Custom Validation Module

In this tutorial, we will look at how to create your own validation module.

As mentioned in theoverview , any custom validation module needs to extend the abstract contractBaseAuthorizationModule . This contract is crucial as it connects your module to essential authorization functions. The following implementation enables a module to receive userOp data and provide validation results back to the SmartAccount.

Let's take a look at the step-by-step guide to create a custom Validation Module.

1. Create the custom validation contract

import

{ BaseAuthorizationModule }

from

"./BaseAuthorizationModule.sol" ;

contract MyCustomValidationModule is BaseAuthorizationModule { string

public constant NAME

=

"My Custom Validation Module" ; string

public constant VERSION

=

"0.0.1" ;

// Function signature of this method will be used as moduleSetupData in Account Factory function

initForSmartAccount ( )

{

} } As per the code

- First, we import the BaseAuthorizationModule and initialize two constants name and version
- Then we declare a method initForSmartAccount which gets used to initialize this module for the smart account. SA calls this function with msg.sender as the smart account address. It updates the relevant storage for the msg sender, It could be ownership information as shown in ECDSAOwnsership Module.
- Implement validateUserOp method

/// @inheritdoc IAuthorizationModule function

validateUserOp (UserOperation calldata userOp , bytes32 userOpHash) external view virtual override returns

(uint256)

{

(bytes memory moduleSignature ,

)

= abi . decode (userOp . signature , (bytes , address)) ; // implement _verifySignature as per the module functionality if

(_verifySignature (userOpHash , moduleSignature , userOp . sender))

{ return

VALIDATION_SUCCESS ; } return

SIG_VALIDATION_FAILED ; } As per the code

- This function requires the userOp and the userOp Hash as the input parameters where userOp signature is the ABI-

encoded signature and validation module address. we extract the module signature and call the verify signature method which we will implement next.

- _verifySignature
- method expects moduleSignature which is a signature that should be processed by a module and made according to the requirements specified by the module that is expected to be processing it.
- validation status constants are imported from AuthorizationModulesConstants contract. VALIDATION_SUCCESS indicates successful validation, while VALIDATION_FAILED signifies validation failure.
- Implement isValidSignature method

function

isValidSignature (bytes32 dataHash , bytes memory moduleSignature)

public view virtual override returns

(bytes4)

{ return

isValidSignatureForAddress (dataHash , moduleSignature , msg . sender) ; }

/// @inheritdoc IEcdsaOwnershipRegistryModule function

isValidSignatureForAddress (bytes32 dataHash , bytes memory moduleSignature , address smartAccount)

public view virtual override returns

(bytes4)

{ if

(_verifySignature (keccak256 (abi . encodePacked ("\x19Ethereum Signed Message:\n52" , dataHash , smartAccount)) , moduleSignature , smartAccount))

{ return

EIP1271_MAGIC_VALUE ; } return

bytes4 (0xffffffff) ; } As per the code

- isValidSignature
- : this method validates an EIP-1271 signature. It internally calls the isValidSignatureForAddress which again calls the _verifySignature method.
- Implement isValidSignatureUnsafe method

/// @inheritdoc ISignatureValidator function

isValidSignatureUnsafe (bytes32 dataHash , bytes memory moduleSignature)

public view virtual returns

(bytes4)

{ return isValidSignatureForAddressUnsafe (dataHash , moduleSignature , msg . sender) ; }

/// @inheritdoc IEcdsaOwnershipRegistryModule function

isValidSignatureForAddressUnsafe (bytes32 dataHash , bytes memory moduleSignature , address smartAccount)

public view virtual returns

(bytes4)

{ if

(_verifySignature (dataHash , moduleSignature , smartAccount))

{ return

EIP1271_MAGIC_VALUE ; } return

bytes4 (0xffffffff) ; } As per the code

- isValidSignatureUnsafe
- : this method validates an EIP-1271 signature but expects the data Hash to already include smart account address information.
- Implement _verifySignature method

/** * @param

dataHash Hash of the data to be validated. * @param

signature Signature to be validated. * @param

smartAccount expected signer Smart Account address. * @return true if signature is valid, false otherwise. */ function

_verifySignature (bytes32 dataHash , bytes memory signature , address smartAccount) internal view returns

(bool)

{

// verification according to signature scheme of this module } As per the code_verifySignature method contains the custom logic for this module, which developer can implement based on their requirements.

