Dublin City University

Computer Applications

Functional Specification - 4th year project

ELECTIO

Applications of Electoral Systems using the Ethereum Blockchain

---

By:
Patrick H Morris
14759021

Supervisor:
Dr. Geoff Hamilton

November 24, 2017

Contents

1 Introduction

1.1 Overview

Electio is a proposed e-voting application that allows election administrators to dynamically design an electoral system and allow voters to vote securely and verifiably from any location with internet access.

1.2 Scope

The system described will be supported via the Ethereum blockchain which will enable execution of election functions on a public and transparent platform by the use of Smart Contracts. Users, of an election system, will interface with these Smart Contracts using an Ethereum client node and a desktop application which can interact with their connecting node.
The desktop application will be divided into three specific views arranged in sequence of before, during and after an election.

- **Election Set-up** - An election administrator will use the set-up view to design an election specific to the rules of the organisation or electorate they work for. Once the election criteria is met, the rules are converted into a Smart Contract and deployed to the blockchain.
- **Registration & Voting** - Both candidates and voters will use this view to register for an election and to vote in that election. If the criteria of an election is to elect people to an office of government or the like, a person may register themselves as a candidate.
  The actions a candidate and voter has with an election is done so on the blockchain and cannot be withdrawn once it has taken place.
- **Election Results** - Post-election, this view can be used to determine the winner of an election. The election administrator would publish the voting key for the election and allow anyone to determine the result of the vote and that their vote was counted.

The reason for blockchain technology to be used in an e-voting application is that it provides immutability of data. The assurance to the voter that a vote cannot be refuted, once it has been registered by the Smart Contract, is the underlining principle of a secure and verifiable voting system.

1.3 Background

1.3.1 The paper-ballot

The original voting system, and the one that is still used by democracies around the world is the paper-ballot system. Voters select their candidate by making a physical mark on a ballot-paper and submitting their vote to a locked ballot-box in the presence of election officials. At the end of the election, the ballot boxes are brought to a central location where votes are then tallied by other election officials and the results are revealed. Security of the paper-ballot system is based on the trust of the election officials to do their jobs correctly. The scale of election-fraud is minimised as it takes a large collusion of officials to successfully compromise an election, which is difficult as all colluders must trust each other that their actions remain secret. By trusting that a large majority of overseers of an election will carry out the functions of that election properly is why many countries still implement the paper-ballot system.

1.3.2 E-voting

However, the paper-ballot system is very expensive to implement and also requires people to vote from a designated voting-station which may impact on voter turnout. Electronic voting or *E-voting* is where voters can vote using electronic machines and transfer their vote over a network where it is then tallied by a central server. This central server then reveals the results of the election. It is accepted that they will reduce expense to carry out an election as they can be used repeatedly and in some implementations allow a voter to vote from anywhere provided they have internet access.
Nevertheless, the paper-ballot system is still preferred as most e-voting systems are not deemed to be as secure and also not satisfactory to voters in proving that their vote was registered. The main security issue is that it gives a potential attacker an opportunity to totally compromise the election if they break into the system undetected.
It also means that a large amount of trust must be given to the people who created and control the voting and tallying machines.

1.3.3 Blockchain in E-voting

From a security standpoint, the traditional e-voting system is exposed in the fact that it is a centralised hierarchical structure. If the central node within the system, (the tallying server) is compromised, then the election fails completely. Blockchain technology is a solution to this as it is completely decentralised
The Ethereum blockchain functions as a global computer that executes code across a distributed network of user nodes. The application of it within an e-voting system is that it is possible for the functions of the election to be executed by everyone on the network and provide the same result with perfect consistency. Along with this, because the blockchain is essentially a record of transactions, it is possible to allow a user privately verify that their vote was counted and what their vote was.
My idea for this project is to apply blockchain technology across all electoral systems by allowing election administrators design the elections the electorate wish to participate in and allow voters and candidates ensure their participation in that election.

1.3.4 E-voting Security requirements

Before designing and implementing an e-voting application, there are requirements to e-voting systems that must first be observed:

- **Integrity** - The outcome should match the voters intent and as such, be verifiable. This can be expanded into two scenarios.
  - **Individual Verifiability** - The voter should be able verify that their vote was:
    1. Cast as intended
    2. Recorded as cast
    3. Counted as recorded
  - **Universal Verifiability** - Any person should be able to verify that the tally is correct.
- **Eligibilty** - Only registered voters can cast votes.
- **Privacy** - nobody can figure out how a voter voted.
- **Receipt-freeness** - a voter should not be able prove how he/she voted. This is to prevent a person in selling their vote.

- **Coercion-resistance** - no person should be able force another person in voting in a particular manner or to abstain from voting.
- **Fairness** - no partial results should be known until after the election has closed. Not doing this would give people who would vote last a choice advantage.

1.4 Document Overview

General Description

This section gives an informal outline of what the application does and how it works.

Functional Requirements

The requirements section outlines the critical components of the system and a description of how interact with one another. This will be have more technical content then the description section.

System Architecture

A description of the high level interactions between users and major components are modelled Also is a component diagram showing the technologies used to implement the system.

High Level Design

This section displays detailed models of the inner workings of the application. Shown are a number of sequence diagrams outlining the possible scenario's users will have with the system.

Project Schedule

The schedule describes the development timeline for the project.

1.5 Glossary

Ethereum - Ethereum is a decentralised network which acts as a global computer. It is akin to the cryptocurrency, Bitcoin which allows for immutable transactions of value over a peer-to-peer network. Ethereum enables immutable execution of data across a peer-to-peer network.

Smart Contract - A Smart Contract defines what the Ethereum network will compute. It is the ethereum programming language in essence.

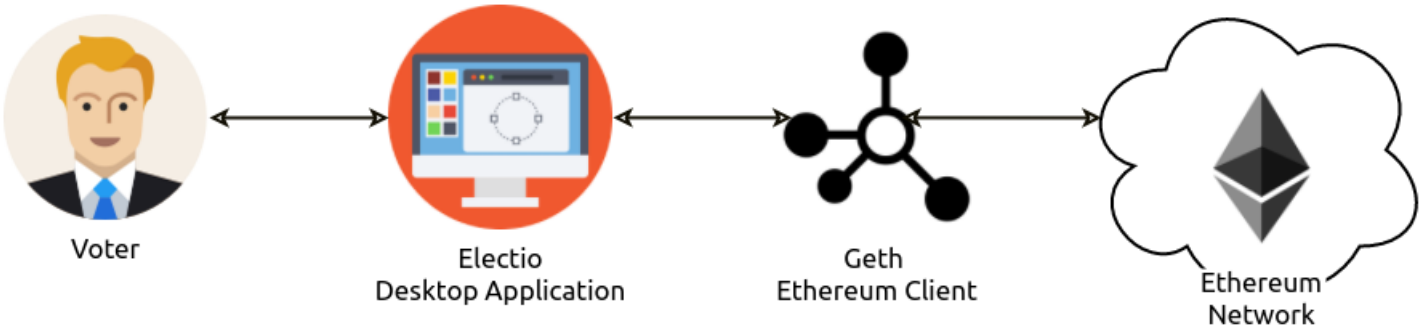
Ether & Gas - Ether is the store of value on the Ethereum network and is analogous to a Bitcoin, it has a real-world value and is a tradeable currency. Gas is a unit of measurement used to calculate the computational cost of executing code on the Ethereum Blockchain. When the gas is calculated, the user who executes a transaction, must pay a portion of Ether to cover this gas value.

Encryption/Decryption - Encryption and Decryption are processes where a secret is applied to a function on a piece of data to hide it (encryption) or to reveal it (decryption). The readable state of the data is called the plain-text and the hidden unreadable state of the data is called the cipher-text

2 General Description

2.1 Functions of the System

The main functions of this system are intended to allow the users both create and operate the functions of an election from their own computer.
In order to do this a user will connect to the Ethereum blockchain through a geth node. (*Geth* is an Ethereum client node written in Go). Through this client node, a person is able to operate the functions of the blockchain such as sending transactions, deploying a contract or mining Ether.
Simplifying this interaction process with the blockchain for the user, we will use a frontend desktop application that uses Web3.js. Web3.js is Ethereum's JavaScript API that interfaces with the blockchain through a client, in this case, the geth node. As it is in JavaScript, we can implement a node.js webserver to supply our frontend and easily integrate the functions of Web3.js into it.
This server is intended to operate locally rather than as a central server. The purposes of this is to ensure that a user interacts only with the election system. By using a central server, a user would have to trust the owner of that server which is against the trustless intention of this system.
The web interface will be provisioned by the Electron framework which is used to create desktop applications using web technologies.



The above infographic displays at a high-level, the core system components. To clarify, both the desktop application and the geth client exist within the userspace and do not interact with any other external system other than the Ethereum network.

