BLOCKCHAIN BASED ELECTRONIC VOTING PROTOCOL

Edison Kagona and Prof. Venansius Baryamureeba

Department of Computer Science and Information Technology, International University of East Africa, Kampala, Uganda.

E-mail: edisonmat15@gmail.com

Cite this article:

Edison K., Venansius B. (2022), Blockchain Based Electronic Voting Protocol. British Journal of Computer, Networking and Information Technology 5(1), 56-115. DOI: 10.52589/BJCNIT-7LST204K.

Manuscript History

Received: 13 April 2022

Accepted: 5 May 2022

Published: 5 June 2022

ABSTRACT: *SQL injection attack is now the most common server-side attack in web applications whereby malicious codes are injected into the database through user input fields by unauthorized users and this could lead to data loss or in the worst case, to database hijacking. The utilization of Blockchain technology in e-voting applications is not a new thing. Many systems have been proposed using cryptography and other security techniques. In those systems, minimal involvement of third party is observed, a problem of coercion resistance and transparency maintenance at the same time is observed and most processes have not been implemented to evaluate the systems further. This paper applies the cryptographic signatures to validate the origin and integrity of the votes by preserving the voter's choices during the election process. Furthermore, authors provide several possible extensions and improvements that can be made as an addition to the scope of this research.*

KEYWORDS: Algorithms, Blockchain, SQL injection, Decentralization, Cryptographic signatures, Data Structures.

INTRODUCTION

Background of the Problem

The use of technology has become common today in helping to meet human needs. Its increased usage has brought new challenges in the process of democracy as some people today have limited trust in the manual processes undergone through in the voting procedures, making elections very important in a modern democracy (Rifa Hanifatunnisa, 2017). To every nation, voting is a primary right and plays a significant role in constructing a democratic society. It gives individuals in a community the opportunity to voice their opinions (Williams, 2019). Therefore, it is imperative to keep our electoral system efficient, safe, and accessible for all citizens casting their votes.

Electronic voting (e-voting) refers to electronic voting machines, voting via the Internet from one's computer, cell-phone, or any digital device. Electronic Voting Machine (EVM) is a simple electronic device used to record votes in place of ballot papers and boxes which were used earlier in the conventional voting systems (D. Ashok Kumar, 2012). Although advocates for electronic voting argue that these systems minimize costs, increase participation, and provide convenience, there are several setbacks. These include skepticism of privacy, lack of transparency, policy mitigation, fraud, and a digital divide (Djina Pavlovic, 2019). The introduction of e-voting raises some of the same challenges as are faced when applying electronics to any other subject, for example, e-government (Caarls, November 2010). Politicians or administrators may perhaps expect that a paper version of a certain service or process can simply be taken and put on the Internet. Unfortunately, the reality is more complex, and nowhere more so than with e-voting. In every democracy, the security of an election is a matter of a national authority. Recognizing that the establishment and strengthening of democratic processes and institutions is the common responsibility of governments, the electorate, and organized political forces, that periodic and genuine elections are a necessary and indispensable element of sustained efforts to protect the rights and interests of the governed and that, as a matter of practical experience, the right of everyone to take part in the government of his or her country is a crucial factor in the effective enjoyment by all of the human rights and fundamental freedoms (Union, 1994). The voting systems that have been utilized in most countries to authorize people to cast their ballots are either paper-based (conventional) or electronic-based.

As an electronic voting system mainly relies on the internet platform, the fundamental challenge for e-voting is the significant security risks it might cause. The security issues of electronic voting are not all centered around online voting systems, though it is well established that electronic voting machines that are often deployed in polling stations can also be hacked. For example, in September 2017 the Hacking Conference DefCon published a report titled "Report on Cyber Vulnerabilities in U.S. Election Equipment, Databases, and Infrastructure" (Graceful, 2019). The report details of the vulnerabilities found in several voting machines, such as compromising an AVS WinVote remotely over Wi-Fi using a vulnerability from 2003. Interestingly though, they also managed to extract 650,000 voter records from Shelby County from an ExpressPoll device, which had not been correctly decommissioned.

Mueller report hearsays that "By at least the summer of 2016, GRU officers sought access to state and local computer networks by exploiting known software vulnerabilities on websites of state and local governmental entities. GRU officers, for example, targeted state and local databases of registered voters using a technique known as 'SQL injection'. In one instance in approximately June 2016, the GRU compromised the computer network of the Illinois State Board of Elections by exploiting a vulnerability in the SBOE's website. The GRU then gained access to a database containing information on millions of registered Illinois voters, and extracted data related to thousands of U.S. voters before the malicious activity was identified" (Robert S. Mueller, 2019).

To moderate risks, in the past 40 years, various procedures related to the ballot-privacy, individual verifiability, eligibility, completeness, fairness, uniqueness, robustness, universal verifiability, and receipt-freeness have been widely proposed (Orhan Cetinkaya, 2019). Besides, the published protocols have implemented a variety of technologies, such as blind signature, homomorphism encryption, Mix-Net, zero-knowledge proof (Xukai Zou, 2017). Back in 2005, Estonia became the first nation to hold a legally binding general election over the internet, something that has become a norm now, while most countries are still only contemplating the option (Plantera, 2019). Other countries like Australia, Canada, France, India, Mexico, Armenia, Panama, Switzerland, and the United States that have tested remote online voting, have mostly done so by introducing individual voting machines, which use vendor-produced software and are more prone to errors (Mulligan, 2017).

In 2010, Washington, D.C. developed an Internet voting pilot project that was intended to allow overseas absentee voters to cast their ballots using a website. Before deploying the system in the general election, the District held a unique public trial, a mock election during which anyone was invited to test the system or attempt to compromise its security. Within 48 hours of the system going live, there was near-complete control of the election server. There was a successful change of every vote and reveal of almost every secret ballot. Election officials did not detect the intrusion for nearly two business days and might have remained unaware for far longer (Scott Wolchok., 2012).

The discussion is no longer theoretical in Africa where an increasing number of countries are turning to electronic voting or including a digital component in the voting process, such as the biometric voter recognition kits and electronic results transmission system deployed in countries, For example, the last two elections in 2012 and 2016, Ghana had a strong digital component while Namibia held the continent's first-ever completely digital election, or "e-vote" in 2014, Kenya in 2013 set up the computer system to verify voters and remit results to the national tally center in Nairobi (Bagnoli, 2017). Governments in many countries are eager to establish electronic voting for a variety of reasons such as convenience, reduced costs, and hope for an increase in turnout, especially amongst young people (Canada, 2017).

For the case of Uganda, the practice of leaders assuming office through elections can be traced to the pre-independence period, when the British colonial government made a statute that allowed Africans to participate in the local elections starting with the Legislative Council (LEGCO) which was a precursor to Uganda's independence in 1962 (Commision, 1958 - 2012). The public general elections are governed by the Electoral Commission. The electoral commission is established under Article 60 and mandate under Article 61 of the Constitution of the Republic of Uganda 1995(as amended) to organize, conduct, and supervise regular, free, and fair elections and referenda, among other functions (Commission, 2019).

With the rising use of internet, web application vulnerability has been increasing effectively. SQL injection attack is an easiest method of attack in which attackers inject some SQL codes to the original code in the database to get sensitive information or to destroy the information. History says SQL injection attack has been there around for years and now this is a popular method to exploit the security system. Different techniques and methods have been developed and used to protect the database. But still attackers use this method very often because they are finding it easy to type a few deformed SQL commands into the front-end as well as back-end application. Types of SQL injection are tautologies, illegal or logically in corrected queries, union queries, piggy backed queries, blind injection, timing attacks (CEH_v9, 2019).

Attackers inject codes using tautology statements into the authentication phase to enter into the database, which says 1=1 is always true and so the injected query becomes true even if the wrong username and password are being entered. Similarly, they use logically incorrect queries to get an error message and this message works as a hint for them to find out some information. Single quotes, double quotes and backslashes are generally used in query to make these incorrect codes work correctly. Union operator is used while injecting codes to join the injected query to the original query. Piggy backed queries are those which use semicolons with injected codes to make duplicate codes work along with original ones. Hackers use blind SQL injection attack by asking some true or false questions if error messages are costumed by programmer. To make a delay in operation, attackers use timing attack and by taking the advantage of this, attacker hacks the username and password with the use of benchmarks (Sonam Panda, 2013).

Cryptography is an absolutely necessary field that ensures the security of databases. By applying encryption method, database attacks can be prevented. Today, Cryptography has been employed to offer several benefits to electronic voting and counting solutions (Marian Stoica, 2016). It is the science of protecting information by transforming it into a secure format. This process, called encryption, has been used for centuries to prevent handwritten messages from being read by unintended recipients. Cryptography is the study of secure communications techniques that allow only the sender and intended recipient of a message to view its contents (Karspersky, 2020). The term is derived from the Greek word kryptos, which means hidden. It is closely associated with encryption, which is the act of scrambling ordinary text into what's known as ciphertext and then back again upon arrival. Besides, cryptography also covers the obfuscation of information in images using techniques such as microdots or merging. Ancient Egyptians were known to use these methods in complex hieroglyphics, and Roman Emperor Julius Caesar is credited with using one of the first modern ciphers.

When transmitting electronic data, the most common use of cryptography is to encrypt and decrypt email and other plain-text messages. The simplest method uses the symmetric or "secret key" system. Here, data is encrypted using a secret key, and then both the encoded message and secret key are sent to the recipient for decryption. If the message is intercepted, a third party has everything they need to decrypt and read the message. To address this issue, cryptologists devised the asymmetric or "public key" system. In this case, every user has two keys: one public and one private. Senders request the public key of their intended recipient, encrypt the message and send it along. When the message arrives, only the recipient's private key will decode it, meaning theft is of no use without the corresponding private key (Karspersky, 2020).

Traditionally, cryptography (from the Greek for "hidden writing") was used to conceal information between two people using a secret key known only to them. Over time, it expanded into the art and science of using mathematics (in the form of algorithms) to hide information, protect privacy, ensure files are not altered and prove the identity of a message's sender. Considering the paramount importance of ballot secrecy and fraud detection, cryptography has proved a useful tool for countries employing election technologies (Institute, 2020). Another solution used to protect the secrecy of stored votes is homomorphic cryptography, which allows the votes in the electronic ballot box to be tabulated while still encrypted. As individual votes are never decrypted, there is no possibility of linking voters to the way that they voted. Votes may even be posted to a public bulletin board for independent tabulation by anyone to verify the outcome of the election.

On April 19, 2018, a public call was published on the Institute for Blockchain Studies website, addressed to associations, universities, research centers, students, and scholars to establish a voluntary working group, in charge of elaborating and proposing objectives linked to the use of blockchain technologies. By the end of this call, more than 300 applications for membership arrived, coming from all over the world" (City Government of Naples, Resolution No. 465 of October 5th, 2018, authors' translation) (Jonathan, 2020). The project of a blockchain voting system was the only one emerging from bottom-up, while the municipality intended to rather focus on administrative transparency, payments, and cryptocurrencies. Indeed, in a period of crisis of the traditional mechanisms of political representation, the innovation of democracy was vindicated, claiming the necessity of a plurality of participatory tools able to ensure inclusion, fairness, and transparency. This push generated a commitment to creating a viable solution that can innovate an already existing legal tool, the local referendum, by making it more affordable for the administrations and resistant to the dynamics of vote coercion or exchange (CoinTelegraph, 2020).

On 13th, September 2018, computer scientist J. Alex Halderman rolled an electronic voting machine onto a Massachusetts Institute of Technology stage and demonstrated how simple it is to hack an election. In a mock contest between George Washington and Benedict Arnold, three volunteers each voted for Washington. But Halderman, whose research involves testing the security of election systems, had tampered with the ballot programming, infecting the machine's memory card with malicious software. When he printed out the results, the receipt showed Arnold had won, 2 to 1. Without a paper trail of each vote, neither the voters nor a human auditor could check for discrepancies. In real elections, too, about 20 percent of voters nationally still cast electronic ballots only (Schwartz, 2018). The value of blockchain technology is now dawning on developers focusing on the accuracy of data results. This has led to a new race of developing new applications of this technology of which one of them is the application of blockchain technology in elections.

Mostly election is still stuck at the stage where voters have to leave their homes. In turn, they have to travel to a location with a ballot system. This has left the voting process open to all kinds of voter fraud. Blockchain technology in elections can bridge the infrastructure gap presently available for voting. Allowing the organizer to move the whole voting process online without the fear of compromise in terms of security. The development of a voting process based on blockchain mitigate the concern of internet connection security and possible voter and election fraud. Votes can be submitted without the need of the voters exposing their political affinity and identity. While the officials can effectively count casted votes with unquestionable conviction with the knowledge that every single ID is assigned to just a single vote. Thereby, preventing the creation of fakes and making tampering improbable. Many organizations like Horizon State are already developing applications that can bring blockchain to the voting process and voters. In the case of Horizon State, the company is proposing that voters will be able to cast from the comfort of their smartphones and computers. This can be done using decision tokens known as HST. Their votes are then entered into an unalterable blockchain that prevents tampering, election manipulation, and errors while recording. This kind of blockchain-based voting application can also see voting costs decrease (Francisco, 2020).

A blockchain is a digital, public ledger that records online transactions. Blockchain is the core technology for cryptocurrencies like bitcoin. A blockchain ensures the integrity of a cryptocurrency by encrypting, validating, and permanently recording transactions, similar to a bank's ledger, but open and accessible to everyone who utilizes the cryptocurrency (Bankrate, 2020). The blockchain serves as a public ledger of transactions that cannot be reversed. All the important consensus of the transaction (i.e. legitimate votes) is achieved through 'miners' agreeing to validate new records being added. Whenever a new insertion is to be made e.g. votes, then a new transaction record is created by a voter adding details of their cast vote to the blockchain. Should it be deemed a valid transaction then the new vote is added to the end of the blockchain and remains there forever. What is neat about this solution is the fact that no centralized authority is needed to approve the votes but rather a majority consensus. Here everyone agrees on the final tally as they can count the votes themselves & because of the blockchain audit trail, anyone can verify that no votes were tampered with and no illegitimate votes were inserted (Curran, 2018).

