Background

Lido was originally deployed such that it specifies a BLS eth2 pubkey as thewithdrawal credential for all deposits to the deposit contract. Because these keys can't specify smart contracts that are written today, this required the current threshold signature based withdrawal mechanism. While the group involved in this ceremony is relatively diverse and respected, this is still a scenario where $10B+ of ETH withdrawals hinge on a k-of-11 entity threshold committee.

A couple months after Lido's release, a change was made to the eth2 specsupporting a way to specify eth1 addresses at deposit time. This allows specifying a smart contract as the withdrawal adddress, allowing trustless withdrawal logic.

It was always in the plans to move to this new scheme, but below I make the argument that it's worth thinking about now and present a sketch for how this could happen.

The result of this would be that all subsequent

deposits to Lido would have a trustless, smart contract withdrawal scheme rather than relying on the 11 member threshold signature group.

Why now?

So, why not wait until the last minute? Why bother doing anything before merge?

The biggest reason is that the longer we wait, the more ETH that relies on the threshold sig for withdrawal vs. a smart contract. Lido has had an impressive growth curve so far, and if it's true that staking pools may be a winner-takes-most category, a nontrivial fraction of ETH supply may end up reliant on an 11 person threshold signature committee. No matter who is in that committee, this situation is non-ideal for the overall safety of Ethereum and by extension, Lido.

Secondarily, moving to smart contract withdrawals will likely improve the decentralization optics of Lido itself, which may (funnily enough) improve user adoption. The ETH community is still composed of many people who care about the permanence/decentralization of the systems we're building and a push to smart contract withdrawal could win a lot of potential depositors over.

Design

To recap, there are two steps involved:

1. writing/deploying a new withdrawal contract
2. through governance, setting the withdrawal credentials to a value corresponding to this new contract

Here's a sketch of what the withdrawal contract may look like:

contract LidoWithdrawals { using SafeMath for uint256;

```
Lido private lido;

constructor(Lido _lido) {
    // Store Lido DAO contract
    lido = _lido;
}

/**
 * @dev user withdrawal fn
 * @param _value amount of stETH to withdraw
 */
function withdraw(uint256 _value) {
    // must be authorized xfer
    require(lido.allowance(msg.sender, address(this)) >= _value)

    (uint256 _, uint256 _, uint256 beaconBalance) = lido.getBeaconStat();

    // someone should check that this is the right calculation :)
    uint256 ethToWithdraw = _value.div(lido.totalSupply()
                            .mul(beaconBalance);

    // must have enough ETH (i.e. enough BLS sig -> eth1 addr withdrawals have taken place)
    // If contract doesn't, user may need to rely on threshold sig mechanism
    require(address(this).balance >= ethToWithdraw);

    // handle withdrawal
    lido.transferFrom(msg.sender, address(this), _value);
    msg.sender.transfer(ethToWithdraw);
}
```

}

This design is optimized for minimal governance actions/changes to the existing contracts. It doesn't keep track of how much ETH is staked to be withdrawn by the old vs. new withdrawal credentials and instead just fails if there isn't sufficient ETH here.

There is a small UX downside to this: stETH holders who are withdrawing their ETH 'last' may have to withdraw tranches separately between the threshold sig withdrawal and the smart contract withdrawal because there won't be enough in either left to cover their entire withdraw. I think this price is worth paying for the contract simplicity.

Once this contract is written/deployed, the withdrawal credentials should be set as specified here with the eth1 address set to the withdrawal contract's address.

Let's discuss!

This proposal is largely meant to be a starting point for discussion. One thing I hope to convey is how simple

this upgrade could be. That being said, I likely am missing edge cases and other considerations; would love to hear what the community thinks and work towards making this happen.