2.2 User Characteristics and Objectives

Defining who will be a potential user of this system depends on the context in which elections are held. Elections can be held in multiple different environments and are used to define a consensus between a set of people, whether that be electing a person to a position or coming to a democratic agreement over a business decision.
Essentially, the context where a voting system can be used is too broad to specify individual users or groups of users by what they are using it for. Therefore it is more accurate to describe users of the system in how they are using it. I define three users of the Electio e-voting system, each with their own purpose in interacting with it.

Election Administrator - The administrator is described in this document as a single individual but can be realised as either a person or group of people in charge of implementing an election. They have multiple objectives in the operation of this system.

- Validate legitimate voters to the register
- Confirm a person's candidacy in an election or outline the proposed motions to be voted on
- Implement and deploy the criteria of the election to the Ethereum network as a smart contract
- Notify the electorate of the necessary voting information
- Generate the election keys and publish them in order
- Process the election contract for spoiled votes

As we can see, the administrator has multiple tasks that they must achieve to implement an election. The application should be make this as easy as possible, automating the marking of spoiled votes and validating voters.

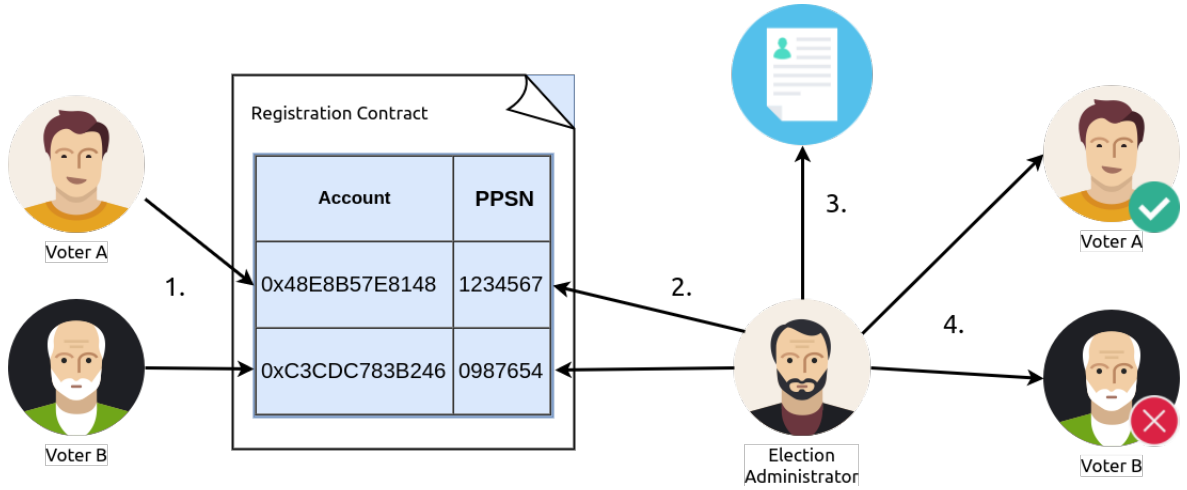
Voter - The primary concern for a voter using an election system is as stated in the security requirements in 1.3.4. Voters using an e-voting system must be satisfied they voted privately and that their vote was cast, recorded and counted.
A voter intends to perform two actions in an election with a possible third in the context of this system. A voter must register and be validated by the administrator in order to vote for the candidates or motions of their choice. The third action is that post-election they intend to self-verify their participation and tally the results of the election.
It is to be noted, that the system is to be designed to include all members of society and so the proper UI requirements should be implemented to cater for people with disabilities.

Candidate - Candidates are not always a user of the system as some elections are motion-based and evidently, a human candidate is not pertinent to them. However, in cases that they are, a candidate will nominate themselves to the administrator who then affixes them to the voting contract.
Often, a candidate has a manifesto on why they should be elected and he/she wishes to communicate this on the electoral ballot as a voter makes his/her vote. This manifesto can be provided using the candidates IPFS address and is given to the administrator.
Otherwise, the candidate acts as a voter in all other interactions with the system.

2.3 Functions of the Election

2.3.1 Registration

The Republic of Ireland, and other democratic countries validate users to vote in national level elections through a document called the *electoral register*. The register is a list of all people who are eligible to vote in these elections and it is a publicly viewable document.
As voters in this system will be represented publicly as Ethereum account addresses, a process must be in place for voters to identify themselves to the election administrator whilst preserving their privacy from the rest of society. For this, an assumption is made that the user has a unique identfication number or code that can be used to identify his or herself to the administrator and that the administrator knows who that code represents. If it were an election in the Republic of Ireland, the unique identifier could be a citizens PPS number. A PPS number is perfect for this as it is already used by the government to identify individuals. In other circumstances it could be agreed verbally or emailed by the voter, either way a unique identifier must be used to identify individuals
Prior to an election, the administrator would generate a Smart Contract which voters can post their PPS numbers to. The administrator will then examine the accounts and PPS numbers with the electoral register and determine if they are a valid voter or not. The list of valid voters is important as it can be referenced by the voting contract as we will see.

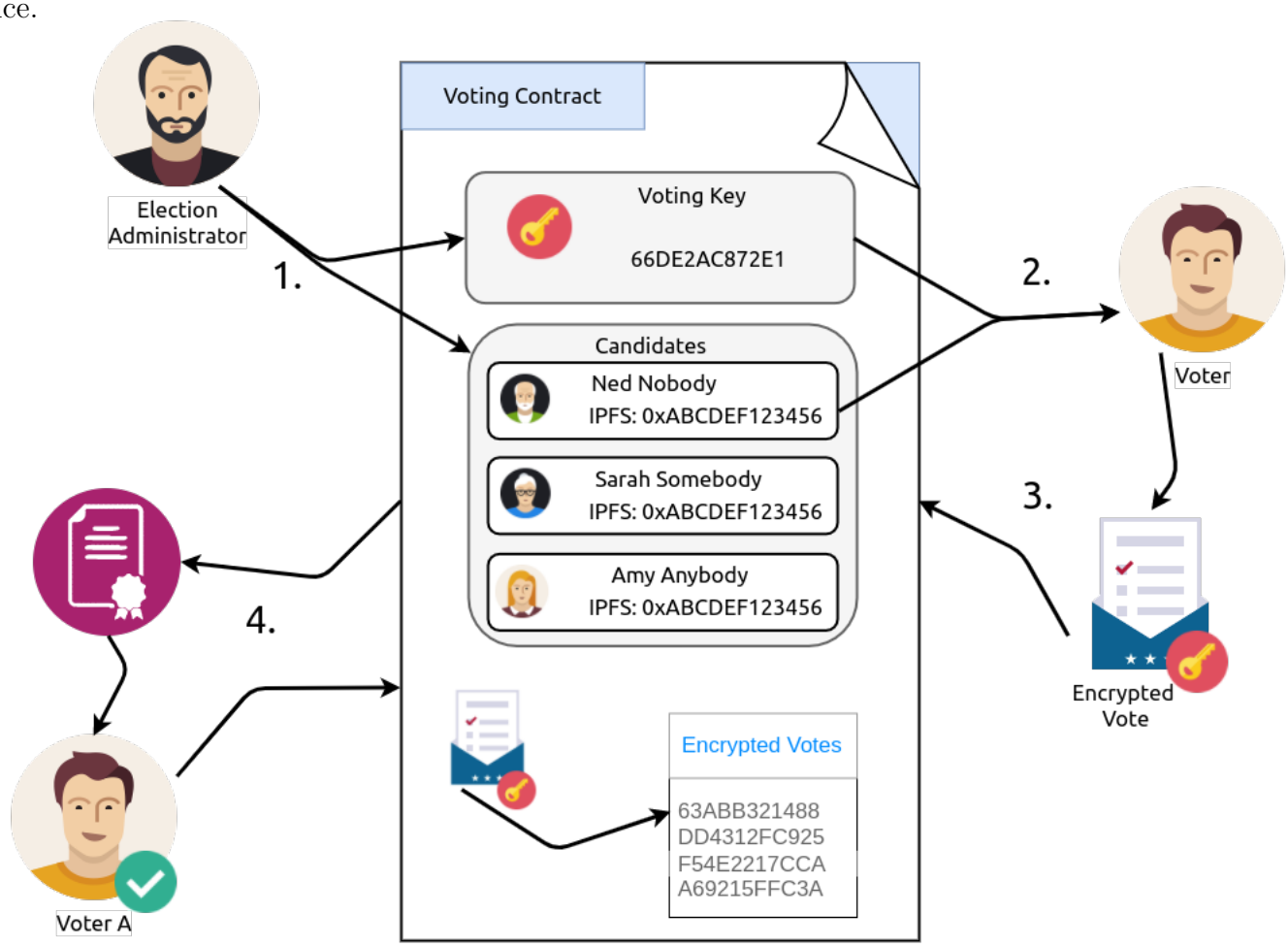


1. Voter A and B publishes their PPSN to the registration contract
2. The election administrator records the PPSN for each account address
3. For each PPSN the administrator determines who it corresponds to and if they are on the electoral register
4. Voter A was on the register and was validated for the voting contract whereas B was not on the register and was therefore refused.

