Subsenate

Doron Zarchy

Abstract

E-voting systems, which operate over a decentralized platform, may not scale well when used for too many elections or when unprecedented numbers of users participate in the process. As an alternative, we propose Subsenate which is a zero knowledge-based scheme for voting over blockchain which grants permission to vote only to a random subset of the registered voters. Subsenate is robust in its ability to defend against election outcome manipulation. Subsenate keeps the identities of the users who are selected as eligible voters hidden both before and after the vote. Subsenate incentivises truthful behavior by rewarding only the voters who vote with the majority.

1 Introduction

A decentralized and secure E-voting system has a clear great deal of potential. Election can be not only for official elects, but can be as well used for specific bill, prediction markets, open disputes, or as an oracle that interfaces the blockchain with e external events. However, system that processes many elections at a high speed that involves many users may not scale well as doing so may require interaction amongst too many users and may consume too much bandwidth. Especially if the election are about bills in which the voter has no background knowledge about.

To address this, we propose a scheme called Subsenate for choosing a community of judges to resolve the challenge of scalability. Subsenate only tallies the votes of a random subset of the users which has been determined in a reliable and manipulation resilient manner where each registered user has an equal change to be chosen as a eligible voter. In some environment such as blockchain, voters are assumed to act as oracles to an external events where event can be easily checked (e.g., who won the election at a certain location and time), Subsenate is designed to incentivise votes to behave truthfully (e.g., in a setting of interfacing with a) by rewarding only the users who vote with the majority.

We also add a mechanism to allow coersion resistant, namely that no coercer or vote buyer can be confident

Democracy, in its philosophical sense, should be governed by the majority while maintaining the rights of the minority. In ordinary elections, when tallying all the registered votes, the majority has the entire power to determines all the outcomes while the minority has no power at all. We believe that power should be distributed more equally. In our setting, there is a chance of a minority to

decide the outcome (the chance depends on the subset size, where larger size create a bias to the majority and the smallest subset of size 1 distributed the power proportionally to the size of the parties)

The requirements for the system are the following:

1. a decentralized mechanism for selecting a random set of the users where each user has equal chance to become elected
2. hiding the identities of the eligible voters
3. hiding the identities of the voters (the eligible voter who practiced their right to vote)
4. rewarding honest voters
5. hiding the identity of election creator

2 Overview of the protocol

We now describe a very high level overview of the protocol:

1. election creator sends a transaction to "creation election" smart contract that contains the list of candidates, majority type (simple or super) , a deposit, and a URL to website that describe the background of the election. The deposit is used for rewarding the honest voters
2. registration to the election is done by sending a special transaction that contains commitment to the voter identity to a smart contract
3. after an election is set, the voters compute a random seed
4. subset of eligible voter is selected according ot the random seed
5. each eligible user publishes a proof attesting membership of the subset of eligible voters and a membership to the set of registered voters
6. miners approve all the votes whose proofs are verified
7. the election outcome is set to be the candidate that gets the majority of the ballots of the approved voters
8. miners unlock the collateral of the election creator and send the coins to the "winners" of the election

3 Assumptions

Although Subsenate grants equal chance to all eligible voters, we assume that the voting power splits proportionally to the stakes. We do it by assuming that anyone who register the system utilize it's entire stakes for voting (by assuming the existence of wallets which split the coins to different addresses

keys) –TBD, privately distribute the deposit according to the stakes, i.e., the reward split according to the stakes. The security of the system is based on the assumption that the majority of the stakes is honest. In addition we assume that the originators of the proofs are unknown. Since some blockchains (such as Ethereum) requires signing the transactions, we assume a mixed network that hides the massage senders – elaborate–.