Expand the code below to see the entire code:

Details import

{ BaseAuthorizationModule }

from

"./BaseAuthorizationModule.sol" ;

contract MyCustomValidationModule is BaseAuthorizationModule

{ string public constant NAME

=

"My Custom Validation Module" ; string public constant VERSION

=

"0.0.1" ;

/* *Smart contract account calls this function with msg.sender as smart account address It updates the relevant storage for the msg sender, It could be ownership information as mentioned in ECDSAOwnsership Module. The function signature of this method will be used as moduleSetupData in Account Factory* / function

initForSmartAccount ( )

{

}

/// @inheritdoc IAuthorizationModule function

validateUserOp (UserOperation calldata userOp , bytes32 userOpHash) external view virtual override returns

(uint256)

{ // in the userOp.signature field we append the moduleSignature with // the Validation Module address

// extract the moduleSignature (bytes memory moduleSignature ,

)

= abi . decode (userOp . signature , (bytes , address)) ; // implement _verifySignature as per the module functionality if

(_verifySignature (userOpHash , moduleSignature , userOp . sender))

{ return

VALIDATION_SUCCESS ; } return

SIG_VALIDATION_FAILED ; }

// moduleSignature is a signature that should be processed by a module // and made according to the requirements specified by the module that // is expected to be processing it.

/** * @param

dataHash Hash of the data to be validated. * @param

signature Signature to be validated. * @param

smartAccount expected signer Smart Account address. * @return true if signature is valid, false otherwise. */ function

_verifySignature (bytes32 dataHash , bytes memory signature , address smartAccount) internal view returns

(bool)

{

// verification according to signature scheme of this module }

/** * @inheritdoc ISignatureValidator * @dev Validates a signature for a message. * @dev Appends smart account address to the hash to avoid replay attacks * To be called from a Smart Account. * @param

dataHash Hash of the message that was signed. * @param

moduleSignature Signature to be validated. * @return EIP1271_MAGIC_VALUE if signature is valid, 0xffffffff otherwise. */ function

isValidSignature (bytes32 dataHash , bytes memory moduleSignature)

public view virtual override returns

(bytes4)

{ return isValidSignatureForAddress (dataHash , moduleSignature , msg . sender) ; }

/// @inheritdoc IEcdsaOwnershipRegistryModule function

isValidSignatureForAddress (bytes32 dataHash , bytes memory moduleSignature , address smartAccount)

public view virtual override returns

(bytes4)

{ if

(_verifySignature (keccak256 (abi . encodePacked ("\x19Ethereum Signed Message:\n52" , dataHash , smartAccount)) , moduleSignature , smartAccount))

{ return

EIP1271_MAGIC_VALUE ; } return

bytes4 (0xffffffff) ; }

/// @inheritdoc ISignatureValidator function

isValidSignatureUnsafe (bytes32 dataHash , bytes memory moduleSignature)

public view virtual returns

(bytes4)

{ return isValidSignatureForAddressUnsafe (dataHash , moduleSignature , msg . sender) ; }

/// @inheritdoc IEcdsaOwnershipRegistryModule function

isValidSignatureForAddressUnsafe (bytes32 dataHash , bytes memory moduleSignature , address smartAccount)

public view virtual returns

(bytes4)

{ if

(_verifySignature (dataHash , moduleSignature , smartAccount))

{ return

EIP1271_MAGIC_VALUE ; } return

bytes4 (0xffffffff) ; }

} Note: We could also create a module's own interface IMyCustomValidationModule to move events and errors there and then make the above module extend this interface. Previous ERC20 transfer Next Migrating to V4