To register via the application, a candidate or voter would enter the registration contracts address. There, they would post their registration information which would then be deployed to the registration contract.

2.3.2 Voting

The voting process is straightforward once the voter has registered for the election. The election administrator will have deployed the voting contract to the blockchain after the registration process and published the contracts address to the electorate. The administrator will also publish the candidates and the voting key which can be used to encrypt a voters vote.
When a voter wishes to vote in the election, he/she will enter the voting contract's address and the desktop application will pull all election data from the blockchain. This will provide the voter with a display of all relevant election data along with the voting key to hide their vote. A voter will make their vote in accordance with the election rules and encrypt it with the key giving a hashcode receipt. This receipt hashcode can be used to validate that their vote was recorded as cast and counted as voted after tallying.
Upon receiving a vote, the contract will check the registration contract to determine if the voter sending the vote was validated during the registration phase. When the vote is taken by the contract, the voter is invalidated on the registration contract by the voting contract so they cannot vote twice.



1. The administrator publishes the voting key and Candidates list to the contract
2. The voter takes the voting key and selects a candidate to vote for
3. The voter encrypts the vote and posts it to the contract
4. Upon receiving a vote, the contract checks the registration contract if the sender is a valid voter. When confirmation is received that the sender is a valid voter, the encrypted vote is then added to the contract.

2.3.3 Tallying

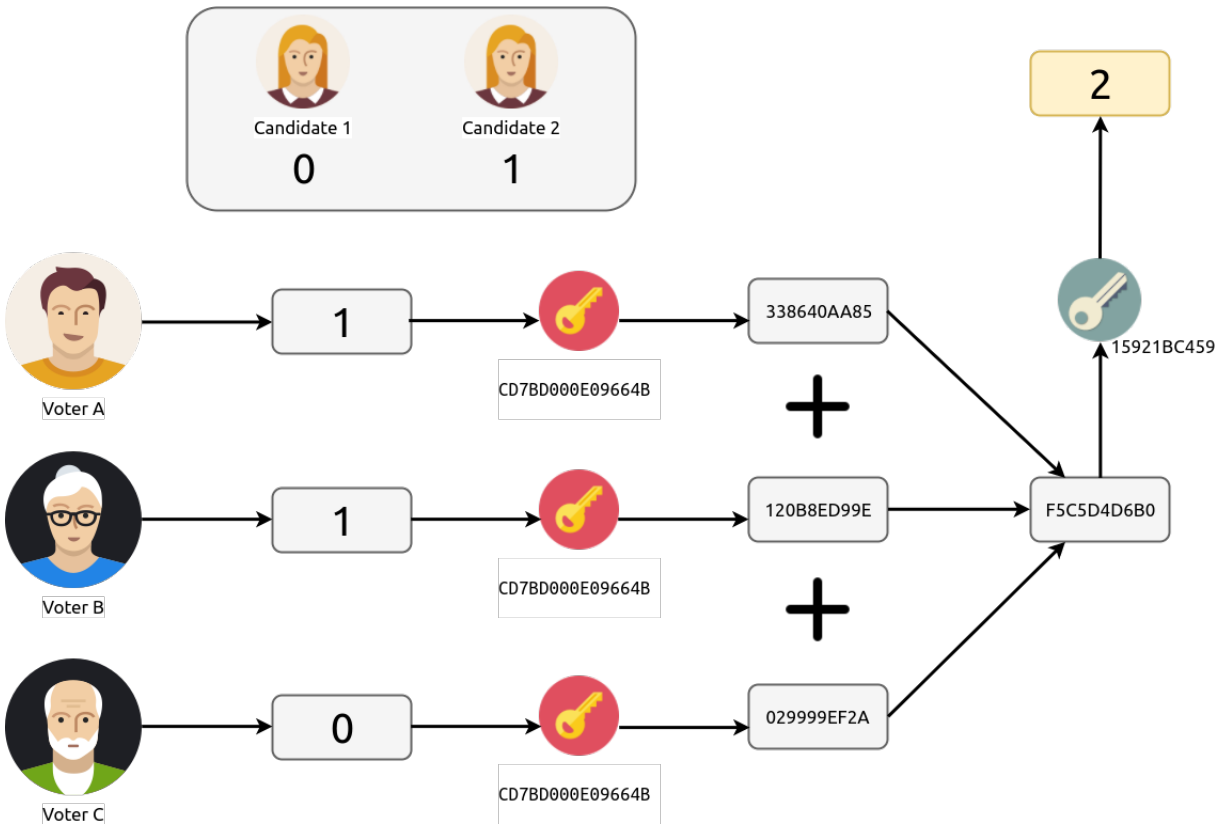
The tallying process is designed with the intent that it is universally verifiable by anyone. In order to this, the system must enforce a homomorphic encryption scheme on every persons vote.

Homomorphic Encryption

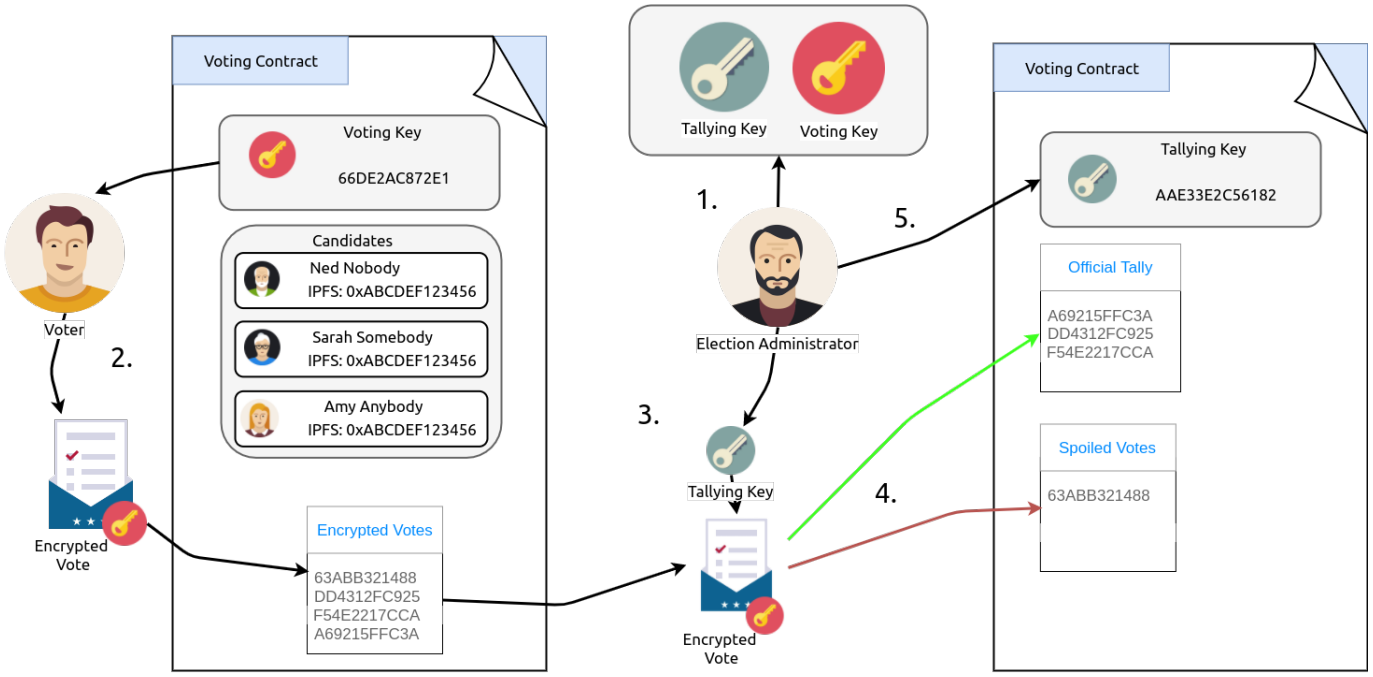
Homomorphic encryption is a type of encryption that enables computation on cipher-texts which mirrors the same computation on the plain-text. The implementation used by this system is the Paillier scheme which is an additive homomorphic cryptosystem, this means given only a public-key and the encryption of m_1 and m_2 one can compute the encryption of $m_1 + m_2$. Crudely this can be explained below where E and D are the encryption and decryption functions:

$$D(E(1) + E(2)) = 1 + 2$$

In its application to an election tally, an example of a 2 candidate election is often used where one candidate is represented as 1 and the other as a 0. All voters encrypt their votes using the public-key and give their hashed vote to the administrator. The administrator computes the sum of the encrypted votes together and decrypts them using the private-key which only he holds. The decrypted number is the sum of all the 1's and therefore the 0's can be calculated by the sum of the votes subtracted by the sum of the 1's. The example below illustrates this and we see the decrypted value of the tally is 2. This is the amount of votes given for Candidate 2, Candidate 1's votes can be calculated by subtracting the sum of votes for Candidate 2 (2), from the number of voters (3) which gives a value of 1. Candidate 2 in this case would be the winner of the election.



