Department of Computer Science and Engineering

The Chinese University of Hong Kong

ESTR4998 Graduation Thesis Report

LYU1803

Opensource E-voting System for 8 million mobile devices

Written by

Chan Maxwell Chun Sum (1155079378)

Supervised by

Prof. Lyu Rung Tsong Michael

23 November 2018

Table of Contents

Table of Figures

Abstract

In this project, we focus on designing and developing an application that integrates electronic voting with blockchain technology. So that the application can promote verifiability and anonymity of e-voting, which can be easily expendable to organize an election of any scale.

There are many pieces of researches on an end-to-end verifiable voting system and blockchain based voting system, but not many of them have proposed the combination of both, plus an implementation of an online remote voting system. Therefore, in this report, we take Helios, one of the popular end-to-end verifiable voting system, as a reference, and proposed improvements on the system with the use of blockchain technology.

In this term, we only implement on the basic part of the designed system. Yet, we believe that after the full implementation in the next term, the public can actually utilize it and learn about the significance of using a secure e-voting system.

Acknowledgements

We would like to express our greatest appreciation to our supervisor Prof. Michael Lyu and advisor Mr. Edward Yau for guiding us on this interesting project. They have given a lot of precious advice and feedback from doing research, designing, implementation to the presentation. This report will not be completed without their help.

Besides, we would also take this opportunity to thanks everyone who has supported us in the project or read this report.

1. Introduction

1.1. Motivation

Voting for many elections in Hong Kong is often paper-based, no matter it is in small scale like electing a student union (Figure 1-1), or in large scale like legislative council election. The problem is that the process uses a lot of time and resources, from marking to counting ballots, and even discourage voter intention to vote in person. In this digital era, almost everyone should have a mobile phone, digitalize the voting process and allowing people to vote on their own devices should be a trend in the future. Therefore, develop an e-voting application is important for Hong Kong, not only to improve the efficiency and accuracy of voting, but also aim to increase the voter turnout rate, which is important to democracy in this digital era.

According to a discussion in the legislative council [1], the reason why e-voting is not popular in Hong Kong is mainly due to the mistrust between people, the government, and computer network. Counting the electronic ballot inside a computer is just like a black box, people cannot easily monitor the process. So, they may not confident with the result, especially when the government is able to control the computer. Besides, with those report of incidents about network security threat and personal data leak, people may concern about the stability of the voting system.

On the other hand, the blockchain technology is getting much more attention recently. Despite the popular application on those cryptocurrencies such as Bitcoin and Ethereum, it is also used commercially to provide reliable and trusted data to the public, just like the flight data by Hong Kong International Airport [2]. Blockchain is popular because it can provide transparency, auditability, decentralization, etc [3]. If an e-voting system contains these characteristics, it will be more secure and reliable. Therefore, we would like to study the possibility of intergrading blockchain technology with an e-voting application, such that it can be used and trusted by the public.

Figure 1-1: A traditional voting booth in CUHK for student union election (Source: www.facebook.com)

1.2. Background

1.2.1. E-Voting consideration

End-to-end verifiability

An e-voting system should be end-to-end verifiable to promote the integrity of ballots [4], such that the result from a tallied election correctly reflect the intention of all voters. The end-to-end verifiability includes three parts (Figure 1-1).

Cast-as-intended verification

A voter should be able to verify the submitted ballot correctly record his intent. In the case of an encrypted ballot, the voter needs to verify the encryption system is working well so that it will be equal to the original ballot if decrypted. Also, if there is any unauthorized modification to the ballot after submission, the voter can notice the change.

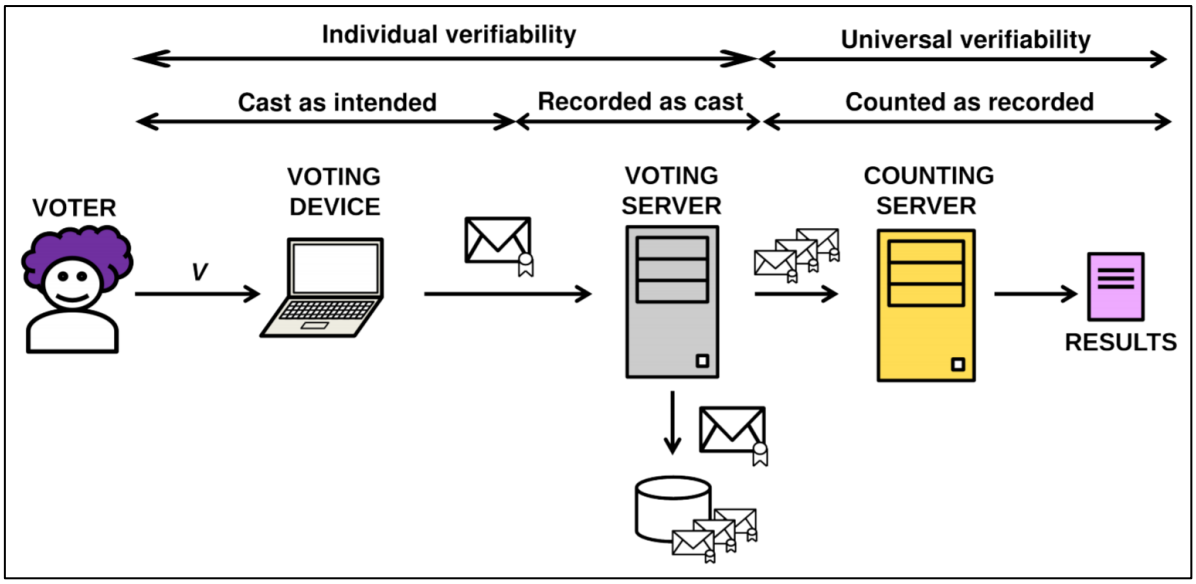


Figure 1-2: End-to-end verifiable voting (Source: fc16.ifca.ai)

Record-as-cast verification

A voter can verify that the ballot saved on the server is the same ballot as he submitted. So that there is no error on ballot transmission, and the ballot has not been deleted any time before tallying.

Tallied-as-recorded verification

A voter should be able to verify his vote is in the tally of an election result. More importantly, any voter can verify that a ballot is tallied if and only if it is valid, which means the tally should only include all ballots that voted by an eligible voter.

Voter authentication & anonymity

There should be an authentication system so that only eligible voter for that election can enter the voting system. However, after authentication, the submission of the ballot should be anonymous, so that nobody can tell the owner of the ballot and which options the voter has voted for.

1.2.2. Blockchain

Blockchain is a way of storing data in a system using a chain of blocks. A basic block should contain the sequence number, data, hash of this block, and the hash of the previous block (*Figure* 1-1). Modification to block data is hard by design, because a minor change in data will change the hash value, and thus affect all hash values in blocks after that. So, verification of the data can be easily done by checking the hash of each block, making the blockchain reliable.

Figure 1-3: An example of blockchain (Source: rubygarage.org)

Besides, blockchain is designed to be distributed and decentralized, so that everyone can become a node in the system and read the data. Therefore, this will require a to agree on the generated block.

In a permission-less blockchain like Bitcoin, usually, a proof-of-work system is used [5] so that a nonce, which can hash to a specific value, is needed to be calculated by the node that generate the block. As the nonce calculation is computationally intensive, so it is also hard to have multiple nodes generates a block at the same time.

In a permissioned blockchain, only some trusted server/node can generate a new block. Proof-of-work usually not used, as the number of these nodes is relatively smaller [5]. Instead, a trusted node will be selected, may be randomly or in an order, to generate new block every time, and store its identity into the new block.

However, if there are some dishonest 'trusted nodes' in the permissioned blockchain, or the blockchain is not run in a trusted network, then a consensuses algorithm is needed for all of them to agree on a node to generate new block every time.

Byzantine Fault Tolerance is one of the algorithms that usually used for this purpose [6], it is proven that having at least two-third honest nodes are enough for the consensuses. Assume that there is a source that tells every node which node the block generator is, but the source is dishonest (Figure 1-2). According to the algorithm, every node will broadcast its piece of information to other nodes. After that, the decision in each node would be the majority of what it has received.

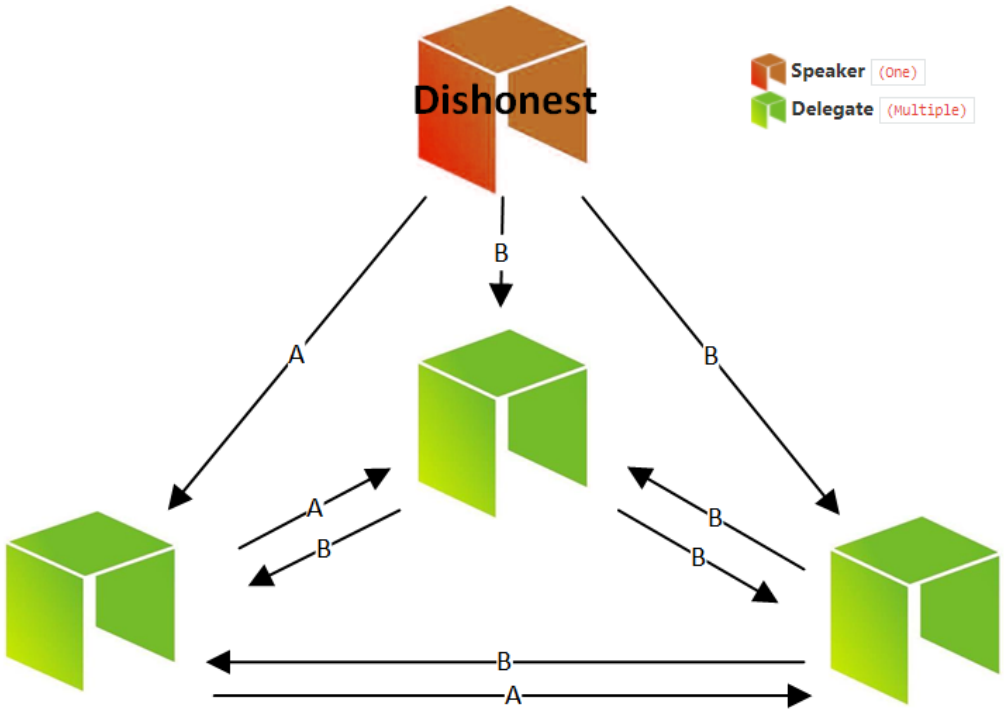


Figure 1-4: Situation when the Speaker is dishonest in this case (Source: steemit.com)

As the nonce or the identity cannot be gotten easily, these consensus protocols make the block data modification nearly impossible, unless more than half of the computers in the network become malicious, or there exists a polynomial time algorithm to crack the cryptography.

1.3. Objective

The goal of the entire project is to design and build an e-voting application that can be used by everyone to vote in an election of any scale. The application should use blockchain and other technologies to enforce all consideration of e-voting, including end-to-end

verifiable, voter authentication and anonymity. We hope, via the release of the application, the public will learn that e-voting can be more transparent and reliable than traditional paper-based voting.

The entire project is expected to be complete in 2 terms. Here's are the objectives for the first term:

- Explore and study the related research and implementation of end-to-end verifiable voting and blockchain voting
- Design the flow of an election, from creating election to releasing result. Also, decide the type of blockchain and it's detail protocol to be used for the e-voting application.
- Implement the backbone of the application and the basic part of the election, so that user can create, vote and tally an election.

2. Related Work

2.1. E-voting in Hong Kong

While there is no large-scale e-voting used officially in Hong Kong, many tiny-scale e-voting is just done by setting up a simple website or using Google Form, which is obviously not fulfilling most of the e-voting consideration stated in Section 1.2.1.

The largest e-voting system in Hong Kong is PopVote which is for conducting civil referendums, such as the unofficial vote on issues related to constitutional reform proposals [7]. PopVote had been using website and mobile application to conduct the voting (Figure 2-1), but it changed to use Telegram Bot last year. No matter which medium is used, the design of voting is not end-to-end verifiable, voters can only trust the system.

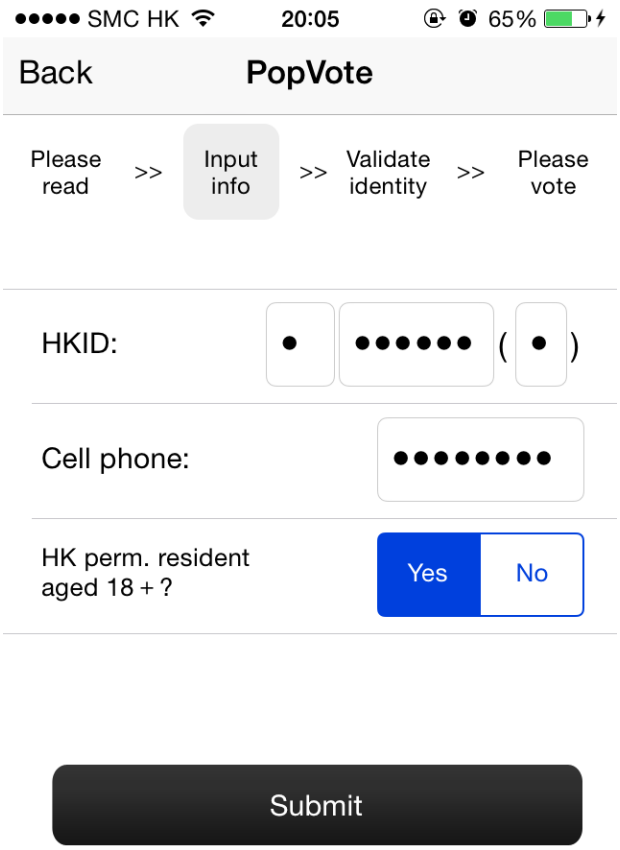


Figure 2-1: Registration in PopVote mobile application (Source: popvote.hk)

Although it involves authenticating voter via HKID with the phone number and claims

that the personal data won't be sent to the server, it can suffer from Denial of Service attack [8] and contains security loophole on disclosing personal data [9].

2.2. End-to-end verifiable voting

2.2.1. Helios

Helios is an opensource voting system with an online implementation (Figure 2-2) which invented by Adida [10]. It uses homomorphic encryption system for secure encrypt and decrypt the ballots, and uses a ballot bulletin board to enforce end-to-end verifiability.

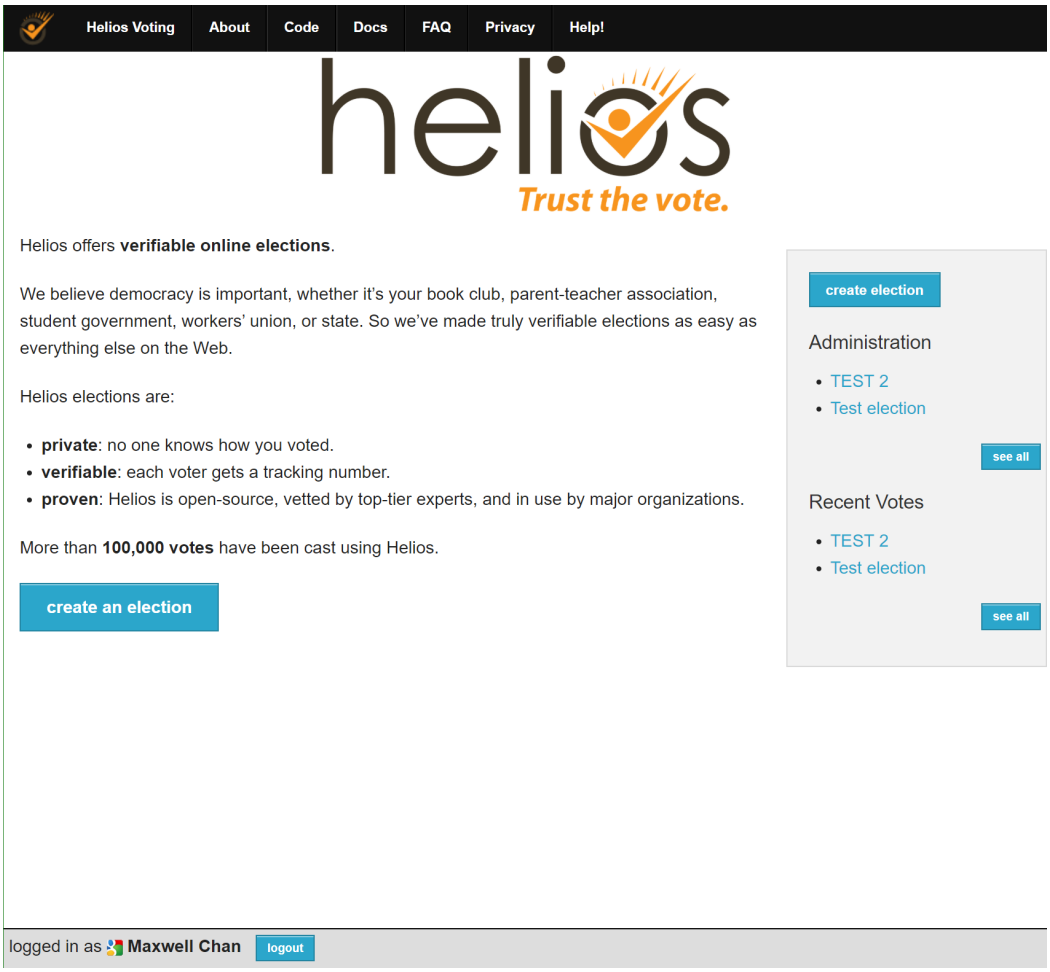
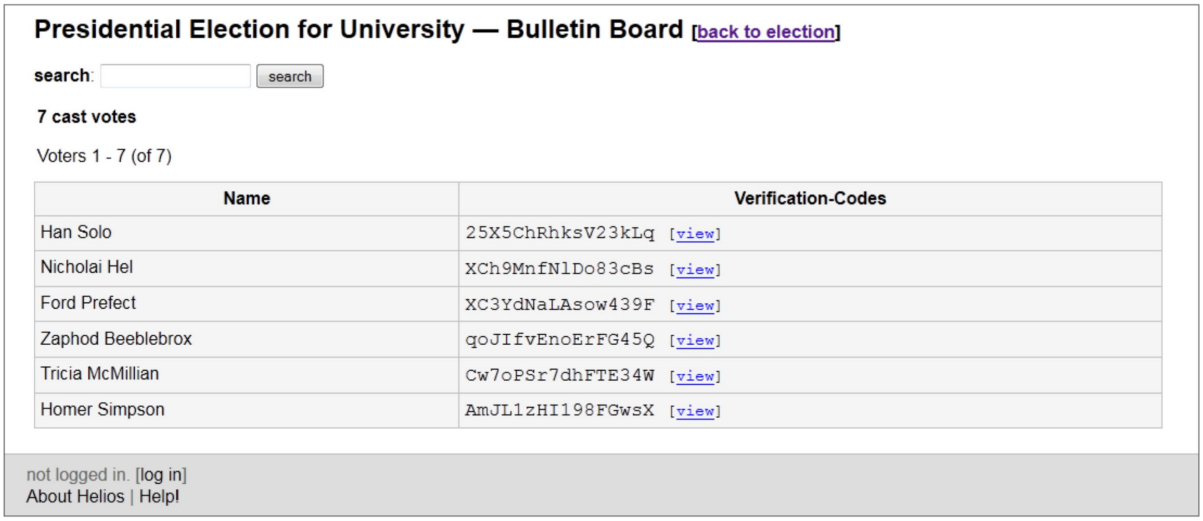


Figure 2-2: The online implementation of Helios

To create an election, besides setting the question and election details, a group of trustees is needed to be selected, so that each of them holds a different public and private

key pair. All the public keys will be aggregated to become the election public key.

Voting in Helios is easy as clicking some radio button. The browser will encrypt the answer with some randomness and election public key, then a ballot fingerprint, i.e. the hash of the ballot, will be generated for the voter. A voter will be authenticated by Facebook or Google account, after that the ballot will be submitted with the voter ID only, which cannot recover the voter identity by others. On the other hand, the ballot bulletin board (Figure 2-3), which accessible by the public, will post all submitted ballot fingerprint, so that voter can verify that his vote is counted in the tally correctly.



