Ideas on how to use capsules for implementing a private-land price feed using anopen-oracle approach without having to rely on storage at all.

Searching for use cases that justify the existence ofcapsules in the Aztec Network,@rahul-kothari and @alexghr came up with the idea of oracles, so I figured I would front-run them and turn it into a post. Hopefully this will be the first of a series Capsules: What are they good for?

.

A quick primer on capsules

Capsules are one of many@LHerskind brainchildren. A capsule is an arbitrary piece of data that a contract in private-land can request from the PXE via an oracle call at any time. Since data is provided directly by the user's PXE, this data is fully unconstrained

, meaning it needs to be somehow verified by the contract. Capsules allow a developer to inject data out-of-band into a contract when needed, though it requires the PXE or Dapp to understand what capsules need to be provided for a given execution run.

A good example use-case for capsules is AuthWits (which are currently implemented as a dedicated oracle call but could very well be built on top of capsules). An account contract may at any time request the PXE to provide a signature for a given action that is to be carried out on its behalf, such as a typical erc20 transferFrom

, which then the account contract verifies against its own stored public keys. This pattern allows implementing a permit-like transfer in Aztec, while keeping the transfer interface simple, since the signed message is injected directly into the contract that needs it.

But @jaosef challenged us to find other valid use cases for capsules, so here we are.

What is an open-oracle?

The open-oracle is an oracle standard developed by Compound in which a trusted entity publishes data off-chain but signed with their Ethereum private key. Anyone can then grab that data and push it to an on-chain contract that validates the signature. This saves the oracle owner from having to pay the gas for every data update, and instead places the burden on whoever is interested in making use of the oracle. This standard was for instance implemented by Coinbase in early 2020.

twitter.com

mariano.eth |

@nanexcool

@spencernoon @coinbase sign every API endpoint data with a secp256k1 private key, make the Ethereum address public so we can verify the data on-chain. Boom, awesome oracle

7:14 PM - 24 Oct 2019 30

1

How would an Aztec private open-oracle work?

The naive implementation would follow the same pattern: anyone can send a transaction to an oracle contract that creates a slow-update entry with the most recent value, and the oracle contract validates the signature of the payload before accepting it.

But we don't need to store each data entry: we can just provide the data on-the-fly every time we query the oracle, validating it every time we use it. And since this happens in private-land, which happens entirely on the client side, there's no extra gas cost for the signature validation.

However, this has a downside: it breaks the expected interface of an oracle, which should be as simple as a get()

. Let's say you wanted to deposit funds on an aggregator's vault, which in turn called a defi protocol, which in turn called a handful of oracles to gather prices on the assets you're working with. Having to add an extra set of arguments that needs to be passed from one function to the next is annoying to say the least.

But we can easily solve this with capsules: both the data and its signature can be injected directly via a capsule, so the oracle could work without the need for storage and without requiring a different interface.

Show me the (pseudo)code

In a few lines, the oracle would look something like this:

contract LazyOracle: immutable public_key

def get_oracle_data(): payload = get_capsule() signature = get_authwit_for(payload) # remember: get_authwit is just a specialized get_capsule! assert valid_signature(payload, signature, self.public_key) context.set_max_block_number(payload.valid_until) return payload.data

How else could we build this?

An alternative to using capsules, that allows for a simple interface and not using storage, would be to implement an equivalent to Ethereum's TSTORE

, where we store data that is only valid for the span of the transaction. The user could then make a multicall from their account contract, where they first seed all the oracles they need to, and then execute the action that depends on them. Personally, I found TSTORE

more difficult to reason about and more error prone that capsules. But an oracle built on that would be more or less like:

contract LazyOracleWithTStore: immutable public_key transient data

def set_oracle_data(data, valid_until, signature): assert valid_signature(payload, signature, self.public_key) context.set_max_block_number(payload.valid_until) self.data = data

def get_oracle_data(): return self.data

While the code is also relatively simple, the orchestration here feels more complex, since the caller needs to pre-seed the oracle (which takes more function calls, which increase proving time). And introducing TSTORE makes me nervous, but maybe that's just me.