4 Election Creation and Registration

An election creator sends a transaction to a smart contract which specifies the following:

1. a deposit of the election creator (which will be later distributed among the honest voters)
2. the type of election rule can be one of the following:
   (a) simple majority, i.e., candidate who received most of the ballots (if there are two candidate then its a simple majority)
   (b) super-majority, defined by a parameter α. The election is finalized only if more than α-fraction of the voters voted to some candidate.
3. the duration where voters are allowed to vote. Specifically, the smart contract sets an interval $[T_1, T_2]$ for which voter's vote is considered only between blocks t_1 and t_2.
4. the size, in expectation, of the subset of voters, denoted R
5. (*) a URL whose website contains content of the election/dispute.
6. list of candidates V.

Registering to the election requires sending a transaction to a smart contract that contains a commitment to the voter's identity and a solution to some cryptographic puzzle (e.g., finding a preimage of some hash function) which serves as an anti-spam mechanism.

5 Voting

The voter cannot send the vote in plaintext as it would expose its identity, hence a commitments of both to the voter's identity as well as to the vote are published (Subsenate does not expose the link between the voter and his commitment). The attestation to the commitment is done via zkSNARK.

The voting proceeds in two phases: In phase one, a the voter sends a commitment to its identity and a commitment to its vote. Formally, let $\mathbb{F}_p$ be a field of prime order p. The user computes a commitment cm to voter's identity as follows:

1. select a secret key sk uniformly from $\mathbb{F}_p$
2. select nonces t_1 and t_2 uniformly from $\mathbb{F}_p$
3. computes the commitment $cm = H(sk, t_1, t_2)$
4. sends cm and to the smart contract

Miner can now create a Merkle tree of registered voters.

The actual voting is done by publishing a vote commitment c_v that is computed as $c_v = H(sk, v, SR, u, pk_{new})$ where

1. $v \in V$ is the vote.
2. pk_{new} is a derivation from the new public key (for the subsequent address), i.e., $pk_{new} = g^{sk}$, where $g \in \mathbb{G}$ is a generator of a multiplicative group
3. select u uniformly from $\mathbb{F}_p$. u is a nullifier which guarantees that all votes are unique (similar to the use in Zcash)
4. SR is a serial number of the election

5.1 Selecting the Subset

A subset of size r is randomly selected from a set of voters according to some random beacon that is computed by some distributed protocol. Several distributed random number generation (distRNG) protocols has been proposed for computing random beacon that guarantees unpredictability, bias-resistance, and public-verifiability of the protocol transcript that are based on Byzantine fault-tolerant random number generator (BFT-RNG) and Public Verifiable Secret Sharing (PVSS). In particular these protocols ensures that the result cannot be manipulated despite the presence of adversaries who can halt or deviate from the prescribed protocol. Each of these protocols makes an assumption on the number of parties the adversary controls. Several schemese achieves a resiliency of $f < \frac{N}{2}$ number of parties controlled by the adversary [**?**], [**?**], and also implemanted [**?**]. Among these, in recent work by Cascudo et al. [] they designed a system called Albatross that generates $\Theta(n^2)$ uniformly random values where n is the number of parties execute this protocol (with an amortized computational complexity of $\Theta(logn)$ per party). This, not very expensive cost may allow the participation in the distRNG of all voter. Then, even if the majority of the parties is dishonest then, the system works as expected since the majority are to decide the outcome in an election anyway. A cheaper approach is to have some trusted parties that execute the distRNG protocol (this may be enforced by deposit in a blockchain, if some party halts or no cooperate). After the distRNG protocol outputsa random number S, it can be used as a seed to selecting the subset. A simple approach is to consider voter to be eligible if $H(S, t_1) < R$ where t_1 is the nonce which the user committed to in phase one (or a verifiable random function can be used).