Presidential Election for University — Bulletin Board [back to election]

search: search

7 cast votes

Voters 1 - 7 (of 7)

Name	Verification-Codes
Han Solo	25X5ChRhksV23kLq [view]
Nicholai Hel	XCh9MnfNlDo83cBs [view]
Ford Prefect	XC3YdNaLAsow439F [view]
Zaphod Beeblebrox	qoJIfvEnoErFG45Q [view]
Tricia McMillian	Cw7oPSr7dhFTE34W [view]
Homer Simpson	AmJL1zHI198FGwsX [view]

not logged in. [log in]
About Helios | Help!

Figure 2-3: Ballot bulletin board in Helios (Source: www.usenix.org)

To compute the result, all ballots are aggregated homomorphically. Then, all trustees are required to provide his private key one by one to totally decrypt the aggregation. This process can be monitored by the public, as everyone can verify the aggregation. More importantly, no single ballot is decrypted in the whole process, unless all trustees are malicious. Therefore, even in the case of failure of the authentication system and voter identity is disclosed, his vote intention won't be leaked out.

2.2.2. Pretty Good Democracy

Pretty Good Democracy was proposed by Ryan and Teague [11]. This voting system uses special 'code voting' method to provide end-to-end verifiability, as well as proving voter and server identity

Prior to election begins, each eligible voter will receive a unique ‘code sheet’ (Figure 2-4), which provide a ‘vote code’ and a ‘acknowledge code’ for each option. All the codes are randomly assigned, so to act as a shared secret between all trustees and each voter.

Candidate	Vote Code	Acknowledgment Code
ALCHEMIST	5962	218931
ANARCHIST	2168	854269
BUDDHIST	3756	129853
MARXIST	1247	875391
NIHILIST	9881	039852

ID: 4896327

Figure 2-4: Code sheet in Preety Good Democracy (Source: arxiv.org)

In order to vote, a voter needs to input the ‘vote code’ for the option he chooses into the system. Then, the system can the authenticate user via the code and it should respond the corresponding ‘acknowledge code’. The voter can compare it with the one on the ‘code sheet’, so as to ensure the server is not malicious. Notice that this mechanism can also prevent someone sniffing in the middle, as the sniffer can deduce nothing meaningful with only these codes.

Also, ballot bulletin board is used to show the ID of submitted ballots for voters to verify. To calculate the result, a mixnet is used to further anonymize the ballot, tallying can be done only after that.

2.2.3. In-person voting systems

There are more end-to-end verifiable voting systems that require perform voting in a voting booth. Usually in these systems, after voter marked his choice on the physical ballot, the ballot will be processed by a computer, while the voter will get a receipt for verification at a later stage. These systems include Prêt à Voter [12], Scantegrity [11], Punchscan [13], Votegrity [11], etc.

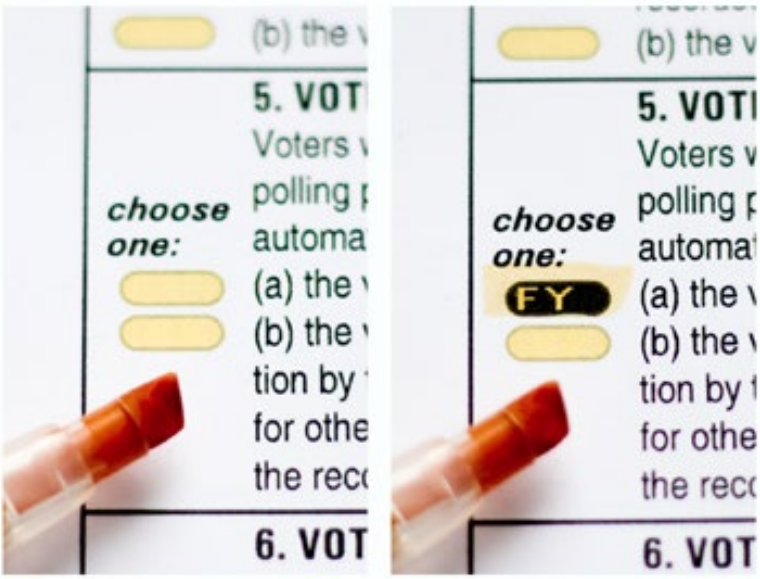


Figure 2-5: Using special pen for voting (Source: en.wikipedia.org)

Take Scantegrity as an example. In its physical ballot, a random code is assigned to each option, these codes are different for each ballot and each option. Moreover, the code is printed with invisible ink, which only becomes visible when voters mark the choice using a special pen (Figure 2-4). Therefore, voters can get the code together with the ballot ID as a receipt, which they can use to verify online if the submitted ballot is really in the database.

2.3. E-voting using blockchain

2.3.1. A vote as a coin

There are proposals [14] [15] and implementation [16] of treating a vote as a coin in cryptocurrency, such that submitting ballot will be like making a transaction (Figure 2-5).

Basically, the idea is that every voter and candidate have their own wallet. Initially, the number of coin in the voter's wallet will be equal to the number of choices they can vote. In order to vote, a voter needs to transfer a coin into the candidate wallet. It is not allowed to transfer coin into another voter's wallet or transfer more than one coin to the same candidate from the same voter. In this design, tallying is easy, as the number of coin in a candidate wallet directly reflect the number of votes he receives.

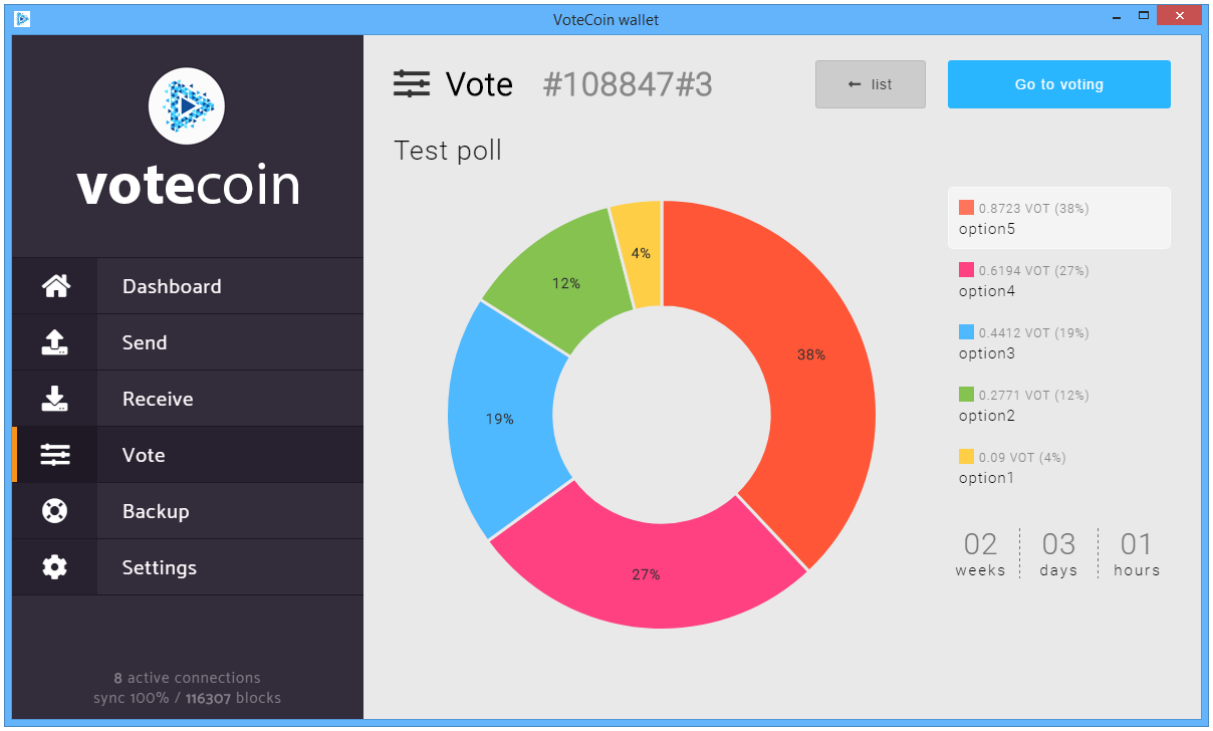


Figure 2-6: VoteCoin application (Source: www.votecoin.site)

Because of the transparency characteristics of blockchain, everyone can easily monitor and verify all activities in the blockchain network, so it can achieve more than end-to-end verification. Just like cryptocurrency, every wallet is identified by an address, only one with the corresponding private key can spend coin in the wallet. Therefore, voter authentication and anonymity also achieved.

However, the problem with this design is the disclosure of intermediate result. In the blockchain, everyone knows how many coins voters and candidates have. So, one can easily get a real-time result of the election. This is not appropriate because it can affect the vote intention of those voters that not yet voted, making the election unfair. Another problem to this blockchain is that voter can prove to others which candidate he votes for, just by telling others his wallet address. This can make vote-buying easily available, causing corruption in the election.

2.3.2. Use blockchain for secure storage

Regardless of the encryption and decryption algorithm used for an election, other proposals [17] [18] [19] are trying to use blockchain mainly for a non-modifiable storage to

support their election algorithm.

Use the research by Panja [20] as an example, it uses two blockchains. One blockchain will be used to store voter authentication details, this will be a private blockchain that only accessible by trusted parties. The hash value of the last block will be shared among them to verify the blockchain during the election. Another blockchain is a public one for storing ballot, it uses proof-of-work for consensus algorithm. The first block in this blockchain will store the election public key and signed by the administrator, for public verification purpose.

3. Design

3.1. Overview

After studying different end-to-end verifiable voting systems, we decided to use Helios' algorithm as the reference of our design. The main reason is that Helios is opensource and has a workable online implementation, which can demonstrate its merit and loophole easily, and also able to make obvious comparison with our final product. Besides, it uses homomorphic encryption to ensure no decryption of a single ballot, which is crucial to anonymity. While Helios has its limitations, we will discuss later and propose possible modifications.

For the blockchain part, we will use it as a storage for election details and encrypted ballots in Helios, so that everyone can monitor the entire election.

3.2. Algorithm related to the voting process

3.2.1. Helios cryptography

Helios uses homomorphic El Gamal encryption scheme, which is a variant of El Gamal encryption [21].

Create election

To set up a new election, one needs to generate the following parameters, they will be used consistently throughout the election:

p: a prime number

g: a primitive root[1] of p

[1] If g is a primitive root of n, then for every integer a coprime to n, there is an integer k such that $g^k = a \bmod n$; a is coprime to b if the greatest common divider of them is 1

For every trustee i, he need to generate:

x_i: trustee private key, where $0 < x_i < p - 1$

y_i: trustee public key, where $y_i = g^{x_i} \, mod \, p$

All trustees' public key will be submitted to calculate the election public key y:

$y = y_1 y_2 \ldots y_n \, mod \, p$, where n is the number of trustee

Ballot preparation

For each option in each question, the following are needed to be calculated:

r: a random number, where $0 < \mathrm{r} < \mathrm{p} - 1$

$$m = g^i \, mod \, p,$$

where $i = 1$ if voter choose this option, $i = 0$ if otherwise

$$c_1 = g^r \, mod \, p$$

$$c_2 = y^r m \, mod \, p$$

Then, the encrypt answer (c_1, c_2) will be submitted for each option.

The random number is important, as we do not want the voters that vote to the same option get the same encrypt answer, thus violating voter privacy. Therefore, the random number will not be shown to voter for security reason. However, if a voter is concerned about the trustworthiness of their browser, Helios allows a voter to view it for cast-as-intended verification, but then the answer will be re-encrypted using another set of random number.

Tallying election

The following result computation process is repeated for each option in each question. As the trustees' private key will not be sent to the server, so firstly the server need to compute:

$$c_1 = c_{1,1} c_{1,2} \ldots c_{1,a} \, mod \, p$$

$$c_2 = c_{2,1}c_{2,2} \dots c_{2,a} \; mod \; p$$

where a is the total number of voter,

$(c_{1,1}, c_{2,1})$ is the encrypt answer for voter #1

While on the client side:

$$g^m = c_2({c_1}^{x_1} \, {c_1}^{x_2} \dots {c_1}^{x_n})^{-1} \; mod \; p$$

where n is the number of trustee[2],

x_1 is the private key for trustee #1

m on the above equation is the actual number of voter that vote for this option. In order to get m from $g^m (mod \; p)$, we need to perform a discrete logarithm of g^m with base g. While there is no polytime algorithm to do discrete logarithm, m must be smaller than or equal to the total number of voters. So, what Helios use is just trying all possibilities, this will not be very time consuming when the total number of voters is not too huge.

The focus point of the entire encryption and decryption flow is on the g^0 and g^1 encoding in c_2 when encrypting voter choice. As $g^a g^b = g^{a+b}$, when we do the aggregation of c_2 for all voters, it will be able to produce g^m, so that we don't need to decrypt individual ballot one by one.

3.2.2. Limitation on Helios and proposed modification

Denial of service attack

Currently, Helios implementation is not designed to be run on distributed servers and databases. So, there is a risk that the server may under attack and become unresponsive or even making changes to the database. Thus, Helios also claim that it is not suitable for large-scale election.

That's why we would like to implement Helios on a blockchain, so that the signal point of failure is removed from the system [22]. Everyone can make a copy of the database so that in case of all servers fail, the ballots won't be lost or modified. Besides, if any server is

[2] The $^{-1}$ is the equation means modular multiplicative inverse, so that $a^{-1} = x$ when $ax \; mod \; p = 1$

under attack, one can easily set up another server by just joining the blockchain network with a computer. Therefore, overall availability and integrity will be improved.

Moreover, using blockchain make tracing error easier. Previously in Helios, if a voter suspects that there is inconsistency in the ballot bulletin board, it is hard for a voter to tell where the error is come from, as he does not know what is going on in the server. With distributed server and database, one can have more information to check the record of all data.

Slow tallying

As described in Section 3.2.1, computing result in Helios needs to perform discrete logarithm. While the speed is acceptable when number of voter is small to medium, it can be slow in a large-scale election, since this action needed to be done for every option in every question. Another bottleneck will be in aggregating the encrypted answer on server side, as it will involve lots of multiplication.

So, we suggested allowing decrypting ballots in batch. When tallying, divide all ballots into serval batch (Figure 3-1). As we have distributed servers, we can assign different servers to be responsible for different batches, such that the decryption of all ballots can really be done in parallel, thus reducing the overall time of tallying.

This will not violate ballot anonymity as long as the group size is not too small to make guessing of individual ballot intention possible. Also, everyone in the blockchain network can verify the aggregation done by the servers, to make sure they won't give a single ballot to trustees for decryption.

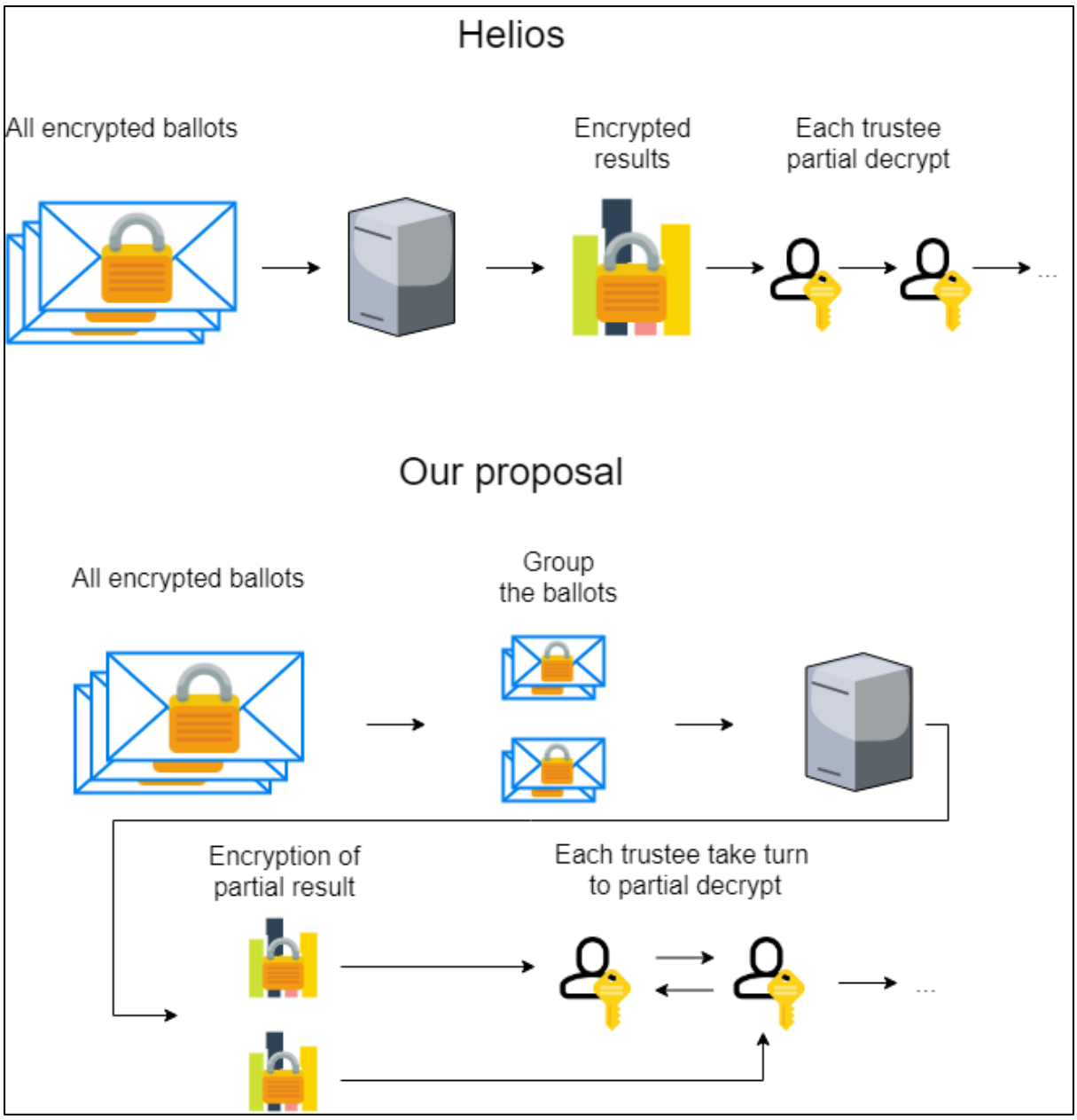


Figure 3-1: Our proposal on allowing decrypt an election in batch

Coercion

Coercion in voting means that a voter is forced to vote for some candidate. This can be done in serval ways: voter prove to coercer that what he had voted, coercer sits next to voter to force him on voting, and voter gives his credentials to the coercer. The first way is not possible in Helios, as no one can deduce the selected options from the encrypt ballot, even with the help of the voter.

However, the second and the third way is still possible in Helios. This is actually a common problem in a remote voting system, the server never knows who is actually sit next to the voter. Helios try to reduce the possibilities of coercion by allowing re-voting [21]. A voter can vote as many time as he wants in an election, only the last submitted ballot will be tallied. So that if a voter is being coerced previously, he can vote again when the coercer has left. Nonetheless, the coercer can still notice the change in ballot fingerprint if he knows the ID the voter.

While we will keep this re-voting mechanism, we think it is not enough. So, we proposed to add an option for in-person kiosk voting when setting up an election. If election organizer uses this option, he needs to provide an extra RSA public key, while keeping private key secret, when creating election. Election organizer needs to set up some kiosks, just like voting booths in paper-based voting, for a voter to vote in-person. Inside the kiosk, voter just needs to perform voting on a computer as normal. The only difference is that this computer contains the private key, which will sign the ballot and then send to the blockchain network (Figure 3-2). Therefore, the voter cannot vote again on their device after they vote in the kiosk.