Applying this property to our system allows the administrator to generate key pairs, a voting key for voters to encrypt their vote, and a tallying key to generate the result. As mentioned, the voting key is published on the voting contract for the voters but the tallying key must be withheld by the administrator until after the election. Publishing the private key to the voting contract would mean that anyone can decrypt anyone else's vote but as everyone's identity is obfuscated by their voting address, their privacy is preserved.
The manner of how the votes will be stored on the contract is also relevant as no relational mapping structure will be used to store the votes. Instead of mapping the hashes of a voter's vote to the voters address, the hashed votes will just be deposited in an array.
As the administrator has ownership of the private-key until after the election, he/she can validate the votes for spoiled votes. Although the desktop application will channel a user to only vote legitimately, there is little possible to deter any bad voters in attempting to interact with the contract manually if the contracts address and functions are known. Taking the homomorphic example, if a voter wanted the candidate represented by 1 to win, he/she could encrypt 10,000 for example and it would be the equivalent of adding 10,000 votes to that candidate. As the administrator can decrypt a vote to see the contents, the contract could enable him/her to determine these kind of votes as spoiled.



1. Prior to the voting contract is generated, the election keys are generated. The voting-key is published to the contract.

2. Voters use the voting-key to encrypt their vote and deposit the encrypted hash into an array structure on the contract.

3. To ensure no spoiled votes took place, the administrator decrypts the votes with the tallying key that only he knows at this stage.

4. If a vote is a valid value, the hash is added to the official tally. If the vote is not a valid value for the election, then the hash is added to a list of spoiled votes. It is to be noted that the number of official tally votes and spoiled votes must equal the number of votes deposited to the contract.

2.4 Operational Scenarios

To examine a context of how this system may be applied in reality I will explain how a GAA club would elect a new chairperson using Electio.

Need for an Election

A GAA club has 200 adult members and are tasked with electing a chairperson among them as the previous chairperson has resigned. The club secretary proposes to use the Electio election system as some members are out of the country and would not have the opportunity to vote locally by the clubs traditional method of a paper ballot.

Election Registration

The secretary takes on the role of election administrator and asks for all members to directly message him a code so that they can be verified for the election of the new chairperson when they register. The secretary begins using the Electio application, creates the registration contract and sends a group email to all club members of the registration contract's address. Over the next two days, 180 members are pending approval to be registered with 20 members abstaining to vote. The secretary uses the Electio application to view all the pending members account addresses and correlates the codes that were directly sent to him to the ones fixed to the addresses. Through the process, he notices that 3 members are underage players and it is against club policy for anyone under the age of 18 to vote in club affairs. The secretary does not validate the addresses corresponding to the underage players.

Election Voting

After processing, the secretary generates a pair of keys, a voting key and a tallying key. He then generates a voting contract using the Electio election form as per the club rules and publishes the voting key and the two candidates for the vote. He keeps the tallying key safe and private and sends a group email to all club members, informing them of the voting contract's address. When a club member goes to vote they are able to view the voting key and candidates list.
When the voting window is over, the contract has stored 177 unique hashes. Taking these hashes, the secretary computes their sum and decrypts them using the tallying key. The results show that one candidate got 120 votes, while the other candidate got 70. As there were 190 votes for 177 people, a possible scenario is that one voter encrypted a value of 14 to fraud the election in favour of their candidate.

Election Tallying

The secretary begins decrypting all of the encrypted votes stored in the voting contract until he finds the one with a value not equal to 1 or 0. He decrypts one vote and finds it has a value of 14. The hash it was decrypted from is flagged as a spoiled vote and is not counted in the tally. The tally now represents results of 106 to 70 for the two candidates. The secretary announces to the club of the results and publishes the tallying key to the voting contract so that any person may wish to validate the results for themselves.

2.5 Constraints

2.5.1 Ethereum Blockchain

The primary constraint the system is that it is entirely dependant on the performance and reliability of the Ethereum Blockchain. Ethereum has been aggressively developed over the last three years and it is to be made aware that any critical changes done to Ethereum may affect the Electio system. Also to be noted is how Gas, the cost of using the Ethereum network, is measured by the efficiency of the contract in the actions it performs. Performing insertions and deletions to data structures should be optimised as much as possible.

2.5.2 Timeline

Development of the application is projected to begin in December with a strict implementation deadline on the 21st of May 2018. By the middle of February to the 1st of March, an end-to-end functional implementation of one electoral system should be in operation.

2.5.3 Scalability

The Ethereum system is analogous to a global computer and it, like a normal computer CPU, has a clock speed in how it can execute computational functions. This clock speed is known as block-times in blockchain. The *block* in blockchain is important to understand as it is how Ethereum transactions are collected and executed. It involves a user broadcasting his/her transaction to the network and relying on a miner to pick it up and execute it in his/her copy of the block. The network randomly picks some miners block every 10-15 seconds on average in Ethereum.
In the case of Electio, an execution of the contract may not be instantaneous and interactions may become queued. This is to be kept in mind and investigated in Electio's development.

2.5.4 Electoral Systems

The objective of this system is to enable administrators apply their election simply and effectively. This infers that the application must be designed with the intention to support as wide a range of election systems as possible. Initially the objective is to develop a single election system on Ethereum but the final product should cover most election scenarios.

3 Functional Requirements

3.1 Election Contract Generation

- **Description**
  This involves generating default registration and voting contracts. The idea is that these contracts will encompass the full functionality of the system and will be accessible from the blockchain itself. The system will use a deployer contract which will pass election rules as constructor arguments for a new instance of a voting contract. Multiple voting contracts for different electoral systems may be accessible from the deployer and will have different constructor variables dependant on the nature of the election.

- **Criticality**
  The system entirely depends on the presence of the two types contracts and hard-coding the basic functionality of them is of the highest priority.

- **Technical Issues**
  Development of these will mean defining a point of what should be customisable by the election administrator and what should be uniform functionality across all elections. The election rules will be described by the administrator on the interface and will be passed as constructor variables in the creation of the contract.

- **Dependencies**
  The templates themselves will exist as fixed documents on the Ethereum network. To interact with the system they depend on a deployer contract to generate an instance of each type.

3.2 Election Keys Generation

- **Description**
  This will be an implementation of the Paillier cryptosystem that will generate a public and private key pairing. The public key will be the voting key, used by all voters to encrypt their vote. The private key will be the tallying key used to tally the votes.

- **Criticality**
  This is a critical feature of the system and is necessary to hide a person's vote.

- **Technical Issues**
  Implementing a cryptographic protocol requires that it does exactly as intended by the definitions of its creator. This would mean that the implementation is vigorously tested.

- **Dependencies**
  The generator will programmed in Java so the user must have Java installed on his/her machine.

3.3 Communicating with the Blockchain

- **Description**
  The application backend is required to process all communication between a user of the system and the contracts on the blockchain. The backend is a local server that can interact with the Geth node and will serve as a foundation for the user interfaces.

- **Criticality**
  The backend is a central component in the system and will perform all the heavy-lifting in terms of communicating with the contracts.

- **Technical Issues**
  The server will be implemented using node as it is quite accessible and has an abundance of tools and features. Web3.js is the JavaScript API for Ethereum and using that in conjunction with a JavaScript server is more manageable. Web3.js makes function calls to the network using the client node.

- **Dependencies**
  The backend will have user dependencies such as needing to have nodejs installed on their machine.

3.4 Generating an Election

- **Description**
  The administrator interface will be the frontend for an election administrator to generate an election. It must provide the administrator a manageable way in describing the rules of the election and performing the functions of the election he/she is required to do.

- **Criticality**
  The interface is critical and of the three listed interfaces, is the most dense in terms of content. This will be the most difficult to implement and will be the crux of the election process as it is at this interface the election is defined.

- **Technical Issues**
  The administrator will have a multitude of options and actions he/she can perform through the interface and this may be difficult to understand. Providing annotations and information pages will be necessary for the user to understand how the system works.

- **Dependencies**
  The interface will depend on the backend to generate the contracts and make function calls to the blockchain.

