# Decryption & Sealing

When an app wants to read some piece of encrypted data from a Fhenix smart contract, that data must be converted from its encrypted form on chain to an encryption that the app can read and the user can decrypt.

There are two ways to return encrypted data to the user:

1. Sealed Box Encryption
2. The data is returned to the user usingsealed box encryption
3. from NaCL. The gist of it is that the user provides a public key to the contract during a view function call, which the contract then uses to encrypt the data in such a way that only the owner of the private key associated with the provided public key can decrypt and read the data.
4. From a contract perspective, this is done by using theFHE.sealoutput
5. function, which takes the data to be sealed and the public key of the user, and returns an encrypted blob.
6. This data can then be decrypted using fhenix.js, or manually by using the caller's private key.
7. Standard Decryption
8. Alternatively, Fhenix supports standard decryption as well. If some data needs to be decrypted for public access, that can be done as well and a plaintext value is returned to the caller.
9. This can be done using theFHE.decrypt
10. function.

## Sealed Data Format

If you are usingfhenixjs , you don't have to worry about parsing the raw sealed data that is returned fromsealoutput orseal .

However, developers have the option to unseal this data manually. As we described above, the data is encrypted usingsealed box encryption .

The data itself is encoded in the following format:

{ "version": "x25519-xsalsa20-poly1305", "nonce": "", "ephemPublicKey": "", "ciphertext": "" }

### Metamask Compatability

The encryption schema and structure matches the one used by Metamask'seth_decrypt function. This means that we can consume sealed data directly from metamask, which can provide a more engaging experience for the user of a dApp - You can fetch an address's public key using theeth_getencryptionpublickey method, and seal data for that specific public key (either as a permit or by using the public key directly), and then use Metamask'seth_decrypt call to provided a guided decryption experience.

Warning Metamask'seth_getencryptionpublickey andeth_decrypt methods are deprecated. We provide this compatability to demonstrate compatability with native wallet encryption/decryption. We aim to maintain compatability as new standards for Encryption on Ethereum emerge

## Examples

### Sealed Box Encryption

import {FHE} from "@fhenixprotocol/contracts";

function sealoutputExample(bytes32 pubkey) public pure returns (bytes memory reencrypted) { euint8 memory foo = asEuint8(100); return foo.seal(pubkey); }

### Decryption

import

{ FHE }

from

"@fhenixprotocol/contracts" ;

function

sealoutputExample ( )

public pure returns

```
( uint8 decrypted )
{ euint8 memory foo =
asEuint8 ( 100 ) ; return
FHE . decrypt ( foo ) ; }
```

## Metamask Unsealing

```
async
getPub ( )
{ const provider =
new
BrowserProvider ( window . ethereum ) ; const client =
new
FhenixClient ( { provider } ) ; const accounts =
await
window . ethereum . request ( {
method :
'eth_requestAccounts'
} ) ; const keyResult =
await provider . send ( 'eth_getEncryptionPublicKey' , [ accounts [ 0 ] ] ) ; const pk =
0x { this . base64ToHex ( keyResult ) } ; this . showNotification ( pk ) ; } async
unseal ( )
{ const provider =
new
BrowserProvider ( window . ethereum ) ; const client =
new
FhenixClient ( { provider } ) ; const accounts =
await
window . ethereum . request ( {
method :
'eth_requestAccounts'
} ) ; const result =
await provider . send ( 'eth_decrypt' ,
[ this . sealedInput , accounts [ 0 ] ] ) ; const plaintext =
this . toString ( result ) ; this . showNotification (Unsealed result: { plaintext } ) ; }
```

Taken from the encryption & unsealing tool Edit this page

Previous Next 🌳 Select Inputs If...else