Blockchain based E-voting protocol

Noam Gottlieb, Gili Lior

August 2020

1 Introduction

In this project we have implemented a novel protocol of electronic voting, based on smart contracts and blockchain properties.

Many e-voting schemes have been proposed before. Some are even in use in some countries, but the ones that are in use are not blockchain-based. There are still some challenges to overcome before we can say that e-voting system are as reliable as ballot voting, but we beleive that once those gaps are closed, e-voting protocols over a dedicated blockchain can make the e-voting system much stronger and reliable than the traditional ballot voting.

Why blockchain? The transparency and decentralization of the blockchain lets us supervise the authorities that run the elections, as well as making it possible to any participant to detect a fraud, and therefore it will be a much more reliable system.

Why smart contract? Everyone see the code that is running, convince himself that it is reliable, or pointing out problems in its implementation. Also, the smart contract does not require us to depend on a specific participant's honest behaviour, besides the one that wrote the contract, which we believe it is a reasonable assumption.

There are some papers that offer an e-voting protocols using the blockchain technology, but most of them are either not detailed enough (no implementation), or fits a very small scale.

Our work is mainly influenced by 2 articles. We used concepts that are detailed in those articles, and implemented them:

(1) "A privacy-preserving, decentralized and functional Bitcoin e-voting protocol" by Zijian Bao et al. https://arxiv.org/pdf/1809.08362.pdf

(2) "E-Voting with Blockchain: An E-Voting Protocol with Decentralisation and Voter Privacy" by Freya Sheer Hardwick et al. https://arxiv.org/pdf/1805.10258.pdf

2 E-voting principles

In order for an e-voting protocol to be considered secured and trustable, the following properties must hold:

1. Fairness: No early results, the results cannot be determined before the voting stage is over.
2. Eligibility: Only eligible voters allowed to participate and influence the voting results.
3. Privacy: The way that an individual voter voted should not be revealed to anyone.
4. Equality: Each voter can only vote once.
5. Individual verifiability: An individual voter has the ability to verify that his vote was counted.
6. Universal verifiability: Anyone can verify the final results.

We will show later how does our protocol satisfy each one of these properties.

3 Protocol

In our protocol there are 3 main players:

- CA - Certificate Authority. He is the manager of the elections process. He has all the information regarding to eligible voters and the valid candidates. One of the vulnerabilities that we encountered in most of the e-voting proposed schemes is the centralization of the CA in the elections process. We agree that there has to be some of centralization in such processes, but we tried to make it as less centralized as possible.
- Candidates - All possible players that the one to get the most votes will win the elections. A valid vote can only be to one of the determined candidates.
- Voters - Players that participate in the voting process itself, and can add a single vote to one of the participating candidates.

Our protocol is divided to 4 stages: Registration, Voting, Reveal keys, Count.

3.1 Registration

The following steps takes place in the registration stage:

1. A permissioned blockchain is being initialized (we use ganache for simulation).
2. Candidates communicate with the CA and sign up.
3. Voters communicate with the CA and sign up.

4. The CA deploy the voting contract (VotingContract.sol). In the contract initialization the CA must pass the list of the candidates names, a list of the candidates addresses, and a list of all the eligible voters addresses. The candidates lists will be used for passing the voters the information they need regarding to candidates, as well as for validating function calls that must be called only from candidates addresses. The voters list will be used for ensuring that only eligible voters influence the elections result. In addition, the CA also passes his own RSA public key, that will be used for votes encryption in the voting stage.

5. Each candidate publishes to the voting contract his RSA public key, that will be used for votes encryption in the voting stage.

6. The CA divides the eligible voters to voting groups, each group holds a single VotingObject, that will later hold a shuffled list of this group's votes, as well as the voters and the CA's signatures over it.

In our project, we did not focus on the registration stage and its robustness, since all these actions are being preformed outside the blockchain, and we believe they are possible to be implemented using existing tools of the government to identify the citizens through the web.

3.2 Voting

This stage begins once all candidates publish their RSA public keys. The CA must change the voting contract status to VOTING.

