial userOp' * , * partialUserOp *)

1. Once we have the partial User operation, we can execute it using following methods.

// get the paymasterData, user can also use ERC20 mode const biconomyPaymaster = biconomySmartAccount . paymaster

as

IHybridPaymaster < SponsorUserOperationDto

;

try

{ const paymasterAndDataResponse = await biconomyPaymaster . getPaymasterAndData (partialUserOp , { mode :

PaymasterMode . SPONSORED , smartAccountInfo :

{ name :

'BICONOMY' , version :

'1.0.0' } , }) ; partialUserOp . paymasterAndData

= paymasterAndDataResponse . paymasterAndData ; if

(paymasterAndDataResponse . callGasLimit

&& paymasterAndDataResponse . verificationGasLimit

&& paymasterAndDataResponse . preVerificationGas)

{

partialUserOp . callGasLimit

= paymasterAndDataResponse . callGasLimit ; partialUserOp . verificationGasLimit

= paymasterAndDataResponse . verificationGasLimit ; partialUserOp . preVerificationGas

= paymasterAndDataResponse . preVerificationGas ; } }

catch

(e)

{ console . log ("error received " , e) ; } const userOpResponse =

await biconomySmartAccount . sendUserOp (partialUserOp) ; const transactionDetails =

await userOpResponse . wait ( ) ; 1. Once the above userOp (which contains the upgrade transaction) is mined successfully, Import BiconomySmartAccountV2 class from accounts package as shown in theexample 2. CreateBiconomySmartAccountV2 3. instance. When initialising BiconomySmartAccountV2 with the default auth module as ECDSAOwnershipModule instance, the generated default address will differ. However, It’s preferred to use the V1 address here because the account has been upgraded. 4. There are two ways to do this. Either the account address can be overridden in thebiconomySmartAccountConfigV2 5. , or ascanForUpgradedAccountsFromV1 6. flag can be passed that detects the V1 account address that has been upgraded to V2 implementation (with the help of the AddressResolver contract) 7. CreateBiconomySmartAccountV2 8. instance with overridden address (if V1 address known from the same script) or enable the flag to detect the same.

const ecdsaModule =

await

ECDSAOwnershipValidationModule . create ({ signer : signer ,

// same signer used in V1 moduleAddress :

DEFAULT_ECDSA_OWNERSHIP_MODULE

// imported from modules package })

const biconomySmartAccountConfigV2 =

{ chainId : config . chainId , rpcUrl : config . rpcUrl , paymaster : paymaster , bundler : bundler , entryPointAddress :

DEFAULT_ENTRYPOINT_ADDRESS , defaultValidationModule : ecdsaModule , activeValidationModule : ecdsaModule , senderAddress :

await biconomySmartAccount . getSmartAccountAddress ( )

// if the address is already known

scanForUpgradedAccountsFromV1 :

true

// OR use this flag if not overriding like above } ;

const biconomySmartAccountV2 =

await

BiconomySmartAccountV2 . create (biconomySmartAccountConfigV2) ; This way it will use V2 account but with the context of V1 address which has been already upgraded. Previous Execute a Gasless Transaction Next Paymaster