Handling Encrypted Inputs

Fully Homomorphic Encryption (FHE) contracts handle encrypted data differently from standard Solidity contracts. Developers can receive encrypted inputs primarily in two ways:

1. UsinginEuintXX
2. Structs:
3. function
4. transferEncryptedToAccount
5. (
6. address to
7. ,
8. inEuint32 calldata encryptedBalance
9.)
10. public
11. {
12. _updateAccountBalance
13. (
14. to
15. ,
16. FHE
17. .
18. asEuint32
19. (
20. encryptedBalance
21.)
22.)
23. ;
24. }
25. Using Raw Bytes:
26. function
27. transferEncryptedData
28. (
29. address to
30. ,
31. bytes calldata encryptedData
32.)
33. public
34. {
35. _storeEncryptedData
36. (
37. to
38. ,
39. FHE
40. .
41. asEuint32
42. (
43. encryptedData
44.)
45.)
46. ;
47. }

Encrypted Data Structures

Different types of encrypted data can be defined:

- inEbool
- : Encrypted boolean.
- inEuint8
- : Encrypted unsigned 8-bit integer.
- inEuint16
- : Encrypted unsigned 16-bit integer.
- inEuint32
- : Encrypted unsigned 32-bit integer.

Advantages ofinEuint

Over Raw Bytes

UsinginEuint over raw bytes ensures type safety and readability. It also provides a structured approach that integrates well with the FHE.sol library's functions.

Examples

1. Voting in a Poll:
2. function
3. castEncryptedVote
4. (
5. address poll
6. ,
7. inEbool calldata encryptedVote
8.)
9. public
10. {
11. _submitVote
12. (
13. poll
14. ,
15. FHE
16. .
17. asEbool
18. (
19. encryptedVote
20.)
21.)
22. ;
23. }
24. Setting Encrypted User Preferences:
25. function
26. updateUserSetting
27. (
28. address user
29. ,
30. inEuint8 calldata encryptedSetting
31.)
32. public
33. {
34. _applyUserSetting
35. (
36. user
37. ,
38. FHE
39. .
40. asEuint8
41. (
42. encryptedSetting
43.)
44.)
45. ;
46. }

inEuint

vs.euint Types

- inEuint
- types are used for handling incoming encrypted data.
- euint
- types are used for data already processed and in use within the contract.

Conversion Necessity

Conversion frominEuint toeuint is necessary for the data to be compatible with FHE operations. This ensures that only correctly formatted and expected encrypted data is processed.

This is done using theFHE.asEuintXX function, whereXX is the bit size of the encrypted data.

Gas Cost Implications

Attempting to storeinEuint types directly in storage can lead to prohibitively high gas costs due to the large size of encrypted data. It's generally recommended to avoid storing these directly and instead process them as needed.

Best Practices

When handling encrypted inputs, always ensure the data integrity and security of your contract's operations. Use the structuredinEuint types for clearer and safer code, and be mindful of gas costs when designing your contract's data handling strategies. Thorough testing and consideration of security implications are essential in maintaining the robustness and reliability of your FHE-based smart contracts. Edit this page

Previous Next Getting Started Outputs