A useful voting system needs to balance out a range of key features. Security is one of the most critical factors to prevent any rivals or self-interested parties from being able to manipulate the results and also to ensure that the counted votes are authentic, otherwise, the result won't be fair and democratic. Despite how crucial security is to the voting process, it still needs to be balanced out with other requirements (Lake, 2019). For the final vote count to accurately represent the will of the people, the other properties that need to be considered in voting systems are:

Accuracy – The final vote count to accurately represent the choices of the people.

Verifiability – To be able to check the accuracy of the vote and determine whether an election has been tampered with.

Anonymity –The need for votes to be anonymous. If anyone else could find out who an individual voted for, it may lead to intimidation or coercion. This would compromise the integrity of the vote.

Accessibility – To take all voters into account. Allowing everyone to vote from the comfort of their own homes or offices to make the process easier.

Speed – To receive the results in a relatively short period. If it took a year to calculate the final vote tally, the will of the people at the time of the results may be very different from what it was at the time when the votes were cast.

Cost-effectiveness – We could have the most secure or accurate system in the world, but if it costs 10 times a nation's GDP to implement, it wouldn't be practical.

Considering the security requirements for a more trusted and secure electronic voting system, this thesis proposes an electronic voting protocol based on blockchain. The hypothesis is that blockchain establishes a system of creating a distribution agreement in the digital online world. This allows participating entities to know for certain that a digital event happened by creating a convincing record in a public ledger. It opens the door for developing a democratic open and scalable digital economy from a centralized one that is susceptible to server-side attacks.

LITERATURE/THEORETICAL UNDERPINNING

Electronic voting

Electronic voting fascinated much interest in the recent past and a variety of schemes were introduced. The first electronic voting system was introduced in the early eighties by David Chaum. The system used public-key cryptography, which was used to cast votes and keep voters anonymous. The blind signature theorem was used to make sure there were no links between voters and ballots (Chaum, 2019). Since then, many scholars have continued to show interest in the subject and a lot of research has been done (Taher Elgamal, July 1985), (Ayed, May 2017), (Sanson, October 16 - 19, 2001). According to our study of some of the previously developed models, most of the research done on the field focused on the direct recording electronic system and internet voting systems. Electronic systems can make casting a vote easier and more convenient, and increase the number of voters. However, technical threats to the e-voting system have always been a concern.

Several digital voting systems are presently in use in countries around the world and different security protocols have been proposed by many researchers world-wide. We researched some of these systems and the security protocols to acquaint ourselves with the current implementations as well as the blockchain technology and how it can be used in the voting procedures.

Existing electronic voting systems

Estonian I-Voting System

Estonia was the first country where citizens were able to cast their votes using only the internet and an electronic national identification card. Estonia’s voting over the internet is very straightforward. Like all the other digital services in the country, the internet voting system is made possible via the Estonian ID cards or Mobile-ID that enable secure remote authentication and legally binding digital signatures (Estonia, 2017). The ID card was designed to run on an integrated circuit, a Java chip platform, and protected with a 2048-bit PIN (Ben, 2017). The card can create signatures using SHA1/SHA2, easily usable for authentication, encryption, and signatures. The voter has to download the voting application, authenticate it using the electronic ID, and if the voter is eligible to vote a list of candidates will be displayed and a vote could be cast. The vote will be encrypted using the election’s public key and signed with the voter private key. As soon as the vote is cast it will be sent to a vote storage server controlled by the Estonian government (Springall, 2014). Voters could vote multiple times, and only the last vote will be considered valid. This is done to prevent vote-buying.

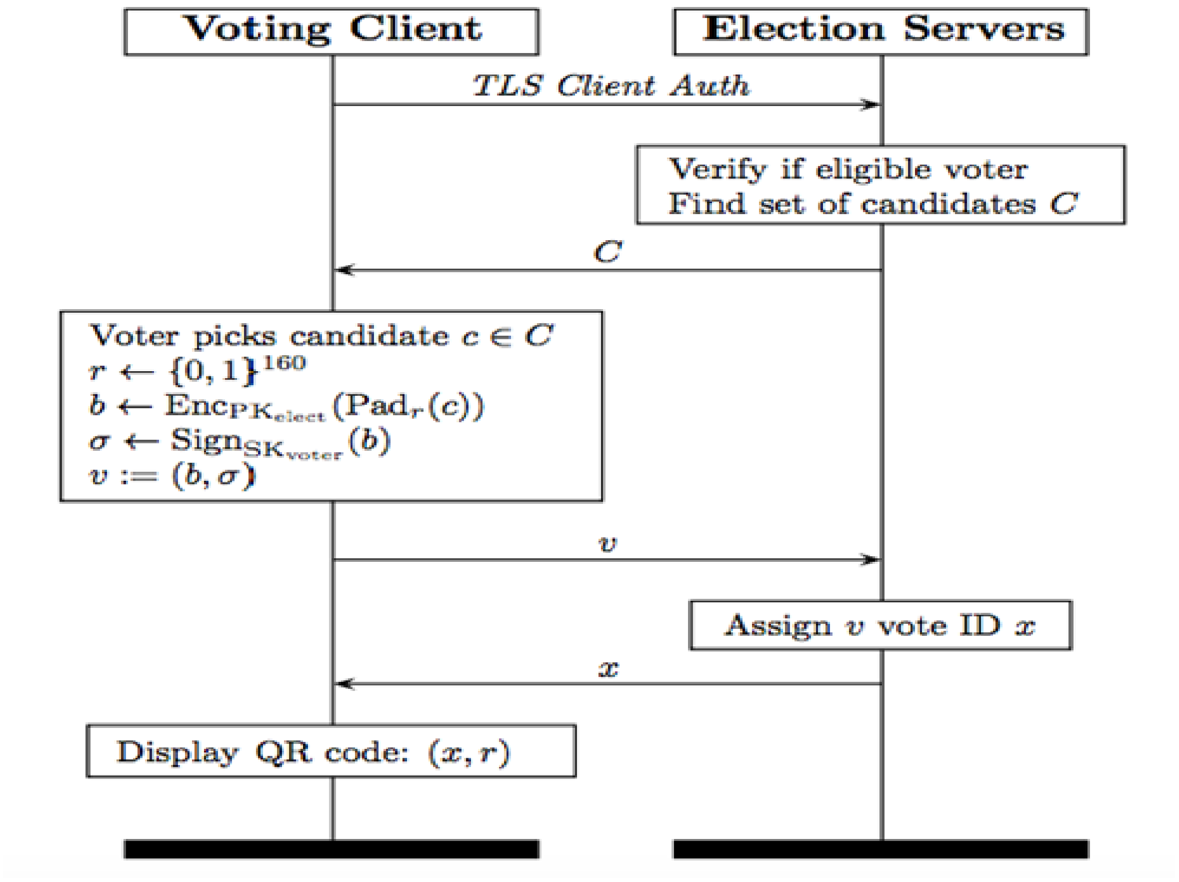

Voting Client
Election Servers
TLS Client Auth
Verify if eligible voter
Find set of candidates C
C
Voter picks candidate $c \in C$
$r \leftarrow \{0,1\}^{160}$
$b \leftarrow \mathrm{Enc}_{\mathrm{PK}_{\mathrm{elect}}}(\mathrm{Pad}_r(c))$
$\sigma \leftarrow \mathrm{Sign}_{\mathrm{SK}_{\mathrm{voter}}}(b)$
$v := (b, \sigma)$
v
Assign v vote ID x
x
Display QR code: (x, r)

Figure 1: Vote Casting Process in the Estonian I-Voting System (Drew Springall T. F., November 2014)

During the 2013 Local Election, researchers observed and studied the I-voting process and highlighted several potential security risks with the system. One such risk was the possibility of malware on the client-side machine that monitors the user placing their vote and then later changing their vote to a different candidate. Another possible risk is for an attacker to directly infect the servers though malware being placed on the DVDs used to set up the servers and transfer the votes (Drew Springall T. F., 2014). However, this report has also come under criticism from the Estonian Information Systems Authority.

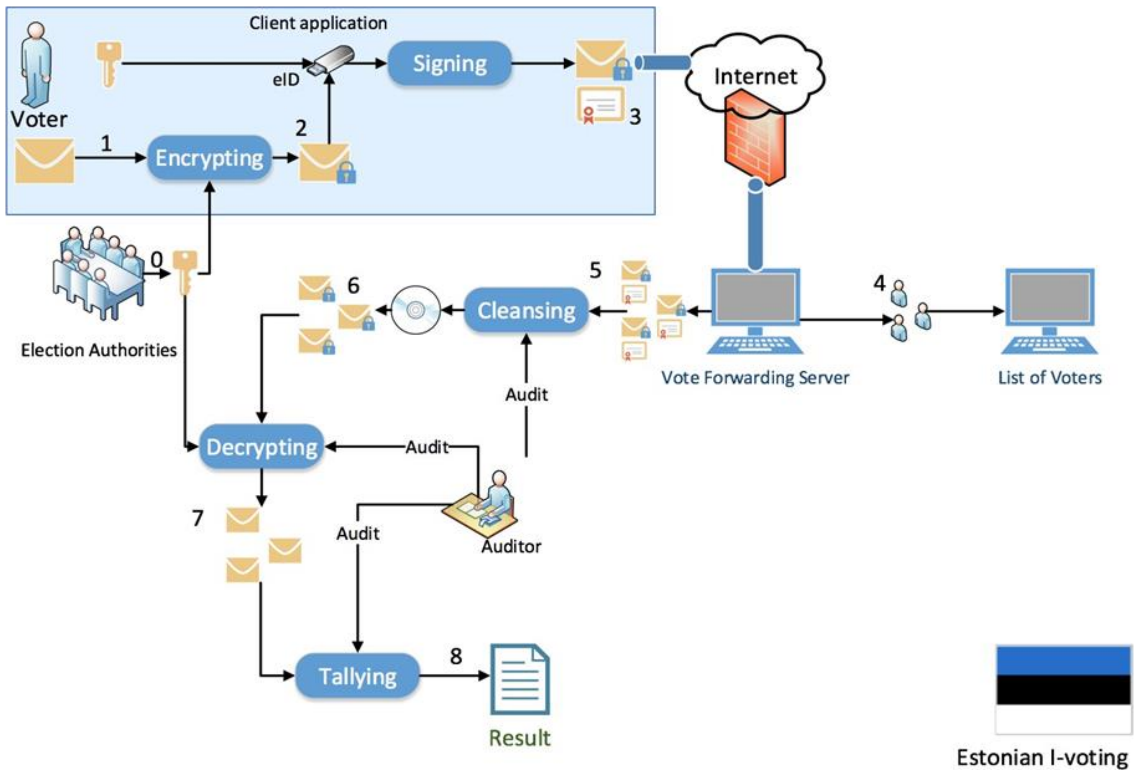


Figure 2: Estonian Digital Voting System (Source: R. Verbij. "Dutch e-voting opportunities." Master thesis, University of Twente, 2014)

Norwegian I-Voting System

In 2011 Norway used an electronic remote voting system for the country council elections. The system was developed by e-voting vendor Scytl and was very similar to the Estonian electronic voting system. However, in 2014, the country discontinued its I-Voting project due to security concerns (Ayed, May 2017). One of the main criticisms the Norwegian I-Voting system faced was the fear of votes going public in case of a cyber-attack.

New South Wales iVote System

In 2015, about 280,000 eligible citizens placed their votes using the iVote system in the New South Wales State election (Ayed, May 2017). iVote was developed by Scytl as well but had a different design from the Norwegian system. To cast a vote, citizens had to undergo four steps, either of the two was optional.

- The voter had to register with authorities, receive a voter ID, and choose a six-digit PIN.
- The voter logins in the system using his ID and PIN, cast a vote, then receives a 12-digit receipt number as a confirmation.
- The voter enters his ID, PIN, and receipt number to verify that his vote went through. This step is optional.

- After the election is over, the voter still can use his 12-digit receipt to check if his vote was included in the final count. If the vote was not counted, a reason will be displayed. This is an optional step as well.

D.C Digital Vote-by-Mail Service:

In 2010, Washington D.C developed a pilot electronic voting system and performed a dummy election to test the security of the system. Many critical issues were found; therefore, the project was canceled and never used in any official elections (Scott Wolchok, Feb. 2012).

Attacks on the existing e-voting systems and protocols

The secrecy of the voter's ballot is a critical defense against voter coercion and vote-buying. The I-voting system implements relatively strong protection against in-person, individual coercion by allowing voters to cast replacement votes online or to cancel their electronic ballot entirely and vote in person on election day. More sophisticated attacks remain possible, however, including spyware on the voter's PC or smartphone, as well as server-side SQL injection attacks.

Based on the criticism made on the existing systems, server-side SQL injection attacks on ballot secrecy are particularly troubling, since preserving ballot secrecy is the main goal of the system's cryptographic double-envelope architecture. I-voting design attempts to ensure that votes remain private by breaking the association between voters' digital signatures from their plaintext votes. The encrypted ballots are separated from the signatures and copied to an isolated machine before being decrypted and counted. Note that this machine, the counting server, have access to the complete association between the encrypted ballots and the plain text votes. An attacker who can smuggle this information out through a covert channel can compromise every voter's secret ballot.

Unfortunately, the tabulation procedures offer multiple possibilities for exfiltrating this information. When tabulation is complete, officials use the counting server to burn a DVD containing both vote totals and log files. Suppose for simplicity that the attacker is a dishonest insider with access to this DVD and to the complete set of signed, encrypted ballots (e.g. from a backup disk) and some mechanism for infecting the counting server with malicious code, such as the routes discussed above. The counting server malware can sort the encrypted ballots and leak the voter choices corresponding to each as a sequence of integers in the same order. Since there is typically only one race, only a few bits per ballot are needed to determine the choices of all voters. The malware could steganographically encode this data into the log files through the order of entries, or it could simply write this information to unallocated sectors of the disc. The attacker can then decode this information and use it to associate every voter's digital signature (and hence, their identity) with their vote.

- Server-side SQL injection attack on Estonian I-voting system.

