As an alternative to contract upgrades for account contracts, we can allow account contracts to dynamically execute scripts rather than just running its deployed code. Most importantly, we can run these scripts in the context of the current address, without having to implement DELEGATECALL

s. This requires tweaks in the private kernel circuit, as we'll note in the next paragraph, but these shouldn't be too complex.

We allow a contract to make function calls not to an address & selector, but to a vkhash

(ie verification key hash, which should be enough for identifying a circuit). This requires a small change in the call stack item of the app circuit and private kernel circuit. However, it allows us to call into a circuit that's dynamically provided by the execution oracle. The kernel circuit then verifies that the vkhash matches, instead of verifying membership in the contracts tree.

How do we authenticate these calls to dynamic scripts? Same as we authenticate anything else: we let the user's account contract take care of it. If the account contract has a code path that allows a user to provide a vkhash to call along with a valid signature, then that account contract will be able to execute arbitrary scripts supplied by its oracle. This allows us to build a Safe-like modules system, only that modules do not need to be predeployed and preregistered. We just run code and the kernel verifies it, that's it.

This scheme could also be used for implementing internal functions, removing them from the contracts tree. Since we only allow a contract to call to a vkhash in the context of self, having a call to a hardcoded vkhash in the contract's code is equivalent to having a call to an internal function, since no one else can call it and its code is fixed at contract deployment time.

It's unclear what use cases this unblocks, but at the time of this writing the main reason for building something like this is because we can

.

This idea is heavily inspired by @Mike's bytecode abstraction proposal, copied here from an internal discussion via Slack:

Imagine you have a keypair (csk, Cpk)

I'll over-simplify for now. Imagine your 'contract address' is simply Cpk

.

You can sign over any function bytecode that you wish to become "a part of your account contract", using csk

(this is why it doesn't align with account abstraction, because I just accidentally enshrined a signature scheme).

You call the bytecode that you signed over, and you provide the initial private kernel circuit with the signature over the bytecode, to verify against Cpk

.

You basically say "Hey, Mr. Initial Private Kernel Circuit, I now want this to be my bytecode." It might ruin your account contract's state variables (which I guess would be stored against Cpk

). It might break everything. But you get contract upgradeability for free, because the bytecode is whatever you want, whenever you want.