Select vs If...else

Writing code with Fully Homomorphic Encryption (FHE) introduces a fundamental shift in how we handle conditionals or branches in our code. As you already know, with FHE, we're operating on encrypted data. This means we can't use typicalif...else branching structures, because we don't have visibility into the actual values we're comparing.

For example, this will not work:

euint32 a

FHE . asEuint32 (10) ; euint32 b =

FHE . asEuint32 (20) ; if

(a . lt (b))

{ return

FHE . decrypt (a) ; }

else

{ return

FHE . decrypt (b) ; } When writing Solidity contracts for our blockchain, you'll need to consider all possible branches of a conditional at the same time. It's somewhat akin to writing constant-time cryptographic code, where you want to avoid timing attacks that could leak information about secret data.

To handle these conditionals, we use a concept called a "selector". A selector is a function that takes in a control and two branches, and returns the result of the branch that corresponds to the condition. A selector is like a traffic signal that decides which traffic to let through based on the color of the light (control signal).

In Fhenix, we utilize this by calling theselect function. It's a function that takes in a condition and two inputs, and returns the input that corresponds to the state of the condition. You can think of this like a ternary boolean conditional (condition ? "yes" : "no"), but for encrypted data.

Let's take a look at an example ofselect usage from a Blind Auction Smart Contract: TBD(ADD LINK):

ebool isHigher

existingBid . lt (value) ; bids [msg . sender]

=

FHE . select (isHigher , value , existingBid) ; In this snippet, the bidder is trying to place a new bid that is higher than their existing one. Thelt function checks if the existing bid is less than the new value and assigns the result toisHigher (the result is of typeebool).

ThenFHE.select takes over. IfisHigher is true (remember, this is still an encrypted boolean, not a plaintext one), it returns thevalue (the new bid), otherwise, it returnsexistingBid . This gets assigned tobids[msg.sender] , effectively replacing the old bid with the new one if the new one is higher.

The crucial part here is that all these operations take place on encrypted data, so the actual value of the bids and the result of the comparison stay concealed. It's a powerful pattern to handle conditional execution in the context of FHE data, maintaining privacy without sacrificing functionality. Edit this page

Previous Next Outputs Require Statements