The e-voting system places complete trust in the server that counts the votes at the end of the election process. Votes are decrypted and counted entirely within the unobservable "black box" of the counting server. This creates an opportunity for an attacker who compromises this server to modify the results of the vote counting. The researchers demonstrated that they can infect the counting server with vote-stealing malware. In this attack, a state-level attacker or a dishonest election official inserts a stealthy form of infectious injection code onto a computer used in the pre-election setup process. The infection spreads via software DVDs used to install the operating systems on all the election servers. This injected code ensures that the basic checks used to ensure the integrity of the software would still appear to pass, despite the software having been modified. The attacker's modifications would replace the results of the vote decryption process with the attacker's preferred set of votes, thus silently changing the results of the election to their preferred outcome (Estonia, 2014).

- Client-side attacks: A bot that overwrites your vote

Client-side attacks have been proposed in the past, but the researchers found that constructing fully functional client-side attacks is alarmingly straightforward. Although Estonia uses many security safeguards, including encrypted web sites, security chips in national ID cards, and smartphone-based vote confirmation, all of these checks can be bypassed by a realistic attacker.

A voter's home or work computer is attacked by infecting it with malware, as millions of computers are every year. This malicious software could be delivered by pre-existing infections (botnets) or by compromising the voting client before it is downloaded by voters by exploiting operational security lapses. The attacker's software would be able to observe a citizen voting then could silently steal the PIN codes required to use the voter's ID card. The next time the citizen inserts the ID card, say, to access their bank account, the malware can use the stolen PINs to cast a replacement vote for the attacker's preferred candidate. This attack could be replicated across tens of thousands of computers. Preparation could be well in advance of the election starting by using a replica of the I-voting system, as the team did for their tests.

One core strength of the I-voting system is Estonia's national ID card infrastructure and the cryptographic facilities it provides. While the ID cards cannot prevent every important attack, they do make some kinds of attacks significantly harder. The cards also provide an elegant solution for remote voter authentication, something few countries do well.

Drawbacks and Security issues with the existing e-voting protocols and systems.

- One of the main critics of both Estonian and Norwegian electronic voting systems is the secrecy of critical parts of the code. The script to post the vote on the Estonian I-Voting system is made close to what raises questions about transparency. An open-source e-voting system is a must for a trusted election.

- The centralization of the I-Voting system makes it vulnerable to DDOS attacks what could make the elections inaccessible to voters.

- Estonian I-Voting system, one such risk was the possibility of malware on the client-side machine that monitors the user placing their vote and then later changing their vote to a different candidate. Another possible risk is for an attacker to directly infect the servers though malware being placed on the DVDs used to set up the servers and transfer the votes.

- Intelligence Agencies have access to a wide range of network traffic and enough computing power to analyze voting data for a potential alteration. Even with enhanced security, State level attacks are possible in all previously motioned systems.

Existing electronic voting protocols

The utilization of Blockchain technology in e-voting applications is not a new thing. Many schemes have been proposed using cryptography and other security techniques. In such cases, minimal involvement of third party is observed and a problem of coercion resistance and transparency maintenance at the same time is observed. However, most of those protocols have not been implemented to evaluate the protocols further.

In the past years, securing data on the electronic voting systems, the blind signature theorem was used to make sure there were no links between voters and ballots (Chaum, 2019). Since then, many scholars have continued to show interest in the subject and a lot of research has been done. For example, Yi Liu and Qi Wang proposed a decentralized e-voting protocol without the existence of a trusted third party. The protocol was designed but required further optimization and implementation according to the specified conclusion and the balancing of transparency and coercion-resistance was a possible future work (Wang, 2017).

Antony Lewis et al. (2018) described blockchain as an open, distributed ledger of historical records that uses cryptography and digital signatures. In his paper he also mentioned the logic of blockchain and how it works. On explaining the aftermath of re-solving conflicts, he introduced an idea of not broadcasting a block intentionally. Two blocks can be created, and one can be left as being not broadcasted, the un-broadcasted block can be broadcasted when desired (Lewis, 2018). Clement Chan Zheng Wei and Chuah Chai Wen (2020) proposed a blockchain-based electronic voting protocol. This protocol was implemented using blockchain to turn election protocol into an automated control system without relying on any single point of entity (Clement Chan Zheng Wei, 2020). However, this proposed protocol does not give a step by step description of the solution and does not include any proposed algorithm or encryption standard used in registration, client side or server-side security of the voting process.

Another proposed e-voting procedure is a blockchain based secured e-voting by using the assistance of smart contract, this protocol utilizes smart contract into the e-voting system to deal with security issues, accuracy and voters' privacy during the vote. The protocol results in a transparent, non-editable and independently verifiable procedure that discards all the intended fraudulent activities occurring during the election process by removing the least participation of the third party and enabling voters' right during the election (Kazi Sadia, 2020). However, the proposed solution does not show how the voters' data is securely captured and verified using an encryption standard or hashing algorithms.

Electronic voting protocol based on blockchain

The previously developed protocols do not specify the actual security gaps they solve in the electronic voting systems, they do not provide a step-by-step solution and some do not include well defined and proposed algorithms or encryption standards used in securing voters and their choices how they are captured, stored and verified. Almost all the existing protocols have not been implemented to further test the applicability of the designed protocol.

Having analyzed the gaps in the existing developed and proposed electronic voting protocols and systems, the proposed electronic voting protocol based on blockchain focuses on mitigating SQL injection attacks by improving the quality of protections through the decentralization of electronic voting using distributed databases on a linked network chain creating a public and transparent voting process while protecting the anonymity of the voter's identity.

Information Security and Cryptography

Information security is the process that describes all measures taken to prevent unauthorized use of electronic data, whether this unauthorized use takes the form of destruction, use, disclosure, modification, or disruption. Additionally, information security and Cryptography share the common services of protecting the confidentiality, integrity, and availability of the information ignoring data form (electronic document, printed document) (Menezes A, 1996).

Cryptography in E-voting

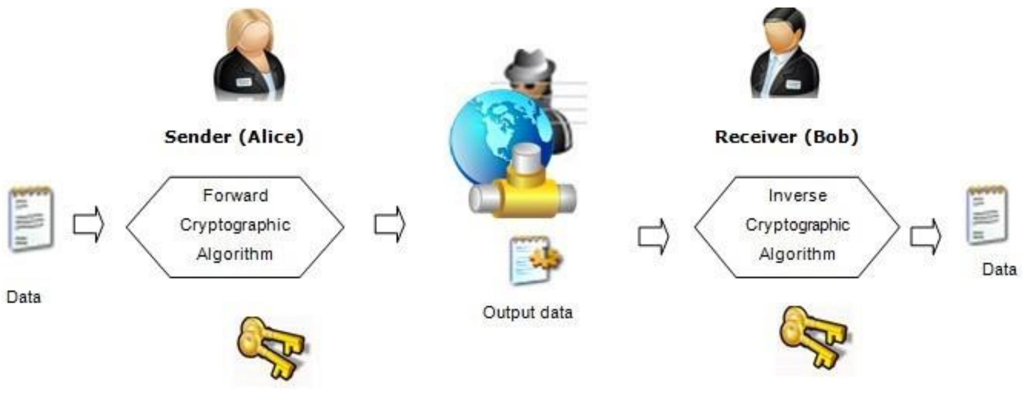

Sender (Alice)
Forward Cryptographic Algorithm
Data
Output data
Receiver (Bob)
Inverse Cryptographic Algorithm
Data

Figure 2: Cryptographic scheme

Cryptography is used to protect the confidentiality of the message using methods called encryption and decryption. To hide the secret of the message, encryption is used. The message is encrypted using a secret key and generates a ciphertext. To reveal the secret message, decryption is used. The ciphertext is decrypted using a secret key to get back the secret message. Therefore, the security of the cryptosystem relies on the secret key, this is best known as Kerkhof's principle (J. Hoffstein, 2008). It is a method of transferring private information and data through open network communication (refer to Figure 3. However, cryptography provides many services such as confidentiality, authentication, integrity, non-repudiation, and accessibility.

Cryptography provides information security for other useful applications such as in encryption, message digests, zero-knowledge proof of identity, key-sharing, and digital signatures. The length and strength of the Cryptography keys are considered an important mechanism. The keys used for encryption and decryption must be strong enough to produce strong encryption. They must be protected from unauthorized users and must be available when they are needed. It also contributes to Computer Science, particularly, in the techniques used in computer and network security for access control and information confidentiality (Branovic I, 2003). Cryptography is also used in many applications encountered in everyday life such as Electronic

Voting, computer passwords, ATM cards, and electronic commerce (Alia M, 2007). Generally, Cryptography may be divided into two main categories;

- Asymmetric / two key/ public-key: Ciphering and deciphering using a pair of keys.
- Symmetric / one key/ secret-key: Ciphering and deciphering using the same key (or without key – in the case of Hash function).

Secret-key (Symmetric) Algorithms

Secret-key (refer to Figure 4) is also known as a single-key or one-key algorithm. Secret-key is an encryption scheme consisting of sets of encryption and decryption algorithms. The plaintext is encrypted by key e and the ciphertext is decrypted by key d, where e is the encryption key and d is the decryption key. In a secret-key scheme, key d must be equal to key e as shown in Figure 4. The Data Encryption Standard (DES) is an example of the secret-key scheme (Standards N. B., 1977).

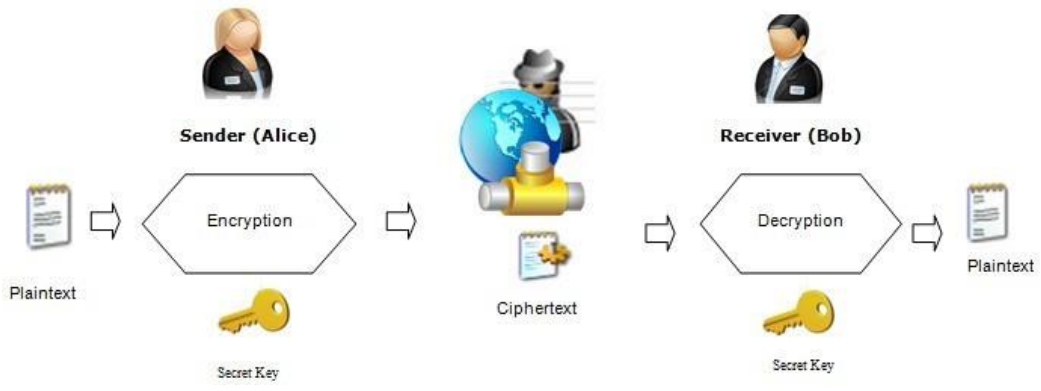


Figure 3: Secret-key cryptographic scheme

Public-key (Asymmetric) Algorithms

In public-key algorithms (refer to Figure 5), there is a pair of keys, one of which is known to the public and used to encrypt the plaintext to be sent to the receiver who owns the corresponding decryption key, known as the private key.

Every public-key cryptosystem is based on a mathematical problem that is in some sense difficult to solve. These problems are called "hard problems" and are classified into two major categories according to the Cryptography classifications, as P (Polynomial) and NP (Non-deterministic polynomial) (Laboratories, 2007). The problem is considered to be a P hard problem if the problem can be solved in polynomial time, while a problem is considered to be an NP-hard mathematical problem if the validity of a proposed solution can be checked only in polynomial time. The three major types of mathematical hard problems that had been successfully used in Cryptography are described in the following subsections of this part. These problems are;

i. The Integer Factorization Problem (IFP)
ii. The Discrete Logarithm Problem (DLP)
iii. The Elliptic Curve Discrete Logarithm Problem (ECDLP)
iv. Chaotic Hard Problem (CHP)

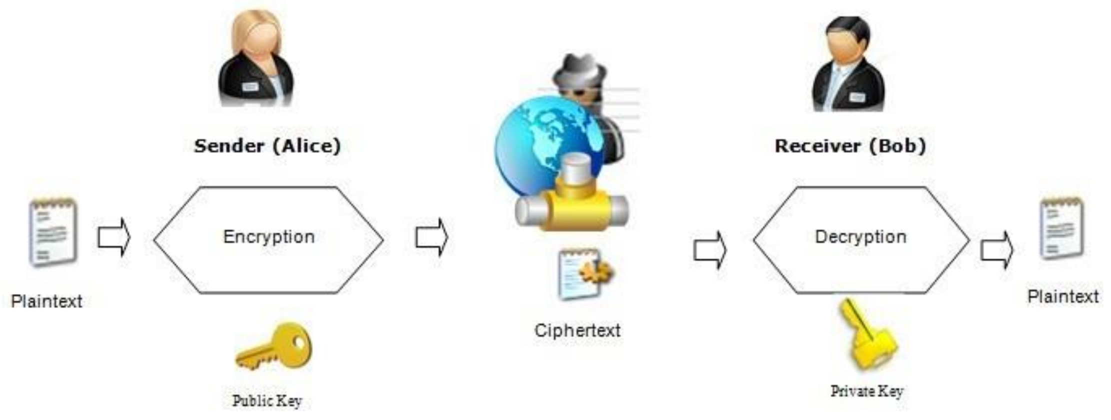


Figure 4: Public-key cryptographic scheme

Public key encryption

The public-key cryptosystem concept was developed by Diffie-Hellman in 1976 (Diffie W, 1976). The RSA algorithm is the first encryption protocol based on the public-key concept, which was published by Revist, Shamir, and Adleman in 1978 (Rivest R A, 1978). In RSA, one key is known to the public (receiver's public key) and is used to encrypt the information by the sender. The other key is known as a private key, and it is used to decrypt the encrypted data received by the receiver (receiver's private key). There are many other public-key encryption algorithms published since the RSA was made public. Among them are ElGamal (ElGamal, 1985), Elliptic Curve (Koblitz, 1987), etc. Only a few public-key algorithms are both secure and practical. Of these, only some are suitable for encryption. While the others are only suitable for digital signatures. This can be seen in the following list:

- Integer factorization (RSA, Rabin).
- Discrete logarithm problem (ElGamal).
- Knapsack (subset) (Merkle-Hellman, Chor-Rivest).
- Probabilistic method (Blum-Goldwasser, Goldwasser- Micali).
- Elliptic Curve (Elliptic Curve, modified Elliptic Curve).

- Algebraic code theory (Mc Eliece).
- Fractal system (Newton Raphson law).

RSA Public-Key Encryption Protocol

The RSA protocol is the most widely used public-key encryption algorithm (R. A. Rivest, 1978). It may be used to provide both secrecy and digital signatures. The RSA security is based on the intractability of the integer factorization problem. However, there are three integers e, d, and n used in the encryption and decryption algorithm, where n = p × q, with p and q being large primes. Below are the details of the RSA algorithm.

