this is a copy of what I wrote here:offchain voting

Why

Onchain governance is always expensive because it involves so many transactions. The goal of this document is to discuss an approach to bring most of the voting mechanisms off chain but still have the same level of verifiability with a few assumptions. The goal here is to describe those assumptions, see if they are reasonable or not and maybe get feedback on how to improve this approach even further.

The properties we are looking for are:

near constant gas cost for a vote round

reasonable cost to create a vote

simple approach

on chain verification / audit possible if something doesn't seem right

Design

The goal here is to create two merkle tree:

The vote definition merkle tree. It represents every voting rights

The vote results merkle tree. It represents the result of the vote

The vote definition root is being submitted at the same time as we create the vote onchain.

The vote results are being submitted once the voting has ended.

VoteRoundId

Each round needs to be uniquely identified. To do this, we use the function: hash(Moloch address, proposalId)

which is always unique.

Voting

A Voter should be able to vote as well. To do this, the user needs to signs with his key the following data:

for yes: hash(voteRoundId, 1)

for No: hash(voteRoundId, -1)

for abstain : hash(voteRoundId, 0)

The vote definition merkle tree

Its structure is as follows:

leaf: voter address, voter weight. The hash representation of the leaf (used for parent nodes) is hash (voter address, voter weight)

node: standard merkle tree node, i.e. hash of its children

Each leaf is sorted by the address

The vote results merkle tree

Its structure is as follows:

leaf

it has the entries:

- voteSignature. // the actual vote

- balance (1, 0 , -1)

The hash being used by the parent node is hash(voteSignature, balance)

node

It has the same structure but the function is a bit different

- hashOfChildren
- accumulation (sum of accumulation of its children)

root

then the hash of the root is being published with the accumulation result. The accumulation result is the result of the voting

Happy path

If everyone is happy with the voting result (no fraud has been committed) then the DAO just take the result and acts accordingly.

Fraud

This is the most important part. We describe here the fraud / attack vector, and how this system can respond to them.

Wrong data

If someone tries to vote with the wrong data (not signed by the right key, not the right weight etc …) anyone can publish the proof that the leaf was wrong by publishing the incriminating leaf with its path to root, the proof in the vote definition and show that the data doesn't match.

unknown address

if someone shouldn't be able to vote, anyone can publish a proof that the address doesn't exist in the vote definition tree.

The way to prove that is as follows:

- get an address that should be before and after the missing address
- get all the parents to come with it and prove it's part of the tree

if. the address is smaller or higher than any other address, just prove that the smaller / highest address in the tree definition is not this address and is higher / smaller than the one that has voted

Failsafe

if anything goes wrong, we should make it possible to invalidate an offchain voting and switch back to onchain if a quorum of voters mark it as fraudulent. TBD: do we want to just cancel the vote or switch it to on chain ? maybe configurable ?

Staking

It seems to me that using some form of staking to reward the participants and also having a way to punish bad behaviour makes sense but the details of this part is still TBD

Assumptions

In order for this to work, we make a few assumptions:

- The vote definition tree is a sorted Merkle tree and we assume it is the case (it is possible to add a way to prove fraud here (by showing two nodes that are not in order)
- The vote definition tree is available for anyone to check
- Once the vote happened, the vote result merkle tree is also available to anyone to check

Smart Contract design

TBD: Let’s start with the high level design and then I can work on the smart contract itself

I know there are still a few things that need to be defined but I wanted to have a sense.

Does this approach make sense?

Am I missing something?

Is there an optimisation that is easy enough to do that I’m not seeing here ?