3.5 Participating in an Election

- **Description**
  Participating in the election by this system is where both the voter and candidate users can access the functions of the election contracts, registration and voting.

- **Criticality**
  Human candidates are not always critical as not every election is a vote on electing a person. The voter functionality is a core component of the system and must be looked at first before the candidate features can be employed.

- **Technical Issues**
  As every person does not have an ethereum account, it may be necessary to add functionality for a person to do so here. The metamask plugin is a wallet for Chrome and Firefox browsers and can be implemented into the application. This can be used for a user to generate an

account.
For a candidate, the election manifesto will be linked by them registering their IPFS address. This will have to be displayed on this interface when the election data is pulled.

- **Dependencies**
As with the administrator interface, it requires the backend to do most of the heavy-lifting.

3.6 Validating an Election

- **Description**
The tallying interface will be a simple page where a user can input the address of a contract and use it to display the final results of the election. This page will display data like the tallying key, the candidates, the vote hashes and a means of calculating the tally result with this data and thereby validating the election.

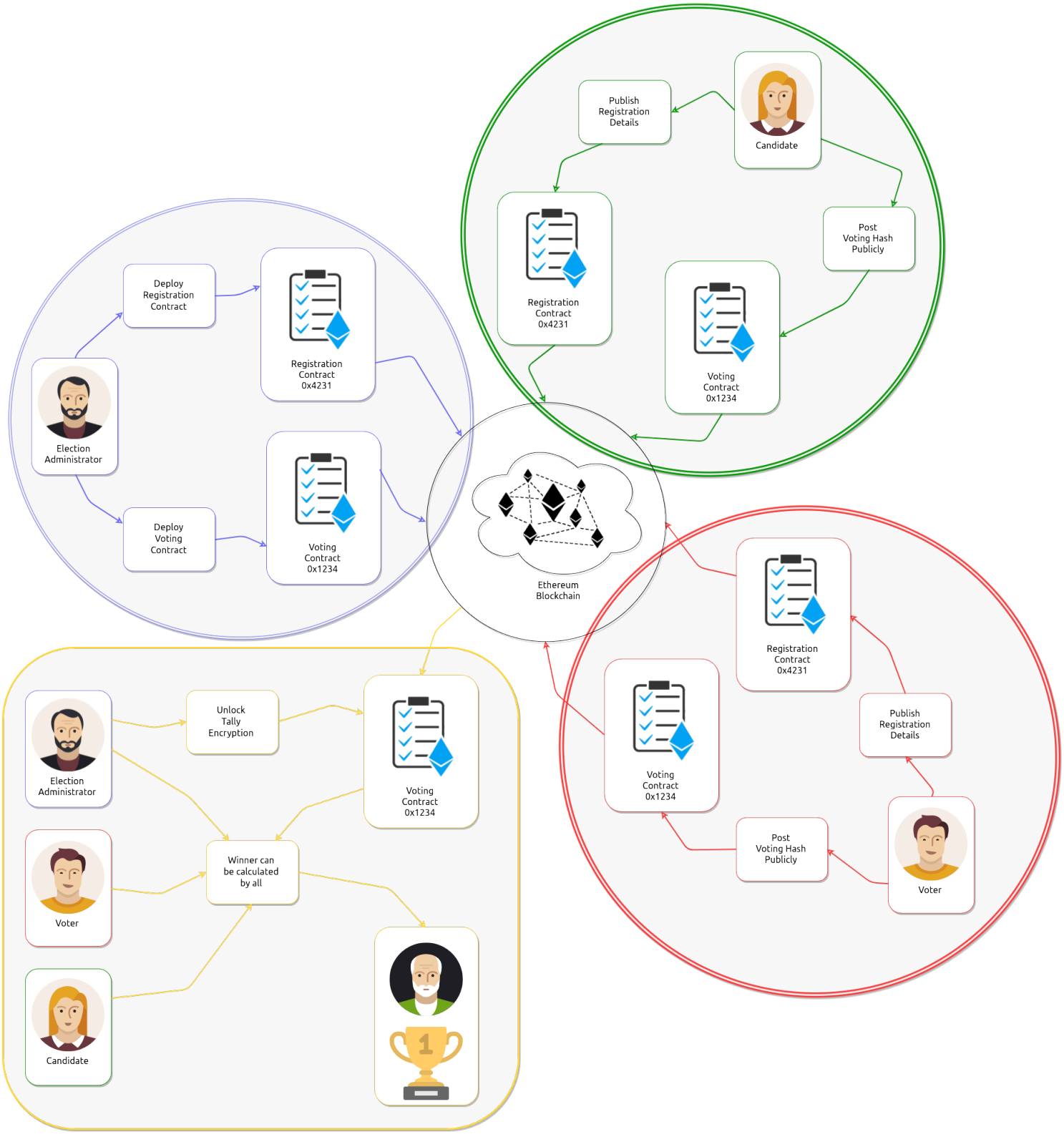
- **Criticality**
This page is critical in the fact that an election must have a provable result. All necessary data of the election should be observable from here.

- **Technical Issues**
From a technical standpoint, implementing this interface will only require extracting the relevant information from a contract given its address.

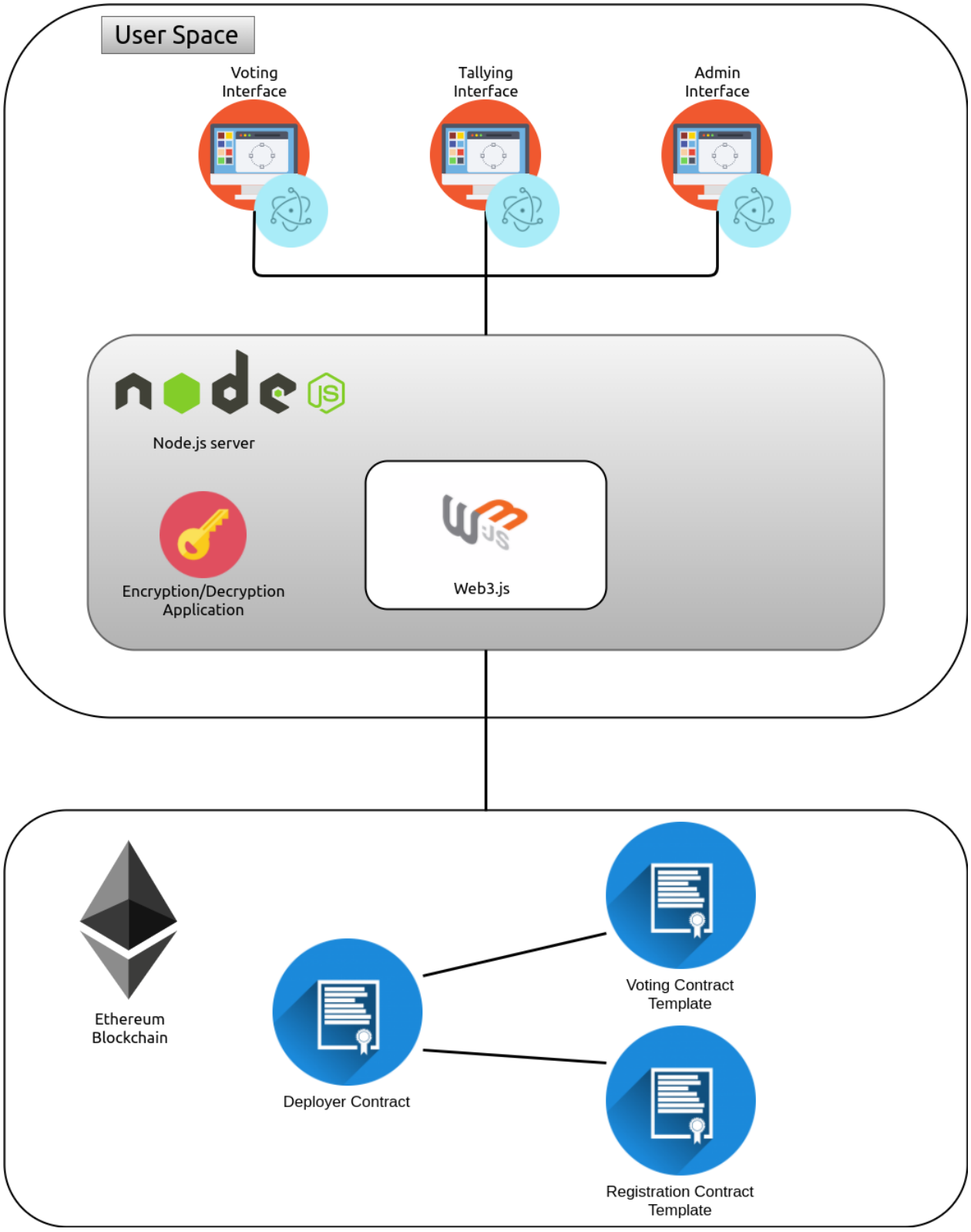
- **Dependencies**
This page is dependant on the backend to make the function calls to the contract.