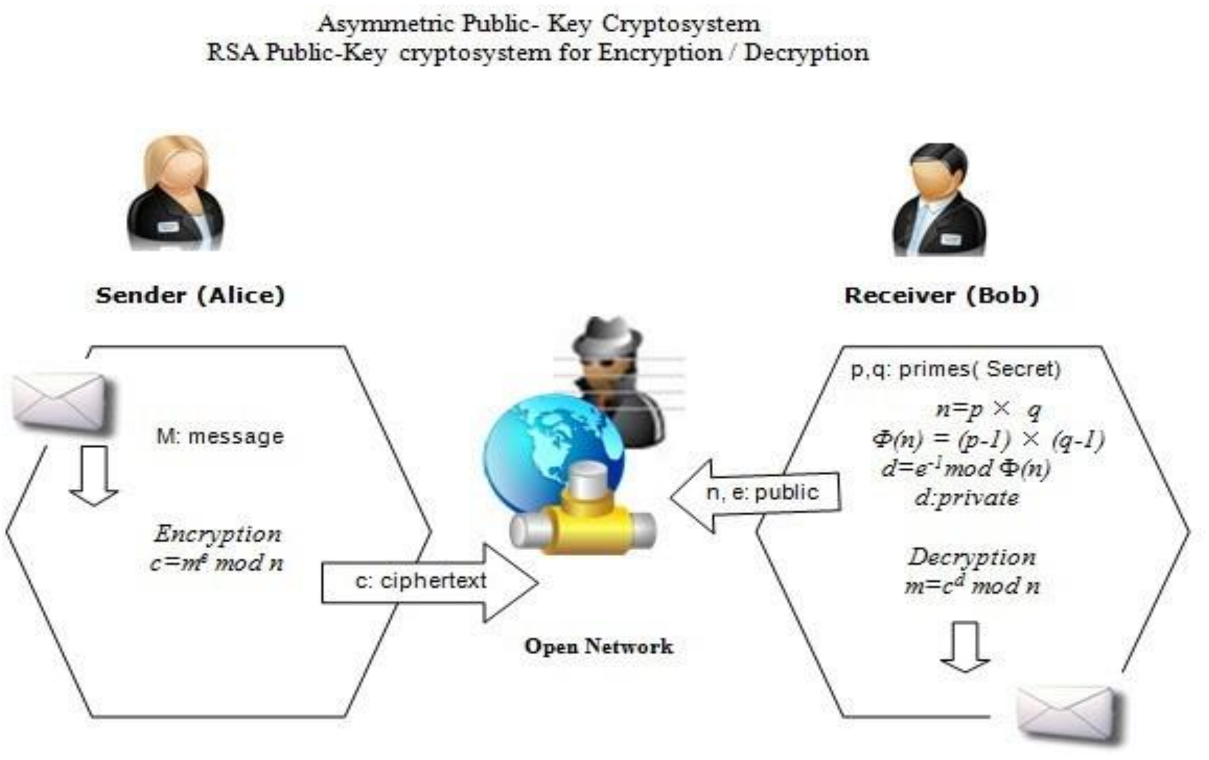


Figure 5: RSA Encryption protocol

Blockchain uses cryptography to secure the identity of a sender and ensuring the records are tamper-proof. Therefore, implementing cryptography into electronic voting may ease the privacy of electronic voting.

Blockchain

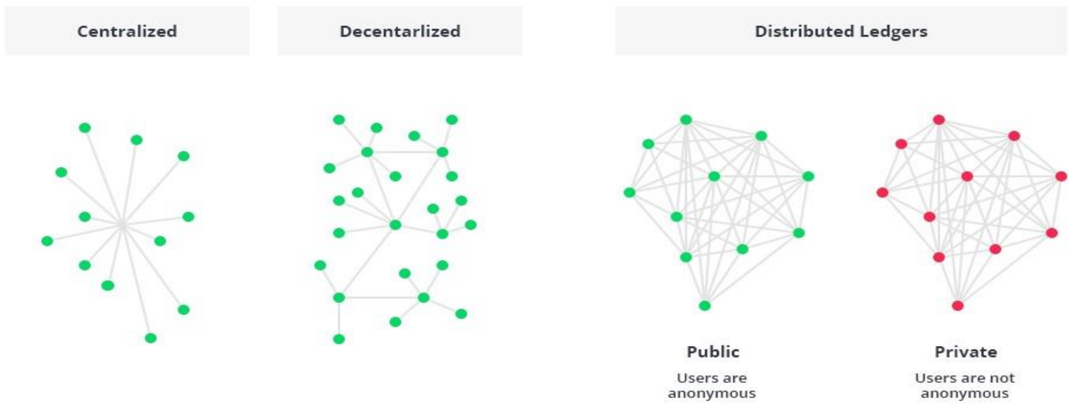


Figure 6: Blockchain structure representation

The term blockchain was first described back in 1991. A group of researchers wanted to create a tool to timestamp digital documents so that they could not be backdated or changed. Further, the technique was adopted and reinvented by Satoshi Nakamoto (MLSDev, 2019). In 2008, Nakamoto proposed a peer-to-peer payment system that allows cash transactions through the Internet without relying on trust or the need for a financial institution, the blockchain-based project called Bitcoin. Blockchain is secure by design, and an example of a system with a high byzantine failure tolerance (Hardwin spenkelink, 2014). Bitcoin is considered the first application of the Blockchain concept to create a currency that could be exchanged over the Internet relying only on cryptography to secure the transactions. Blockchain is an ordered data structure that contains blocks of transactions. Each block in the chain is linked to the previous block in the chain. The first block in the chain is referred to as the foundation of the stack. Each new block created gets layered on top of the previous block to form a stack called a Blockchain.

Table 1: Comparison of traditional databases to blockchain

CONSTRAINTS	BLOCKCHAIN	DATABASES
ARCHITECTURE	Blockchain uses a distributed ledger network architecture.	The database utilizes client-server architecture.
DATA HANDLING	Blockchain utilizes Read and Write operations.	The database supports CRUD (Create, Read, Update, and Delete).
INTEGRITY	Blockchain data supports integrity.	Malicious actors can alter database data.
TRANSPARENCY	Public blockchain offers transparency.	The database is not transparent. Only the administrator decides which data the public can access.

COST	Blockchains are comparatively harder to implement and maintain.	The database being an old technology is easy to implement and maintain.
PERFORMANCE	Blockchain is bobbed down by the verification and consensus methods	Databases are extremely fast and offer great scalability.
BEST USE CASES	1. Transfer value 2. Storage value 3. Monetary transactions 4. Voting systems 5. Decentralized apps (dApps)	1. Apps or systems that utilize the continuous flow of data. 2. Storing confidential information. 3. Online transaction processing that needs to be fast. 4. Apps or systems where data verification is not needed. 5. Rational data.

METHODOLOGY

The blockchain logical architecture

Into the technical aspects of the blockchain, each block contains a spreadsheet that comprises of the hash of the previous block, miner address, list of unconfirmed transactions, and a random number. This entire spreadsheet of information then feeds through a cryptographic hash function.

For the network chain to accept the block, the output of the hash must be small enough. The random number in the block spreadsheet determines the size of the hash output. The only way to find a small hash is by trial and error. If a block with larger hash output is broadcasted into the network, it automatically gets rejected. Sometimes, there will be different nodes that find different solutions to the same block at the same time. Then the blockchain performs a temporary split where some nodes accept a version of the solution while the other part of nodes accepts the other. Blockchain follows a rule of "Longest blockchain is the correct blockchain." This concept is based on the consensus of every miner in the network. Once a certain node chain is longer than the other, the blockchain on the entire network will ditch the old blockchain and accept the new longest chain as the correct blockchain.

Blockchain in the electronic voting protocol

Introducing blockchain technology in the E-voting protocol can strengthen the security of the e-voting process and protect the privacy of each voter as well as minimize the server-side SQL injection attacks experienced by the existing voting software programs. The blockchain-based e-voting protocol is decentralized and does not need to rely on human trust. The registered voter has the right to vote using their electronic devices connected to the Internet. All the vote records are publicly distributed and can be verified by any intended personnel. No one can corrupt the voting process.

Hash Functions in Blockchain

A cryptographic hash function is used to ensure the integrity of the blockchain. In the blockchain, each block is processed one at a time with the hash function, each time combining a hashed from the previous block. To have the node on the entire network to accept a new block, each block must consist of the correct hash value from the previous block and the hash value of the current block.

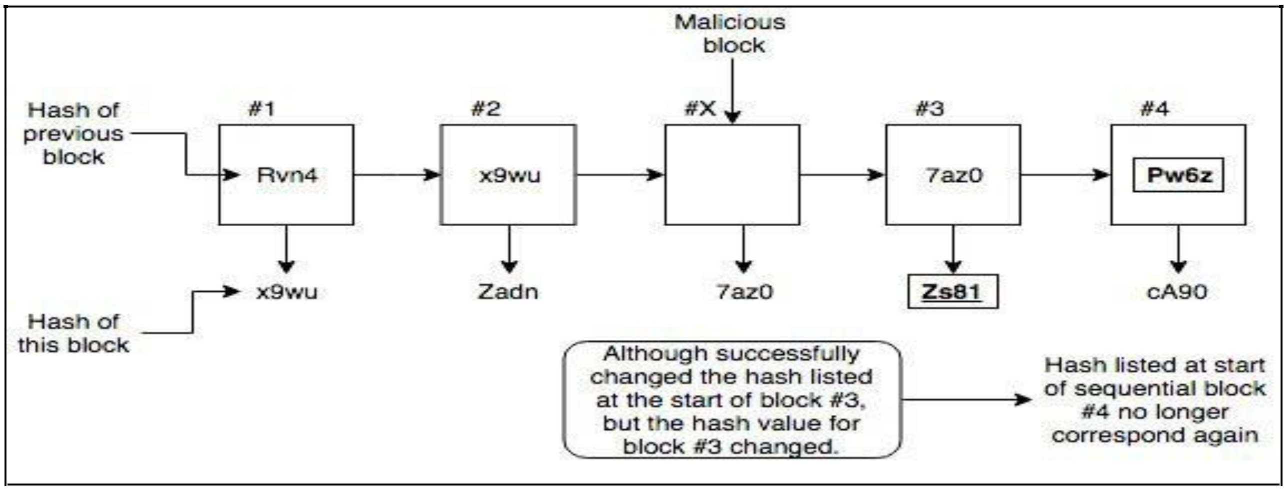

Malicious block
Hash of previous block
#1
Rvn4
#2
x9wu
#X
#3
7az0
#4
Pw6z
x9wu
Zadn
7az0
Zs81
cA90
Hash of this block
Although successfully changed the hash listed at the start of block #3, but the hash value for block #3 changed.
Hash listed at start of sequential block #4 no longer correspond again

Figure 8: Malicious block added into the chain

When a malicious block is added into the middle of the blockchain, the hash output listed at the start of the next block for the "malicious block" no longer corresponds as shown in The blockchain network will never accept this "malicious" chain until every block in the chain follows the correct hash value corresponding to its previous block hash. The extra block must fulfill the characteristic to be accepted into the chain, but it will be extremely difficult to do so.

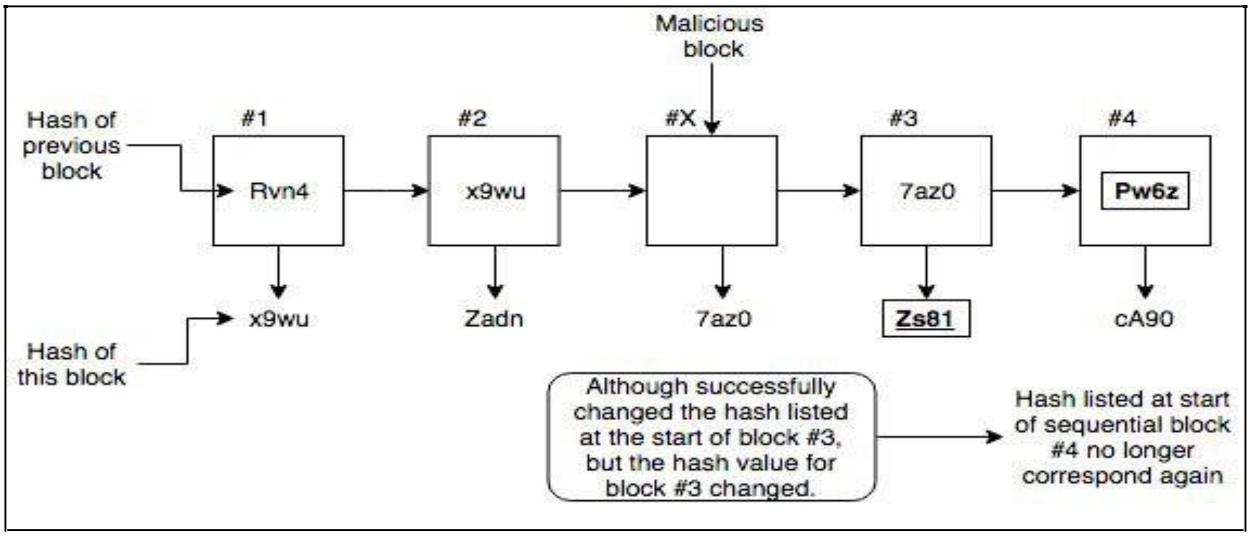

Malicious block
Hash of previous block
#1
Rvn4
#2
x9wu
#X
#3
7az0
#4
Pw6z
x9wu
Zadn
7az0
Zs81
cA90
Hash of this block
Although successfully changed the hash listed at the start of block #3, but the hash value for block #3 changed.
Hash listed at start of sequential block #4 no longer correspond again

Figure 9: Retrospective changes in the sequential block

If the malicious attacker manages to solve the extra block and change the hash output listed at the start of the next block to correspond to the "malicious" block, the attacker will also have to change the output hash of the sequential block to match the block he/she previously solved as shown in *Figure 9*. To successfully make any changes to a blockchain, every subsequent block has to be solved again by the attacker. The attacker must have the computing power greater than the majority of the network to successfully alter the chain. Because by the time the attacker solves all the blocks in the "malicious" chain, the other nodes of the chain would have already solved new blocks and have a longer blockchain. Therefore, the hacker's "malicious" chain will be rejected because it is shorter than the correct blockchain consent by the entire network.

