ð Parallel Decryptions

In Fully Homomorphic Encryption (FHE) schemes, decryption can be slow and computationally expensive, especially when using aThreshold Network . Additionally, since a Threshold Network functions as a sort of network in itself and serves as an oracle for the main blockchain, it cannot perform decryption synchronously, since decryptions may take too long to be practical or the Threshold Network might be busy or unavailable. To address this issue, Nitrogen introduces Parallel Decryptions. This feature performs asynchronous decryption while providing the experience of synchronous decryption. Instead of using callbacks, as in traditional oracle-based protocols, Fhenix allows developers to write code normally, without needing to be concerned that a decryption request is being sent to an external component.

Better developer and user experienceâ

This approach improves the whole experience in two key ways:

- Developers can write clean, cohesive code without worrying about asynchronous decryption.
- Users and dApps avoid the need for multiple transactions to handle decryption callbacks.

euint32 encrypted ; uint32 decrypted ;

function

decryptInFhenix ( )

public

{ // Beginning of function logic ..

decrypted

FHE . decrypt (encrypted) ;

// Continue of function logic.. }

// -------- Otherwise.. -------- // (*simplified* example of callback-based decryption)

function

requestDecrypt ( )

public

{ // Beginning of function logic .. FHE . decryptWithCallback (encrypted) ; }

// This function will be called after some time in a different TX function

receiveDecryptCallback (uint32 result)

public

{ decrypted = result ; // Continue of function logic.. } As shown in the example above, Fhenix's parallel decryption approach allows for clean, straightforward code that reads like normal synchronous operations. Developers can write their business logic in a natural flow, with decryption seamlessly integrated into their functions. This makes the code more maintainable and easier to understand, while Fhenix handles all the complexity of asynchronous decryption behind the scenes.

In depthâ

Behind the scenes, decryption requests are detected in transactions and sent to the appropriate decrypting party (whether Threshold Network or local decryptor). While these decryption requests are processed, the transactions are pushed to a special queue (a sort of tx-mempool). After all the decryption requests for a transaction are completed, the transaction is included in a block and executed.

note Since Fhenix is an L2 solution with a sequencer, no traditional mempool is in place. As a result, there may be a slight delay before the transaction is included and executed in a block, potentially leading to a timeout of the original transaction (tx) receipt. To handle this, periodically query the tx hash in your UI. // Example of handling parallel decryption transactions in a frontend application async

function

handleDecryptTransaction (contract , methodName ,

```
... args )
{ try
{ // Send the transaction that includes decryption const tx =
await contract [ methodName ] ( ... args ) ;
// Get initial transaction receipt let receipt =
await tx . wait ( ) ;
// If the receipt times out, periodically check the transaction status if
( ! receipt || receipt . status
===
'pending' )
{ receipt =
await
pollTransactionStatus ( tx . hash ) ; }
return receipt ; }
catch
( error )
{ console . error ( 'Transaction failed:' , error ) ; throw error ; } }
// Helper function to poll transaction status async
function
pollTransactionStatus ( txHash , maxAttempts =
20 )
{ let attempts =
0 ;
while
( attempts < maxAttempts )
{ const receipt =
await provider . getTransactionReceipt ( txHash ) ;
if
( receipt && receipt . status
===
1 )
{ return receipt ; }
// Wait 3 seconds before next attempt await
new
Promise ( resolve
=>
setTimeout ( resolve ,
```

3000)) ; attempts ++ ; }

throw

new

Error ('Transaction timeout: Decryption taking longer than expected') ; }

// Usage example const contract =

new

ethers . Contract (address , abi , signer) ;

try

{ const receipt =

await

handleDecryptTransaction (contract , 'decryptInFhenix') ; console . log ('Transaction completed:' , receipt) ; }

catch

(error)

{ console . log ('Failed to process decryption:' , error) ; }

Edit this page

Previous ð¡ï¸ Security Zones Next ð¯ Useful Tips