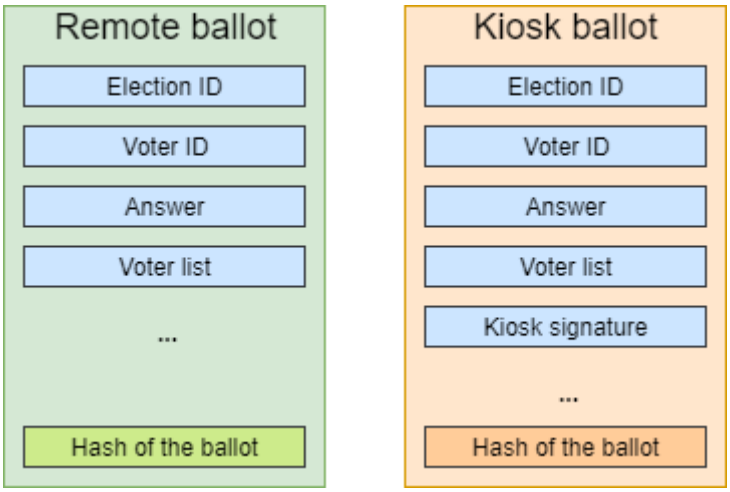


Figure 3-2: Difference between a ballot encrypted by kiosk to that by personal devices

We thought that the risk of coercion increases with election scale. For example, coercion risk is low for student union election, while it is high for district council election. Also, in a large-scale election, even for e-voting, some kiosk will be needed to set up for voters not having electronic devices or having technical issues. So, what we have just proposed simply add a functionality to these kiosks, which allow voters who are really

concern about the risk of coercion to vote in the kiosk.

Authentication method

As stated in Section 2.2.1, Helios authenticate voter via Facebook or Google account. After authentication, the corresponding voter id and hash is put into the vote. The problem here is that the public can only trust the authentication by Helios, they cannot further verify.

Therefore, we suggested having ballot signature binded with the submitted ballot. When creating an election, every voter needed to generate an RSA private and public key pair. The public keys of all voters are stored in the blockchain together with their voter ID. So, when preparing the ballot, one hash will be generated for all encrypted answers, and voters are required to provide their private key to sign on the hash before submitting. Such that everyone can verify the hash by using the corresponding public key.

Nevertheless, the one using the private key may not be the one who owns the private key, the coercion problem may come again. Hence, we encourage election organizer to use valuable credentials for further authentication.

Valuable credentials mean that the credentials cannot be created easily, and the voter may use these credentials for other important services' authentication, so that they are less willing to give it to others. For example, if it is an election for students in CUHK, then CUHK OnePass credentials will be valuable, as students use it for all IT services in CUHK like email (Figure 3-1). Another example would be biometric information like fingerprint and facial data, as people nowadays may use it for unlocking the mobile phone and even online banking. A counterexample would be Facebook and Google account, as creating a new account is very easy.

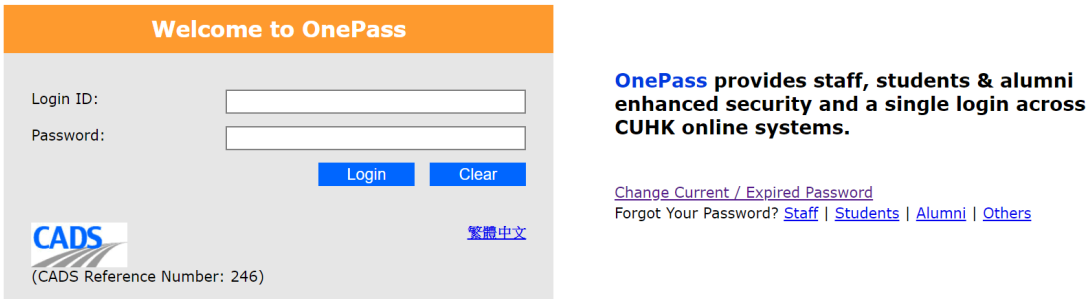


Figure 3-3: CUHK OnePass login interface

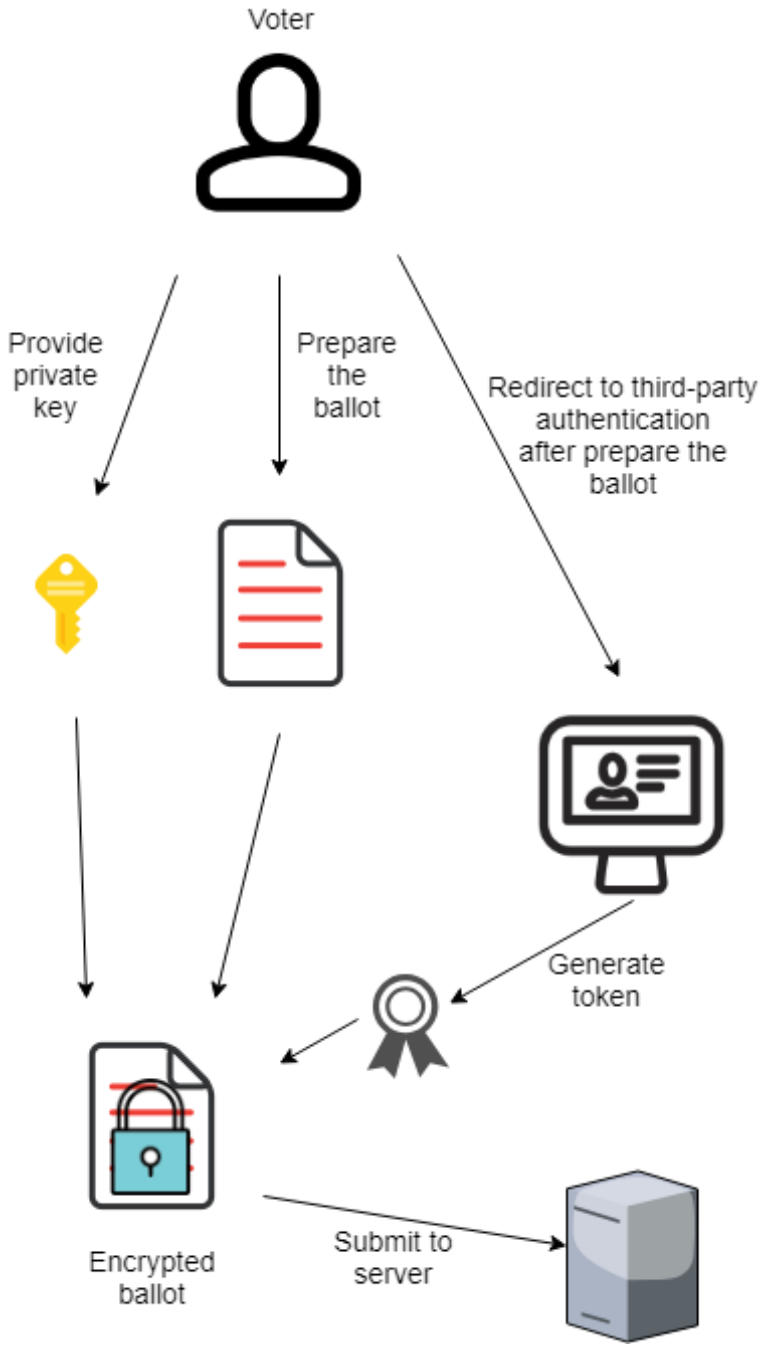


Figure 3-4: Overview of steps for using third-party authentication

As we want our voting application to be generic enough for an election of any scale, we will not enforce the authentication method stated above. Instead, we planned to offer an API[3] for any election organizer to provide their addon authentication method. Before a voter submits the ballot, he will need to pass through the third-party authentication first. In case of successful authentication, it will sign on the voter ID and submitted with the

[3] Application Programming Interface

ballot (Figure 3-4).

Knowledge of who has voted

In order to achieve record-as-cast verification and tallied-as-recorded verification, a ballot bulletin board is used in Helios, showing all election fingerprints with the corresponding voter ID or voter alias[4]. While no one can deduce voters' intention, one can still know whether a voter has been voted or not, if he knows the owner of a voter ID.

We believe that this is not a significant problem, as in any traditional paper-based voting system, it is always possible to know whether a voter has been voted or not by monitoring the entrance of voting booth. This may become a problem only when the voting intention of a voter is obvious, and his voter ID is known by a coercer which has opposite voting intention, then the coercer may coerce him not to vote.

Thus, we will not make changes to the mechanism of Helios here. Instead, we will encourage the election organizer to implement three measures. Firstly, voter ID should not be guessable and do not reuse the voter ID from one election to another election, so it is less likely for one's voter ID and his voting intention to be known by others. Secondly, always add an 'Abstention' option in an election, so that a voter intention is more difficult to guess by whether he has voted or not. Thirdly, educating voter not to disclose their voter ID, as this may infringe their own privacy.

3.3. Blockchain related protocol

3.3.1. Type of blockchain

Since the blockchain will be used for storing ballots and all election details, we want it to be publicly verifiable, so it must not be a totally private blockchain. Thus, we need to choose between permissioned and permission-less blockchain.

Permission-less blockchain works well in cryptocurrency and smart contract.

[4] Election organizer can assign alias to voters instead of generating voter ID

However, when it comes to voting, the 51% attack [23] may be easily possible. Notice that everyone in a permission-less blockchain have a right to verify a ballot and put it into a block. Although the longest chain will be chosen as the right one, if there is a fork in the blockchain of the same length, the one that majority nodes have will be chosen. Consider a situation that there are only two candidates in an election, one of them has more computational power and own more than half of the nodes. These nodes can just approve the ballots from 'trusted voters', who is the voters of their side, and disapprove all other ballots. As a result, the election will not be a fair one.

Another reason that permission-less blockchain is not quite suitable is that, usually the consensuses algorithm they use is proof-of-work, which is quite computationally intensive. As we want our blockchain to be suitable for an election of any scale and easy to use, it is not a good idea to use proof-of-work, or purchasing this kind of service.

Therefore, we decided to use permissioned blockchain. The only trust here is on the trustees, just like the Helios' algorithm rely the trust on all trustees. As long as not more than half of the trustees are malicious, it is still a fair election.

3.3.2. Roles and permission

There are 3 different stakeholders here: trustee, voter, and the general public. Trustees will have the right on both reading and writing on the blockchain, while voter can only read the blockchain and submit their vote via trustees. The general public may have the right to read the blockchain for an election, depends on whether the election organizer wants the election to be known by everyone.

Trustees' controlled computer should connect to each other, so that the consensuses will make quicker. For voters or the general public, usually they should connect to any these trustee's computer to read the blockchain, but they can also connect to another one's computer to relay the blockchain if they trust it (Figure 3-2).

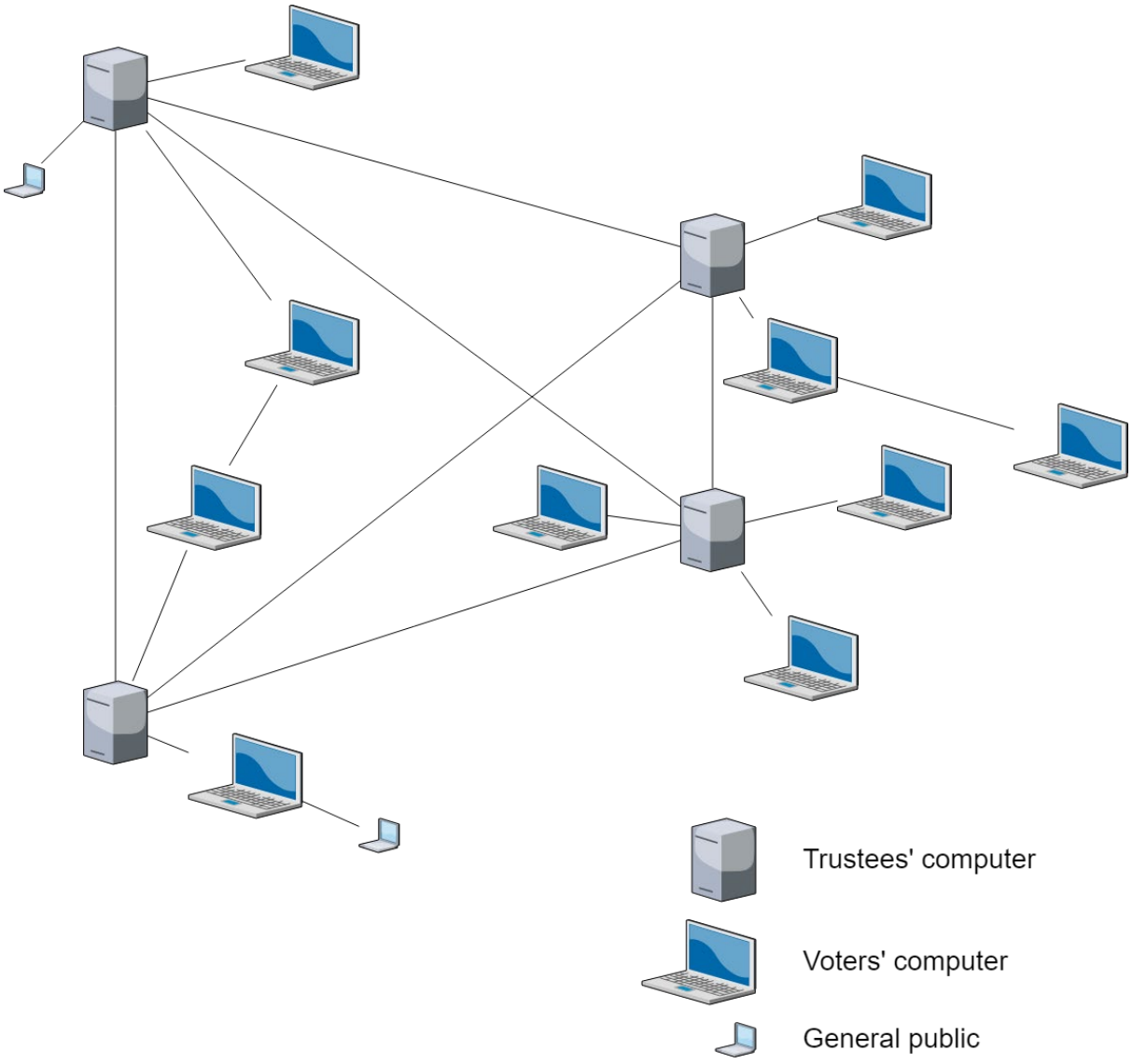


Figure 3-5: An example of connections among different computers

3.3.3. Design a blockchain protocol for voting

Unlike public blockchain like Bitcoin and Ethereum, which has many different implementation and library available online, there are not many opensource libraries for permissioned blockchain. One of them is Hyperledger Fabric [24] (Figure 3-3), which is contributed by IBM. However, its first release is only one year ago, and it is still constantly developing. Although for our application we can use the Hyperledger Fabric with suitable modification, there may be some security loopholes that we didn't notice, as it contains more functionality than we actually need.

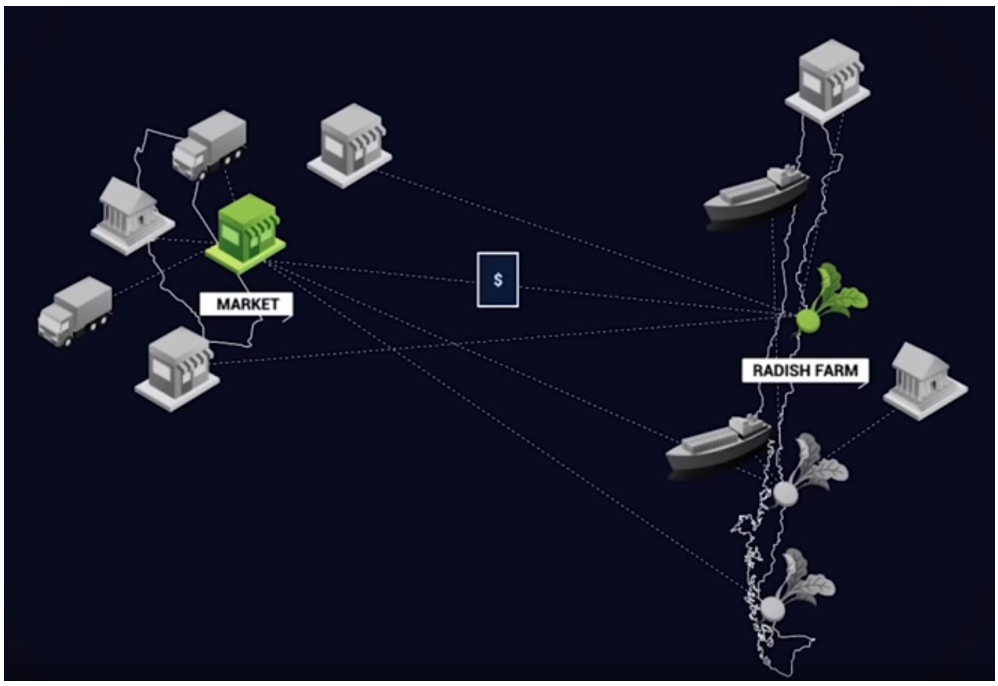


Figure 3-6 Screenshot from Hyperledger Fabric's introduction video

Hence, we would like to define our blockchain protocol for the voting application. In fact, this will bring us more advantages. First, it will be more lightweight, as we will just design what we need. Second, it will be more easy to run the functions related to voting, such as vote aggregation, since we can simply define the block structure as we want.

Although making new protocol may impose new vulnerabilities, we will try our best to search for these and take relative measures. More importantly, our application will be opensource and will not rely on security by obscurity, such that others can also prove the security.

3.3.4. Block design

Every election will have its own blockchain, since the trustees involved may be different and some elections may not be publicly available.

The first few blocks will contain all details required for an election, such as election ID, questions, election public key, list of voter ID, voters' public key, etc. Election organizer can make changes to these details by including new data into a new block, so that the latest piece of information will be used.

Until a block contains a key-value pair with the key of ”Frozen At” and value of current time, election organizer may no longer make changes to the election details. The hash value of this block will become the hash value of the election details, for later verification purpose. After that, every block will contain a set of ballots that are submitted and verified before that block is generated (Figure 3-4). A block will be generated in a regular time interval, with a value preset before.

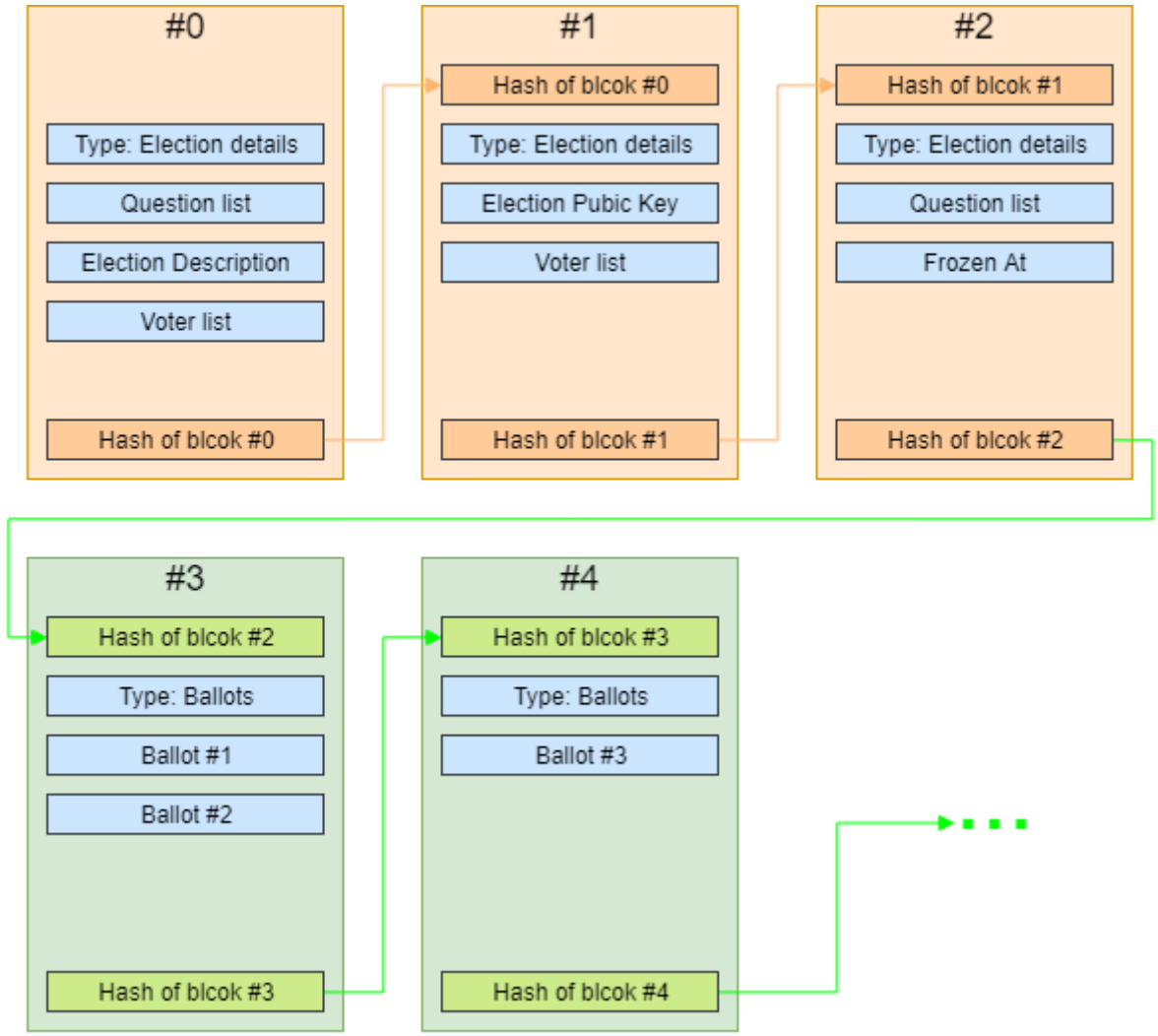


Figure 3-7: An example of blocks in the blockchain

3.3.5. Handshake protocol design

Here is the procedure of how a new node *X* join an existing network (Figure 3-5):

i. *X* get the address of any node *Y* in the network and send a connect request

ii. *Y* verify the request from X and save the address of *X* into the database

iii. *Y* send back addresses of all nodes in the network to *X*

iv. *X* save all the address into its database and ping all other nodes

v. Other nodes save the address of *X* when receiving its ping request

All nodes in the network will ping each other periodically to check their liveliness, a node will be deleted from the database if it does not response from ping.