4 System Architecture

4.1 High Level Overview

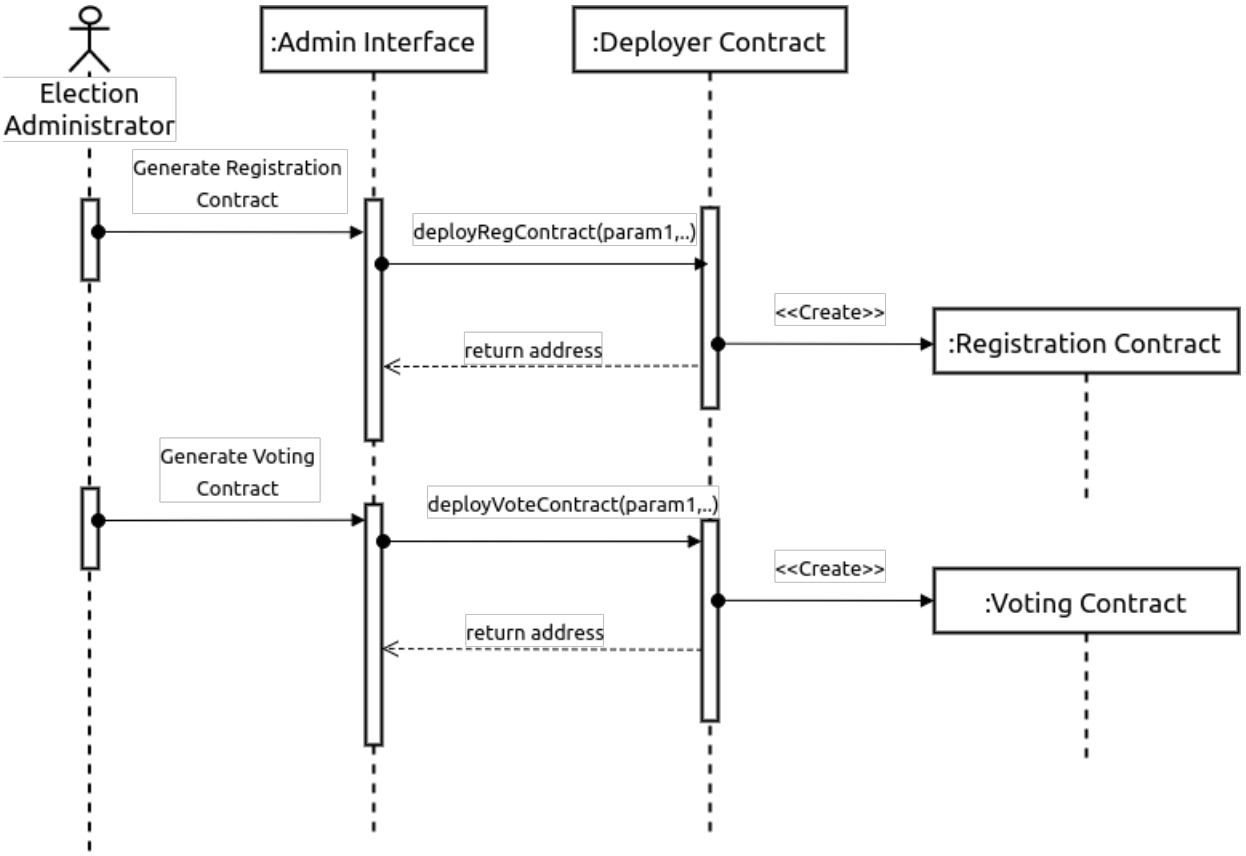


4.2 Component Diagram



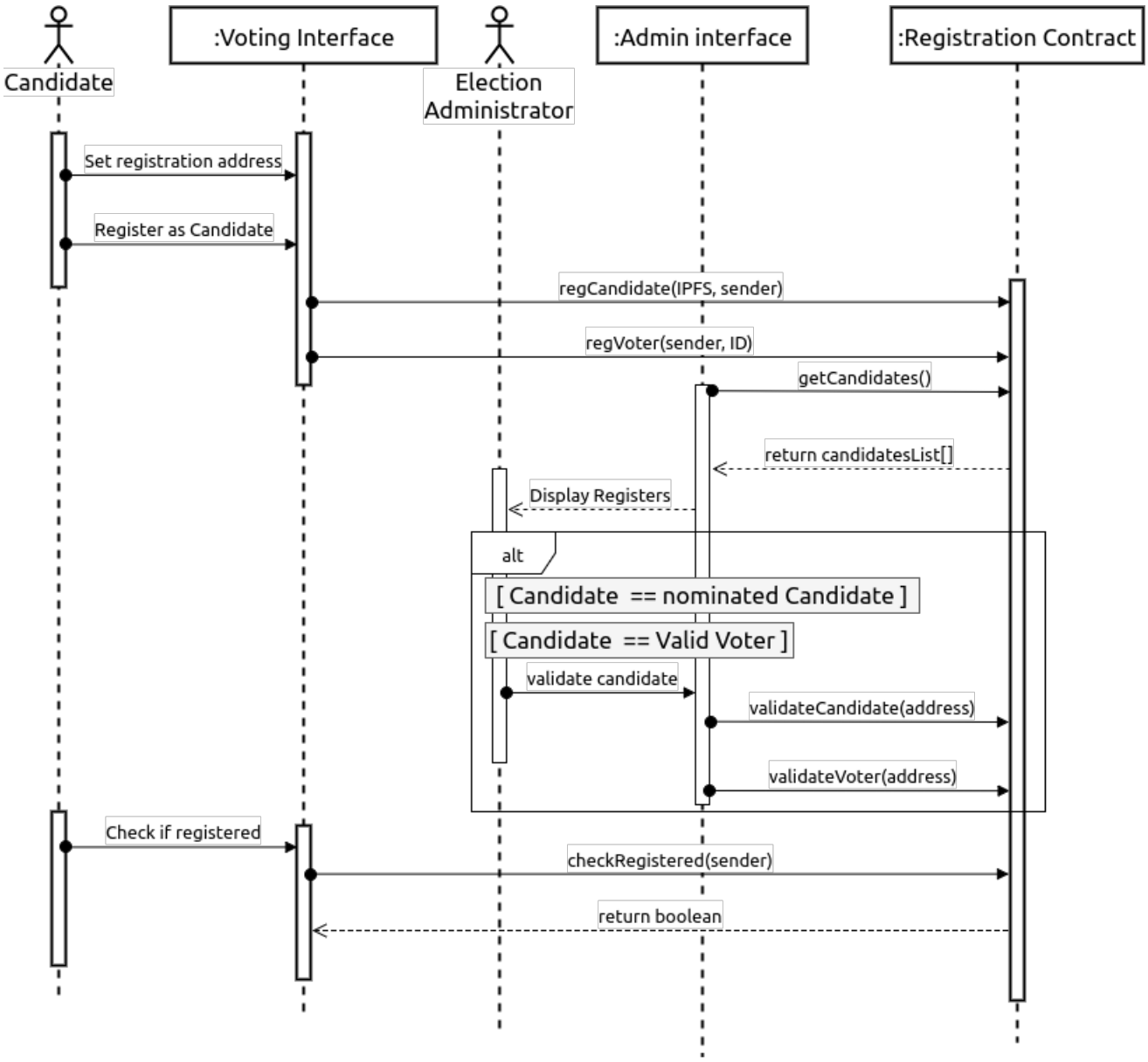
5 High-Level Design

5.1 Contract Deployment



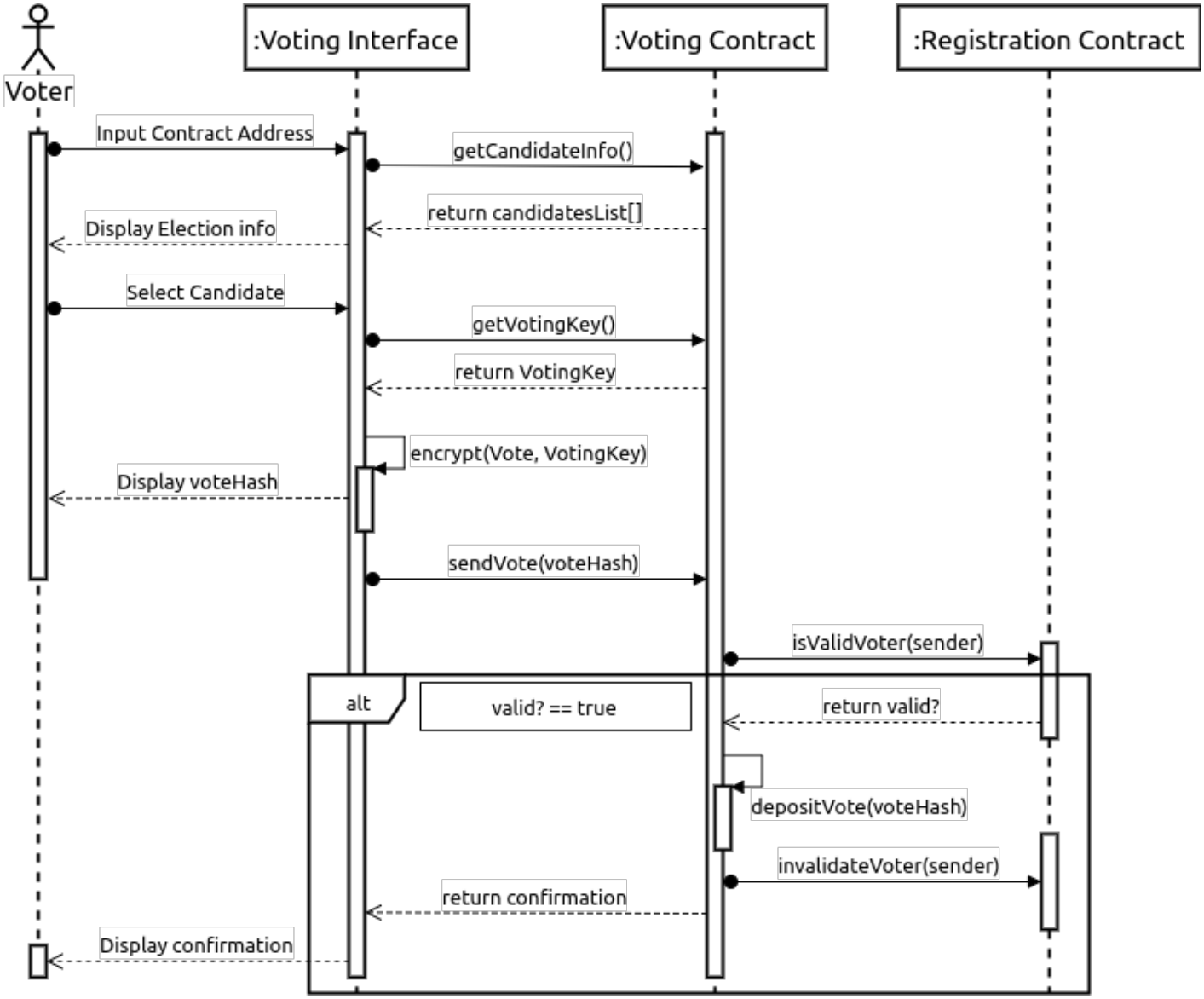
The above image simply illustrates the sequence of how a contract is generated. The Registration and Voting contracts are both implemented via the deployer contract.

