ð¡ï¸ Security Zones

Why we need Security Zonesâ

In various use cases, scenarios, and applications, different levels of security, trust, and performance may be required. However, it is preferable to have everything operate within a single blockchain, as deploying across multiple networks (blockchains) is cumbersome and has its own unique challenges, such as bridging, coordination and more. Furthermore, different parts of a single application may require unique combinations of various configurations. Settling on a single configuration would typically compromise overall security and performanceâleaving the application only as secure as its weakest or as fast as its slowest component.

What are Security Zonesâ

Fhenix developed Security Zones to address this problem. With Fhenix, multiple encryption key sets (i.e., Security Zones) can coexist on a single network. Previously, we were limited to optimizing a single cryptographic key set to meet average security and performance needs. Now, with multiple Security Zones, Fhenix can tailor encryption configurations to specific use cases, eliminating the need to compromise on security, performance, or trust.

For example, highly sensitive information can be protected with maximum security guarantees by using a Threshold Network for encryption. For less sensitive data, a local key set controlled by the sequencer may suffice, offering better performance with slightly lower but acceptable trust assumptions. Similarly, some applications may prefer an encryption scheme that specializes in small ciphertext sizes but slower computation, while apps may need to optimize for transaction price above everything else. Security zones will allow having different schemes with different configurations, all under one roof (or rather, one network - Fhenix).

Important Dev Hint An important nuance of Security Zones is that two ciphertexts that are encrypted using two different security zones are not compatible, since every Zone represents a different set ofFHE keys . For example, we cannot compute enc(a) + enc(b), where a and b were encrypted using different Zones. Note Currently, creating new Security Zones is not dynamically supported, which means that users cannot implement this feature on their own. Rather, they must choose between the existing, pre-generated Security Zones available with Nitrogen on launch. However, we may add a dynamic Security Zones feature in the future.

Security Zone in Nitrogenâ

In Nitrogen, we introduce two Security Zones to demonstrate the concept:

Threshold Network (zone 0 - default)â

This zone uses a Threshold Network. The benefit of this approach is that it provides the end user with additional protection from malicious actors. For further reading on the Threshold network,see here .

Local (zone 1)â

This zone uses a local key set held by the sequencer (this is how previous Fhenix testnets handled the decryption key). The intention of this zone is to offer faster decryption but require more trust from the user with the tradeoff of assuming a trusted sequencer.

Dev Hint Whether you are porting a contract from a previous Fhenix version, or creating a new one - you don't have to do anything in particular to use the Threshold Network. Security Zone 0 is the default one - meaning that you don't need to modify your code to use it! See the code example below ð

Code Exampleâ

Contract.sol function

addSecZone0 (n1 inEuint32 , n2 inEuint32)

{ euint32 first = FHE . asEuint32 (n1) ;

// This implicitly uses Security Zone 0 euint32 second = FHE . asEuint32 (n2 ,

0) ;

// This also uses Security Zone 0

euint32 result

first + second ;

// result will be marked 'security zone 0' automatically }

function

addSecZone1 (n1 inEuint32 , n2 inEuint32)

{ euint32 first = FHE . asEuint32 (n1 ,

1) ;

// To use a non-default SZ, you have to be explicit euint32 second = FHE . asEuint32 (n2 ,

1) ;

euint32 result

first + second ;

// result will be marked 'security zone 1' automatically } index.js const client =

new

FhenixClient ({ provider }) ;

let first =

await client . encrypt_uint32 (5) ;

// This implicitly uses Security Zone 0 let second =

await client . encrypt_uint32 (1332 ,

0) ;

// This implicitly uses Security Zone 0 await contract . addSecZone0 (first , second) ;

first

await client . encrypt_uint32 (5 ,

1) ;

// To use a non-default SZ, you have to be explicit econd =

await client . encrypt_uint32 (1332 ,

1) ; await contract . addSecZone1 (first , second) ;

When can I specify a Security Zone?â

InSolidity , you will notice that you can specify a Security Zone only when calling functions that are creatingeunitX types, meaning inFHE.asEuintX() functions. This is because theeuintX types include metadata about the Security Zone that the ciphertext was encrypted with.

Infhenix.js , you can specify a Security Zone only in theencrypt() functions because you need to encrypt the data with the desired Security Zone's public key.

Note The current security zones are implemented to demonstrate the concept with tradeoffs between security and performance. However, they are not indicative of a production deployment. Edit this page

Previous ð§âãï¸ Types and Operations Next ð Parallel Decryptions