5.2 zkSNARK statement

The voter generates a zkSNARK proof $\Pi = (\Pi_1, \Pi_2, \Pi_3, \Pi_4)$ for the following four NP statements:

1. $cm = H(sk, t_1, t_2)$ where cm is the commitment published in the smart contract when the voter registers the election. The voter proves her identity by showing that she knows random nonces t_1 and t_2.
2. $c_v = H(sk, v, SR, u, pk_{new})$, the commitment to the vote v also includes the secret key of the voter, the serial number of the vote, the nullifier, and the public address for which assets from deposit should be transferred.
3. $Root = \mathbb{H}(mp, cm)$. Proving that cm is a member in the set of commitments by proving that the Merkle root is a result of the hashing cm and the witness (Merkle path) . The Merkle root is public.
4. $H(S, t_1) < R$ proves that the voter is eligible.

where the public input is:

1. Root: Merkle root of the commitment set
2. u: a nullifier
3. v: vote
4. pk_{new}: voter's new address
5. N: expected number of voters
6. S: random seed
7. SR: serial number of the vote

the private input:

1. cm: commitment to the vote
2. mp: a Merkle path from cm to root
3. t_1 and t_2: random nonce
4. sk: private key (but not used for signature)

The sender can simply send the proof Π along with nullifier u, a vote v, and the serial number of the election. Note that in Ethereum like systems where the sender's address is part of the transaction, the transaction can be used to break anonymity. Therefore a mixed network is required (a simpler solution use a coordinator which acts as an address that obfuscate the voters's addresses, where the coordinator establishes an offchain connection with the voter which allows the voter to send the proof privately to the coordinator who then publishes the proof).

6 Verifying the proof

Each miner that verifies the proof

- constructs a Merkle tree based on the vote commitments $cm_1, ..., cm_k$ listed in the smart contract of the election (with their defined order). and derive Root.
- computes a random seed S
- verify the zkSNARK proof based on the verification key, proof Π and the public parameters.

Then the miners reward the new accounts pk_{new} of the winner voters, i.e., voters whose ballots are within the majority then it receives equal share of the deposit of the election creator.

7 Security (very high level)

The privacy of the system is based on the security of Groth16 []. Users cannot vote more than once as it requires issuing different nullifiers for the same commitment which is as hard as finding collision of CRHF. The rewards received by the honest voters do not break anonymity as all winner users gets equal amount of coins. UC security can be proven via...TBD

8 Choosing the size of the subset

We claim that the sample has no large bias from the real distribution of the honest voters among the entire set of the users.

Let N be the number of users and r be the size of the subset that is selected for voting. The probability that a majority of the selected users are honest is

$$\sum_{i=\lfloor \frac{r}{2} \rfloor + 1}^{r} \frac{\binom{\frac{N}{2}}{i}\binom{\frac{N}{2}}{r-i}}{\binom{N}{r}} \tag{1}$$

We want to check what is the probability of an error, namely the chance that some candidate that only receive a minority of the votes, get elected. We begin with an election of two candidates a and b, where N_a is the number of voters who voted for a and N_b is the number of voters who voted for b. Let r_b be the number of voters that voter b in the sample r. Given a rule that a candidate wins only if she gets a δ-super majority of the votes (r.g., if $l_a \geq l_b + \delta$, and suppose that $l_a > l_b$, what is the probability that b wins? Let $P_b = \frac{N_b}{N}$ and $\hat{p} = \frac{r_b}{r}$.

Then for large enough r, according the central limit theorem, the discrete distribution in eq. 1 behave as a normal distribution, i.e. , $P(\frac{\hat{p_b} - P_b}{\frac{\sigma}{\sqrt{n}}} < z) = \Phi(z)$ where σ is the standard deviation, where in our case is $\sqrt{(p(1-p))}$. Since

$l_a > l_b$, $P_b < 0.5$. If b wins, then by the super majority rule, $\hat{p}_b > 0.5 + \delta$. Therefore $\hat{p}_b - P_b > \delta$. Hence, $P\left(\frac{\delta}{\frac{\sqrt{(p(1-p))}}{\sqrt{n}}} > z\right) = 0.0001$

