Supported Types and Operations

The library exposes utility functions for FHE operations. The goal of the library is to provide a seamless developer experience for writing smart contracts that can operate on confidential data.

Types

The library provides a type system that is checked both at compile time and at run time. The structure and operations related to these types are described in this sections.

We currently support encrypted integers of bit length up to 32 bits.

These encrypted integers behave as much as possible as Solidity's integer types. However, behaviour such as "revert on overflow" is not supported as this would leak some information of the encrypted integers. Therefore, arithmetic oneuint types isunchecked , i.e. there is wrap-around on overlow.

In the back-end, encrypted integers are FHE ciphertexts. The library abstracts away the ciphertexts and presents pointers to ciphertexts, or ciphertext handles, to the smart contract developer. Theeuint types arewrappers over these handles.

name Bit Size Usage euint8 8 Compute euint16 16 Compute euint32 32 Compute inEuint8 8 Input inEuint16 16 Input inEuint32 32 Input

Operations

There are three ways to perform operations with FHE.sol:

Using Direct Function Calls

Direct function calls are the most straightforward way to perform operations with FHE.sol. For example, if you want to add two encrypted 8-bit integers (euint8), you can do so as follows:

euint8 result

FHE . add (lhs , rhs) ; Here, lhs and rhs are your euint8 variables, and result will store the outcome of the addition.

Using Library Bindings

FHE.sol also provides library bindings, allowing for a more natural syntax. To use this, you first need to include the library for your specific data type. For euint8, the usage would look like this:

euint8 result

lhs . add (rhs) ; In this example, lhs.add(rhs) performs the addition, using the library function implicitly.

Utilizing Operator Overloading

For an even more intuitive approach, FHE.sol supports operator overloading. This means you can use standard arithmetic operators like +, -, *, etc., directly on encrypted types. Here's how you can use it for adding two euint8 values:

euint8 result

lhs + rhs ; With operator overloading, lhs + rhs performs the addition seamlessly.

Comparisons

Unlike other operations in FHE.sol, comparison operations do not support their respective operators (e.g.> ,< etc.). This is because solidity expects these operators to return a boolean value, which is not possible with FHE. Intuitively, this is because returning a boolean value would leak information about the encrypted data.

Instead, comparison operations are implemented as functions that return anebool type.

tip Theebool type is not a real boolean type. It is implemented as aeuint8

Supported Operations

tip A documented documentation of each and every function in FHE.sol (including inputs and outputs) can be found inFHE.sol All operations supported by FHE.sol are listed in the table below:

Note that all functions are supported in both direct function calls and library bindings. However, operator overloading is only supported for the operations listed in the table (solidity please support operator overloading for boolean return types!).

name FHE.sol function Operator Addition add + Subtraction sub - Multiplication mul * Bitwise And and & Bitwise Or or | Bitwise Xor xor ^ Division div / Reminder rem % Shift Right shr

> Shift Left shl << Equal eq n/a Not equal ne n/a Greater than or equal gte n/a Greater than gt n/a Less than or equal lte n/a Less than lt n/a Min min n/a Max max n/a Negative neg n/a Not not n/a Require req n/a Decrypt decrypt n/a Seal Output sealOutput n/a danger At the moment it is not possible to doebool result = (lhs == rhs) and others that return a boolean result. This is because FHE.sol expects aebool , while Solidity only allows overloading to return a regular boolean. Instead, we recommendebool result = lhs.eq(rhs) . danger Using require and decrypt in a TX is dangerous as it can break the confidentiality of the data. Please refer toUseful-Tips to read some more tip Division and Reminder of devisions by0 will output with an encrypted representation of the maximal value of the uint that is used (Ex. encrypted 255 for euint8) Edit this page

Previous Type Conversions Next Useful Tips