Testing on Fhenix

During this phase, we will focus on deploying the contract, wrapping tokens, and executing transactions using the FhenixJS library and Hardhat.

FhenixJS is injected by the Fhenix Hardhat plugin and can be used automatically by tests.

We will break down each step, providing code snippets and explanations to ensure you understand how to test the contract effectively.

note At the moment, only Hardhat is supported as a full testing environment. Stay tuned for Foundry support in the future.

Step-by-Step Guideâ

1. Set Up the Test Environmentâ

First, import the necessary modules and define the initial variables.

```
import

{

WrappingERC20

}

from

"../types/contracts/WrappingERC20" ; import

hre ,

{ ethers }

from

'hardhat' ; import

{

Permit

}

from

"fhenixjs" ;

describe ( 'Test WERC20' ,

( )

=>

{ let

contractAddr : string ; let

contract :

WrappingERC20 ; let

permit :

Permit ; let

owner : string ; let

destination : string =

"0x1245dD4AdB920c460773a105e1B3345707B4834A" ;
```

const amountToSend =

BigInt (1) ;

2. Test Contract Deploymentâ

In this phase, we will deploy theWrappingERC20 contract and initialize the permit using FhenixJS.

it (Test Contract Deployment ,

async

( )

=>

{ const

{ ethers , fhenixjs }

= hre ; const

{ deploy }

= hre . deployments ; const

[signer]

=

await ethers . getSigners ( ) ;

// Set the owner to the signer's address owner = signer . address ;

// Deploy the WrappingERC20 contract const token =

await

deploy ("WrappingERC20" ,

{ from : signer . address , args :

["Test Token" ,

"TST"] , log :

true , skipIfAlreadyDeployed :

false , }) ;

// Get the deployed contract address contractAddr = token . address ;

// Generate the permit using FhenixJS permit =

await fhenixjs . generatePermit (contractAddr ,

undefined , signer) ; contract =

(await ethers . getContractAt ("WrappingERC20" , contractAddr))

as unknown as

WrappingERC20 ;

console . log (contractAddr: , contractAddr) ; }) ; Explanation:

- FhenixJS Injection:
- Thefhenixjs
- object is automatically available through Hardhat's runtime environment (hre
-). This means you don't need to explicitly import or initialize it.
- Permit Generation:
- ThegeneratePermit
- function from FhenixJS is used to create a permit for interacting with the contract. This permit is essential for

performing private operations on the contract, such as viewing encrypted balances.

3. Wrap Tokensâ

Now, we will test the wrapping functionality of the contract.

it (Wrap Tokens ,

async

( )

=>

{ // Get the balance before wrapping let balanceBefore =

await contract . balanceOf (owner) ; let privateBalanceBefore =

await contract . getBalanceEncrypted (permit) ; console . log (Public Balance before wrapping: { balanceBefore }) ; console . log (Private Balance before wrapping: { privateBalanceBefore }) ;

// Wrap the tokens await contract . wrap (amountToSend) ;

// Get the balance after wrapping let balanceAfter =

await contract . balanceOf (owner) ; let privateBalanceAfter =

await contract . getBalanceEncrypted (permit) ; console . log (Public Balance after wrapping: { balanceAfter . toString ( ) }) ; console . log (Private Balance after wrapping: { privateBalanceAfter . toString ( ) }) ; }) ; Explanation:

- Public and Private Balances:
- Before wrapping tokens, we check both the public balance (visible on the blockchain) and the private balance (encrypted and only visible with the permit).
- Wrapping Tokens:
- Thewrap
- function is called on the contract to wrap the specified amount of tokens.
- Encrypted Balances:
- After wrapping, we again check both balances to ensure the wrapping process worked as expected.

4. Execute Transactionâ

Finally, we will test the transaction execution using encrypted amounts.

it (Execute Transaction ,

async

( )

=>

{ // Get the private balance before sending let privateBalanceBefore =

await contract . getBalanceEncrypted (permit) ; console . log (Private Balance before sending: { privateBalanceBefore }) ;

// Encrypt the amount to send const encrypted =

await hre . fhenixjs . encrypt_uint32 (Number (amountToSend)) ;

// Transfer the encrypted amount await contract . transferEncrypted (destination , encrypted) ;

// Get the private balance after sending let privateBalanceAfter =

await contract . getBalanceEncrypted (permit) ; console . log (Private Balance after sending: { privateBalanceAfter }) ; }) ; }) ;
Explanation:

- Private Balance Check:
- Before sending tokens, we check the private balance to verify the initial state.
- Encryption:
- The amount to send is encrypted using theencrypt_uint32
- function from FhenixJS. This ensures that the amount is securely transmitted.
- Encrypted Transfer:

- ThetransferEncrypted
- function is called on the contract to transfer the encrypted amount to the destination address.
- Balance Verification:
- After the transfer, we check the private balance again to confirm the transaction.

Conclusionâ

This guide provided a step-by-step explanation of how to test a contract on Fhenix using Hardhat. By following these steps, you should be able to deploy a contract, wrap tokens, and execute transactions using the FhenixJS library. FhenixJS simplifies handling encrypted operations and permits, making it easier to integrate privacy features into your smart contracts.
Edit this page

Previous Deploying Next FHERC20