Encryption

fhenix.js provides an easy-to-use function to encrypt your inputs before sending them to the Fhenix blockchain.

tip Encryption in Fhenix is done using the global chain key. This key is loaded when you create a fhenix.js client automatically When we perform encryption, we specify the type ofeuint (Encrypted Integer) we want to create. This should match the expected type in the Solidity contract we are working with.

First, initialize the library -

import

{ FhenixClient }

from

'fhenixjs' ; import

{ BrowserProvider }

from

"ethers" ;

const provider =

new

BrowserProvider (window . ethereum) ;

const client =

new

FhenixClient ({ provider }) ; Then, you can use the created client to encrypt

import

{ FhenixClient , EncryptedType , EncryptedUint8 }

from

'fhenixjs' ;

const result : EncryptedUint8 =

await client . encrypt (number , EncryptionTypes . uint8) ; const result : EncryptedUint16 =

await client . encrypt (number , EncryptionTypes . uint16) ; const result : EncryptedUint32 =

await client . encrypt (number , EncryptionTypes . uint32) ; Or, we can use the lower-level type specific functions

const resultUint8 =

await client . encrypt_uint8 (number) ; const resultUint16 =

await client . encrypt_uint16 (number) ; const resultUint32 =

await client . encrypt_uint32 (number) ; The returned types from the encrypt function will be of the typeEncryptedUint8 ,EncryptedUint16 orEncryptedUint32 depending on the type you specified.

TheEncryptedUint type sound scary, but are actually pretty simple. It's just a

export

interface

EncryptedNumber

{ data : Uint8Array ; }

export

interface

EncryptedUint8

extends

EncryptedNumber

{ } These types exist in order to enable type checking when interacting with Solidity contracts, and to make it easier to work with encrypted data. However, feel free to use thedata field directly if you prefer. Edit this page

Previous Installation & Basics Next (Un)Sealing