Multichain Validation

This module acts as an extension of theECDSA Module . The added benefit of this module is you now have the ability to sign once and authorize any number of userOps across multiple blockchain networks.

What is the Multichain Validation Module?

This module allowsExternally Owned Accounts (EOAs) to authorize and sign user operations (UserOps) for Biconomy Smart Accounts on Multiple chains with a single ECDSA signature.

Benefits

- Significantly improves UX for deploying and setting up Smart Accounts on several chains
- Significantly reduces user friction for multichain operations.

tip You can also use this module as thedefault module just like the ECDSA Validation Module, as it has all its capabilities for sending userOps on the particular chain.

Key Functions

- ECDSA Signature Scheme
- : Utilizes the ECDSA secp256r1 curve for secure signing.
- Flexible Signer Authentication
- : Supports various signer solutions like Privy, Fireblocks, Arcana Auth, Capsule, Lit protocol, Turnkey, Web3Auth, Particle, Magic, Portal, etc.
- EIP-1271 Compliance
- : Allowing Smart Accounts to sign Ethereum messages for logging into dApps.
- Sign once, Execute on multiple chains

Use Cases

Enables use cases which require several actions to be authorized for several chains with just one signature required from user.

1. Multichain intent for example exit all my native tokens for USDC
2. Allows a dApp to require just one signature from their user to configure & deploy smart accounts on multiple chains and delegate certain actions with permission via session keys on all those chains
3. Cross-chain state sync actions e.g transfer ownership of a smart account, change ENS, similar actions on a protocol that are deployed on multiple chains
4. Onboard a user on x chains by deploying their smart account and some action like registering in a raffle

How does it work?

1. Collecting the set of userOps for signing
2. Calculate userOpHash
3. Every userOpHash is now a leaf of a Merkle Tree.
4. Instead of signing individual hashes, the user signs the Merkle Root.

Now this signed Merkle Root serves as on-chain evidence, proving any userOp within the tree is authorized. With this, Users can authorize any number of userOps with just one signature over the Merkle Tree Root.

Smart Contract Deep Dive

You have the option to transmit a multi-chain signature, enabling the sending of various user operations on distinct chains. This signature is then validated by the module. Alternatively, if a plain ECDSA signature (65 bytes) is sent, the module interprets it as a standard ECDSA validation.

User Operation Validation (validateUserOp

)

function

validateUserOp (UserOperation calldata userOp ,

bytes32 userOpHash)

external

view virtual override returns

(uint256)

{ (bytes

memory moduleSignature ,

)

= abi . decode (userOp . signature ,

(bytes ,

address)) ;

if

(moduleSignature . length ==

65)

{ return

_verifySignature (userOpHash , moduleSignature ,

address (uint160 (sender)))

? VALIDATION_SUCCESS : SIG_VALIDATION_FAILED ; }

(uint48 validUntil , uint48 validAfter , bytes32 merkleTreeRoot , bytes32 [ ]

memory merkleProof , bytes

memory multichainSignature)

= abi . decode (moduleSignature ,

(uint48 ,

uint48 ,

bytes32 ,

bytes32 [ ] ,

bytes)) ;

bytes32 leaf =

keccak256 (abi . encodePacked (validUntil , validAfter , userOpHash)) ;

if

(! MerkleProof . verify (merkleProof , merkleTreeRoot , leaf))

{ revert ("Invalid UserOp") ; }

return

_verifySignature (merkleTreeRoot , multichainSignature ,

address (uint160 (sender))) ?

_packValidationData (false , validUntil ==

0

?

type (uint48) . max : validUntil , validAfter) : SIG_VALIDATION_FAILED ; } All the other methods are inherited fromECDSAOwnershipRegistryModule

References

1. Source code
2. Merkle Trees

How to Guide - Biconomy SDK

Integrate the Multichain Validation Module into Biconomy Smart Accounts using the SDK.

