We want to allow developers to make their Aztec contracts upgradeable. This means they'll be able to change the code and behavior of their contracts while keeping the same contract address and state. For example, a token contract might add a new mechanism for transfers while retaining all prior balances.

There's two alternatives under consideration to achieve this:

- delegatecall

and proxies, which is how Ethereum upgrades work. These come with a number of known issues, and are discussed in detail here.

- enshrining upgrades themselves, which unlike Ethereum we can do as we have not finalized the protocol. The overall approach and open questions are discussed in this other document.

This document provides an overview of both approaches to allow us to come to a decision. You can think of it as

TL;DR, how do we decide

[delegatecall

and Proxies](https://forum.aztec.network/t/deep-dive-into-delegatecall-proxy-upgrades/4581).

Pros

- A shared mutable storage solution (eg SlowJoe) for storing class ids is not enshrined, so we could migrate to a better solution in the future.
- Contracts that do not require "concurrent" access (eg account contracts) can use vanilla private storage for storing their class ids, at the expense of an extra commitment and nullifer per call.
- It's a similar model to Ethereum, though said model is not well-liked or easy to understand.

Cons

- Need to develop and enshrine fallback functions (3 flavors of it), just for upgrades.
- Need to develop and enshrine [delegatecall

](https://forum.aztec.network/t/deep-dive-into-delegatecall-proxy-upgrades/4581#delegatecall-3) (3 flavors of it), just for upgrades.

- Proxies will remain obscure, an existing problem in Ethereum.
- Extra function call in all upgradeable contracts (longer proofs, more expensive txs)
- We still need some kind of shared mutable storage, same as when enshrining upgrades, and get all the undesired side-effects (e.g. 'downtime' close to an upgrade).
- Tooling will be complex in order to deal with obscurity (e.g. in block explorers) and detect upgradeability.

Open Questions

- Can we get rid of top-level unconstrained contract functions? If so, we'd only have two execution modes (private and public).

Enshrined Upgrades

Pros

- Easy upgrade setup - just call the upgrade

opcode.

- Simple upgradeability detection.
- Minimal runtime overhead.

Cons

- Account contracts leak the fact that they are account contracts on an upgrade (if they reuse contract classes).
- Requires a new upgrade

opcode (could be pushed for later if we are fine with no detection and broken SlowJoe invariants).

- Enshrined SlowJoe means all calls to upgradeable contracts have a max_block_number

, restricting long proof use cases.

- We need to choose a global delay for upgrades, or make it customisable per contract which makes SlowJoe more complex.
- Slightly more complexity on the Kernel circuit for dealing with upgradeable contract classes.

Open Questions

- Can we provide better privacy for Account Contracts? e.g. hashing with a secret salt
- How much do we enforce network-wide upgrade time coordination?
- Can we get away with not adding a new opcode?
- How do we mark non-upgradeable contracts, so that these don't require the max_block_number

check? e.g. a metadata flag in the preimage

Our Suggestion

Enshrined upgrades feels like the cleaner solution. The added complexity to the kernel circuit is not significant, and the AVM team has said that adding an upgrade opcode would not have a big impact. The opposite is true for proxies, where both the fallback function and delegatecall

require changes to multiple components, and the code of the proxy itself is relatively tricky.