Replacing our solution hash

So far in this tutorial, the user sends the plaintext solution to the crossword puzzle smart contract, where it's hashed and compared with the known answer.

This works, but we might want to be more careful and avoid sending the plaintext solution.

Why?

Blockchains rely on many computers processing transactions. When you send a transaction to the blockchain, it doesn't immediately get processed. In some Layer 1 blockchains it can take minutes or longer. On NEAR transactions settle within a couple seconds, but nonetheless there's a small period of waiting time.

When we previously sent the crossword puzzle solution in plain text (via the parametersolution tosubmit_solution) it means it's visible to everyone before it gets processed.

At the time of this writing, there haven't been outstanding incidents of validators "front-running" transactions, but it's something to be aware of. Front-running is when a validator sees a transaction that might be profitable and does it themselves.

There have been several incidents of this and it continues to be an issue.

Real-life example of a puzzle being front-run. ReadAnish Agnihotri's thread

How?

We're doing to do something unique — and frankly unusual — with our crossword puzzle. We're going to use function-call access keys in a new way.

Our crossword puzzle smart contract will add a function-call access key to itself. The private key is derived from the solution, used as a seed phrase.

What's a seed phrase, again? A private key is essentially a very large number. So large that the number of possible private keys is approaching the estimated number of atoms in the known universe.

It would be pretty long if we wrote it down, so it's often made human-readable with numbers and letters. However, even the human-readable version is hard to memorize and prone to mistakes.

A seed phrase is a series of words (usually 12 or 24 words) that create a private key. (There's actually a bit more to it .)

Seed phrases typically use aBIP-30 wordlist , butthey do not need to use a wordlist or have a certain number of words. As long as the words create entropy, a crossword puzzle solution can act as a deterministic seed phrase. So when we add a new puzzle, we'll use theAddKey Action to add a limited, function-call access key can thatonly call thesubmit_solution method.

The first user to solve the puzzle will essentially "discover" the private key and call that method. Think of it like a safe that contains a function-call access key.

Open the safe using answers to the puzzle, revealing the function-call access key. Art bysoulless.near

Our methodsubmit_solution no longer needs to hash the plaintext answer, but instead looks at the key that signed this transaction. Cool, huh!

Onboarding

In the previous chapter we implemented login to the crossword, but this requires a person to have a NEAR account.

If the end user is discovering a key that exists on the crossword contract, they don't even need a NEAR account, right? Well, that's partly accurate, but we'll still need to send the prize in NEAR somewhere.

What if we could make the winner an account on the fly? Is that possible? Yes, and that's what we're going to do in this chapter. Edit this page Last updatedonAug 9, 2022 byDamián Parrino Was this page helpful? Yes No

Previous Overview Next Seed phrase logic