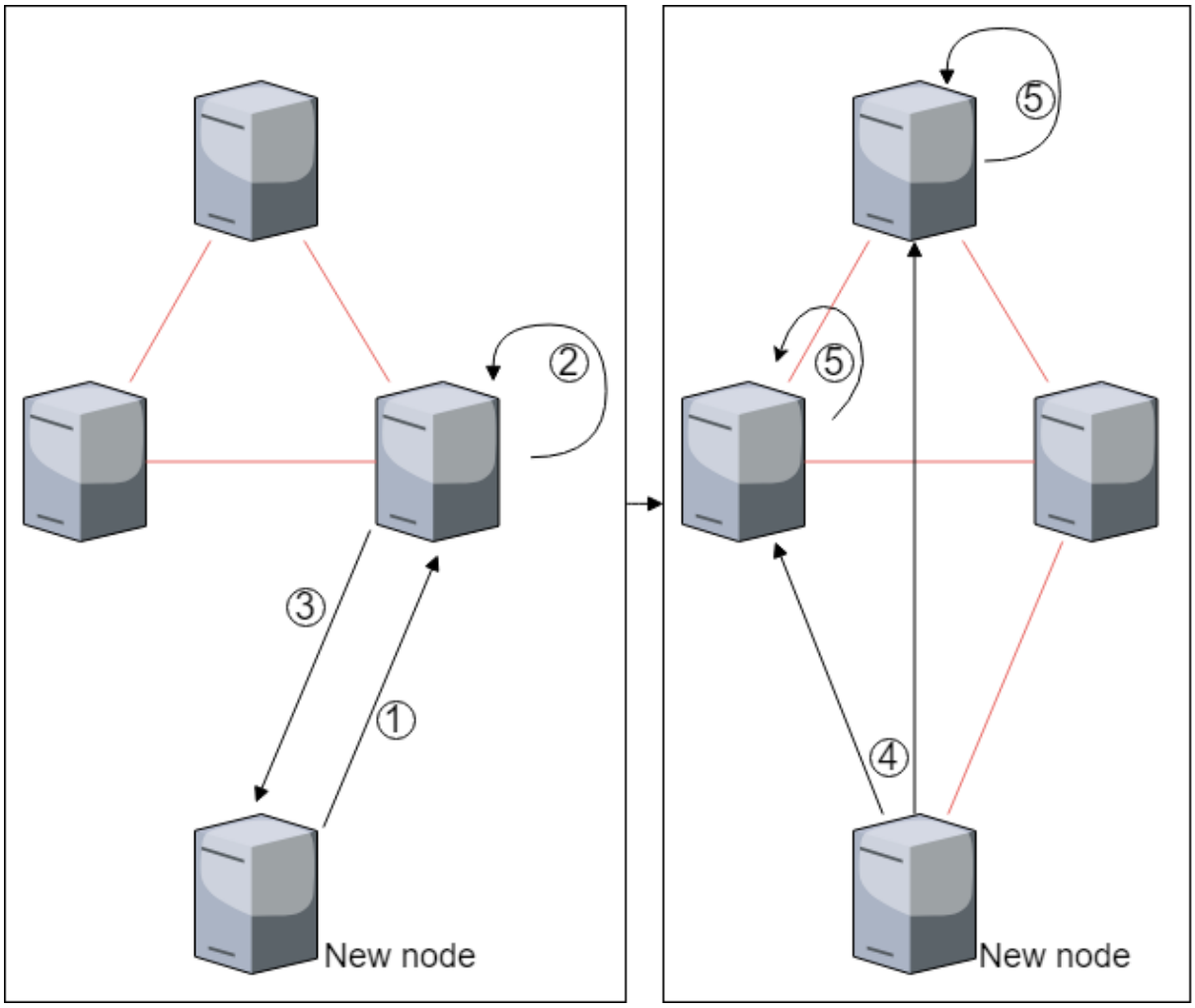


Figure 3-8: Steps of the handshake protocol

3.3.6. Ballot submission protocol design

Here is the procedure when a node *X* in the blockchain network receive a ballot from

a voter (Figure 3-6):

i. *X* verify the ballot using the voter public key and save it into the local database

ii. *X* broadcast the ballot in the network

iii. All other nodes will verify the ballot using the voter public key and save it into its own database

iv. If the ballot is valid, all nodes include *X* will sign on the ballot hash and broadcast the signature in the network

A ballot is considered verified when more than half of the trustees have signed on it.

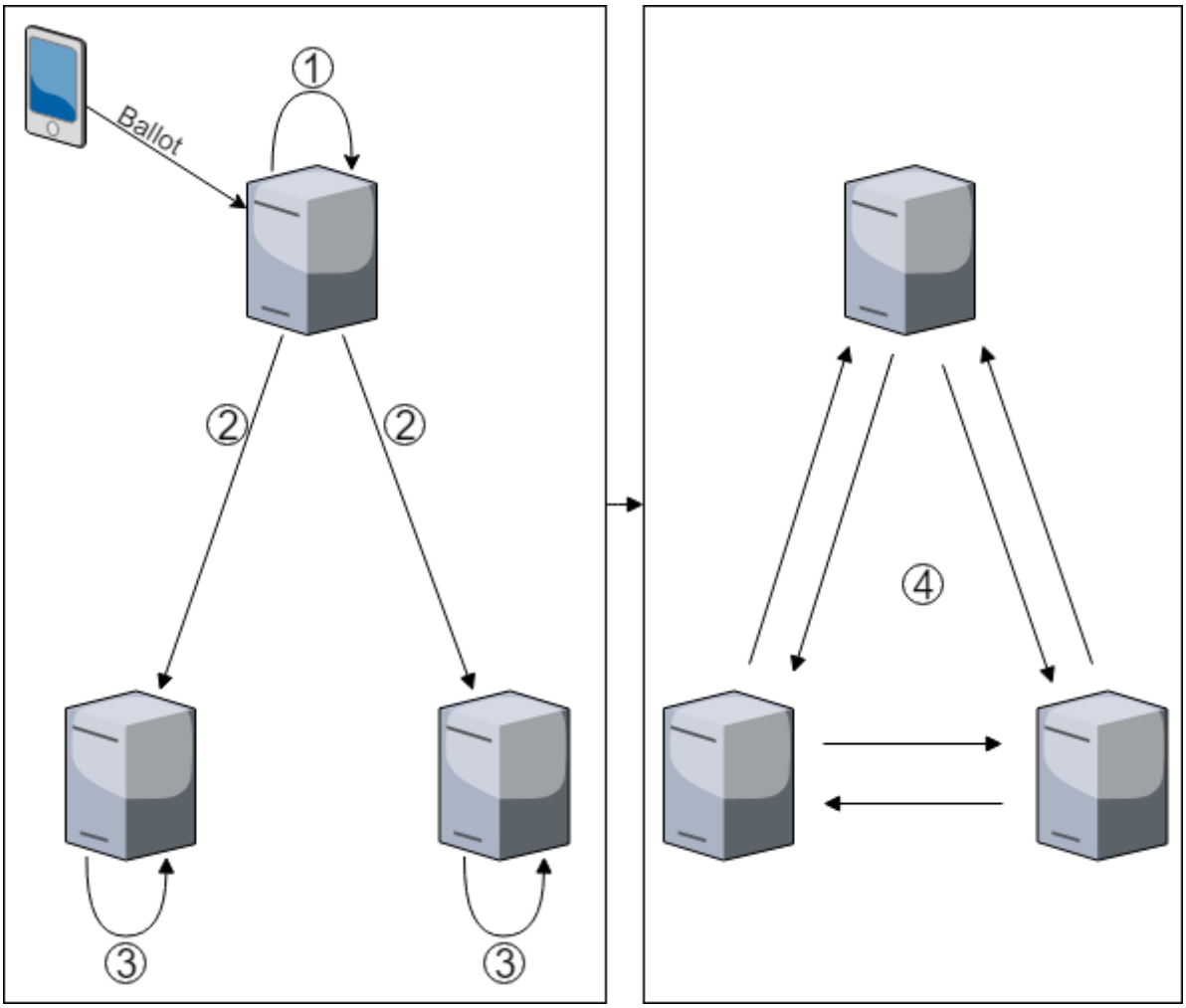


Figure 3-9: Steps of the ballot submission protocol

3.3.7. Block generation process design

Node selection protocol

As a block is generated in a regular time interval, all nodes need to select a node to be the generator every time. While always generating block from the same node will work, assigning the job to different nodes every time will make the blockchain more trustable, as no single trustee is controlling the blockchain. The selection process is as follow (Figure 3-7):

i. Sort all nodes by IP address and port number in the address table

ii. Get the hash h from the last verified ballot in this time interval

iii. Let n be the total number of nodes, and $i = h \, mod \, n$

iv. The node on the $(i + 1)$ row of the sorted address table will be selected

A ballot may get verified just at the time of node selection, causing inconsistency in different nodes. Therefore, we suggested that there will be a little time buffer when defining 'last verified ballot'

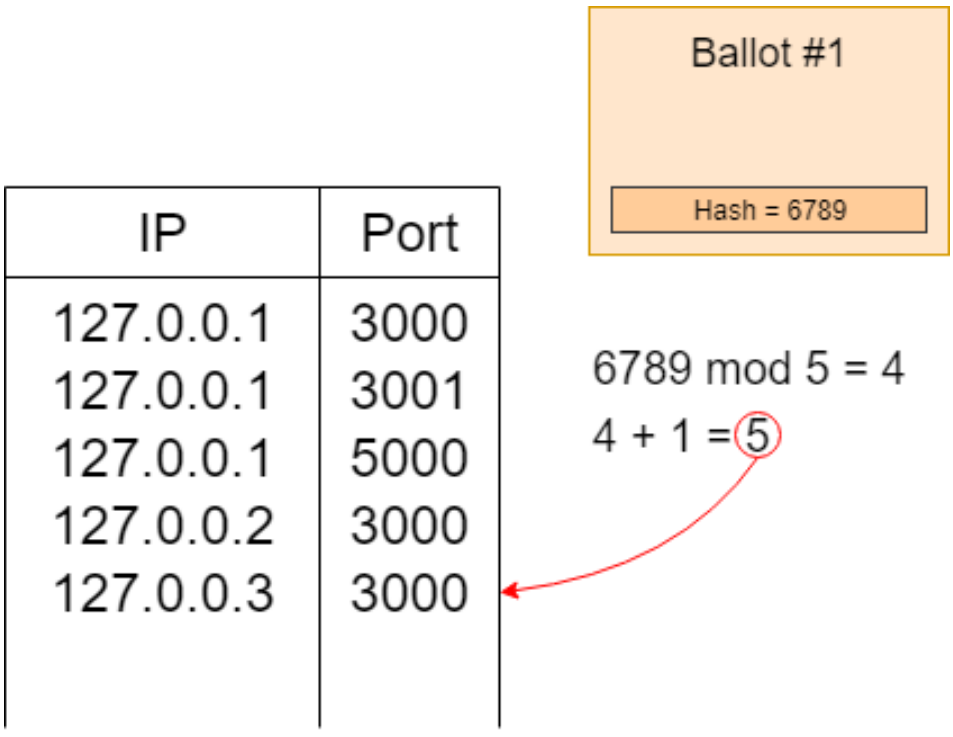


Figure 3-10: An example of selecting a node by the hash of the last verified ballot

Consensuses protocol

Usually, the selection from all nodes will be the same unless some nodes join/leave the network at the time of selection. Therefore, it is possible to results in different nodes being selected, so that a consensuses protocol is needed to select the majority of them to be the one who generates the new block.

We use the Byzantine Fault Tolerance algorithm for the consensuses, which should be work as follows (Figure 3-8):

i. Every node broadcast its result of the node selection protocol

ii. Determine the majority of them by every node itself. In case of a tie, the smallest one will be chosen

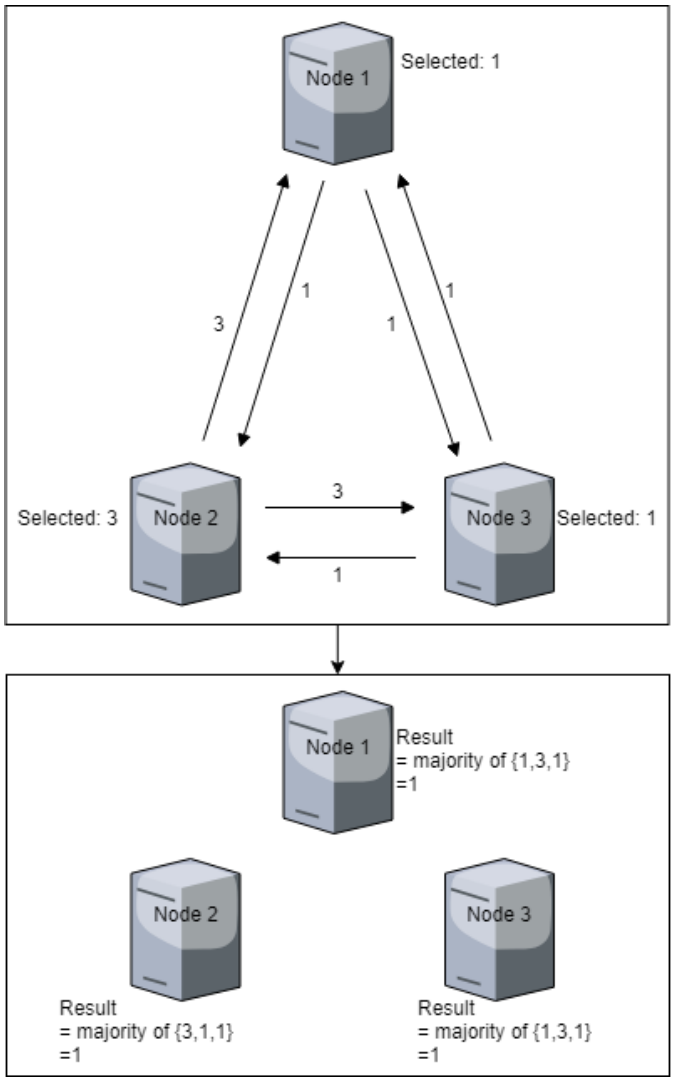


Figure 3-11: An example of the consensuses protocol

A node does not need to wait for the broadcast message from all nodes, as some of the nodes may go offline during the process and the message may get lost. The final result can be determined after receiving the same result from more than or equal to half of the total number of nodes (Figure 3-9).

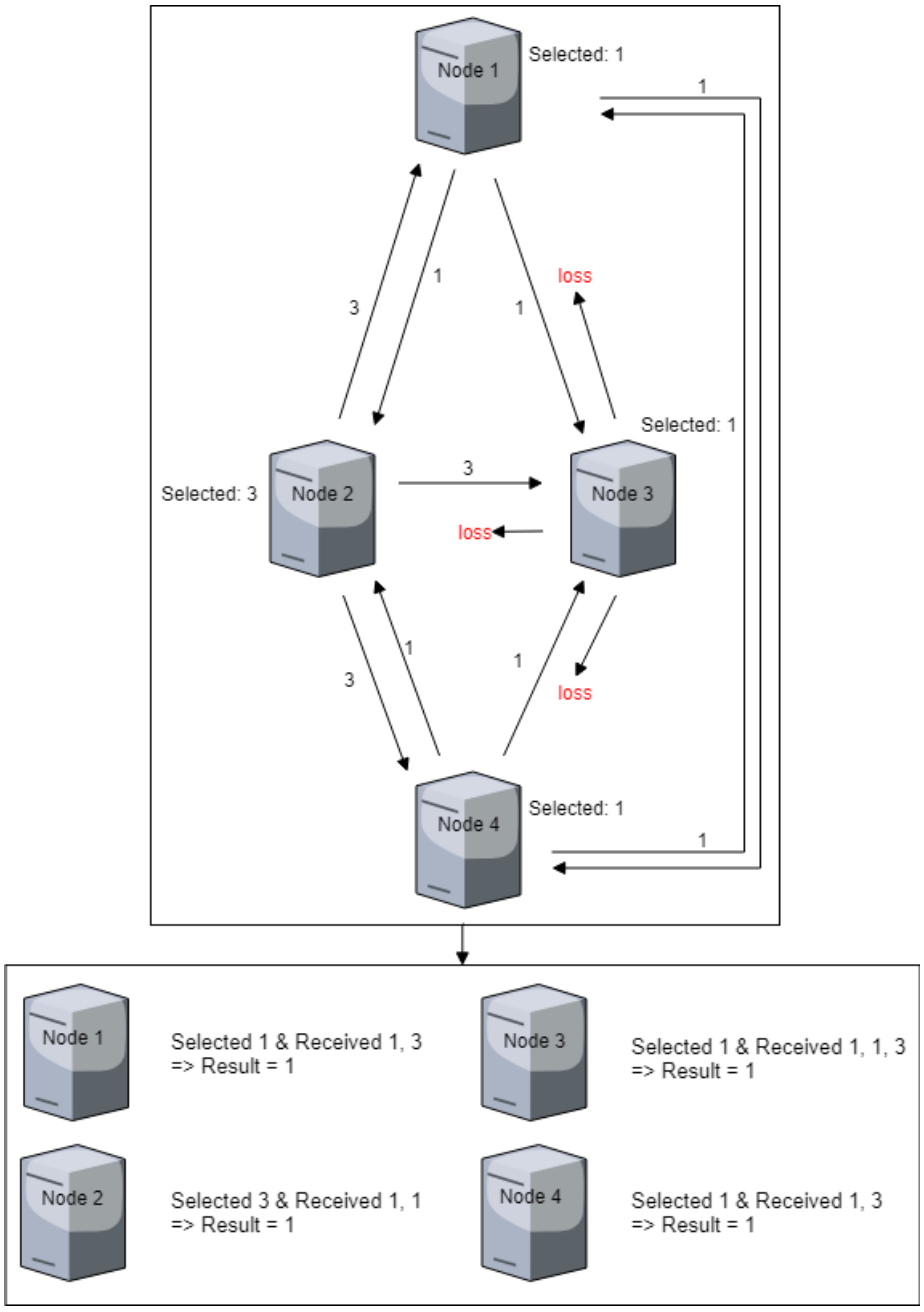


Figure 3-12: An example when some messages are lost

In case of timeout or the block generating node go offline just after the consensuses protocol, then just ignore and leave it for the next round.