Digital Signature in Blockchain

A digital signature is used to authorize the blockchain. A valid digital signature requires two keys. First is the private key, a long random string used to gain access to all information stored on the account. The private key must never be shared with anyone. Second is the public key corresponding to the private key which is referred to as the address of the account. The public key is distributed to anyone.

The way how digital signature works is remarkable. First, the message is hashed to get the hash value. Next, the hash value is signed using the sender private key. This process is known as a digital signature. Lastly, the message and digital signature are sent to the recipient. The recipient verifies the sender's digital signature using the sender's public key. For an attacker who tries to alter a block in the blockchain, they need to change the contents in a block first then generate a digital signature to match the "malicious" changes. The only way to do so is by trial and error because the hash function is a "one-way" function. To successfully find the correct hash for the "malicious" changes take an extremely long time. As an example, guessing every combination of solutions through the SHA-256 hash function will take roughly 1050 years. Thus, the digital signature proved to protect the integrity of a block in the blockchain.

Database vs. Blockchain Architecture

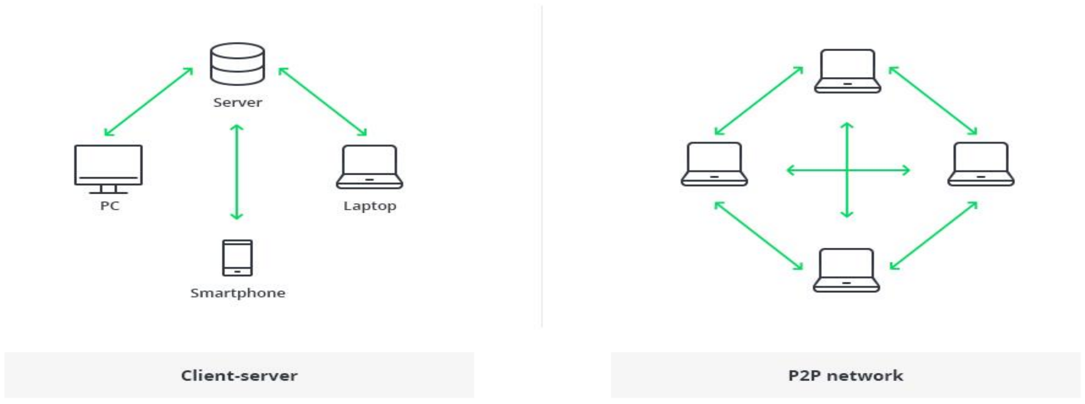


Figure 7: Database vs Blockchain Architecture

The blockchain is a decentralized, distributed ledger (public or private) of different kinds of transactions arranged into a P2P network. This network consists of many computers, but in a way that the data cannot be altered without the consensus of the whole network (each separate computer). The structure of blockchain is represented by a list of blocks with transactions in a particular order. These lists can be stored as a flat file (txt. format) or in the form of a simple database. Two vital data structures used in blockchain include;

- **Pointers** - variables that keep information about the location of another variable. Specifically, this is pointing to the position of another variable.
- **Linked lists** - a sequence of blocks where each block has specific data and links to the following block with the help of a pointer.

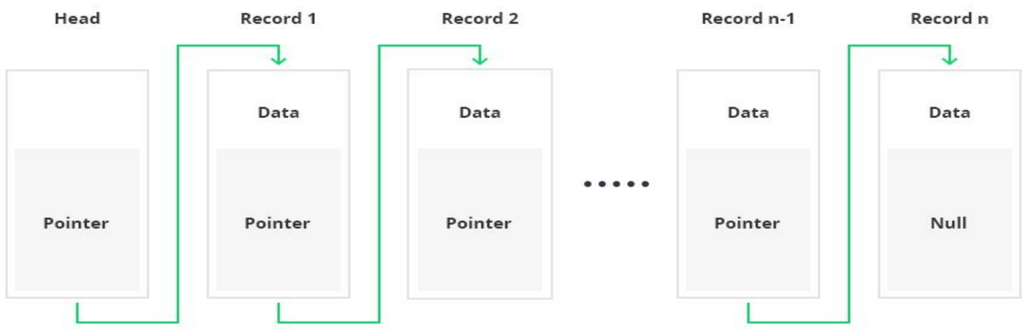


Figure 8: Structure representation of Pointers and linked lists

Logically, the first block does not contain the pointer since this one is the first in a chain. At the same time, there is potentially going to be a final block within the blockchain database that has a pointer with no value. The following blockchain sequence diagram is a connected list of records:

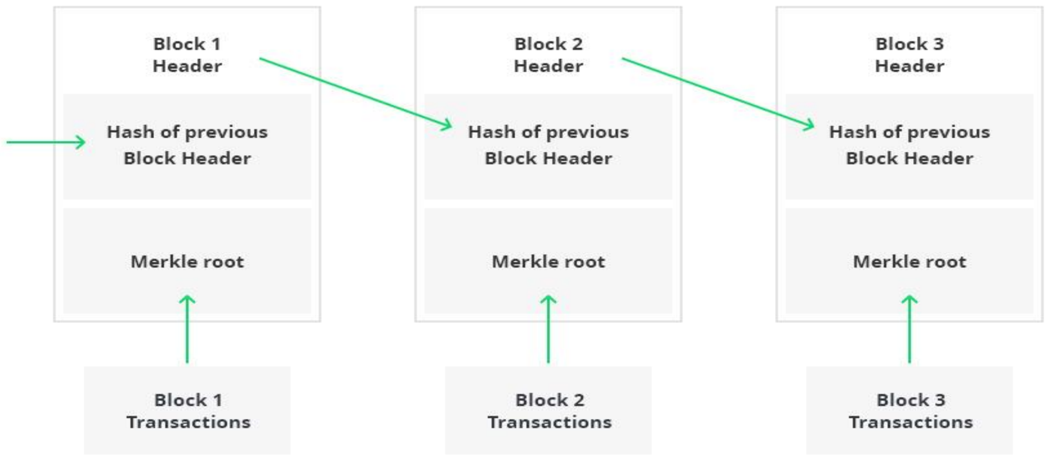


Figure 9: Logical view of the blockchain

Blockchain architecture can serve the following purposes for organizations and enterprises:

- **Cost reduction** - lots of money is spent on sustaining centrally held databases (e.g. banks, governmental institutions) by keeping data current secure from cybercrimes and other corrupt intentions.

- **History of data** - within a blockchain structure, it is possible to check the history of any transaction at any moment in time. This is an ever-growing archive, while a centralized database is more of a snapshot of information at a specific point.

- **Data validity & security** - once entered, the data is hard to tamper with due to the blockchain's nature. It takes time to proceed with record validation since the process occurs in each independent network rather than via compound processing power. This means that the system sacrifices performance speed, but instead guarantees high data security and validity.

Each block in the stack is identified by a hash placed on the header. This hash is generated using the Secure Hash Algorithm (SHA-256) to generate an almost idiosyncratic fixed-size 256-bit hash. The widely used algorithm was designed by the National Security Agency (NSA) in 2001 and was used as the protocol to secure all federal communications (Marko Hölbl, 2018). The SHA-256 will take any size plaintext as an input, and encrypt it to a 256-byte binary value. The SHA-256 is always a 256-bit binary value, and it is a strictly one-way function.

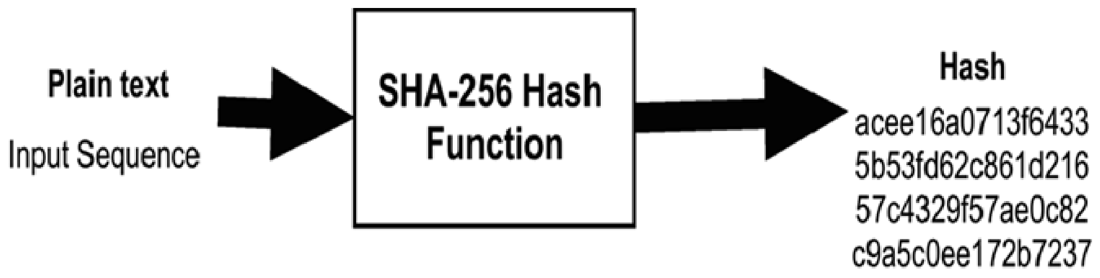


Figure 10: The basic function of the SHA-256 Hash

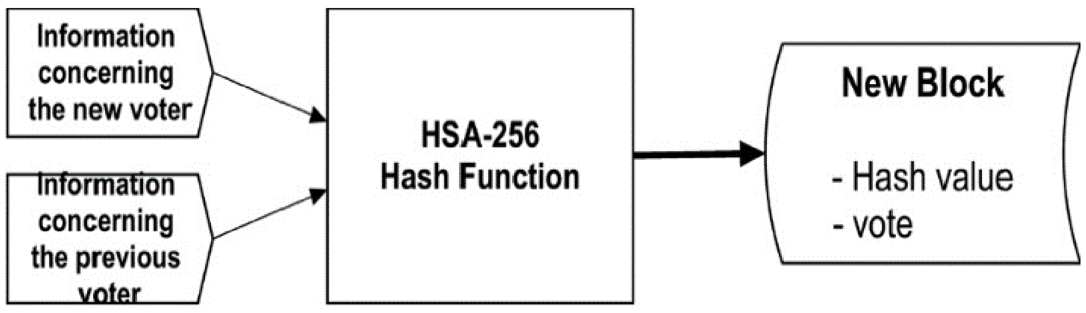

Information concerning the new voter
Information concerning the previous voter
HSA-256 Hash Function
New Block
- Hash value
- vote

Figure 14: New voter and Previous voter information

Each header contains information that links a block to its previous block in the chain, which creates a chain linked to the very first block ever created, which is referred to as the foundation. The primary identifier of each block is the encrypted hash in its header. A digital fingerprint that was made combining two types of information; the information concerning the new block created, as well as the previous block in the chain. As soon as a block is created, it is sent over to the Blockchain. The system will keep an eye on incoming blocks and continuously update the chain when new blocks arrive.

Design of the protocol logical architecture

Based on the outcomes from the above review of the blockchain logical architecture, we integrated the blockchain paradigm into the e-voting procedure and came up with a feasible and general e-voting protocol.

More precisely, we list the properties that our protocol satisfies as follows.

- Public Verifiability. Everyone involved in the election, including spectators, who can see the voting process (recorded on the blockchain), can verify the whole election's procedure and its outcome.
- Individual Verifiability. Each voter can verify individual voting procedures, e.g., whether his/her ballot has been cast and recorded successfully, counted in the final tally, etc.
- Dependability. Guaranteed by the cryptographic algorithms and the practical consensus mechanisms of blockchain, the protocol protects the voting procedure against dishonest behaviors and attacks.
- Consistency. Supported by the practical consensus mechanisms of blockchain again, all participants involved in the election, hold the same record of the voting procedure, and thus accept the same outcome of the election.
- Auditability. The whole voting procedure recorded on the blockchain is auditable after the election.
- Anonymity. Only voters themselves know the information of their votes, and all ballots in the ballot box have no connection with their voters.
- Transparency. Due to the transparency of blockchain, the whole procedure is open to the public. This leads to more fairness and validity.

Techniques employed in the blockchain-based electronic voting protocol

An Ethereum transaction needs to be included in a block and mined before it is processed and saved on the blockchain. As a result, on-chain transactions take time and costs gas to compensate miners for their work. In contrast, off-chain computation lets you perform actions instantly without waiting for transactions to be mined and does not cost any gas. In this protocol, we perform off-chain computation using Ethereum signatures. The cryptographic signatures are used to validate the origin and integrity of the votes by preserving the voter's choices during the election process. Blockchain, a data structure derived from Bitcoin as "public use", guarantees the transparency of the election procedure.

Use cases of off-chain computation such as decentralized interactions, state channels, and meta transactions are surveyed. In our protocol, the Ethereum signatures are implemented using blind RSA signatures.

Blind RSA Signature

In cryptography a blind signature, as introduced by David Chaum, is a form of digital signature in which the content of a message is disguised (blinded) before it is signed. The resulting blind signature can be publicly verified against the original, unblinded message in the manner of a regular digital signature. Blind signatures are typically employed in privacy-related protocols where the signer and message author are different parties. In our protocol, the integrity requires each Voter to be certified by an election authority (EA) before they can be accepted to vote. This is considered because it allows the authority to check the credentials of the voter to ensure that they are legible to vote.

At the same time of voting, it is important that this authority does not learn the voter's selection. An un-linkable blind signature provides this guarantee, as EA will not see the contents of any ballot it signs, and will be unable to link the blinded ballots it signs back to the un-blinded ballots or decrypted results. More formally a blind signature scheme is a cryptographic protocol that involves two parties; a user Alice that wants to obtain signatures on her messages, and a signer Bob that owns his secret signing key. At the end of the protocol, Alice obtains Bob's signature on message (*m)* without Bob learning anything about the message. This intuition of not learning anything is hard to capture in mathematical terms. The usual approach is to show that for every (adversarial) signer, there exists a simulator that can output the same information as the signer.

Therefore, in our protocol the signer does not view the message content, but a third party can later verify the signature and know that the signature is valid within the limitations of the underlying signature scheme. The message in this case is the voter's choices on ballots since they have to be signed by an authority. The blind RSA signature is demonstrated in the following way;

Select the two large prime numbers

P, q

Compute: $n = p*q$,

$$v = (p - 1) * (q-1)$$

Select small odd integer k, relatively prime to v.

$$\gcd (k, v) = 1.$$

Compute d such that $(d * k) \% v = (k*d) \% v = 1$.

Public Key is (k, n)

Private Key is (d, n).

Using the above formulated public key and private key in message signing between 2 parties Alice and Bob.

Suppose C (Alice) wants to obtain a signature from D (Bob) on a message blindly. In this case the signer D owns his secret signing key.

Suppose D's public key is (e, n) and his private key is (d, n).

D $\longrightarrow$ Public key = *(e, n)*

D $\longrightarrow$ Private key = *(d, n)*

C first blinds the message m, by multiplying it by k^e *mod n*, where *k* is a randomly chosen number called the blinding factor.

$k^e \longrightarrow$ Randomly chosen number.