1. A shuffling process of the votes starts. Before we explaing the CoinShuffle scheme, we will explain some notations in use:

   - n: The number of voters in a single voting group.
   - v_i: The $i'th$ voter in a single voting group.
   - c_{v_i}: The candidate choice of the $i'th$ voter.
   - pk_t: The public RSA key of the t, where t can be any voter v_i, any candidate c_{v_i}, or the CA.
   - sk_t: The secret RSA key that matches pk_t.
   - $ENC_{pk_t}(m)$: The RSA encryption of the message m, using the key pk_t.
   - $DEC_{sk_t}(m)$: The RSA decryption of the message m, using the key sk_t, s.t. $DEC_{sk_t}(ENC_{pk_t}(m)) = m$.
   - $ENC(v_i)$: This will represent the encrypted vote of v_i. The content of a vote is the name of the candidate to choose, concatenated with a unique nonce - $c_{v_i}|nonce$. This will be encrypted twice, first with $c'_{v_i}s$ public key, and the result of this encryption will be encrypted with the $CA's$ public key. The structure is as follows:
     $ENC(v_i) = ENC_{pk_{CA}}(ENC_{pk_{c_{v_i}}}(c_{v_i}|nonce))$.

The CoinShuffle (VotesShuffle) explanation:

In the registration stage The CA divided the voters to voting group. Each voting group includes n voters. The CA decides of an arbitrary order within all voters in the group.

The shuffling process starts with v_1. $ENC(v_1)$ is his encrypted vote with the candidate's and the CA's public keys. He will encrypt $ENC(v_1)$ in the following way:

$ENC_{pk_{v_2}}(ENC_{pk_{v_3}}(.....(ENC_{pk_{v_n}}(ENC(v_1)))).....))$

v_1 will send the encrypted message to v_2, that first will encrypt his own vote:

$ENC_{pk_{v_3}}(.....(ENC_{pk_{v_n}}(ENC(v_2)))).....)$

Then, v_2 will decrypt v_1 vote's first layer:

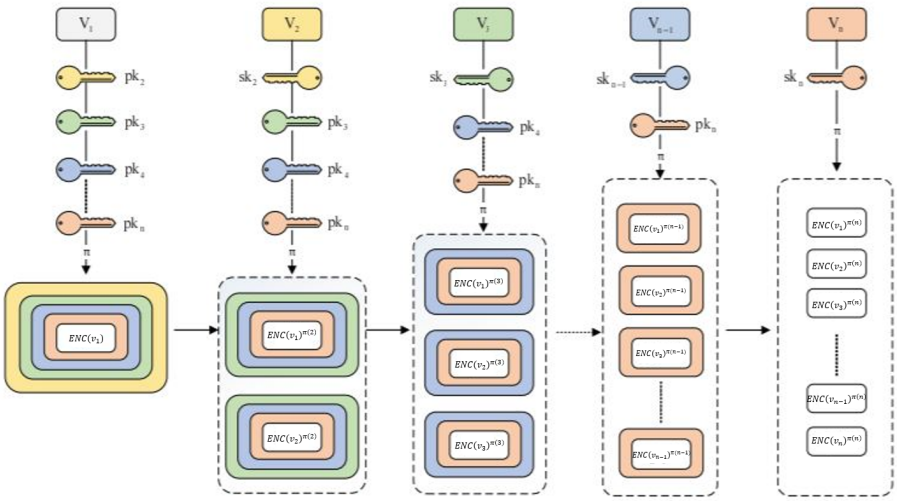
$DEC_{sk_{v_2}}(ENC_{pk_{v_2}}(ENC_{pk_{v_3}}(.....(ENC_{pk_{v_n}}(ENC(v_1)))).....)))$

which is exactly $ENC_{pk_{v_3}}(.....(ENC_{pk_{v_n}}(ENC(v_1)))).....)$

v_2 will shuffle his vote and v_1 vote, and will send it to v_3. v_3 will encrypt his own vote, and decrypt the next layer of v_1, v_2 votes. He will then send the 3 encrypted votes in a shuffled order to v_4.

This process will end after v_n decrypts the last layer, and holds a list of shuffled votes, encrypted with the candidates public keys and the CA's public key. See the attached figure 1 for demonstration.

Then, the final list of votes is being transferred again between the voters in the group. Each voter validates that his encrypted vote appears in the list, and if so, he signs over this voting object. The nonce that was used by each voter is known to him, so he can validate that his encrypted vote $ENC(V_i)$ appears on the blockchain, and thus being counted.



2. Once shuffling process is over, the CA goes over each VotingObject, and validates the voters signatures over the votes. If all is valid, the CA signs this VotingObject, and the votes list is ready to be deployed to the contract.

