One immediate challenge in the notarization mechanism as describedhere is: why would notaries even bother to check if a collation is available? Given that most collations are available, isn't it too easy to just skip the availability checking step and always vote 1?

One answer comes in the form ofcollation availability traps: if you vote 1 on the availability of some collation, then later on anyone can challenge you by providing the index of some chunk in the collation, and in order to either (i) recover your deposit, or (ii) notarize any further messages, you would need to respond to the challenge by providing the Merkle branch corresponding to that chunk. If a notary fails to respond within some period (eg. two months), then the notary's entire deposit is lost, and the challenger can get some portion (eg. 33%). This creates an incentive for proposers to sometimes publish unavailable proposals that try to "trap" lazy non-verifying notaries into accepting them, thereby extracting their deposits from them.

But we can strengthen this mechanism further by adding a "proof of custody" mechanism. This works as follows. A notary is required to provide not just a signature of some hash root of the notarized data, but also a Merkle root of an altered data tree, where each 32 byte chunk of data D[i]

is replaced by xor(D[i], x)

for some value x

; the notary must also commit to H(x)

.

When challenged with any index i

, the notary must provide (i) a Merkle branch of D[i]

in the data tree, and (ii) a Merkle branch of xor(D[i], x)

in the altered tree; from these two values x

can be recovered and checked against the hash commitment provided. This ensures that the notary must not just be confident that the data will be available (eg. because a few other notaries have already signed off on it), but also that the notary actually has that data themselves at that specific time

, as otherwise they have no way to generate the altered data tree.

Notaries should also be required to publish the preimage of every key that they submit at some point (eg. if the keys are generated in some deterministic fashion, like k1 = SHA3(masterkey + 0x01)

, k2 = SHA3(masterkey + 0x02)

, etc, then they would have to eventually publish masterkey

); this allows auditors to generate the proofs of custody client-side, and check for every collation if the Merkle roots match.

If an auditor computes the signature root and the root that they compute does not match the root that was provided, then an auditor can determine the index where the notary would not be able to respond to the challenge in maximum log(n) steps. The algorithm is as follows:

- Challenge index 0. Wait to receive the Merkle branch.
- Walk down the Merkle branch that they provide until you find a point where the provided tree and your computed agree (this could possibly be the first point; but it definitely hops at least one step down). At that point, the two trees agree on some node A, but disagree on the parent of A, implying that they disagree on the sister of A. Challenge the index corresponding to the node opposite that point.
- Repeat until you're at the bottom of the tree, and you found a bottom-level index where the provided and computed trees disagree. Challenge that index.

When a challenge succeeds, the deposit of the notary could be split among all challengers, thereby rewarding anyone who helped participate in the binary search process.

The protocol can potentially subsidize some random amount of auditing by giving some proposers the ability to make a "challenge" of whatever index they want to whatever notary they want for free.