Block broadcasting protocol

After a node X is selected, the following procedure will be performed (Figure 3-10):

i. X generates a new block, including all verified ballots in the last time slot

ii. X broadcast the block in the network

iii. All other nodes will verify the block and save it first

iv. If the block is valid, all nodes will sign on the block hash and broadcast the signature in the network

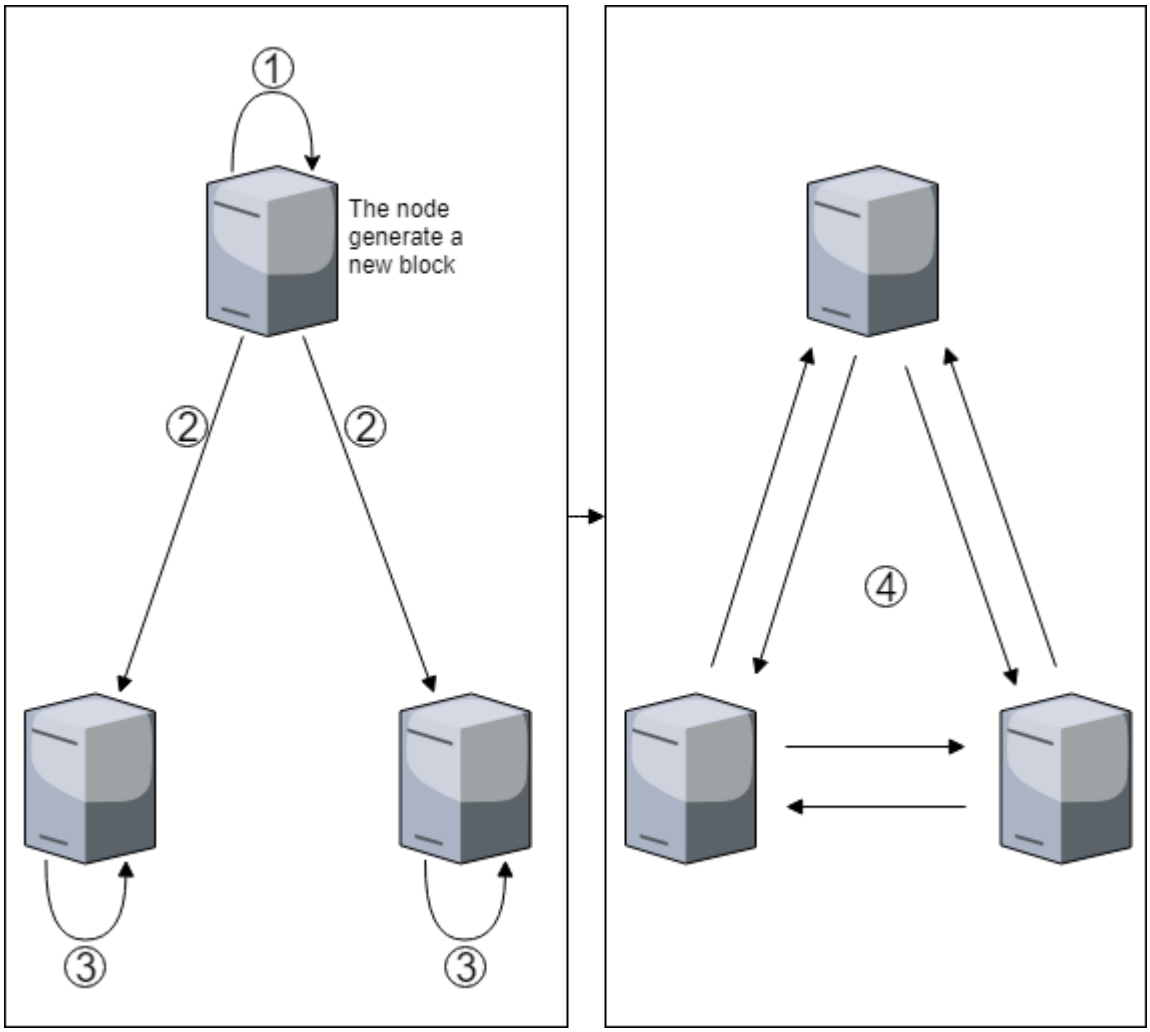


Figure 3-13: Steps of the block broadcasting protocol

A block will be put into blockchain when more than half of the trustee agree and sign

on it. That's why we require at least half of 'honest' node connect to the blockchain network, otherwise it is possible that the blockchain contains unvalidated data, although these unvalidated data may be detected and reported by the public. If a block is proven invalid, all the ballots within this block will consider putting into the blockchain in the next round.

4. Implementation

4.1. Overview of 1st term implementation

From a client-side viewpoint, i.e. a voter or election organizer, we have implemented the end-to-end voting process, which accessible using browser. So, they can create an election by providing public keys, vote in the election and compute the election via providing private keys. Most of the encryption, decryption, and keys related computation are done on the client-side. What a client can do is almost like the Helios voting system, except we have not provided a user-friendly interface this term yet.

From the server-side viewpoint, i.e. a node in the blockchain network, we have implemented the blockchain such that nodes can connect to each other, a ballot can be broadcasted via the network and block can be generated in a regular time interval. For voting-related things, we have done the related functions to support the client's operation as stated above.

4.2. Server-side

4.2.1. Use of programming language

We use Node.js [25], which is a kind of JavaScript, as the programming language. Node.js first release in 2009, and become a popular language nowadays for developing a server-side application to process HTTP request. Because of the popularity, it makes us easier for developing as there are many discussions and solutions online for problems encountered. Also, using a scripting language like JavaScript make debugging simpler. Moreover, Node.js has a huge amount of well-developed and popular modules available online, which we can reuse them to avoid writing a lot of tedious code, thus improving the reusability of our program.

For the database, we decided to use NoSQL[5] type instead of a relational database. The main reason is that NoSQL does not require a predefined structure, which can provide more flexibility. For example, a block in the blockchain may contain ballot or election details, thus not all fields are needed in each block. Also, the tables needed in our design do not require strong relation to each other, such as address table and block table. More importantly, using NoSQL can directly eliminate the risk of SQL injection security attack.

In all NoSQL type of database, we chose MongoDB [26]. It is the most popular NoSQL database nowadays [6], which first released in 2009. Furthermore, MongoDB store documents in JSON format, which is the exact format that JavaScript represent objects, so it should be more efficient when using with Node.js.

4.2.2. System architecture

Architecture design overview

Here is the overall design of a node in the blockchain network (Figure 4-1), i.e. trustee's computer. When there are multiple nodes in the network, every node will share the same design.

[5] Non-relational databases that do not use Structured Query Language

[6] Rank first in November 2018 for NoSQL database model, according to https://db-engines.com/en/ranking/document+store

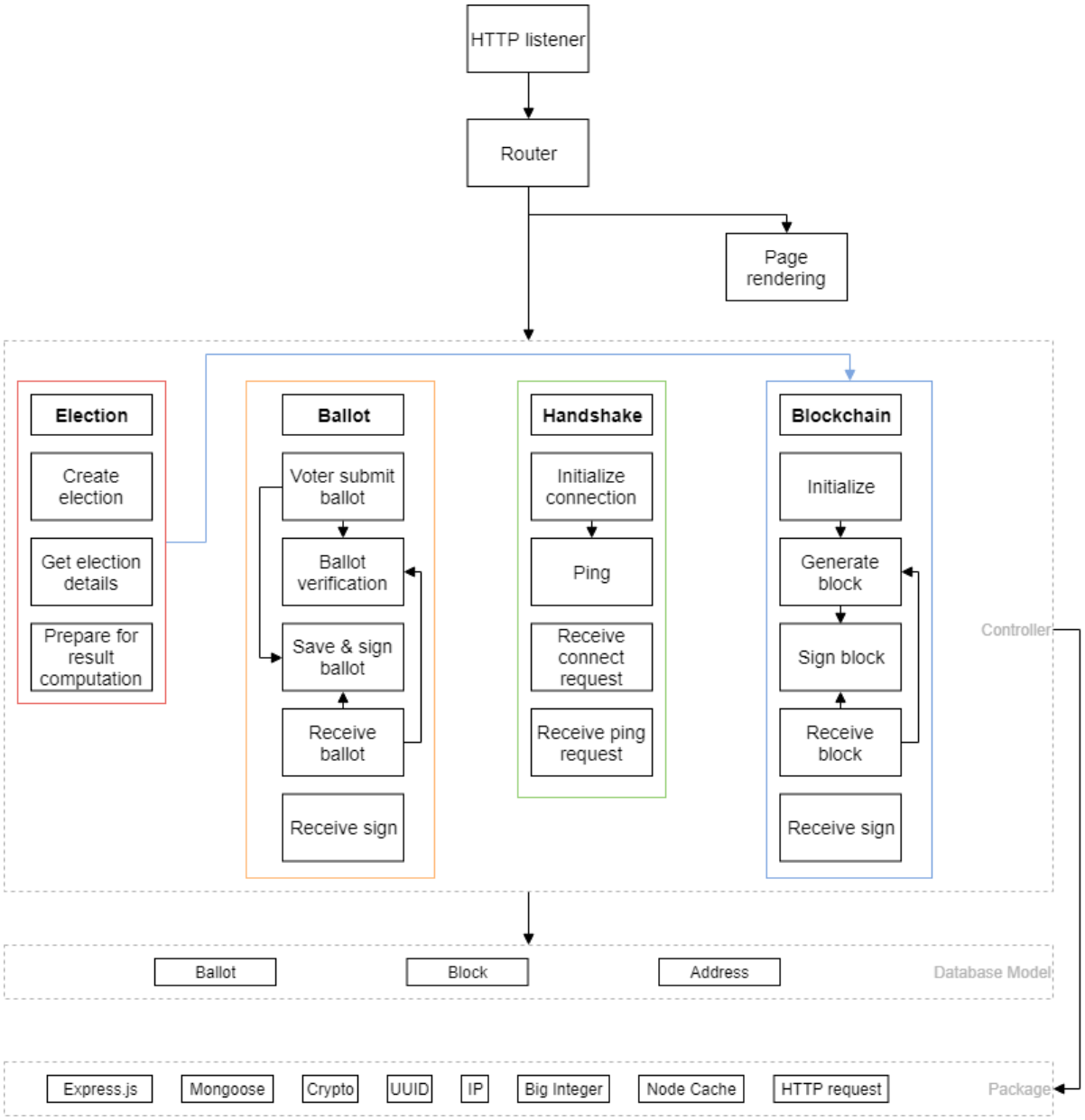


Figure 4-1: Architecture of a node in blockchain

HTTP listener

This is a predefined module that set up the server and listen to HTTP requests. This is the beginning of the system, all request received will proceed to router for processing.

Router

If the request is just a page view, then the router will pass it to page rendering engine,

which will generate the HTML document required from templates and return the request. Otherwise, the request will be sent to corresponding controllers for further processing.

Controller

Each system's component will have its own controller, basically it will cover all back-end code and logic for that component. Different controllers may interact with each other, or access to the database. This will be independent with the front-end code, so a controller cannot invoke by users directly, only via routers.

Database model

A model for a collection defines its fields and related data types. Notice that creating a new collection in MongoDB do not require fields and data types to be predefined, one can always add a document with fields and data types different in the defined database model. However, using a database model can improve the efficiency and consistency in the collections. When adding or editing data in a collection, the system will always match with the defined fields first, unless it could not find one.

Below are the details of the database models that we defined.

Address

To store the network address of all nodes in the blockchain network. (Figure 4-2)

IP: IP address of the node, in IPv4[7] format.

Port: Port number to access the node from the corresponding IP address.

updatedAt: The UTC time of the latest ping request from the node.

[7] Internet Protocol version 4, the format is x.x.x.x where x is integer from 0 to 255

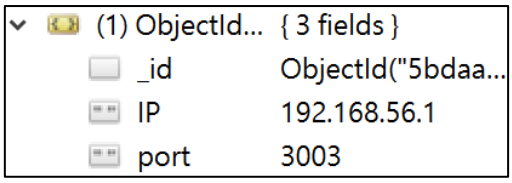


Figure 4-2: An example document in the Address collection

Ballot

To store all ballots received either from voter or from other nodes, this mainly used for storing the ballots before adding them into a block. (Figure 4-3)

electionID: The election ID that this ballot belongs to, in UUID[8] format.

voterID: ID of the voter in this election that submit the ballot.

answers: Array of <question_answer>, number of elements should equal to the number of questions in the election.
<question_answer>: Array of <option_answer>, number of elements should equal to number of options in that question.
<option_answer>: Object with the form of $\{c_1, c_2\}$, where c_1 and c_2 are the encrypted answer as stated in Section 3.2.1. They are encoded using base64[9] format.

voterSign: Signature of voter on the ballot encoded in base64 format. This means the voter use his private key to encrypt the hash of {electionID, voterID, answers} using SHA256 hash function. One can verify the hash via voter's public key.

ballotID: ID of the ballot in UUID format. This is added by the node that first receives the ballot from voter.

receiveTime: The UTC time when the first node receives the ballot from voter.

sign: Array of <trustee_sign_ballot>, number of element should equal to the number

[8] Universally Unique Identifier, the format is xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx where x can be any number or lower-case character

[9] Base64 encode binary data in text, such that every character represents 6 bits. There are 64 choices of character.

of unique signature received by the node.

<trustee_sign_ballot>: Object with the form of {trusteeID, signHash}. trusteeID is the ID of the trustee that belongs to this signature. signHash is the signature in base64 format. This means the trustee use his private key to encrypt the hash of {electionID, voterID, answers, voterSign, ballotID, receiveTime} using SHA256 hash function.

inBlock: A boolean value to indicate whether the ballot is in the blockchain or not.

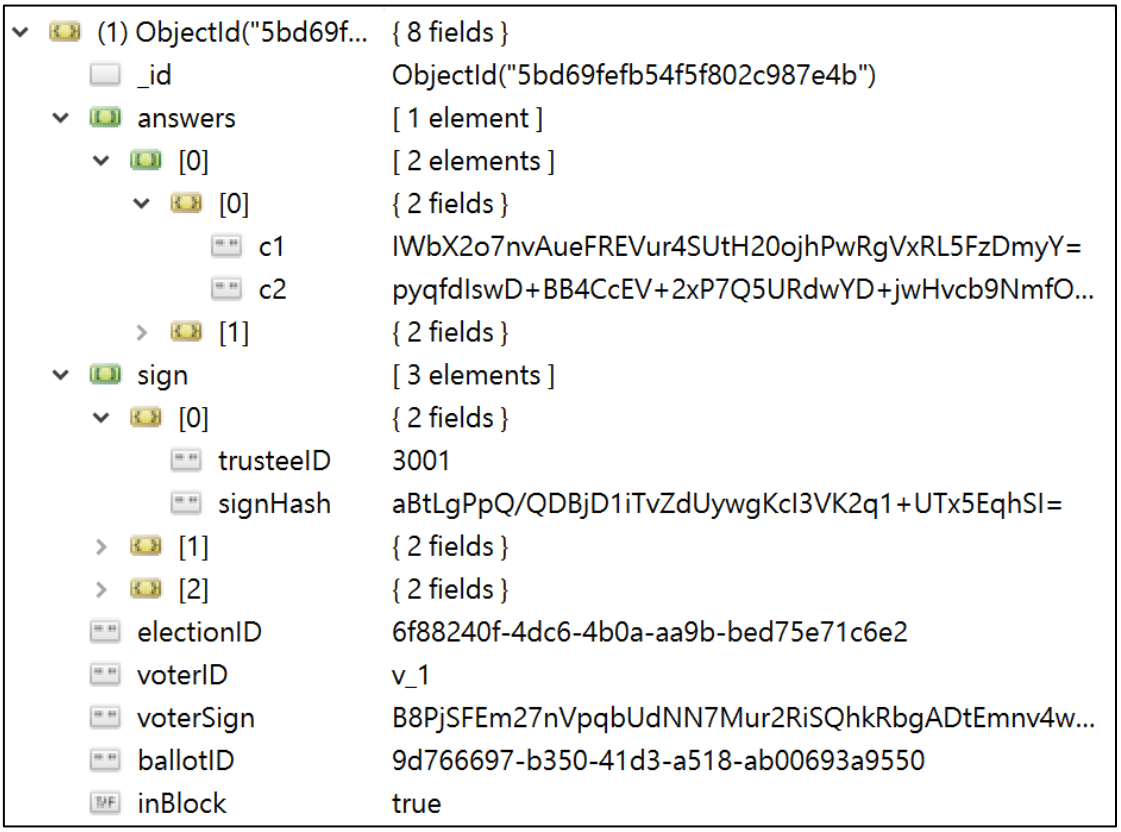


Figure 4-3: An example document in Ballot collection

Notice that the sign array may be different among different nodes, as it is possible that a signature may get lost and not received by some nodes. So, the node that generates the block for this ballot will use its own version of sign array. Also, the inBlock only reflect the local situation. If there is a fork in blockchain among the network, the inBlock value may be different for each node.

Block

To store blocks in all election that the node is responsible for. (Figure 4-4)

blockUUID: ID of the block, in UUID format. This uniquely identify a block, no matter which blockchain it belongs to.

electionID: The election ID in UUID format. This identify which blockchain does the block belongs to.

blockSeq: An integer that represent the position of the block in the chain. The first block is 0.

previousHash: The hash value of previous block, encoded in base64 format.

blockType: Represent the type of data that this block used to store. Either 'Election Details' or 'Ballot' for current stage.

data: An array, the actual structure depends on the block's type, details would be described below.

hash: SHA256 hash value of {blockUUID, electionID, blockSeq, previousHash, blockType}, encoding using base64 format.

sign: Array of <trustee_sign_block>, number of element should equal to the number of unique signature received by the node.
<trustee_sign_block>: Object with the form of {trusteeID, signHash}. trusteeID is the ID of the trustee that belongs to this signature. signHash is the signature in base64 format. This means the trustee use his private key to encrypt the hash of the block using SHA256 hash function.

Similar to ballot, the sign array can be different in different nodes.