For example, suppose that we want to upper bound the error by 0.0001 (i.e., that the minority wins), the z value is at least 3.7. Since $\sqrt{(p(1-p))} \leq 0.5$, it follows that $n \geq \left(\frac{3.7}{2\delta}\right)^2$. If we decide a threshold $\delta = 0.1$ (i.e., that a candidate has to have at least 60% of the sampled votes) it is sufficient to sample a subset of size $\left(\frac{3.7}{2*0.1}\right)^2 = 343$

8.1 Choosing oracle in blockchain

When this system implemented as an oracle in blockchain the winners are rewarded for their efforts. Another way to mitigate the bias and disincentivize the adversary attacking the system, e.g., by bribing other users to vote for some (false) candidate and later sell all the stakes, can be done by allowing any the stake transfers to be performed only after sufficient amount of time (through consensus). Assuming that a after a successful attack of the system the value of the coins will decrease to such extent that selling the stakes will be unprofitable to the attacker.

9 Coercion resistance

We now describe how to transform Subsenate into coercion resistant scheme. The core idea is to encrypt the ballots before sending them, which allows the coerced voter to invalidate it's vote by sending an encryption of the same ballot again without the coercer being able to observe it. In order to do so, the ballots are encrypted by the servers common public key, shuffled, and then check for duplications. The double ballots (i.e., the coerced votes) are detected and removed. In more detail, m servers cooperatively run a distributed key generation (possible threshold) to create a public key pk and shares of private keys $sk_1, ..., sk_m$. Suppose a voter intends to vote v and then some malicious entity coerced the voter to vote v'. In response, the voter creates two ballots b_1 and b_2 where b_1 is a proof of voting v' and b_2 is a proof of voting v (the new ballot structure is discussed in the next paragraph). Then the voter encrypts b_1 twice, i.e., compute $eb_1 = E(b, r_1)$ and $eb_2 = E(b, r_2)$ and send eb_1 to the coercer, along with the witness (private input) that is used to compute b_1 and the random element r_1, and publish eb_2 to the bulleting board (assuming the coerced voter find a time that the coercer does not notice). It also create an encryption $eb_3 = E(b_2, r_3)$ for the vote that should be tallied on the bulletin board. All the ciphertexts are processed through a a mixed network (by el gamal reencryotion, see []) in order to unlink the incoming and outgoing ciphertexts. Then the servers run PET over the ciphertexts and delete all the duplicates ballot (i.e., removes the coerced ballots). The ballot structure should also be updated to handle coersion. The voter needs to update its commitment in a way that allows to prove that it voted for the candidate it is forced to vote for

. To this end, the voter adds a new layer to the Merkle tree that consist of k new children, associated with k commitments $c_{v1}, ..., c_{vk}$ (corresponds to k candidates) to what has been the leaf c_v. Each commitment cm_j is computed as follows: the voter generates a nullifier u and computes $cm_j = H(u, j, id)$ for each $j \in [k]$ and a proof π_j which proves that $L_i = H(cm_1, ..., cm_k)$. Then, when the voter is coerced to vote for candidate j, it produces a proof $\pi_{coerced}$ for the statement that it knows $L_i = H(cm_j, mp)$, and the Merkle root is a result of the hashing cm_j and the Merkle path mp).

Comment: add a fig of the new Merkle tree

10 Tasks

1. add formal definitions. Definnition of zk-snark. zkSnark properties. Descripbe zkSnark building blocks (look at the code). Check https://arxiv.org/pdf/2203.03363.pdf
2. Check this: A BlockchainBased Self-Tallying Voting Protocol in Decentralized IoT. proposed a selftallying protocol that utilizes homomorphic time-lock puzzles to encrypt the votes for a specified duration of time to maintain the privacy of ballots during the casting phase
3. add figures
4. add concrete examples with numbers
5. Add figures
6. make a version that is used for multiuse
7. Try to makes some formal statements (lemmas)
8. Revisit the subset selection and what is the possible attacks and mitigations
9. Add implementation part