C sends the blinded message (m) $\rightarrow$ *m.* k^e *mod n* to D.

Next: D signs the blinded message, resulting in $(m.\ k^e)^d$ *mod n* and sends the signed blinded message back to C.

Result blinded message to C $\longrightarrow (m.\ k^e)^d$ *mod n.*

Finally, C unblinds the message by dividing by *k mod n*, resulting in $(m.\ k^e)^d$ */ k mod n*.

Result of unblinded message $\longrightarrow (m.\ k^e)^d$ */ k mode n.*

Therefore;

$(m.\ k^e)^d / k \text{ mod } n = m^d * k^{ed} / k \text{ mode } n \text{ (mod } n)$

By the reasoning given above

$m^d.\ k / k \text{ mod } n \text{ (mod } n) \longrightarrow m^d \text{ mod } n$ This is D's signature on message (m) from C.

In the above message signing procedure, Alice represented by C obtains Bob's signature on message *(m)* without Bob learning anything about the message. D owns a signing function only controlled by himself.

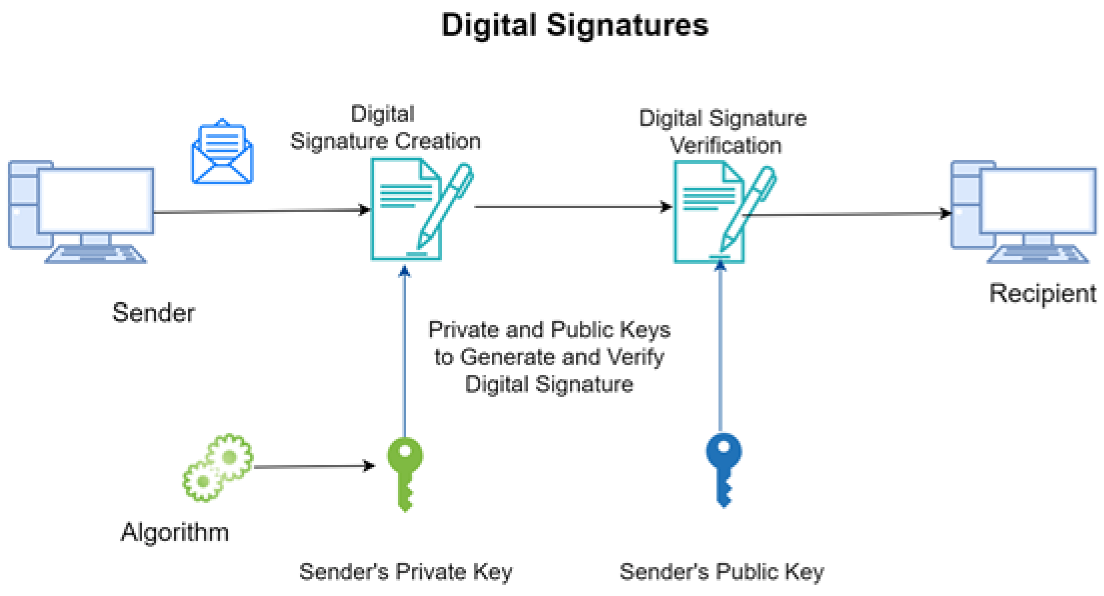

Digital Signatures
Digital Signature Creation
Digital Signature Verification
Sender
Recipient
Private and Public Keys to Generate and Verify Digital Signature
Algorithm
Sender's Private Key
Sender's Public Key

Figure 11: Structure of the Digital signature

An encrypted version of piece of data can be generated on signing it with a private key. Others can verify for-example, decrypt it with your public key. Others can use your public key to sign a piece of data only intended for you. You can decrypt the same with your private key. The encryption algorithms make use of one-way functions -- mathematical functions that are computationally cheap to execute to arrive at an output given two inputs but computationally expensive by many orders to arrive at an expected set of inputs given an output

In our protocol, the signing and verification of data is purely computational and does not require any form of connection to the Ethereum networks.

Message transmission to the blockchain

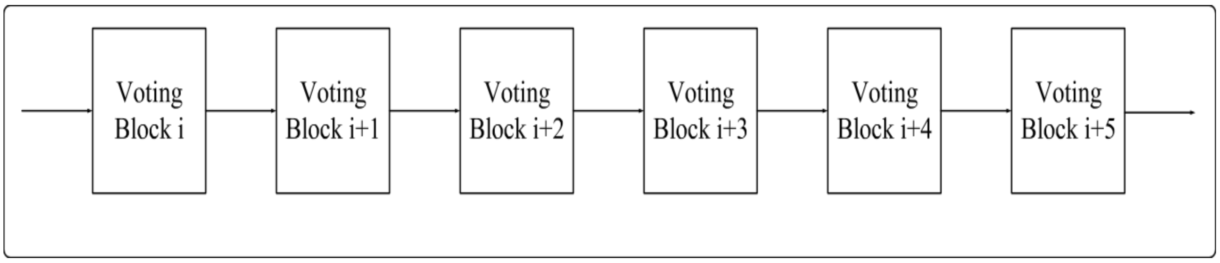


Figure 12: The blockchain for e-voting

Blockchain is a data structure in which data is organized as blocks, and blocks connect chain. Each block's creation is based on the latest block of the most current chain, and these creations are processed by nodes in the blockchain Peer-to-Peer (P2P) network. The blockchain-based electronic protocol is represented as a series of voting blocks sequentially chained to each other.

The first block is called the genesis block. Each block contains the voter's ID, vote, voter's signature, timestamp, and digest (hash) of the previous block, which is depicted in Figure 17 below, and illustrated as follows:

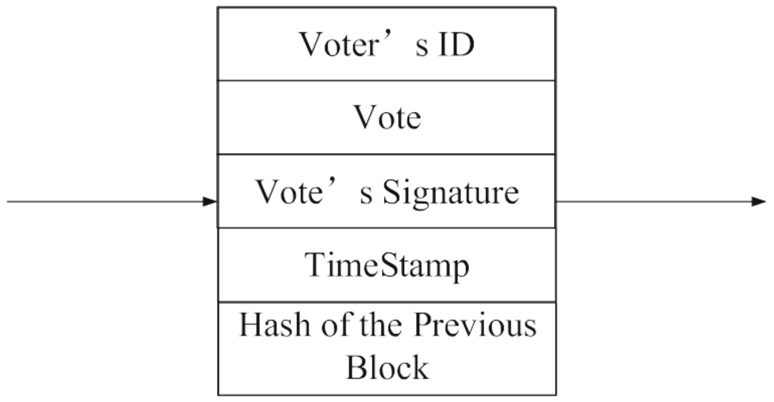


Figure 17: Voting message structure written to the blockchain

- Voter's ID: the person who casts a ballot for his/her chosen candidate is a voter. Ater's ID is randomly assigned to a person who has the right to vote.
- Vote: voting ballot is to state a ballot to the voter's chosen candidate.
- Voter's signature: voting ballot is marked by the voters as a signature so that no one else can find out for whom a citizen is voting. The voter uses his/her private key to sign the hash of the vote, which is used to judge the authenticity of the vote.
- Timestamp: timestamp is used to record the submission time of the block. The block with a higher value of signature is selected over others when they have the same timestamp.
- Hash of the previous block: we used the SHA-256 algorithm to compute the hash value of the previous block. Thus, the blockchain-based e-voting scheme is non- repudiation and is resistant to modification of the data.

Computing the hash value based on SHA-256

We compute the hash value based on SHA-256. By comparing the hash value to the expected hash value, the data's integrity can be determined. SHA-256 is used in the e-voting scheme to compute the hash value, which is depicted in Figure.18 below, and illustrated as follows:

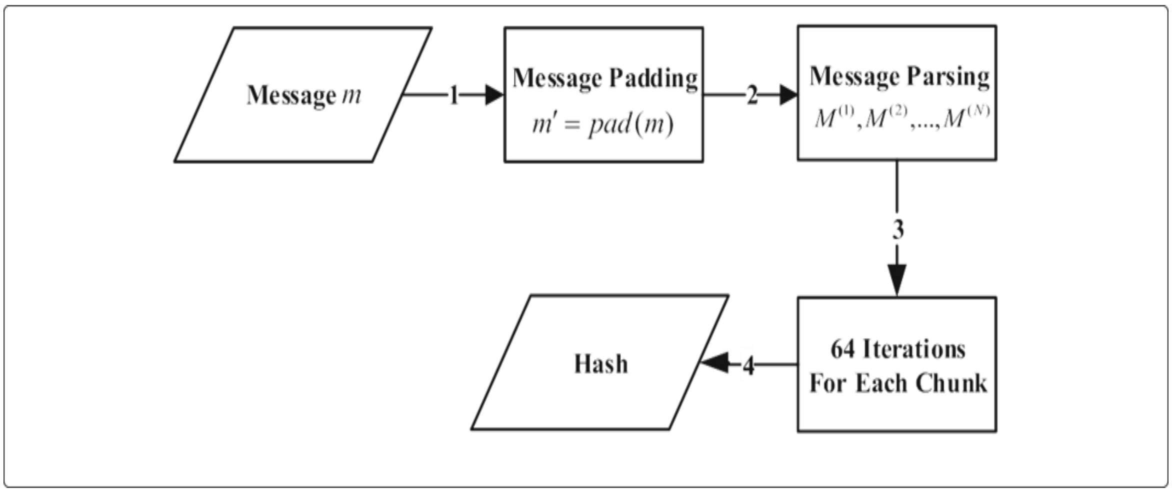


Figure 18: Computing the harsh value based on SHA-256

- The message is denoted by m with binary expression.
- Pad m with 100...000 sequence and the length of m with 64 expressions, i.e., m^r = pad(m).
- m^r is broken into 512-bit chunks, i.e., $M^{(1)}, M^{(2)}, ..., M^{(N)}$.
- 64 constants are used, which are denoted by $W0, W1, ..., W63$, respectively.
- Eight working variables labelled $A = 0x6A09E667$,
- B = $0xBB67AE85$, $C = 0x3C6EF372$, $D = 0xA54FF53A$,
- E= $0x510E527F$, $F = 0x9B05688C$, $G = 0x1F83D9AB$ and $H = 0x5BE0CD19$ are used as the initial hash value.
- Compute the 64-cycle cryptographic iterative computation for the first chunk, i.e., $M^{(1)}$. Repeat the iterative computation for the next chunk based on the result for the last chunk. The result of the last iterative computation is the hash.

Mining and generation of voting blocks

All votes in the blockchain are cryptographically linked block by block. Many secure hash algorithms can be applied to solve the problem of condensing the message in the current block to produce a message digest, such as SHA-256.

A new block is generated by users from the P2P network. The new block generation is based on the proof-of-Work (PoW) algorithm. When a new vote is submitted and verified, the miner generates a new block with the information of vote and broadcasts the new blocks to the network. If new blocks have the same timestamp, the block with a higher value of signature is selected over others.

Message transmission procedure

Now we introduce the message transmission on the blockchain. Every user is associated with an asymmetric key pair, i.e., a public key and a private key. As the abstract message structure depicts below, senders fill out the area with their public keys, receivers' public keys, and message contents. Afterward, senders use their private keys to sign messages and send them to the blockchain P2P network. In this setting, messages are collected and packed into a block during a period. As a message spreads over the network and is recorded on the blockchain, it is obtained from the blockchain by the receivers.

Table 2: Message Structure and An Example

Message	
Sender	
Receiver	
Message	
Sender's signature	

Message	
Sender	*pk_{Voter}*
Receiver	*pk_{EA}*
Message	*Message content*
Sign(hash(msg), sk_{Voter})	

We claim that our protocol can be adapted on either a public blockchain or permissioned blockchain. We may simply hold our elections on some existing public blockchains since the security of such blockchains is assumed to be high.

Protocol notation and definitions

Table 3: Notation structure description

j	Voter j
j_{pub}	Public key of j. Uniquely identifies j, also serves as signature verifying key.
j_{priv}	The private key counterpart of j_{pub} used as a signing key.
EA_{sign} (m)	Digital signature produced by EA over message m.
CH_j	Voter's choice in the election.
DCH_j	Digital commitment for CH_j
OV_j	DCH_j Opening value. CH_j Cannot be derived from DCH_j without the opening value.
x / y	Inclusion of value x and y in a single message.
E_{tokenj}	Eligibility token
E_{tokenj} = EA_{sign} (j_{pub} / DCH_j)	

- Voter: A voter, identified by one's public key, j_{pub}, is considered an object that is permitted to cast a vote towards one of the candidates. Voters can access the electronic voting platform through a voting client-side application in any browser of choice, preference, the security of which is assumed for example the tor-browsers. The terms client and voter will be used interchangeably to describe the object acting as the voter. During elections, using $CH_{j,}$ a voter will make a choice that range from a set of predefined choices. In order to assure fairness, the voter reveals only a digital commitment over the said choice. DCH_j only reveals the choice itself only during the counting stage of the election.

- Electoral Authority (EA). In order for the e-voting protocol to provide declaration that only eligible voters are able to vote, it was deemed necessary for an Electoral Authority to be introduced. This is any voting officer assigned the role to monitor the voting process to do verification and authorization of the eligible voters. For a user to be judged eligible, one must authenticate oneself to the Electoral Authority (EA), and receive a token that proves one's eligibility to vote. The eligibility token, E_{tokenj}, takes the form of a digital signature over a voters j_{priv} and DCH_j.

  $$E_{tokenj} = EA_{sign}(j_{pub} \mid DCH_{j})$$

For the EA to judge whether voters are eligible or not, it is required for it to maintain a list of all the voters that are allowed to participate in the elections. The EA is also required to have access to voter authenticating information in order to have the ability to authenticate eligible users.

- Vote: A vote is a message of predefined structure, that is the equivalent of a bitcoin transaction. A vote is required to include a ballot and any other information that the practical implementation of the protocol requires it to include. Each vote x, once included in the blockchain is identified by a vote id, $V(id_x)$, a value that uniquely recognizes a vote.

- Ballot: A ballot, B_j is the digital representation of the physical ballot, i.e. the paper where the choice of a voter is written on. A ballot is considered sealed when the opening value of the digital commitment has not been revealed and, thus no party, other than the voter, can determine the way a voter voted. Once the opening value has been revealed and the choice of the voter is publicly known, the ballot is considered open. The public key, included in the ballot symbolizes the owner of the ballot and by extension the owner of the vote.