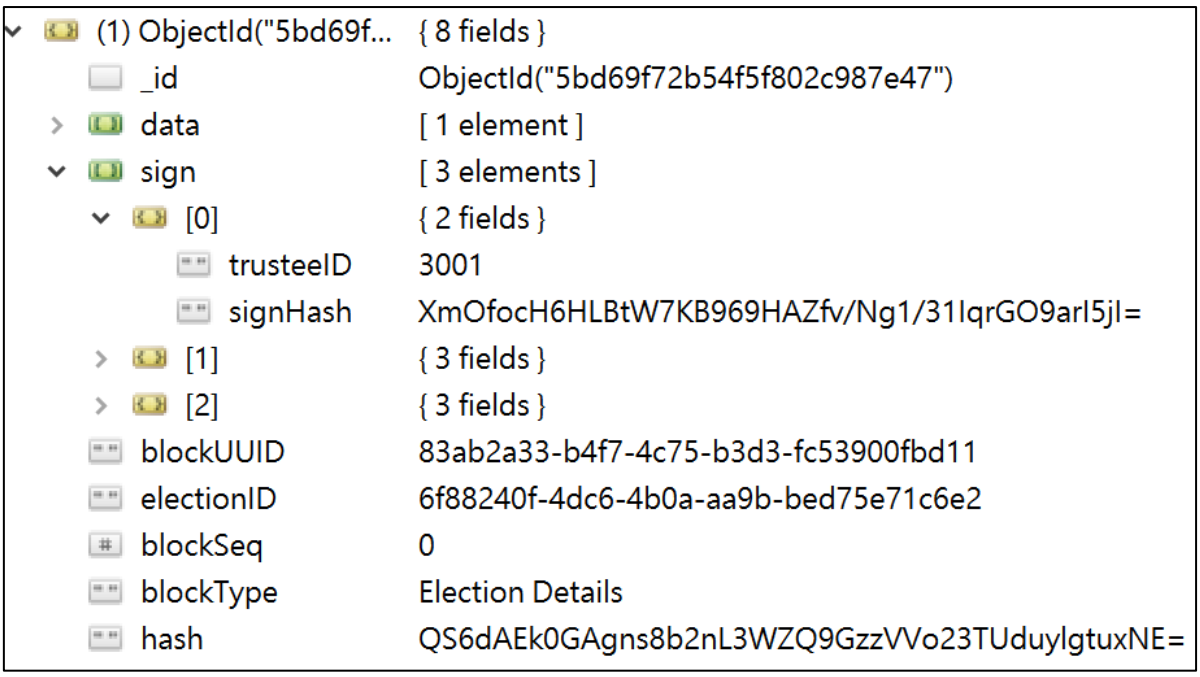


Figure 4-4: An example document in Block collection

Data may have the following fields if it is a 'Election Details' block (Figure 4-5):

name: Name of the election

description: Election description, such as purpose of the election

questions: Array of objects in form of {question, <answers>}
question: Text of the question
<answers>: Array of choice in text for the question

key: An object in form of {p, g, y}, which represent the election public key as stated in Section 3.2.1. They are encoded in base64 format

voters: Array of objects in form of {id, public_key}
id: voter ID
public_key: RSA public key of the voter, in PKCS#8[10] format

frozenAt: The UTC time when all election details are locked

[10] Public-Key Cryptography Standards #8

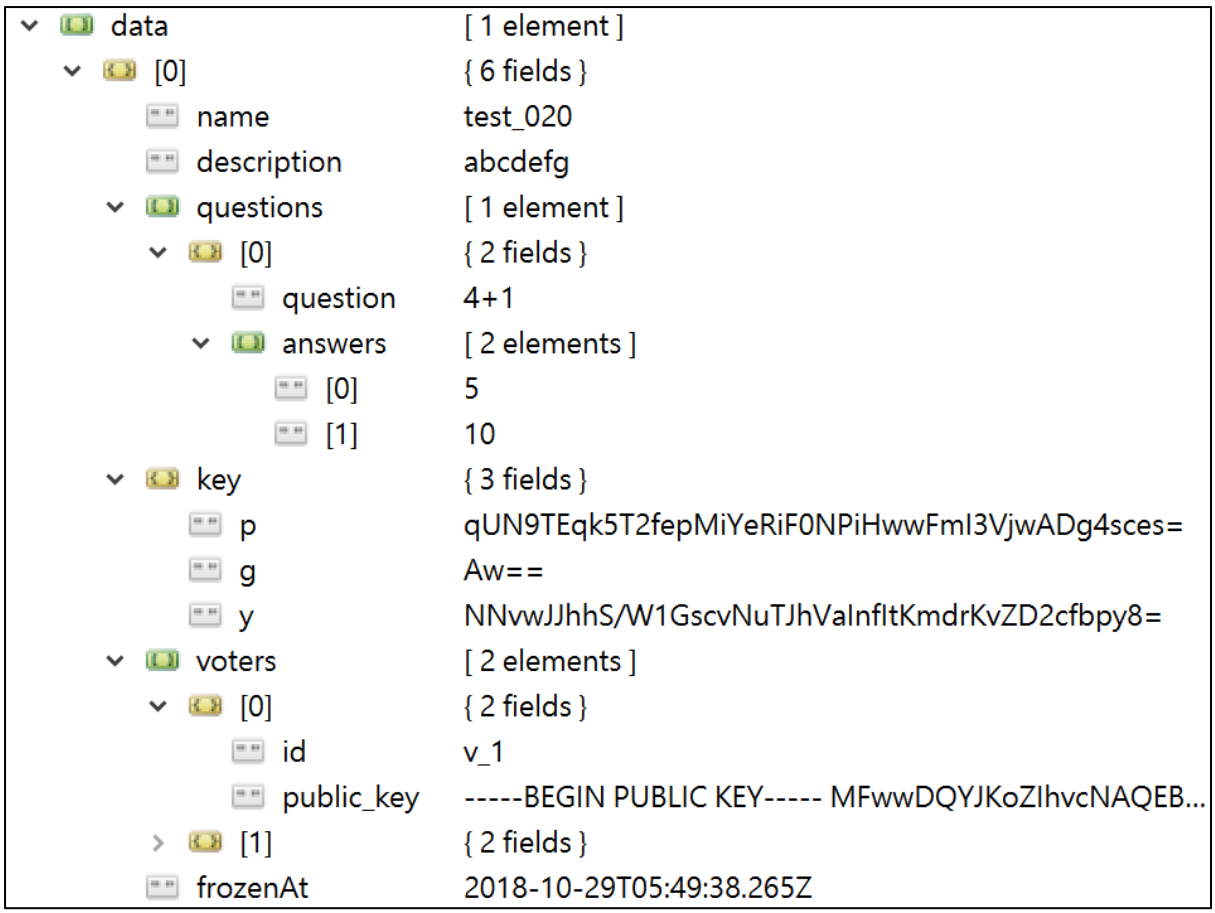


Figure 4-5: An example 'Election Details' object in data field of Block collection

For a 'Ballot' block, data will be array of ballot object, which is the same as descripted in Ballot section, except it will not include the inBlock field.

Package

A package is an opensource Node.js module that allows us to add specific functionality to our program to reduce our amount of work. We always download package carefully by finding out its popularity, stability, and related online discussions, in order to avoid our system being affected by possible vulnerabilities in the package.

Following are the details of each package.

Express.js

This provides a simple but powerful web application framework that helps us to

handle HTTP request, include defining HTTP listener, router and middleware.

Mongoose

This makes the API of calling MongoDB easily available to use on Node.js.

Crypto

It provides all functions required for using popular cryptography algorithms, such as RSA and SHA25. We used them mostly for hashing, signing and verifying hash. However, as the cryptography algorithm of Helios is homomorphic El Gamal, there is no package available for it, so we implement the ballot encryption and decryption ourselves.

UUID

This will generate a Universally Unique Identifier (UUID), which is proven that the probability of having two same UUID is very close to zero [27]. We use it for giving ID for election, ballot and block, such that we can identify them easily in the database.

IP

We use it for getting the IP address of the running node itself, so that it can send the address to others, for setting up connection, broadcasting, and selecting nodes in block generation.

Big integer

Integer in JavaScript is limited to 53 bits. However, when we need to implement the ballot encryption and decryption, we need to handle number more than 256 bits, depends on the size of keys used. Therefore, we will need this package to do calculations for big integer. It can do arithmetic operations on integer of any size, as long as the computer has enough memory.

Node cache

This one does similar things like the cache in browser. When we store objects with this package, we can define the lifetime of that object, such that it will be deleted automatically after the lifetime. This is useful when receiving a signature of ballot or block. Due to network latency, sometime the sign may arrive earlier than the ballot or block, hence we need to temporary save it first. Nevertheless, a signature may be a fake one, but we cannot prove this until the ballot or block actually arrive. Therefore, we save the signature into cache first, if the actual ballot or block don't arrive in a time limit, then it has a big chance that there is an error in the signature.

HTTP request

All communications between the nodes in the blockchain network are done by sending HTTP request, so with this handy package, we can just send the request by giving URL and data.

4.2.3. Detailed explanation of components

In this section, we use sequence diagram to describe how each component interact with each other when connecting a node to the blockchain network (Figure 4-6), creating an election (Figure 4-7), voter submits a ballot (Figure 4-8) and tallying an election (Figure 4-9).

Notice that the process of broadcasting a 'Ballot' block is similar to the diagram for creating an election, except that there is no election organizer involved and no need to generate election public key. So, there is no separate sequence diagram for this action.

Connecting to the network

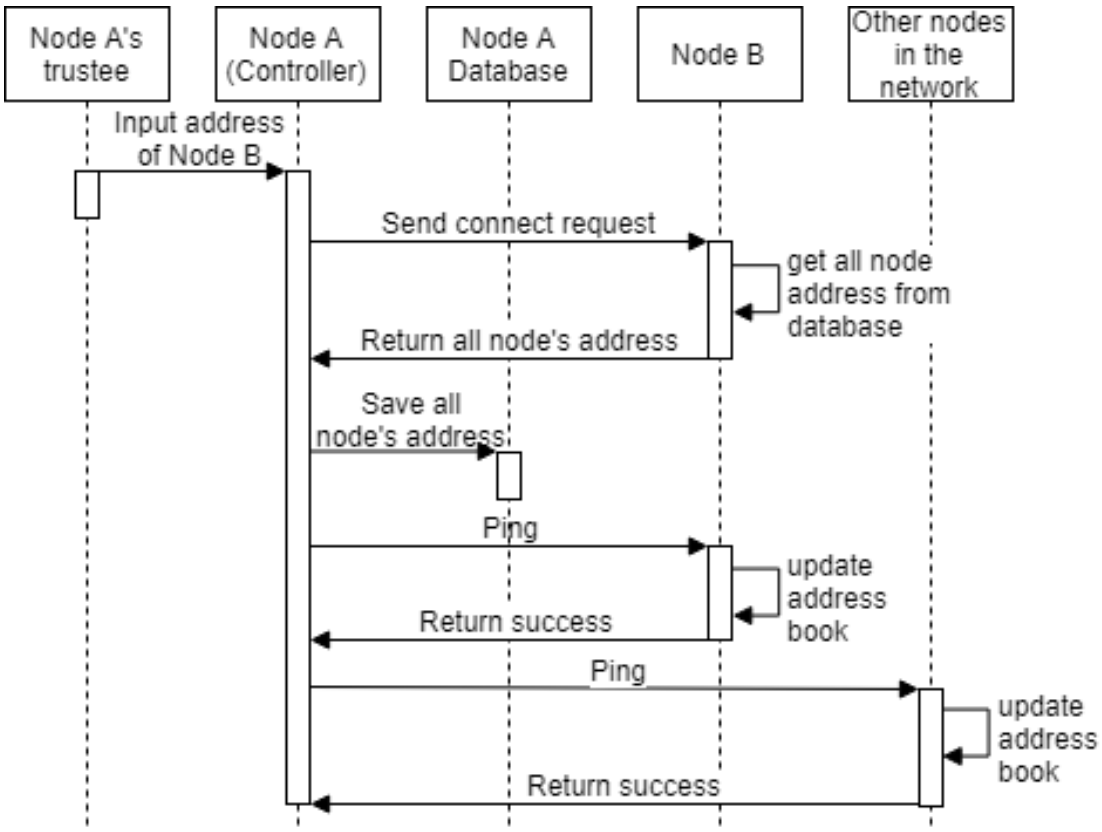


Figure 4-6: Sequence diagram of connecting a node to the blockchain network

Creating election

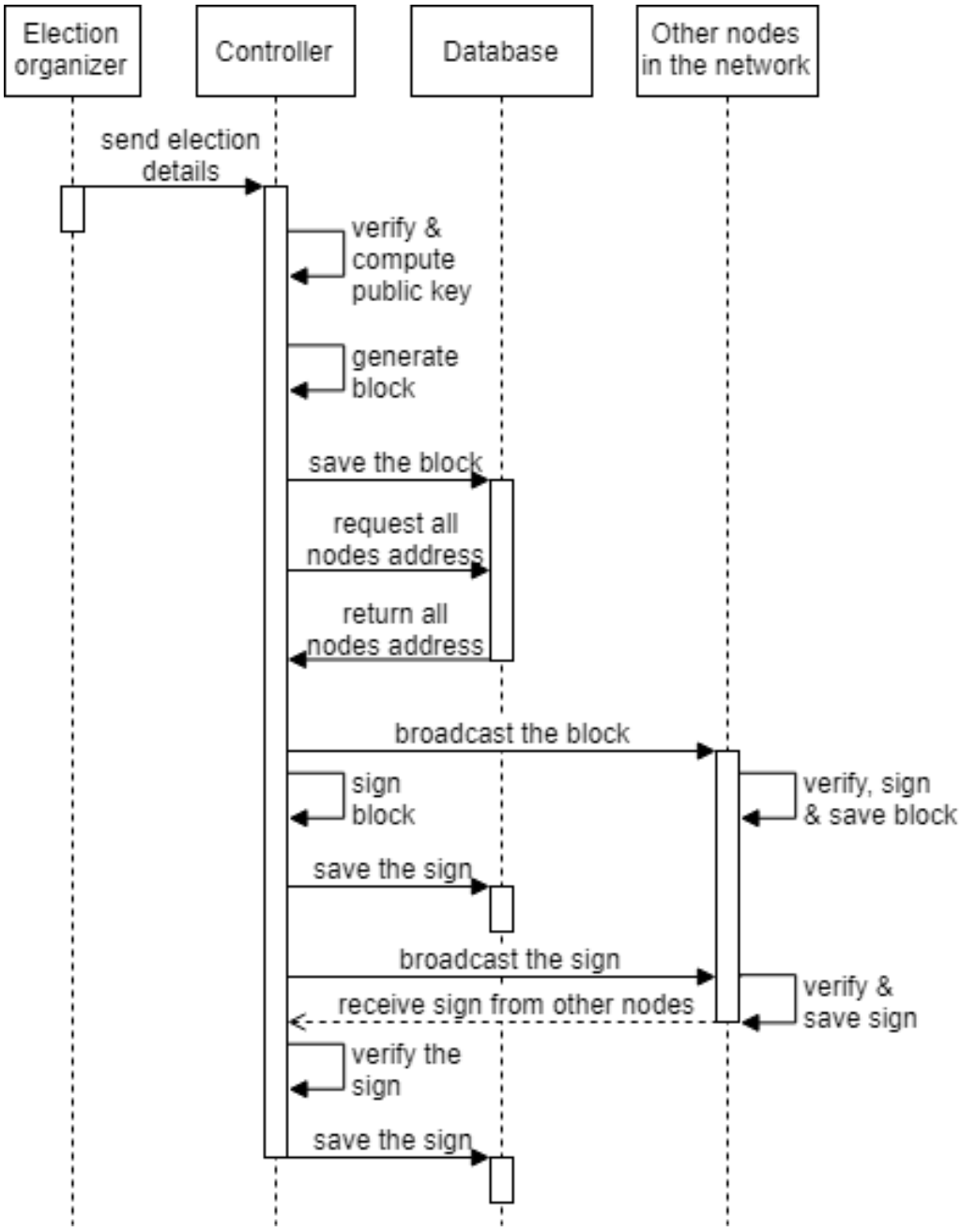


Figure 4-7: Sequence diagram of creating an election

Processing a submitted ballot

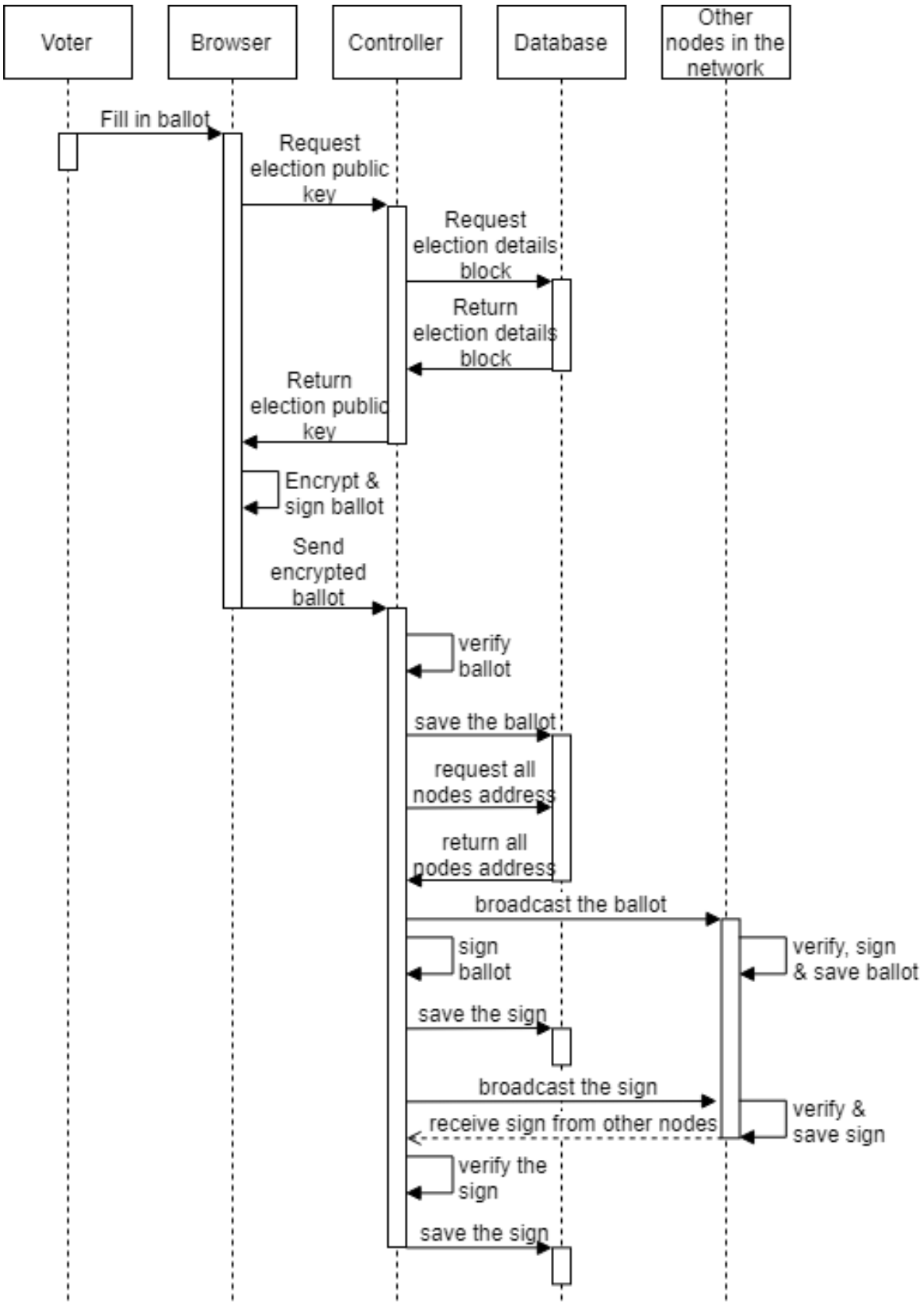


Figure 4-8: Sequence diagram of processing a ballot submitted by a voter

Tallying election

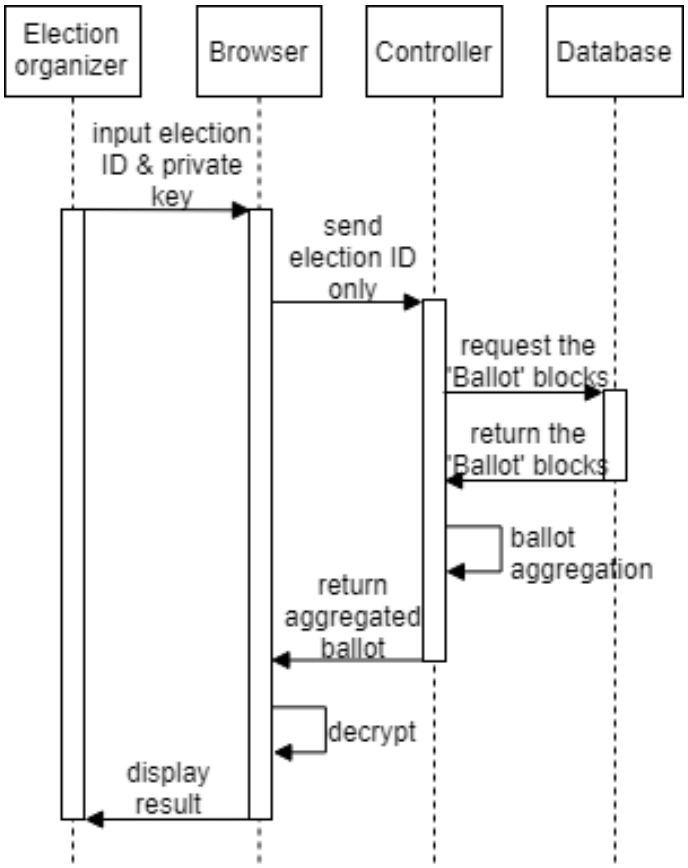


Figure 4-9: Sequence diagram of tallying an election

4.3. Client-side

4.3.1. Use of programming language

As for this term, user interface design is not one of the objectives, the interfaces here are just for the sake of testing server-side functions. So, we use a web interface to achieve this.

The languages used here are HTML, CSS and JavaScript, which are just a standard set of languages for the client-side of a website. For JavaScript, we have included the jQuery JavaScript library, which is the most popular one. It provides handy functions such as sending AJAX[11] request and accessing elements on a webpage. Besides, Big Integer library

[11] Asynchronous JavaScript and XML. A way of sending HTTP request to reduce data size.

also used for the same reason as stated in Section 4.2.2.

