SACI - Simplified anti-collusion infrastructure

MACI is a great specification for anti-collusion infrastructure by Vitalik. In the following, a similar yet simplified specification is proposed for the special use case of voting. Though, it should be usable in any anti-collusion application. The intent is to reduce UX friction and make MACI easier to develop. I would love to get your thoughts on the disadvantages of this solution compared with MACI

Comparison with the original MACI spec:

- Voting does not require paying on-chain fees (unless the operator tries to censor voters)
- No multi-key management
- Same liveness and safety guarantees as MACI
- Simplified circuits: Removing vote decryption in the zk-proof

Specification:

Setup:

The setup is similar to the MACI spec: There is a registry of public keys that are allowed to vote (each public key should hold a bond to disincentive sharing the PK). Additionally, there is a voting operator with a known public key. The operator must collect all the votes and produce a zk-proof that proves a correct tallying up of the votes.

Actions:

There are two actions for each user:

- Voting: Vote for X
- Invalidation: Invalidate all votes from the sending account from this message on

Process:

1. There will be n “voting blocks”. A voting block is a collection of vote messages and invalidations batched together into a Merkle tree with a unique Merkle root hash called the voting_block_root

. The Merkle tree should be sparse at the end of a block, meaning that the last (random) Merkel tree leaves are just empty.

1. Users send signed voting/invalidation messages to the operator valid for 1 voting block, the message can be encrypted using the operator’s public key. In a single block, either only voting messages or only invalidation messages can be included from a certain account.
2. Operator builds the voting block

of signed, decrypted messages and publishes the voting_block_root

on-chain

1. Each voter then receives a Merkle proof of their signed voted/invalidation messages
2. If a voter does not receive their Merkle proof of their signed voted/invalidation, they should assume that their cast was not included and should resubmit their cast for the next voting block.
3. If users are censored by the operator, they can put their vote on-chain in plain text. On-chain vote messages get put into another Merkel tree calculated on-chain, called the on_chain_voting_block_root

At the end, the official voting result will be calculated by the following algorithm: All votes from all members are ordered sequentially by the voting blocks and the leaf position of the vote in the block. The official vote of a voting member is the last vote before any invalidation message from the user. If there is no invalidation message, then the last vote is counted.

After the vote outcome computation, the operator must provide zk proof that shows

- They applied all messages mentioned in the n voting_block_roots

on the initial state root, yielding the first intermediate state root of voting results.

- They applied all messages mentioned in the on_chain_voting_block_root

on the first intermediate state root

- They tallied up all votes from the final state root, and it matches the official vote result

Properties:

Collusion resistant:

- Only the operator and the owner of the private key know whether there was another vote between a shown vote and an invalidation message. Hence, the users are not bribable. Notice that no user will be able to be proof that their vote was the last message included in a block n (and that they sent the invalidation message on block n+1), as the operator puts a random amount of empty leaves at the end of each block.

Censorship resistant:

- As the votes can be published on-chain. Nobody knows, besides the operator and the user, whether these on-chain votes are valid.

UX improvements:

- Gasless voting: Users only need to send signed tx to an operator API instead into the blockchain.
- The footprint of a vote has been reduced to a Merkle root of transactions, instead of publishing all transactions on-chain. Hence, it should be cheaper in general.

Challenges:

- If an operator censors a user, they can force them to vote unencrypted on-chain. Is this worse than in the original specification? Mostly no, as in the original specification, a misbehaving operator could also reveal all votes from one single voter to a bribing party or to the public: The operator could simply show the decrypted votes and zk-prove that these votes are indeed all information for this voter. Though, censoring due to unintended technical difficulties will have a bigger impact on the voting process in the current specification than in the original MACI spec.

Additional:

Several operators:

In some settings, e.g. if some people only trust an entity A and others only trust an entity B to be non-revealing + do not censor, the algorithm can be modified such that there are two operators: entity A and entity B. In such a scenario the process could look like this:

- Each voter chooses their “vote operator entity” on-chain
- Each voter is only allowed to vote with their chosen operator
- The voting process is exactly according to the upper specification
- In the end, the two voting results from the two voting operators are tallied up