Setup and Installation

- npm
- yarn
- pnpm

npm install @biconomy/modules ethers yarn add @biconomy/modules ethers pnpm add @biconomy/modules ethers

Creating a Signer

import

{ ethers }

from

"ethers" ;

const provider =

new

ethers . JsonRpcProvider ("[RPC_Endpoint]") ; const signer =

new

ethers . Wallet ("[Private_Key]" , provider) ;

Importing and Initializing Multichain Validation Module

import

{ createMultiChainValidationModule , DEFAULT_MULTICHAIN_MODULE , }

from

"@biconomy/modules" ;

// Notice this is chain agnostic so same instance can be used on instances of Smart account API on different chains const multiChainModule =

await

createMultiChainValidationModule ({ signer : signer , moduleAddress :

DEFAULT_MULTICHAIN_MODULE , }) ; Once initialized it can be passed to the smart account create method to create instances of the smart account. Check Paymaster integration and Bundler integration sections to create instances of them.

Here is how you can set up Smart Account instances accross multiple chains:

import

{ createSmartAccountClient , PaymasterMode }

from

"@biconomy/account" ;

let baseAccount =

await

createSmartAccountClient ({ biconomyPaymasterApiKey :

"https://docs.biconomy.io/dashboard/paymaster" bundlerUrl :

"" ,

// <-- Read about this at https://docs.biconomy.io/dashboard#bundler-url defaultValidationModule : multiChainModule , activeValidationModule : multiChainModule , }) ;

let polygonAccount =

await

createSmartAccountClient ({ biconomyPaymasterApiKey :

"https://docs.biconomy.io/dashboard/paymaster" bundlerUrl :

"" ,

// <-- Read about this at https://docs.biconomy.io/dashboard#bundler-url defaultValidationModule : multiChainModule , activeValidationModule : multiChainModule , }) ;

Signing and Sending Multichain userOps

// This could be swap action, mint nft, transfer ownership, enable different module as such (check use cases above) which you intend to do on multiple chains // You can also batch with providing just array of transactions // If the smart account is not deployed it would be deployed as part of this transaction. No additional action needed in below payload

import

{ encodeFunctionData , parseAbi }

from

"viem" ;

// Example of creating the callData for the transaction // This could be swap action, mint NFT, transfer ownershi p, enable different module as such (check use cases above) which you intend to do on multiple chains // You can also batch with providing just array of transactions // If the smart account is not deployed it would be deployed as part of this transaction. No additional action needed in below payload

const nftAddress =

"0x1758f42Af7026fBbB559Dc60EcE0De3ef81f665e" ;

const parsedAbi =

parseAbi (["function safeMint(address _to)"]) ; const mintNFTCallData =

encodeFunctionData ({ abi : parsedAbi , functionName :

"safeMint" , args :

[await polygonAccount . getAddress ( )] , }) ;

const transaction =

{ to : nftAddress , data : mintNFTCallData , } ;

// Build partial userOp for chain1 let partialUserOp1 =

await baseAccount . buildUserOp ([transaction] ,

{ // Assuming Sponsorship Paymaster is to be used otherwise leave as it's optional paymasterServiceData :

{ mode : PaymasterMode . SPONSORED , } , }) ;

// Build partial userOp for chain2 let partialUserOp2 =

await polygonAccount . buildUserOp ([transaction] ,

{ // Assuming Sponsorship Paymaster is to be used otherwise leave as it's optional paymasterServiceData :

{ mode : PaymasterMode . SPONSORED , } , }) ;

// Use multichain module to sign once for all ops const returnedOps =

await multiChainModule . signUserOps ([{ userOp : partialUserOp1 , chainId : ChainId . BASE_GOERLI_TESTNET

} , { userOp : partialUserOp2 , chainId : ChainId . POLYGON_MUMBAI

} ,]) ;

const userOpResponse1 =

await baseAccount . sendSignedUserOp (returnedOps [0]

as

any) ; // You can also just wait for transaction hash by waitForTxHash() const transactionDetails1 =

await userOpResponse1 . wait ( ) ;

const userOpResponse2 =

await polygonAccount . sendSignedUserOp (returnedOps [1]

as

any) ; const transactionDetails2 =

await userOpResponse2 . wait ( ) ; info When the Smart Account isn't yet deployed, settingmultiChainModule as the default validation module is beneficial. This activates it by default and retains all functions of theECDSA Module , a popular go-to module.

Conclusion

It makes it easier for dApps to deploy and set up Smart Accounts across multiple chains or issue session keys with different permissions for each chain, reducing user difficulties.

Multichain module allows us to use session keys and setup sessions on multiple chains with single wallet signature.

—Rage Trade Previous ECDSA Ownership Next Session Key Manager