5.2 Candidate Registration



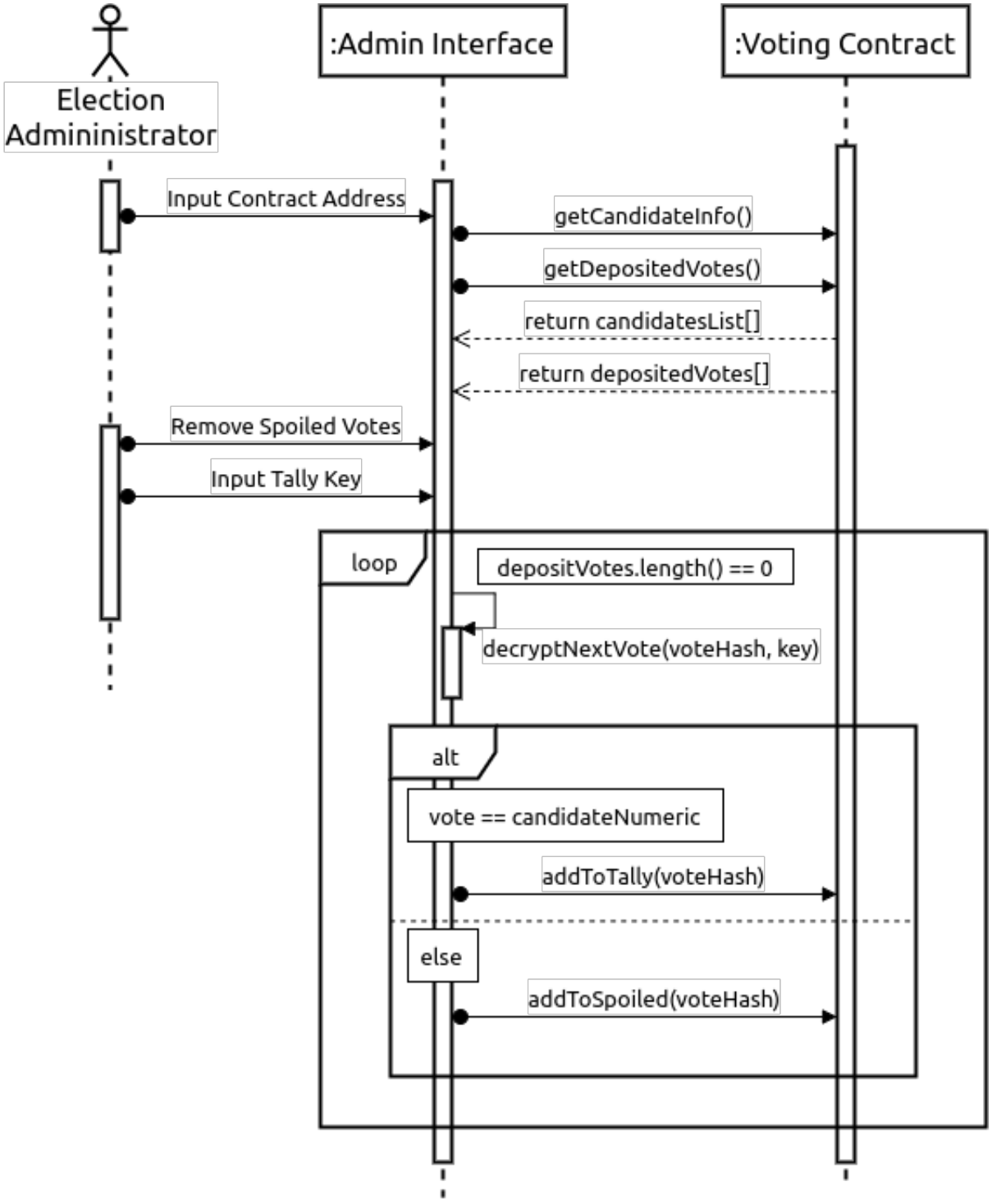
This is the registration process visualised for a candidate. It is to be observed that a voter will register the same way with the omission of the candidate specific functions. Included is the condition that only valid voters and candidates can be added to the election by permission of the election administrator.

