# ECDSA Ownership Module

## What is it?

TheECDSA Ownership Registry Module orECDSA Validation Module is integral toBiconomy's Modular Smart Account , enhancing transaction security and user authentication. This document outlines its functionality, benefits, and use cases.

### What is the ECDSA Ownership Module?

This module allowsExternally Owned Accounts (EOAs) to authorize and sign user operations (UserOps) for Smart Account. It operates similarly to non-modular Smart Account ownership but is reconstructed as aValidation Module within theAccount Abstraction + Modular Smart Accounts ecosystem.

tip The ECDSA Validation Module is the standard choice for Biconomy Modular Smart Accounts unless another module is chosen during setup. It makes things easier for users and helps manage ownership smoothly.

#### Key Functions

- Single Signer Simplicity
- : Offers 1/1 multisig, single-signature control, ideal for both Web2 users and crypto-native users who already possess an EOA wallet.
- ECDSA Signature Scheme
- : Utilizes the ECDSA secp256k1 curve for secure signing.
- Flexible Owner Authentication
- : Supports various signer solutions like Privy, Fireblocks, Arcana Auth, Web3Auth, Magic, Capsule, Turnkey or Particle.
- EIP-1271 Compliance
- : Allowing Smart Accounts to sign Ethereum messages for logging into dApps.

info One of the most popular modules, it bridges the ease of traditional web logins with the security of blockchain technology.

#### Use Cases

1. Secure Transaction Signing
2. : EOAs can securely authorize transactions for Smart Accounts.
3. dApp Interaction
4. : Simplifies the process of logging into dApps using Ethereum messages thanks to EIP-1271 support.
5. Ownership Management
6. : Mirrors traditional ownership systems, providing a familiar framework within a more advanced blockchain setting.

## SDK Guide

### Initializing ECDSA Module

There is no need to do anything in order to use ECDSAOwnershipValidationModule, this module is set by default on every smart account.

Usage

import

{ createECDSAOwnershipValidationModule , createSmartAccountClient , }

from

"@biconomy/accounts" ;

const ecdsaModuleConfig =

{ signer } ;

const defaultValidationModule =

await

createECDSAOwnershipValidationModule ( ecdsaModuleConfig ) ;

// Signer for smart account taken from the module const smartAccount =

await

createSmartAccountClient ( { bundlerUrl , defaultValidationModule ,

// Alternatively this can be ommitted, as it is the default validation module used when accounts are created } ) ;

const saModuleAddress =

await smartAccount . activeValidationModule . getAddress ( ) ; console . log ( saModuleAddress === defaultValidationModule . moduleAddress ) ;

// true Parameters

required params are explicitly mentioned

- signer(Signer
- , required): Pass the signer instance into the validation module which gets used to detect owner of the smart account.
- entryPointAddress(Hex
- ): Defaults to"0x5ff137d4b0fdcd49dca30c7cf57e578a026d2789"
- moduleAddress(Hex
- ): Defaults to"0x0000001c5b32F37F5beA87BDD5374eB2aC54eA8e"
- version(ModuleVersion): Defaults to"V1_0_0"

Returns

- ecdsaOwnershipValidationModule
- (ECDSAOwnershipValidationModule
- ): An instance of the Biconomy ECDSA ownership validation module.

info The Smart Account's address is counterfactually generated based on the signer's public key, which is the owner's address used in the ECDSA Module. This feature allows for the pre-calculation of the account address before deployment. warning To deploy the same Smart Account (with the same address) on different chains, it's crucial to use the same module address, the same initialization data (here, the owner's EOA address) for the ECDSA Module here and the index. This ensures consistency in the account's address across various blockchains.

# Smart Contract Deep Dive

This section dives into theEcdsaOwnershipRegistryModule for Biconomy Smart Accounts, focusing on key functionalities and security aspects.

## Core Functionalities

#### User Operation Validation (validateUserOp

)

// Validates user operations signed by an EOA function

validateUserOp ( UserOperation calldata userOp , bytes32 userOpHash )

external

view virtual override returns

( uint256 )

{ if

( _verifySignature ( userOpHash , userOp . signature [ 96 : 161 ] , userOp . sender ) )

{ return VALIDATION_SUCCESS ; } return SIG_VALIDATION_FAILED ; } note Objective: The function safeguards the authenticity of user operations on Smart Accounts.

Method: It specifically decodes and verifies a segment of the userOp.signature. This segmentation is crucial because userOp.signature contains both the signature and the validation module's address. By extracting only the signature part, the function accurately validates the operation against the intended owner's signature, ensuring that the operation is legitimately authorized.

#### Signature Verification (_verifySignature

)

// Internal function to verify the signature of a smart account function

_verifySignature ( bytes32 dataHash ,

bytes

memory signature ,

address smartAccount ) internal

view

returns

( bool )

{ address expectedSigner = _smartAccountOwners [ smartAccount ] ; // Reverts if no owner is registered if

( expectedSigner ==

address ( 0 ) )

{ revert

NoOwnerRegisteredForSmartAccount ( smartAccount ) ; } // Checks for signature length and recovers the signer if

( signature . length <

65 )

revert

WrongSignatureLength ( ) ; address recovered =

( dataHash . toEthSignedMessageHash ( ) ) . recover ( signature ) ; if

( expectedSigner == recovered )

{ return

true ; } recovered = dataHash . recover ( signature ) ; return expectedSigner == recovered ; } Validates a signature against a data hash and registered owner, supporting EIP-1271 standard.

function

isValidSignature ( bytes32 dataHash , bytes

memory signature )

public

view

returns

( bytes4 ) ; note isValidSignature serves as a key function for external smart contracts to verify signature authenticity. caution Proper handling and verification of signatures are crucial for maintaining the integrity of the Smart Account. TheisValidSignature function is essential for EIP-1271 support, validating that the signature matches the expected signer as per the EIP-1271 standard.

## Security Considerations

- Strict Ownership Rules
- : Only Externally Owned Accounts (EOAs) can authorize transactions, ensuring secure control over Smart Account operations.
- Signature Verification
- : Implements robust methods for signature validation, crucial for preventing unauthorized access.

info TheinitForSmartAccount method doesn't includeisSmartContract checks due to operational constraints. However, this doesn't compromise security as operations require valid EOA signatures. This ensures Smart Accounts remain secure within their operational framework.

### Interaction with Smart Accounts

TheEcdsaOwnershipRegistryModule interacts with Smart Accounts primarily through its core functionalities:

- Initialization and Ownership
- : During deployment, Smart Accounts useinit
- method to invokeinitForSmartAccount
- on modules, setting initial ownership and configurations.
- User Operation Validation
- : When a Smart Account attempts to perform an operation,validateUserOp
- is invoked to ensure the action is authorized by the registered owner.
- Signature Verification
- : The module uses_verifySignature
- to verify any signatures associated with transactions initiated by the Smart Account.Previous Modules Next Multichain Validation