Your First FHE Contract

In this short guide, we'll demonstrate how simple it is to enable confidentiality in your smart contracts using Fhenix.

```
// SPDX-License-Identifier: MIT pragma solidity ^ 0.8 .17 ;

import

"@fhenixprotocol/contracts/FHE.sol" ;

contract EarlyWin

{ uint8 _plaintext ; euint8 public _cipherText ;

function

setCipherText ( inEuint8 calldata _encryptedNumber )

public

{ // convert inEuint8 type structure to euint8 _cipherText =

FHE . asEuint8 ( _encryptedNumber ) ; }

function

setPlainText ( uint8 _number )

public

{ // set standard plaintext _plaintext = _number ; }

function

decrypt ( )

public view returns

( uint8 )

{ return

FHE . decrypt ( _cipherText ) ; } }
```

Code Walkthroughâ

First, FHE is imported directly into your contract with a single line of code. Next, we establish two unsigned integers, with_cipherText being encrypted. This means it will not be publicly accessible by anyone other than the intended viewer. The standard_plaintext uint8 represents a number that is public for all to view.

Step By Stepâ

1. Importing FHE
2. import
3. "@fhenixprotocol/contracts/FHE.sol"
4. ;
5. We can import the FHE precompiles directly into the smart contract with a single line of code. The power of FHE in one single line of copy-paste.
6. Declaring Variables
7. uint8 _plaintext
8. ;
9. euint8
10. public
11. _cipherText
12. ;
13. Line 8 is a familiar way of setting a number in Solidity. However, this unsigned integer will be publicly queryable by everyone with access to the network. The number set on line 9 as the encrypted unsigned integer will not be.
14. Setting the Encrypted Number
15. function
16. setCipherText

17. (
18. inEuint8 calldata _encryptedNumber
19.)
20. public
21. {
22. // convert inEuint8 type structure to euint8
23. _cipherText
24. =
25. FHE
26. .
27. asEuint8
28. (
29. _encryptedNumber
30.)
31. ;
32. }
33. Here, we set the encrypted number via the setter function. We pass aninEuint8
34. as the ciphertext, which represents the number we want to set.
35. Setting the Plaintext Number
36. function
37. setPlainText
38. (
39. uint8 _number
40.)
41. public
42. {
43. // set standard plaintext
44. _plaintext
45. =
46. _number
47. ;
48. }
49. This is the standard way of setting a number via a function call in plaintext Solidity.
50. Decrypting the Encrypted Number
51. function
52. decrypt
53. (
54.)
55. public
56. view
57. returns
58. (
59. uint8
60.)
61. {
62. return
63. FHE
64. .
65. decrypt
66. (
67. _cipherText
68.)
69. ;
70. }
71. Finally, we call the decrypt function to convert the private number to a public one. The method on line 21 represents an example of synchronous decryption. Fhenix will eventually move to an asynchronous decryption call. Don't worry, it will still be possible, and we will update you when the implementation is ready.

Next Stepsâ

If you want to learn more about working with Fhenix, please check out docs for a development tutorial . Here, you will learn how to set up your local dev environment and create an encrypted ERC-20 token!

Have Questions?â

Hop into our Discord and ask questions in the #dev-general or #tech-questions channels! Edit this page

Previous Tutorials Next Shielded ERC20 Token