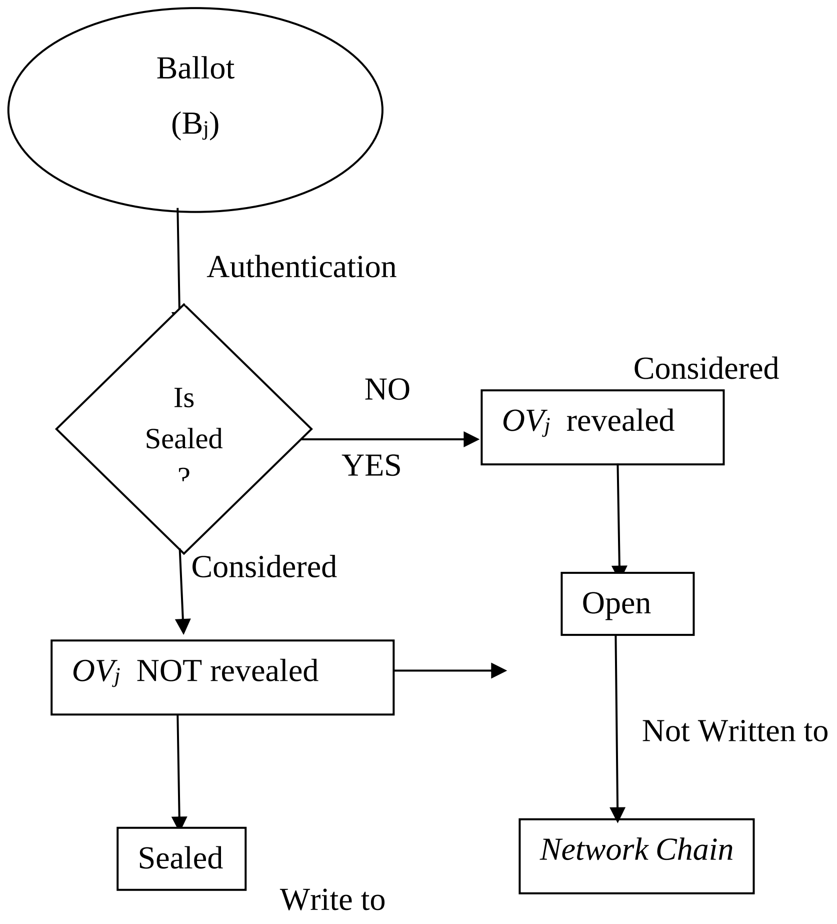


Figure 19: Data flow diagram describing the ballot authentication procedure

Protocol Execution in phases

The protocol has been divided in four distinct phases;

The initialization phases

In this phase, rules governing the elections are pre-determined by the electoral authority (EA). The Ethereum blockchain and all other systems of the protocol are initialized. The voting office under EA decides the duration of the voting process, the phases involved and whether vote cancelation will be granted or not. This communication is published and communicated through their official website and using other communication media houses. The Ethereum infrastructure and the EA are collectively governed by the same rules. The EA during the initialization phase will be provided the list of the eligible voters that are eligible to vote as well as a way to authenticate those users. A pair of signing and verifying keys for the public signature scheme will be generated and the verifying key will be publicized as a system wide parameter. The blockchain will be initialized with an initialization block, that will serve as the genesis block, of the chain. The initialization block does not contain any votes, but instead it contains all the information of the election, including the EA's signature validating key, the set of valid choices the voters can choose from and so on. This way a blockchain is tied to a specific election and all the system parameters become part of the blockchain and thus dispute over them is prevented.

The preparation phases

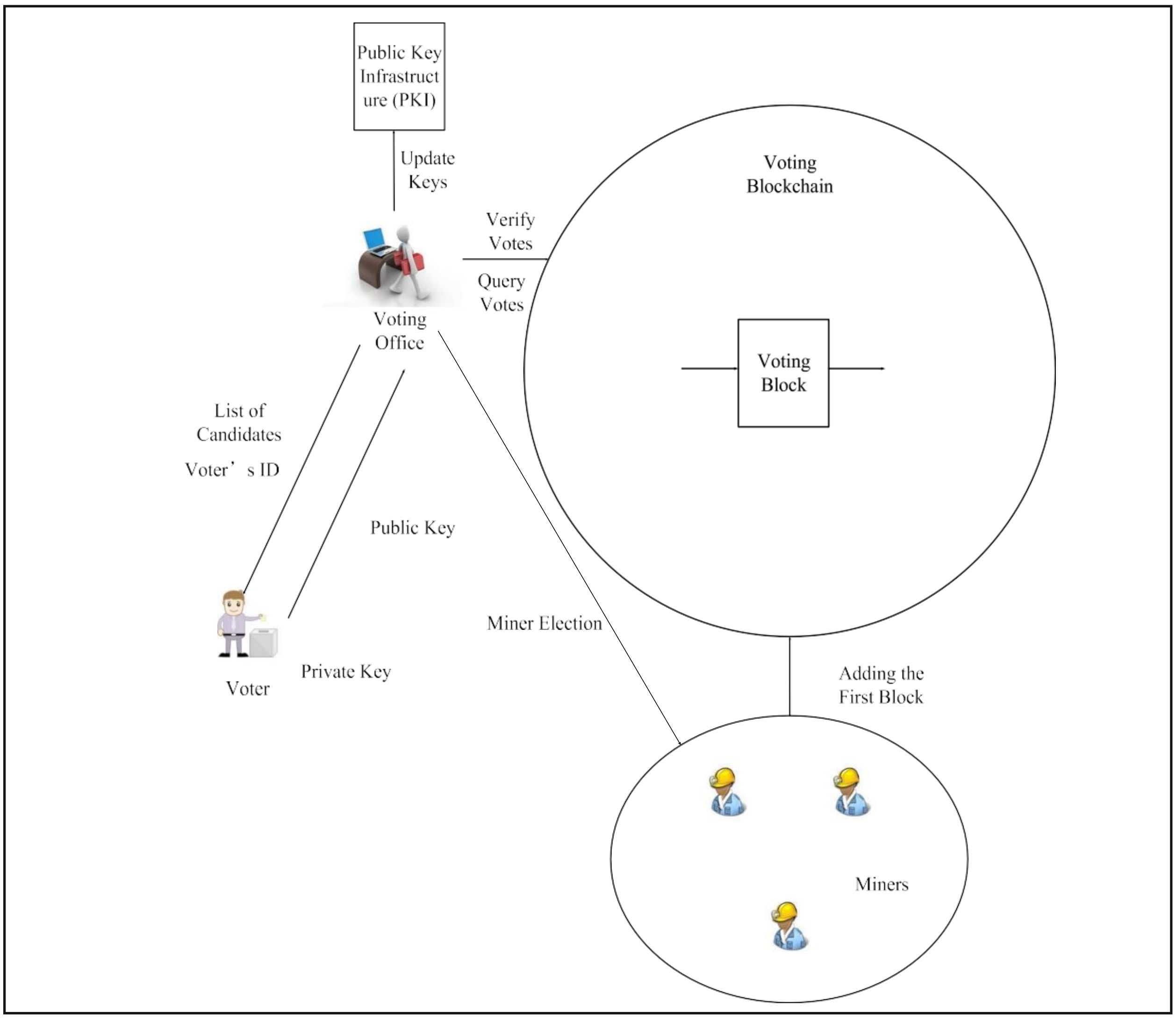
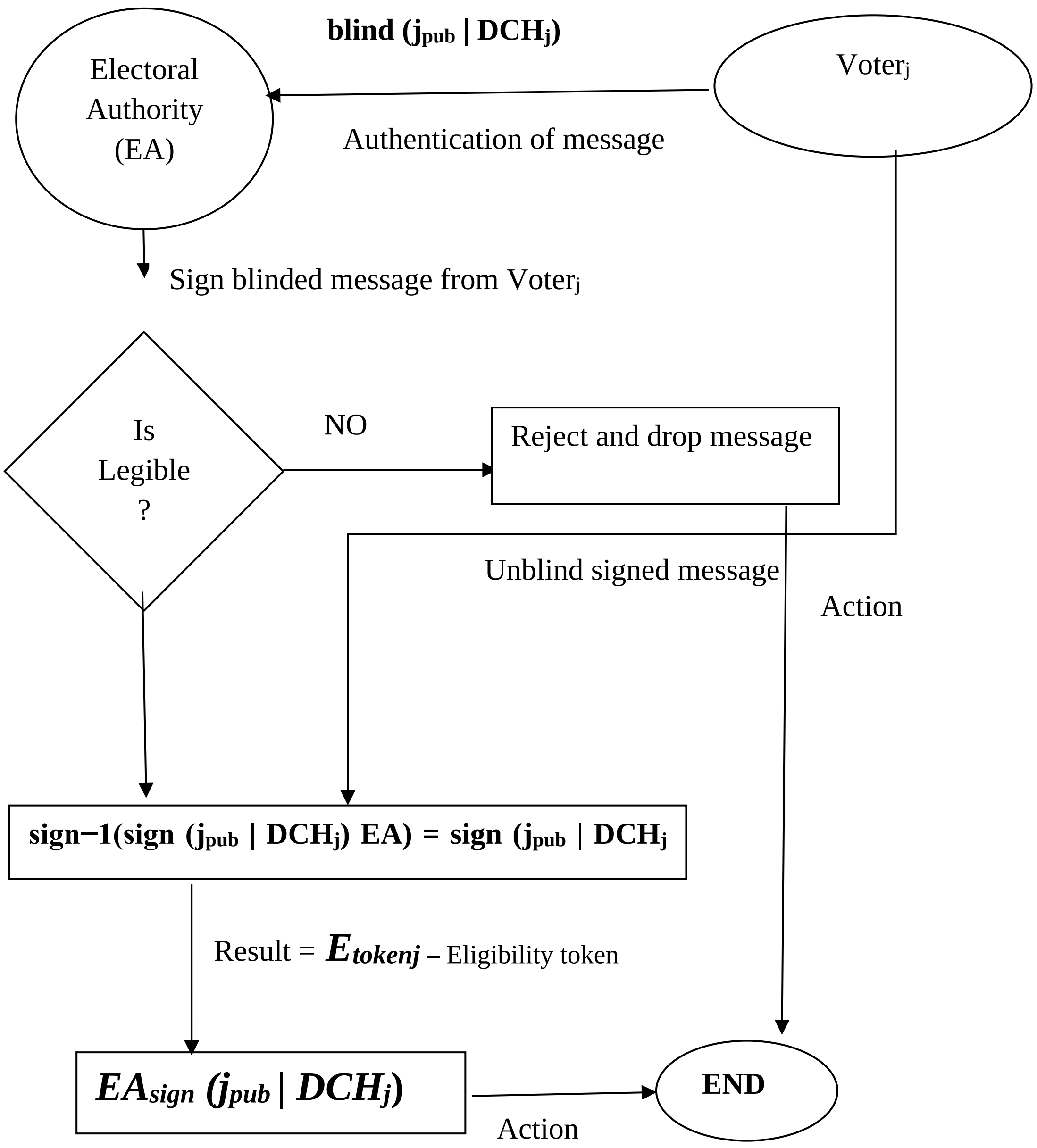


Figure 20: Initialization of the blockchain-based e-voting Protocol

During this phase, the voter, using the client-side application, is called to authenticate oneself to the electoral authority (EA). The EA will use the list of eligible voters along with the authentication information, it acquired during the initialization phase, to determine whether the aspiring voter is eligible to vote. If the voter is judged eligible, the EA will proceed to the following steps, otherwise voter (j) is rejected and the EA, does not proceed with the rest of the phase. All the following information will be exchanged through an authenticated and secure channel. financial transactions, over an unreliable channel. Once deemed eligible, the Voter (j) client will generate a public key pair, whose public counterpart j_{pub}, will be used as a pseudonymous identity of the voter and will also serve as one's verifying key. It will also prompt the voter to make one's choice, CH_j of the predetermined choices that will be accepted by the system and a digital commitment scheme will be used in order to generate DCH_j, the digital commitment of the voter's choice.

To prove one's eligibility, the voter needs to send both DCH_j and j_{pub} to the EA to be signed by it. To avoid the possibility of the EA linking a voter's true identity to one's vote the RSA blinding signature scheme is used at this level. The client will apply a blinding function on the message, blind (j_{pub} | DCH_j) and send it to the EA that will then sign the blinded message and send it back. Once the message is received, the client will unblind the signed message sign−1(sign (j_{pub} | DCH_j) EA) = sign (j_{pub} | DCH_j)EA and will end up with a valid eligibility token E_{tokenj}.