5.3 Voting



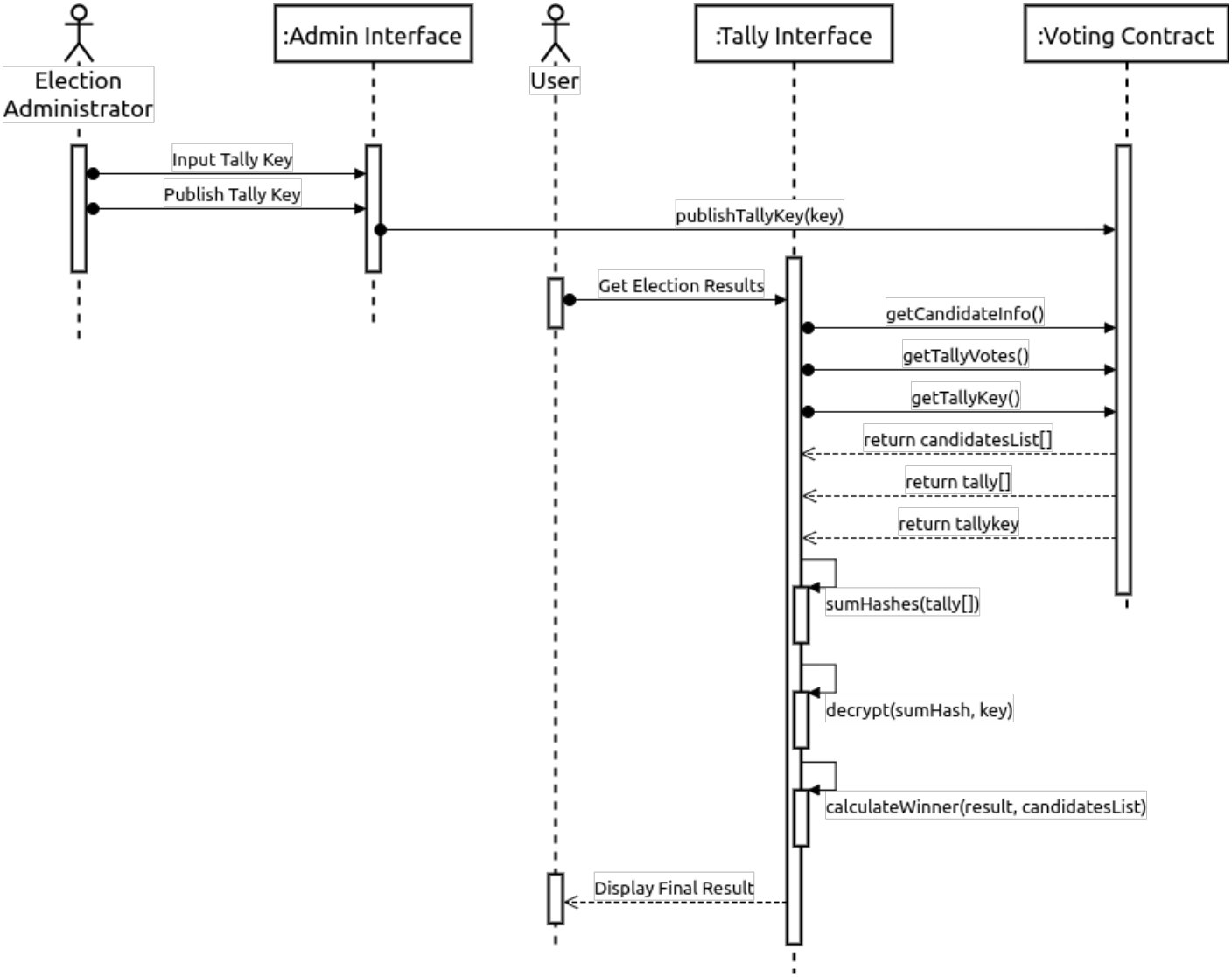
The voting process is clearly straightforward and requires the voter only to know what the contracts address is. The application displays the election data providing the voter with candidate information the candidates have provided. The interface also handles the encryption of the vote and simply deposits a hash value to the contract. There is also an inter-contract operation, determining the validity of voter for that election. When a voter deposits there vote, they are then invalidated from voting again and preventing a voter to vote multiple times.

5.4 Removing Spoiled Votes



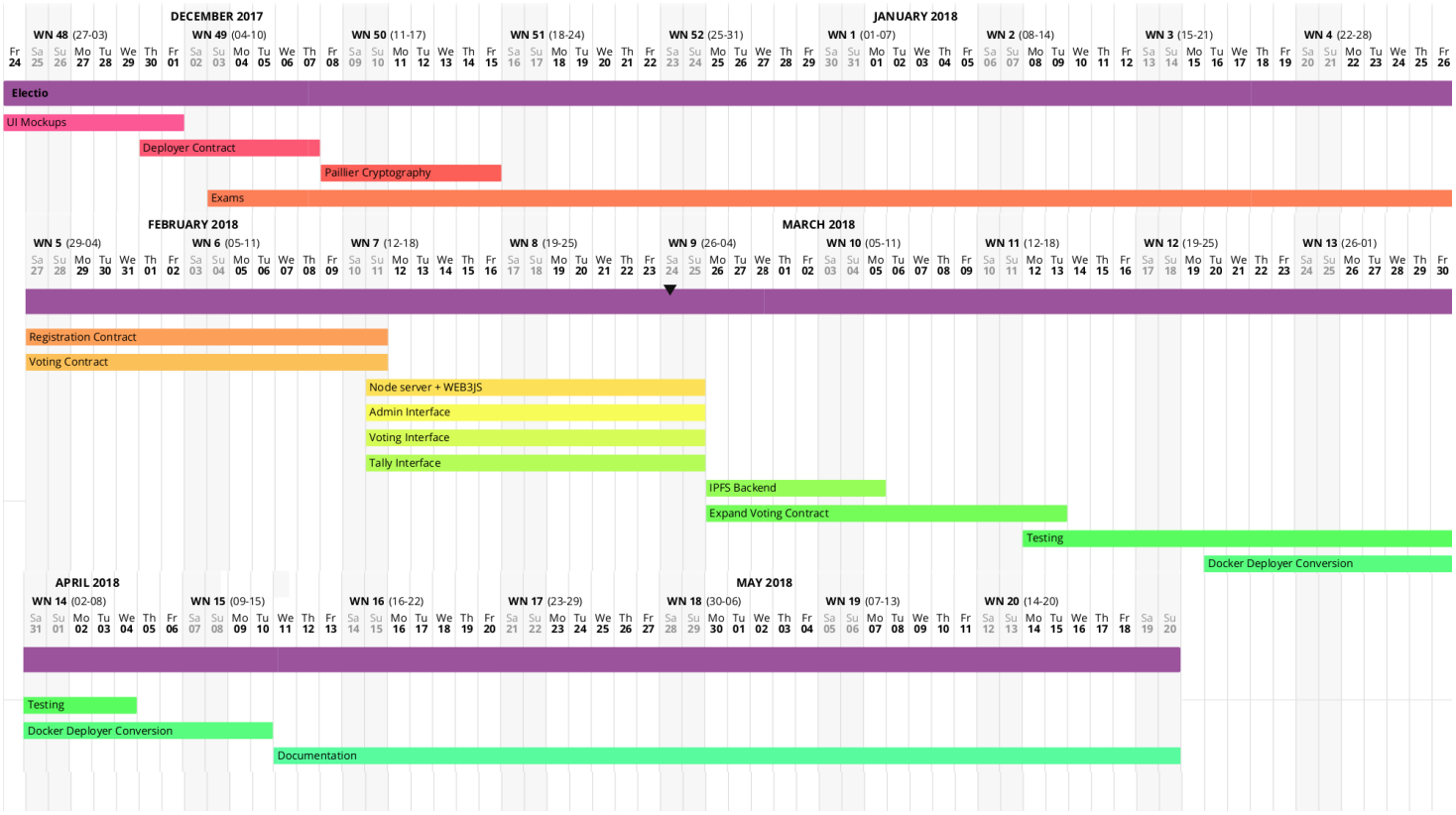
The removing of spoiled votes well be automated by the administrator interface in the application. After the voting window has closed, the administrator will process them to ensure that all votes were deposited correctly. Taking the candidate list, the application will determine the candidate numerics used by the election to represent the candidates. All votes deposited must match one of these numerics. When a vote does, it is added to the official tally on the contract. If it doesn't, it is added to a list of spoiled votes.

5.5 Tally Votes



In order for the tally is to be publicly calculable, the election administrator must have processed the election. In terms of administration, the final action of the election is to post the tally key. Once the tally key is posted, anyone can prove the result of the election as signified by the actor *User*. A person who wishes to tally the election would input the voting contracts address and the application will generate the results.

6 Project Schedule



Schedule

The above Gantt chart displays my proposed project development timeline. The schedule can be broken into multiple parts. Before the January examinations I plan on having the UI mock-ups, the deployer contract and the Paillier key generator working.
Post examinations is when most of the application will be developed, where between the last week in January to the end of February, a working prototype of the described system should be in place. The later stages of development will be expanding the voting contract to accept multiple electoral systems. Testing will be ongoing throughout the whole project timeline but a more direct focus on it will begin here to iron out any remaining problems. The documentation will be the finalisation of the project. With the summer examinations coinciding with the due date for the project, finishing ahead of schedule is the intention.

7 Appendix

[Paillier Cryptosystem]
http://www.cs.tau.ac.il/∼fiat/crypt07/papers/Pai99pai.pdf
Paillier P. 1999. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes

[Open Vote Network]
http://fc17.ifca.ai/preproceedings/paper_80.pdf
McCorry P, Shahandashti S.F, Hao F. 2016. A Smart Contract for Boardroom Voting with Maximum Voter Privacy

[Ethereum White Paper]
https://github.com/ethereum/wiki/wiki/White-Paper
Buterin V. 2014. A Next-Generation Smart Contract and Decentralized Application Platform

[Ethereum Yellow Paper]
http://gavwood.com/paper.pdf Wood G. 2014. Ethereum: A secure decentralised generalised transaction ledger

[Voting Costs Analysis]
https://medium.com/@DomSchiener/voting-on-the-ethereum-blockchain-an-analysis-67701bd1c9f5
Schiener D. 2015. Voting on the Ethereum Blockchain: An Analysis