3. Next, any node can publish the list of votes, list of the addresses of the voters that voted in this VotingObject, and the CA's signature to the contract.

4. The contract validates the CA's signature over the votes and the addresses of the voters. So the signature is valid only if the addresses match the real voters that voted in this voting object. This allows the contract to monitor and make sure that only eligible, first-time, voters are part of this voting object. If all is valid, the contract state is updated with the votes and the already-voted voters addresses.

3.3 Reveal keys

This stage begins once all votes are published to the contract. The CA must change the voting contract status to REVEAL.

In this stage, each candidate must publish his RSA private key that matches the public key he published earlier in the registration stage. This key will be used for decrypting everyone's votes and get the final results.

3.4 Count

This is the last stage of the protocol. Once all candidates had published their private keys, The CA must change the voting contract status to CLOSED. When changing the status, the CA must pass his RSA private key that matches the public key he published when initialized the contract. This key will be used for decrypting everyone's votes and get the final results.

At this point, all the data that is needed for counting the votes is available on contract. each node can get all votes from contract, and decrypt the votes using the private keys (the candidate's and the CA's) that were published in the last 2 stages.

The CA publishes the final result, and each participant can preform the counting by himself to validate the results.

4 How does our protocol satisfies the E-voting principles

1. Fairness: Each voter's vote is encrypted using the CA RSA key, as well as the candidate's key. During the voting stage, the CA is the only one that knows his private key, and each candidate is the only one that knows his own private key. Until the CA publishes his private key, no one can

decrypt the votes, and until all candidates reveal their private key, even the CA cannot decrypt the votes.

2. Eligibility: At the contract deployment, a list of all eligible voter's addresses is being paased to the contract. When a list of votes is being published to the contract during the voting stage, it is considered valid only if all voter's addresses are known to the contract.

3. Privacy: Privacy is kept due to the CoinShuffle scheme. Each voter gets a list of encrypted votes belongs to the voters before him in the group arrangement. He can only decrypt one "layer" of encryption, the one that was done with his own public key, but he cannot see the vote's content. The voters at the beginning of the voting chain can connect between a voter and his specific vote in high probability, but the multiple layers of encryption prevent them to know the content of the vote. In contrast, the voters at the end of the voting chain can know the content of the votes, but at this point the votes already been shuffled multiple times so it is hard to connect betweenn a voter and his specific vote. There are some situations that this principle does not hold and it is possible to connect between a voter and his vote, for example if majority of voters in the group voted to the same candidate. But the bigger and more varied the group, the less of a chance to meet this kind of scenarios.

4. Equality: When the CA signs over a voting object, his signature is also over the list of all the addresses of voters that participated in this specific voting and shuffling group. This means that when publishing votes to the contract, a list of the addresses of voters that participated in this voting object must be passed and validated as part of the signature verification. This allows the contract to save all the addresses of voters that their vote is part of the contract, and monitor that each address does not appear more than once.

5. Individual verifiability: Since all votes are published to the contract, it is easy for a voter to go over all votes and make sure that his encrypted vote appears. Once it appears on the blockchain, it is ensured that his vote will be counted, or the cheating will be caught by a participant that will count the votes correctly.

6. Universal verifiability: Since all votes are published to the contract, as well as all the data that is needed for decryption, any node can decrypt and count the votes to make sure that the CA counted the votes correctly.

5 Vulnerability points of our protocol

There still remains few points that needs to be solved in our protocol. Our approach for this project was "all or nothing"- if someone tried to cheat, the

whole voting process is cancelled. We are certain that with a little bit more work, these scenarios are possible to handle with.

- Invalid votes: In case of an invalid vote, such as ineligible voter or double voting, also the valid votes from that VotingObject will not be counted. This can be solved by implementing a re-organization of voting groups for eligible voters that their vote was not published.

- Voters cooperation: At the stage of mixing the votes, the various votes are encrypted by the keys of all the voters in the group (except the first voter). So if one voter does not preform his decryption and passing it forward, or even if he disengages completely innocently, the list of votes cannot be produced.

- Scalability of voting group size: The bigger the voting group, the more time it takes for the procedure of chain asymmetric encryptions. This fact prevents us from using big voting groups, in a large scale elections, and that is the reason we have splitted the voters to voting groups. Though, it is important to note that there is a trade-off - Big groups gives us better privacy, but higher risk of a single voter to disrupt the shuffling process.