4.3.2. User interface

Create election

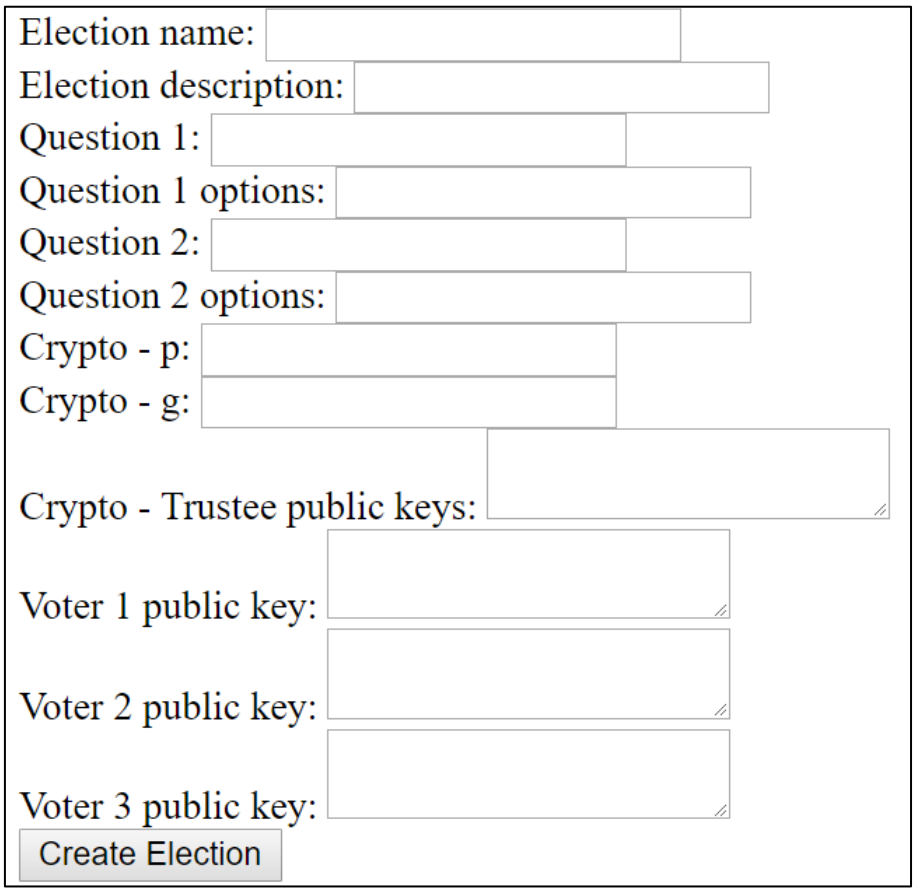


Figure 4-10: The form for creating an election

This is the interface of the 'Create Election' page (Figure 4-6). Users need to fill in the form and click the button, then all data will be sent to the server. For inputting multiple options or multiple trustee public keys, each of them should be separated by a semicolon. It is not required to fill in more than 1 question or voter public key, if an input field leaves blank, then it will just be ignored.

Submit ballot

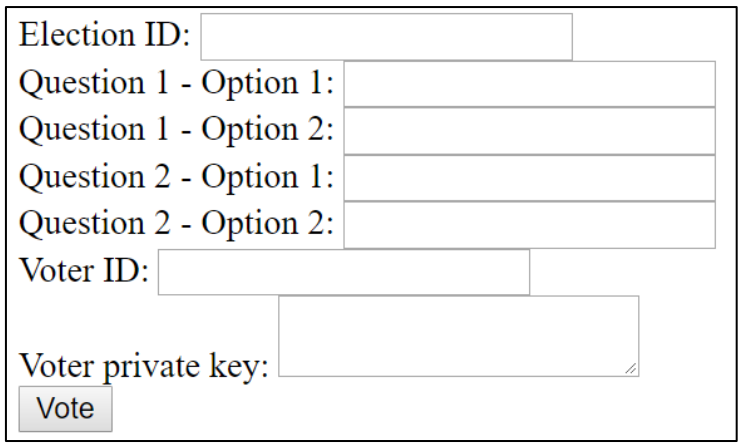


Figure 4-11: The form for prepare and submit ballot

This is the interface where a voter can perform voting (Figure 4-7). Voters need to provide the election ID, voter ID, his private key and his choices to vote. After clicking the 'Vote' button, the ballot will be prepared and encrypted in the browser before sending to the server. For input fields of those options, voters need to input 1 if he votes for that option, 0 if otherwise, just as described in Section 3.2.1.

Compute result

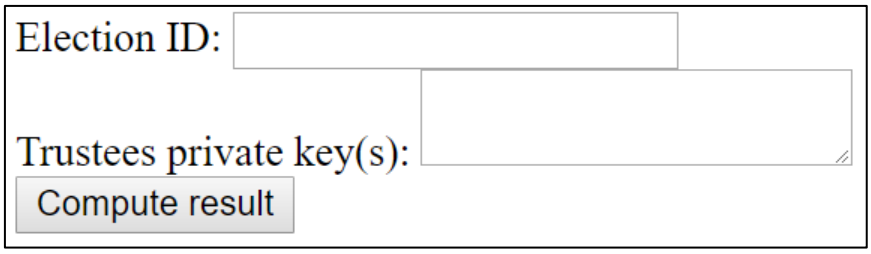


Figure 4-12: The form to submit details for computing election result

The interface for tallying an election is very simple (Figure 4-8), only the election ID and the private key of all trustees are needed. If there are multiple trustees, separate each key by a semicolon. After clicking the button, only the election ID will be sent to the server to get an aggregation of all encrypted ballots, as described in Section 3.2.1. Then, the client-side code will apply those private keys on the aggregations to decrypt the election. The result will be displayed below the form.

4.4. Demonstration

4.4.1. Handshaking in the blockchain network

We will start 3 nodes one by one in the same computer with a different port number, i.e. 3001, 3002 and 3003. They will connect to different databases, so as to simulate 3 different computers running the same program. When starting the second and the third node, we will input the IP address with the port number of the previous node to start a connection. (Figure 4-9)

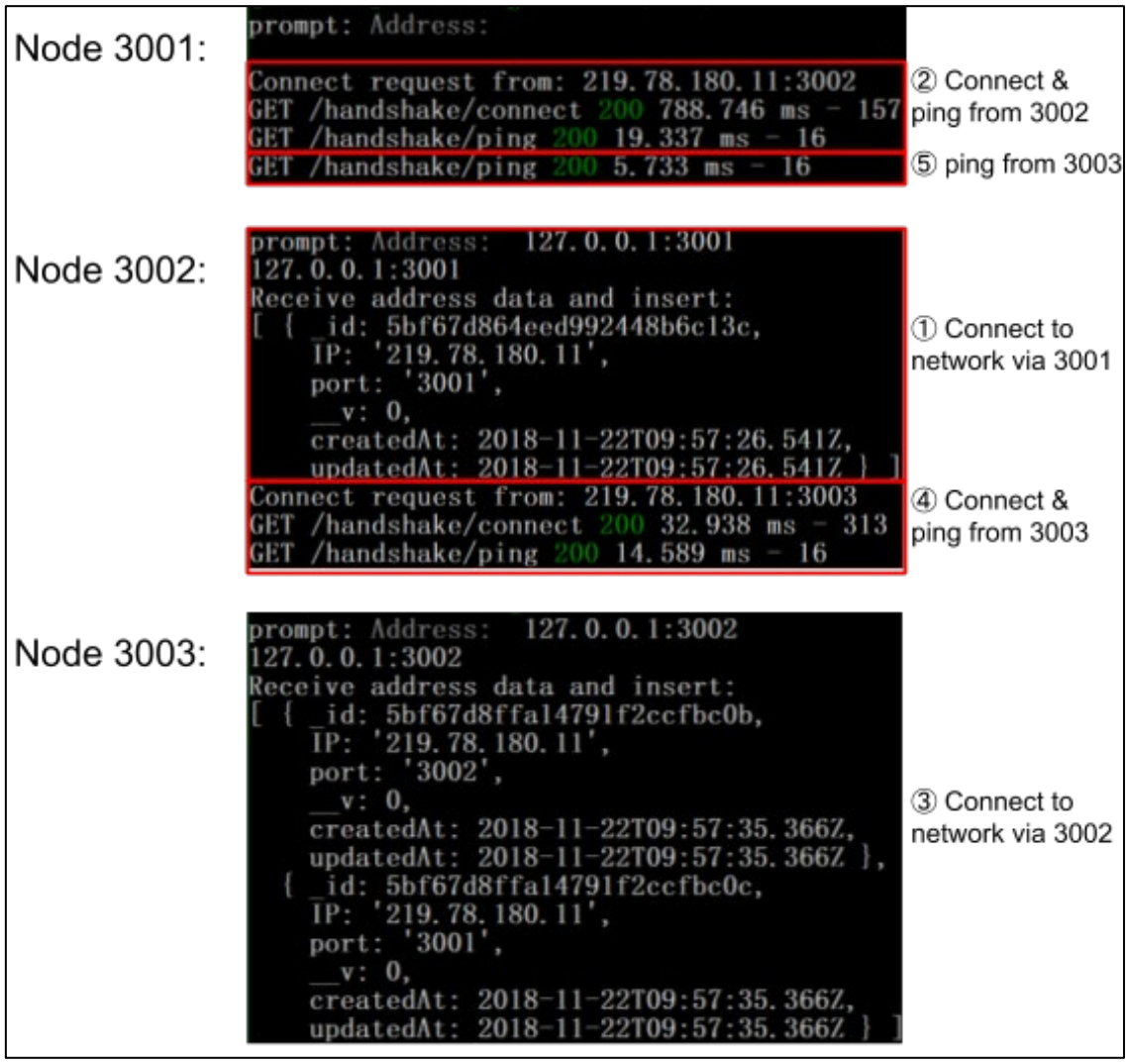


Figure 4-13: Connect 3 nodes to each others

We can see that a node will return its own address book when receiving other's connect request for them to perform ping.

If we terminate one of the nodes, then other nodes will realize its termination during the regular ping. (Figure 4-10)

```
{ Error: connect ECONNREFUSED 219.78.180.11:3003
    at Object._errnoException (util.js:992:11)
    at _exceptionWithHostPort (util.js:1014:20)
    at TCPConnectWrap.afterConnect [as oncomplete] (net.js:1186:14)
  code: 'ECONNREFUSED',
  errno: 'ECONNREFUSED',
  syscall: 'connect',
  address: '219.78.180.11',
  port: 3003 }
Ping fail & deleted: [object Object]
```

Figure 4-14: Output from Node 3001 & 3002 after Node 3003 is terminated

4.4.2. Create an election and generate new block

We create an election via the browser (Figure 4-11). Notice that all the keys are generated somewhere else, as the server is not supposed to know the private keys.

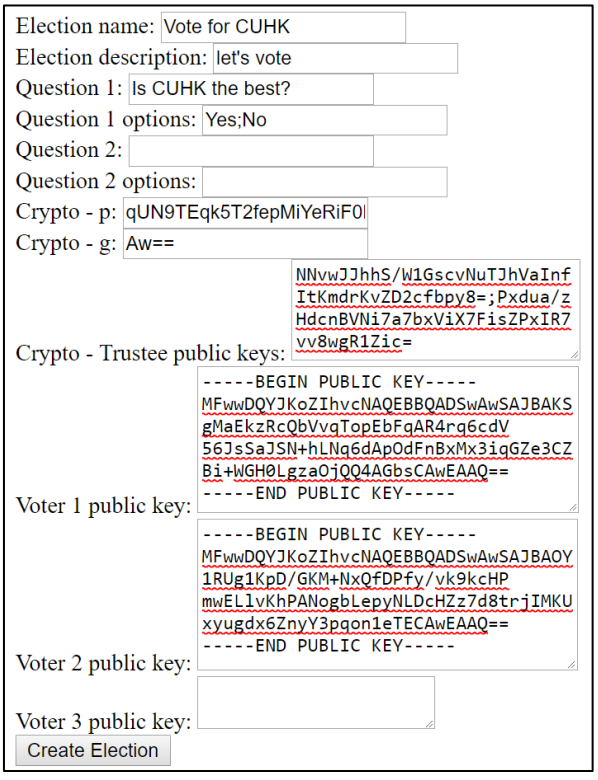


Figure 4-15: An example of filling in the election creation form

The election details are processed in the browser and submitted to one of the nodes,

Node 3001 in this case (Figure 4-12). Unlike a block for storing ballots, currently the node that receives election details will also generate the block. So, we can see that Node 3001 generate and broadcast the block, as well as all nodes broadcast the signature on the block. Observe that Node 3002 receive the signature from 3001 before the block arrives, so as stated in Section 4.2.2, the signature will be put into a cache first and save into the block later.

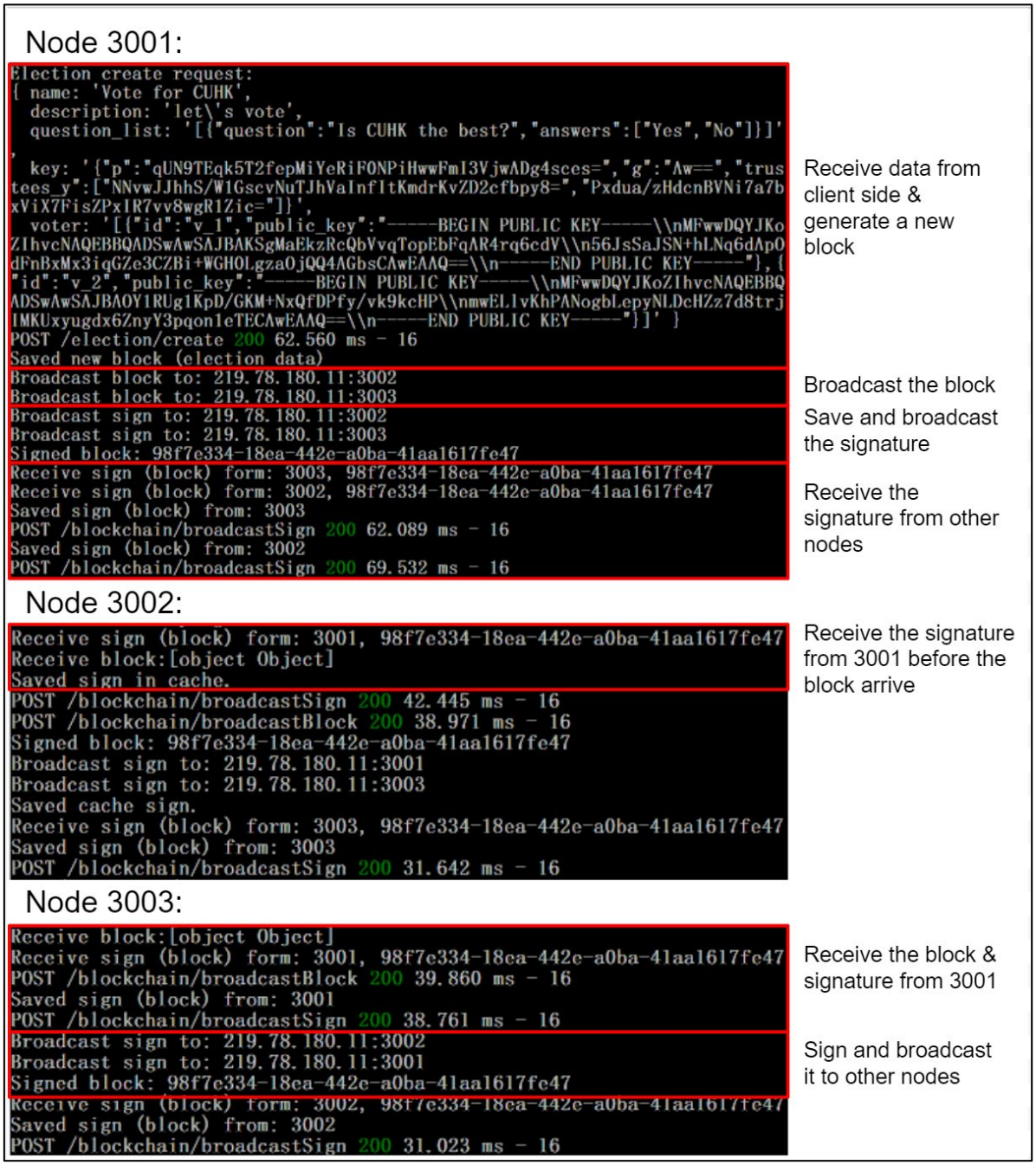


Figure 4-16: Output from the nodes when election details are submitted to Node 3001

4.4.3. Submit a ballot and generate new block

We filled in a ballot which choose the option 1 for the election (Figure 4-13), and submitted to one of the nodes, Node 3001 in this case (Figure 4-14). We can see that every node will verify the ballot before signing on it.

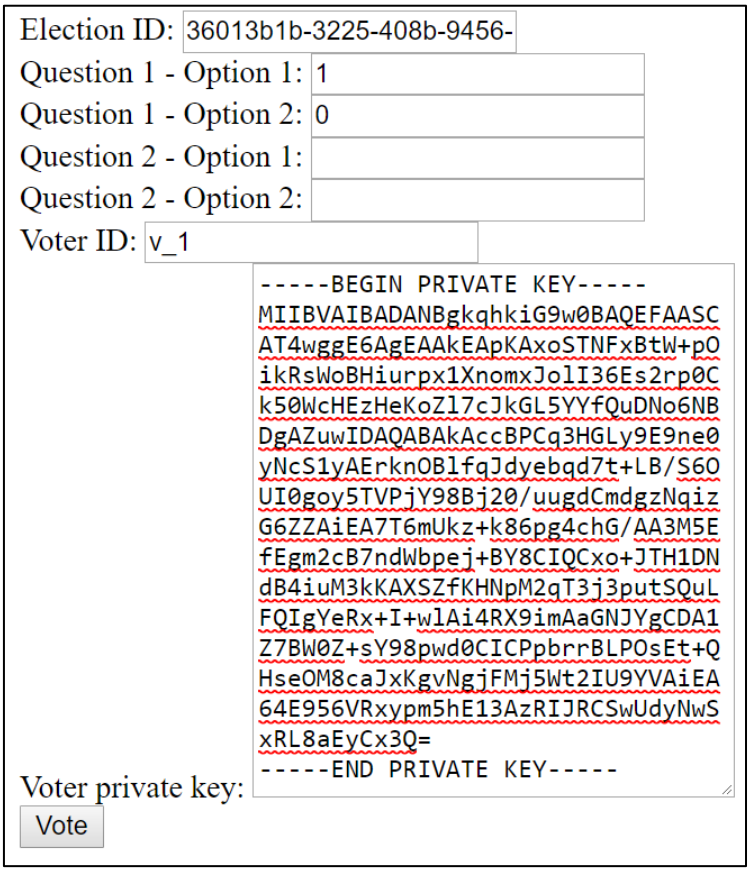


Figure 4-17: An example of filling in the ballot

For the block generation, Node 3002 has been selected to be the new block generator (Figure 4-15). We can see that the block and signatures are being broadcasted to other nodes just like the design as stated in Section 3.3.7.

We have also tried to submit a wrong private key, so that the node fails to verify the ballot and stop broadcasting (Figure 4-16).

