Adding View Functions

Adding Encrypted Balance Retrievalâ

To enhance our contract with secure balance viewing, we're going to implement agetBalanceEncrypted() function. This function will employ permissions to enforce access control, ensuring that only the rightful owner can retrieve and decrypt their encrypted balance.

Defining the Functionâ

We'll start by adding a new function to ourWrappingERC20 contract. This function will use theonlySender(perm) modifier from thePermissioned contract to ensure that only the message sender, validated through a signature, can access their encrypted balance.

function

getBalanceEncrypted (Permission calldata perm) public view onlySender (perm) returns

(uint256)

{ return FHE . decrypt (_encBalances [msg . sender]) ; }

Off-Chain Signature Generationâ

Users will need to generate a signature off-chain, using EIP-712 to sign their balance retrieval request. This signature proves that the user has authorized the retrieval of their encrypted balance.

Executing the Functionâ

When callinggetBalanceEncrypted() , the user includes their off-chain generated signature as a parameter. The function will execute only if the signature is valid and matches themsg.sender , returning the user's encrypted balance.

Putting it All Togetherâ

pragma solidity ^ 0.8 .20 ;

import

"@fhenixprotocol/contracts/access/Permissioned.sol" ; import

"@openzeppelin/contracts/token/ERC20/ERC20.sol" ; import

"@fhenixprotocol/contracts/FHE.sol" ;

contract WrappingERC20 is ERC20 ,

Permissioned

{

mapping (address

=> euint32) internal _encBalances ;

constructor (string memory name , string memory symbol)

ERC20 (name , symbol)

{ _mint (msg . sender ,

100

*

10

**

uint (decimals ( ))) ; }

function

wrap (uint32 amount)

public

{ // Make sure that the sender has enough of the public balance require (balanceOf (msg . sender)

= amount) ; // Burn public balance _burn (msg . sender , amount) ;

// convert public amount to shielded by encrypting it euint32 shieldedAmount =

FHE . asEuint32 (amount) ; // Add shielded balance to his current balance _encBalances [msg . sender]

= _encBalances [msg . sender]

+ shieldedAmount ; }

function

unwrap (inEuint32 memory amount)

public

{ euint32 _amount =

FHE . asEuint32 (amount) ; // verify that our shielded balance is greater or equal than the requested amount FHE . req (_encBalances [msg . sender] . gte (_amount)) ; // subtract amount from shielded balance _encBalances [msg . sender]

= _encBalances [msg . sender]

- _amount ; // add amount to caller's public balance by calling the mint function _mint (msg . sender ,

FHE . decrypt (_amount)) ; }

function

transferEncrypted (address to , inEuint32 calldata encryptedAmount)

public

{ euint32 amount =

FHE . asEuint32 (encryptedAmount) ; // Make sure the sender has enough tokens. FHE . req (amount . lte (_encBalances [msg . sender])) ;

// Add to the balance of to and subract from the balance of from. _encBalances [to]

= _encBalances [to]

+ amount ; _encBalances [msg . sender]

= _encBalances [msg . sender]

- amount ; }

function

getBalanceEncrypted (Permission calldata perm) public view onlySender (perm) returns

(uint256)

{ return

FHE . decrypt (_encBalances [msg . sender]) ; } Edit this page

Previous Writing the Contract Next Deploying