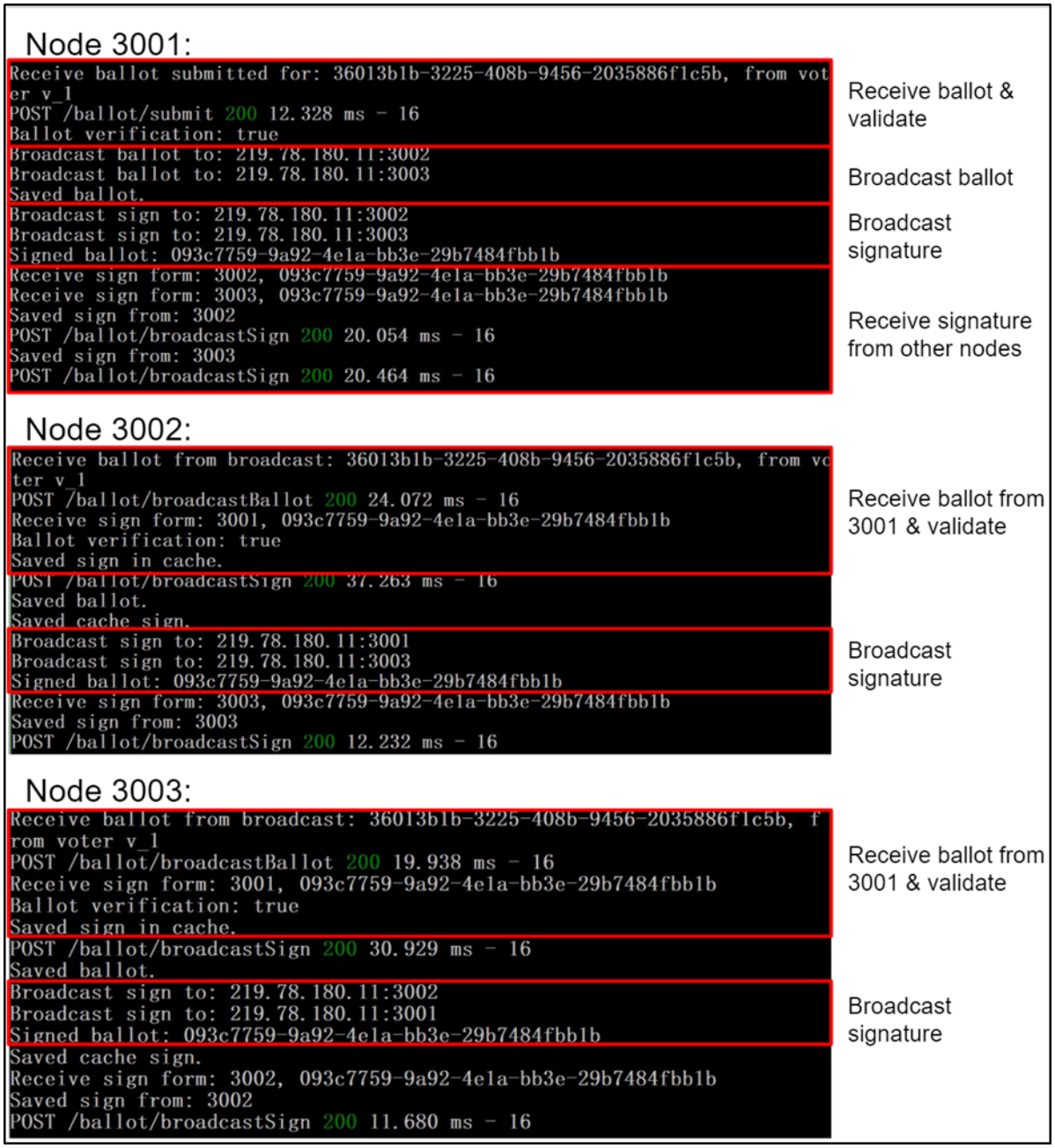


Figure 4-18: Output from the nodes when a ballot is submitted to Node 3001

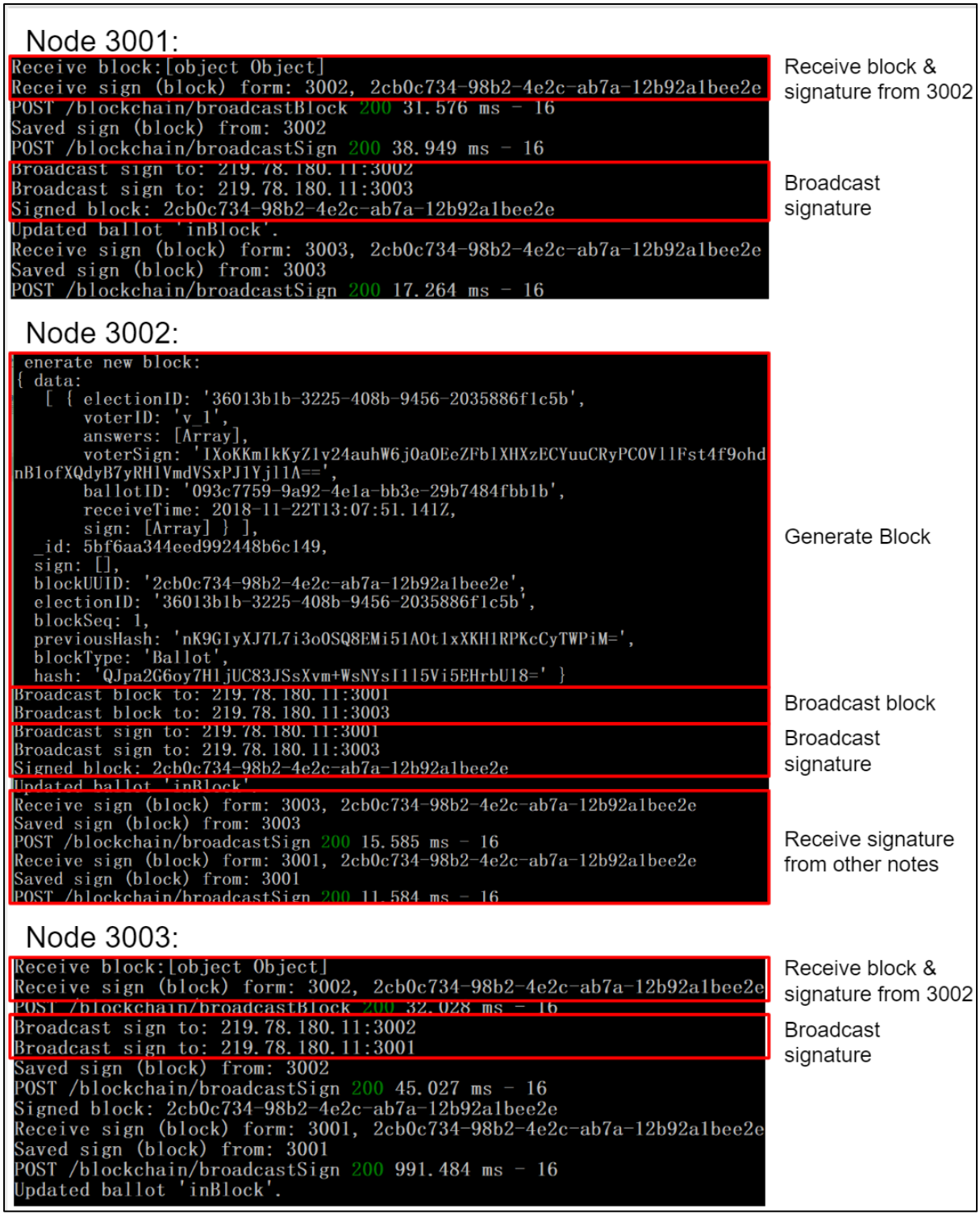


Figure 4-19: Output from the nodes when Node 3002 generate a new block

Figure 4-20: Output from a node when receiving invalid ballot

4.4.4. Compute the result from the election

By providing correct private keys for the election, the browser will decrypt it for us and display the result below (Figure 4-17). As in Section 4.4.3, we vote for the first option, so we have one vote for 'Yes' in the election. Notice that the gm in Figure 4-17 represent g^m in Section 3.2.1, a brute force approach is used for converting it into actual result.

If we provide wrong or insufficient trustee's private key, it can show that the election can't be decrypted (Figure 4-18).

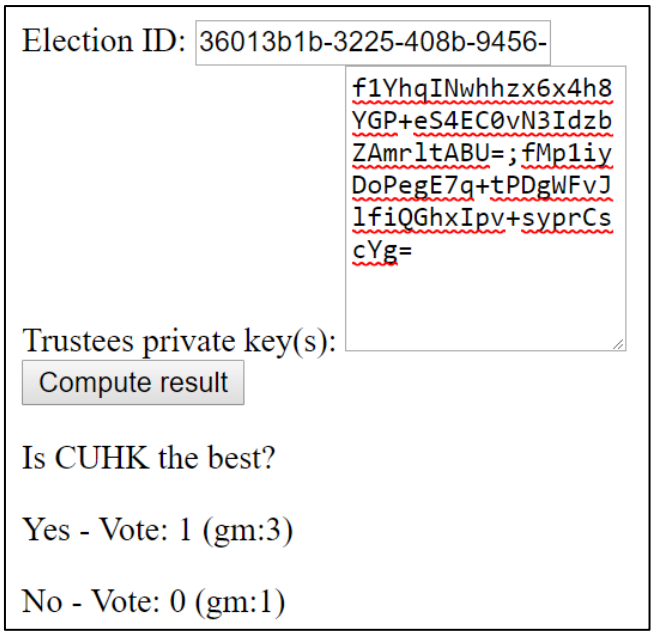


Figure 4-21: Tallying an election

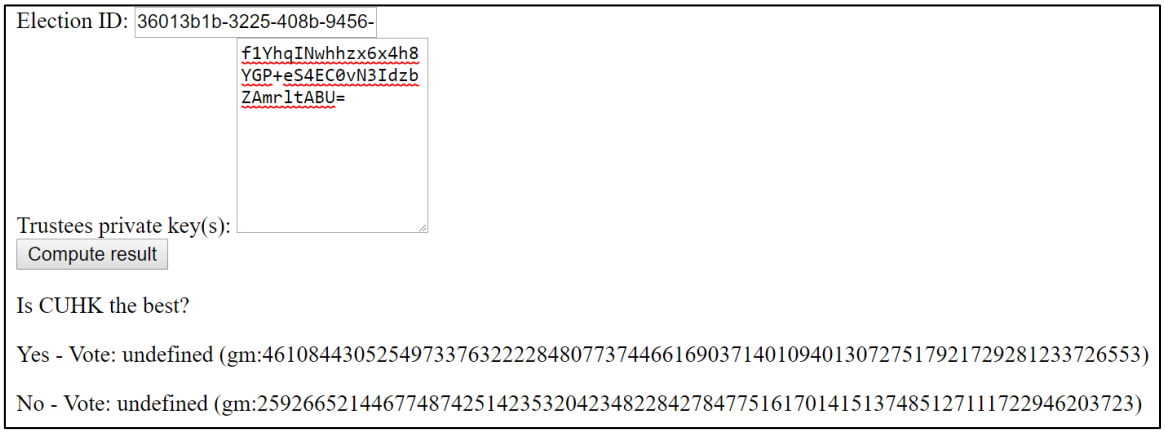


Figure 4-22: Failure when tallying

5. Conclusion

5.1. Summary of 1st term

Voting system itself is not a hot topic in Hong Kong, not many people really concern about the current voting system on its anonymity and verifiability, even after the incident of losing the computer that contains all voter information [28]. Actually, we also learned about all these voting considerations only after we started working on this project. It is interesting to know that e-voting can be much more reliable than we thought. That's why our goal is to educate others about these things. Hopefully, it will be successful after finishing the whole project, with the help of popular blockchain.

To conclude, the objectives for this term is achieved. Firstly, we have researched on end-to-end verifiable voting and blockchain voting technology that proposed by others. While not all of them have been studied in-depth, we found different types of implementation for compare and contrast. Then we discovered Helios, one of the most popular online end-to-end verifiable voting, to use for reference. For the blockchain part, we use it as secure storage.

Secondly, although we used the cryptography algorithm in Helios, we also studied the limitation of its system and suggested modifications to the system design. While for the blockchain part, we have designed our protocol with an end-to-end data flow, i.e. from handshaking of nodes in blockchain network to the consensuses of blocks.

Lastly, we have implemented the backbone of the system like blockchain protocol, and also the end-to-end voting process for voters and election organizer. These match with the target we set earlier. Time for implementation in 1st term is limited, as we spend some time in designing and setting up system environment, but we believe much more implementation would be done on the next term.

5.2. Planned work for the 2nd term

5.2.1. Zero-knowledge proof

A zero-knowledge proof is used for proving someone knowledge on a value, but during the process, the prover should not learn any other information including the value.

In Helios, apart from the cryptography algorithm, it uses zero-knowledge proof to prove the honesty of voters and trustees [21]. So, we will study its methods in-depth and integrate with our application. There are 3 places require the zero-knowledge proof.

Trustee knowledge on private key

While others must not know the private key of any trustee, they should know that trustee actually has the private key that matches with the public key he provides. There are two reasons. First, if the trustee does not know the private key then no one can decrypt the entire election. Second, it is possible that the trustee can manipulate a fraud public key such that he can decrypt all ballots without other trustees' private key. Therefore, this proof is required for achieving voter privacy.

Trustee honest decryption

When decrypting an aggregation of ballots in Helios, the aggregation will be sent to a trustee, the trustee will partially decrypt it using his private key. Then, he will send the partial decryption to another trustee, and so on until all trustees have contributed. No trustees' private key will be disclosed in the process. However, a dishonest trustee can do more operation on the partial decryption than just apply his private key. In consequence, the decrypted result of the election may not reflect the actual result due to this dishonest action. Hence, every trustee should be required to prove their honest contribution to attain integrity.

Voter honest encryption

As stated in Section 3.2.1, voter choice on an option is encoded with 0 or 1, then perform the encryption before submitting the ballot. However, it is possible that voter encodes something else like a negative number. But no one can validate this because it has been encrypted and no individual ballot should be decrypted for checking. Thus, all voter

should submit a proof with their ballot, in order for others to validate the voter honesty.

5.2.2. Full verification on the blockchain

While our blockchain protocol has been designed in Section 3.3, not all verifications have been implemented. For example, ballots re-verification when receiving a newly generated block, trustee's signature verification, connection request validation, etc. All of these are required to make the whole blockchain trustworthy.

5.2.3. User interface

In the 1st term, we focus more on the implementation of server-side, the user interface is only for ease of testing. During the next term, we will design on the user interface such that it is attractive and user-friendly, since we want the public to use it for an electronic election of any size.

Before the design of the interface, we need to decide the deliverable first, whether it should be a web application or a mobile application.

Generally, a mobile application is better for achieving security [29] and privacy [30]. An obvious reason is that people seldom share a personal mobile phone with others, so that any action performed on the mobile application is less likely to be known by others. Besides, a mobile application may need to rely on the security of the operation system, but not the browser. So, SSL certificate can be embedded in the application instead of trusting the browser.

On the other hand, a web application is more portable, either a computer or a mobile phone can access the website as long as it has a network connection. Thus, the application may be able to reach more people, especially when people don't want to download too many mobile applications into their smartphone. Moreover, developing a web application is simpler when compared to a mobile application, so we can spend more time building more functions.

5.2.4. Apply the proposed modifications

As discussed in Section 3.2.2, we have found some limitation on Helios. Therefore, we will actually implement them, so as to prove that these modifications can really bring positive changes to the original Helios system.

Bibliography

[1] P. Tang, “Electronic voting,” Legislative Council Secretariat, Hong Kong, 2017.

[2] Digital Transaction Limited, “AIDB: Authoritative Information Distribution Blockchain,” Hong Kong, 2018.

[3] M. Hooper, "Top five blockchain benefits transforming your industry," IBM, 22 2 2018. [Online]. Available: https://www.ibm.com/blogs/blockchain/2018/02/top-five-blockchain-benefits-transforming-your-industry/. [Accessed 21 11 2018].

[4] J. Benaloh, R. Rivest, P. Y. A. Ryan, P. Stark, V. Teague and P. Vora, "End-to-end verifiability," arXiv preprint arXiv:1504.03778, 2014.

[5] T. Jenks, "Pros and Cons of Different Blockchain Consensus Protocols," Very, 8 3 2018. [Online]. Available: https://www.verypossible.com/blog/pros-and-cons-of-different-blockchain-consensus-protocols. [Accessed 21 11 2018].

[6] G. Konstantopoulos, "Understanding Blockchain Fundamentals, Part 1: Byzantine Fault Tolerance," Medium, 1 12 2017. [Online]. Available: https://medium.com/loom-network/understanding-blockchain-fundamentals-part-1-byzantine-fault-tolerance-245f46fe8419. [Accessed 22 11 2018].

[7] “6.20-29 Civil Referendum,” PopVote, 16 2 2015. [Online]. Available: https://popvote.hk/english/project/vote_622/. [Accessed 21 11 2018].

[8] J. Ma, W. Lee and R. Chung, "PopVote: A Revolution in Gathering Opinions in Hong Kong," RTHK, 12 8 2013. [Online]. Available: http://rthk9.rthk.hk/mediadigest/20130812_76_123023.html. [Accessed 21 11 2018].

[9] Hong Kong Computer Emergency Response Team Coordination Centre, "Advices on the security concerns of the PopVote System," 10 2 2017. [Online]. Available: https://www.hkcert.org/my_url/en/blog/17020901. [Accessed 21 11 2018].

[10] B. Adida, “Helios: Web-based Open-Audit Voting,” *USENIX security symposium,* vol. 17, pp. 335-348, 2008.

[11] S. T. Ali and J. Murray, "An overview of end-to-end verifiable voting systems," in *Real-world electronic voting: Design, analysis and deployment*, 2016, pp. 171-218.

[12] R. Peter Y. A., B. David, H. James, S. Steve and X. Zhe, "Prêt à Voter: a Voter-Verifiable Voting System," *IEEE transactions on information forensics and security,* vol. 4, no. 4, pp. 662-673, 2009.

[13] S. Popoveniuc and B. Hosp, "An Introduction to PunchScan," *Towards trustworthy elections,* vol. 6000, pp. 242-259, 2010.

[14] C. Meter, "Design of Distributed Voting Systems.," arXiv preprint arXiv:1702.02566, 2017.

[15] N. Faour, "Transparent Voting Platform Based on Permissioned Blockchain.," arXiv preprint arXiv:1802.10134, 2018.

[16] VoteCoin team, "Vote Coin: Anonymous Crypto Democracy," 2017.

[17] Y. Liu and Q. Wang, "An e-voting protocol based on blockchain," IACR Cryptol. ePrint Arch., Santa Barbara, CA, USA, 2017.

[18] F. Þ. Hjálmarsson and G. K. Hreiðarsson, "Blockchain-Based E-Voting System.," in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, 2018.

[19] F. S. Hardwick, G. Apostolos, N. A. Raja and K. Markantonakis, "E-Voting with Blockchain: An E-Voting Protocol with Decentralisation and Voter Privacy.," arXiv preprint arXiv:1805.10258, 2018.

[20] S. Panja and B. K. Roy, "A secure end-to-end verifiable e-voting system using zero knowledge based blockchain.," 2018.

[21] O. Pereira, "Internet Voting with Helios," in *Real-World Electronic Voting: Design, Analysis and Deployment*, 2016.

[22] R. Gupta, Hands-on Cybersecurity with BlockChain : Implement DDoS Protection, PKI-Based Identity, 2FA, and DNS Security Using BlockChain, Birmingham: Packt Publishing Ltd., 2018.

[23] "51% Attack," Investopedia, [Online]. Available: https://www.investopedia.com/terms/1/51-attack.asp. [Accessed 21 11 2018].

[24] "Hyperledger Fabric," Hyperledger, 2018. [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/latest/whatis.html. [Accessed 21 11 2018].

[25] S. Tilkov and S. Vinoski, "Node. js: Using JavaScript to build high-performance network programs," *IEEE Internet Computing,* vol. 14, no. 6, pp. 80-83, 2010.

[26] K. Chodorow, "MongoDB: The Definitive Guide: Powerful and Scalable Data Storage," O'Reilly Media, Inc., 2013.

[27] P. Leach, M. Michael and R. Salz, "A universally unique identifier (uuid) urn namespace," no. RFC 4122, 2005.

[28] "Report of the Task Force on the Computer Theft Incident of the Registration and Electoral Office," Legislative Council Secretariat, Hong Kong, 2017.

[29] D. Bressler, "4 Reasons a native mobile app can be more secure than a mobile browser based app," 8 3 2016. [Online]. Available: http://davidbressler.com/2016/03/08/4-

reasons-native-mobile-app-can-secure-mobile-browser-based-app/. [Accessed 21 11 2018].

[30] M. Kassner, "Apps vs. mobile websites: Which option offers users more privacy?," TechRepublic, 30 9 2016. [Online]. Available: https://www.techrepublic.com/article/apps-vs-mobile-websites-which-option-offers-users-more-privacy/. [Accessed 21 11 2018].