Data flow diagram describing the signing procedure and Eligibility token generation.


blind (j_{pub} | DCH_j)
Electoral Authority (EA)
Voter$_j$
Authentication of message
Sign blinded message from Voter$_j$
Is Legible ?
NO
Reject and drop message
Unblind signed message
Action
sign−1(sign (j_{pub} | DCH_j) EA) = sign (j_{pub} | DCH_j
Result = E_{tokenj} – Eligibility token
EA_{sign} (j_{pub} | DCH_j)
END
Action

Voting phase

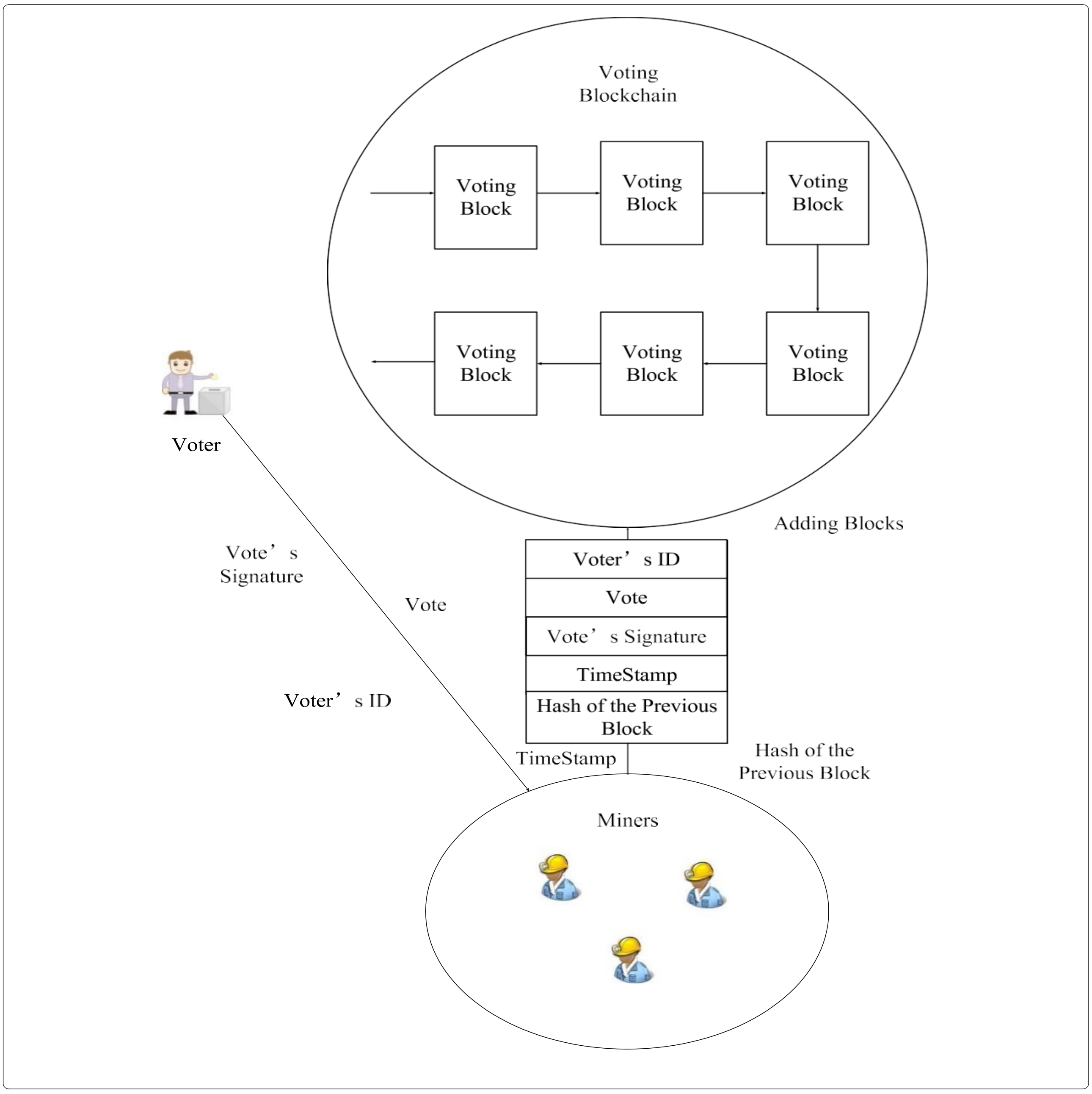


Figure 21: Voting process

- Voter uses SHA-256 to generate the hash value of H = Hash (ID + Vote + Timestamp).
- The voter uses his/her private key to generate a signature S of the hash value H.
- Voter sends ID, Vote, Timestamp, S to the miner.
- The miner obtains the public key from the PKI according to the voter's ID.
- The miner uses SHA-256 to generate the hash value of H =Hash (ID+Vote+Timestamp).
- The miner uses the public key to verify S *and* get H^{r}.
- The miner compares H and H^{r}. If H and H^{r} are the same, S is accepted. Otherwise, it is rejected.
- The miner queries and verifies that the voter has the right to vote.
- The miner generates a new block with the previous block's hash value and the information of vote and adds it to the blockchain.

During the voting phase, every Voter construct and then broadcasts to the network their vote. Each voter is also responsible for collecting votes, validating them and inserting the valid ones in the blockchain. In order for a voter to accept a vote as a valid one and include it in a block, one will make sure that the owner of the vote has not previously cast that vote. One will also have to make sure that EA's signature included in the ballot is validated and that the vote adheres to the predefined structure. If any of those checks fail the vote is discarded as an invalid one.

The counting phases

Counting phase: During the counting phase, all voters are called to reveal their final choice by broadcasting to the network a ballot opening message, OB_j, containing the V(id) of their final vote in the blockchain, the opening value of their vote commitment, and a signature over both values.

$OB_j = V(id_x)|ov_j|\ sign(V(id_x)|\ ov_j)_{EA}$

All nodes of the network are responsible for collecting the ballot opening messages and verifying that the signature validates with the public key of the owner of vote $V(id_x)$. If the signature is verified, the voters will then broadcast the messages to their adjacent peers. And proceed with including the vote in their count. All peers should reach the same result since they operate on the same blockchain.

Testing and evaluation of the blockchain based electronic voting protocol

In this section we examine the extent to which the protocol satisfies the e-voting design properties.

- Eligibility. For a vote to be considered cast for it to be accepted and written on the blockchain, it needs to include a valid ballot. Each ballot needs to include a valid signature of the EC over j_{pub} and DCH_j otherwise it is dropped by the network. The EC provides signatures only to authenticated voters that have been included in the list of eligible voters compiled during the initialization phase and that haven't requested to vote before. This means only eligible voters can vote and they can acquire only one eligibility token and thus cast only one valid vote. The eligibility property also breaks when an eligible voter succeeds in casting a vote more than once. This, however is not possible since all nodes, during the voting phase, will refuse to include to the blockchain a vote that has already been cast.

- Privacy. The protocol guarantees that no party can determine how a voter voted at any point during the protocol run. The only link between the real identity of the voter and one's vote is an individual's public key that acts as a pseudonymous identity. The only entity that would possibly expose said link would be the EC, since it is the only entity that the voter would need to reveal one's true identity to in order to obtain proof of one's eligibility, during the preparation stage. A blind signature is used to avoid any party from being able to verify how a voter voted. Blind signatures provide a way for the electoral commission (EC) to produce a valid signature on the digital commitment ($DCHj$) and public key of a voter without being able to determine neither the public key (j_{pub}) nor the digital commitment ($DCHj$).

- Individual verifiability: Due to the public nature of the ledger, each voter can verify that one's vote has been inserted in the blockchain, thus has been counted. Each voter is also responsible for counting the votes and thus one can ensure that the result includes one's vote.

- Universal verifiability: Since the ledger is public, every voter can verify that the votes have been counted correctly, by simply counting the votes. External auditors can also verify the results by obtaining a copy of the blockchain, making sure that the votes in it are legitimate, e.g. that signatures are validated, duplicates don't exist etc. and once all checks are complete, auditors can count the votes and compare their results against the official election tally. The fact that rules governing the election are included in the genesis block of the blockchain, further facilitates the election's verifiability since their integrity is guaranteed and thus disputes over them become irrelevant.

Since only voters are allowed to participate in the elections, the suggested blockchain to be used is a Permissioned one. The voters can use their eligibility tokens, as proof that they can participate in the blockchain. This makes the environment integrally more secure.

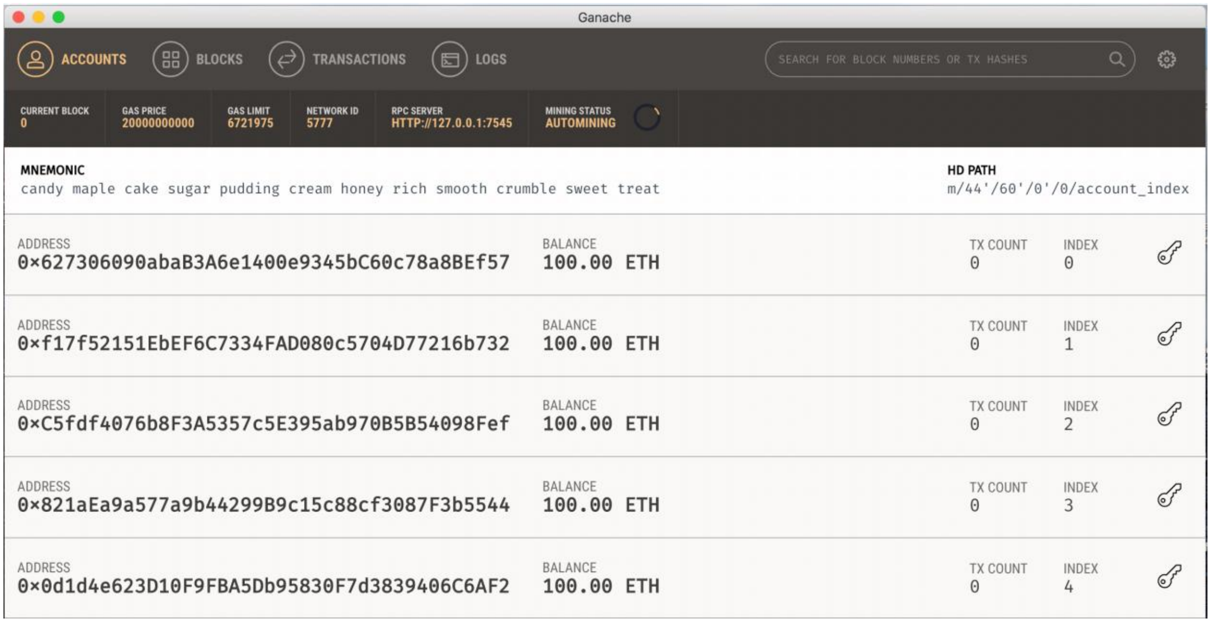
Prototype Implementation and testing

To implement a part of the protocol in a prototype for testing and evaluation, we used a few dependencies;

Node Package Manager (NPM). This dependence comes with Nodejs.

- Truffle Framework. This allowed us to build a decentralized demo on the Ethereum blockchain. It provided a suite of tools that allowed us to write smart contracts with the Solidity programming language. It also enabled us to test our smart contracts and deploy them to the blockchain.

- Ganache. This dependency provided us with addresses on our local Ethereum blockchain. Each account was preloaded with 100 fake ether for testing purposes. Below is a screenshot of the Ganache. Each account address serves as a unique identifier to each voter in this demo of our Election using the above protocol.



CURRENT BLOCK	GAS PRICE	GAS LIMIT	NETWORK ID	RPC SERVER	MINING STATUS
0	20000000000	6721975	5777	HTTP://127.0.0.1:7545	AUTOMINING

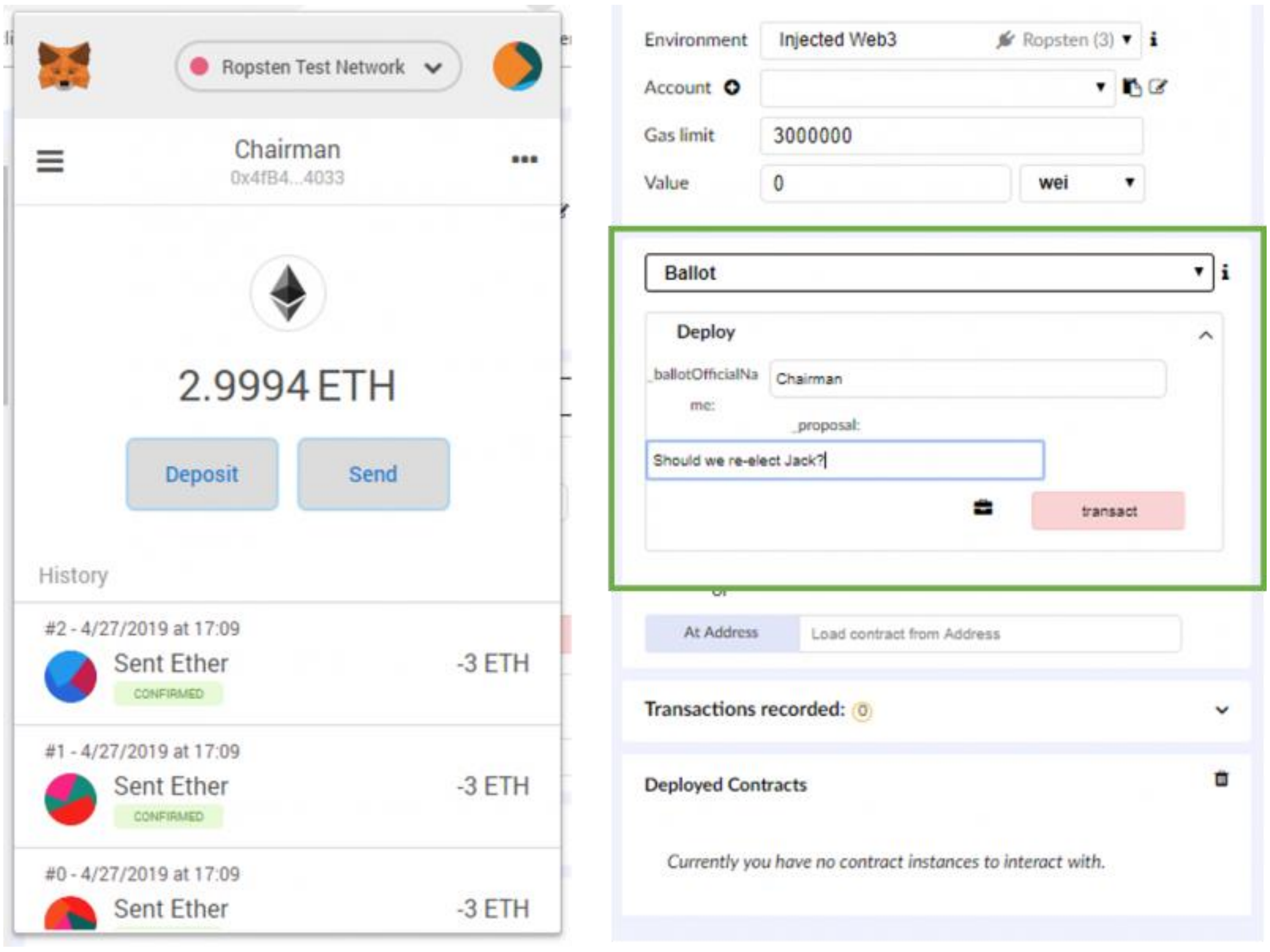
MNEMONIC
candy maple cake sugar pudding cream honey rich smooth crumble sweet treat

HD PATH
m/44'/60'/0'/0/account_index

ADDRESS	BALANCE	TX COUNT	INDEX
0×627306090abaB3A6e1400e9345bC60c78a8BEf57	100.00 ETH	0	0
0×f17f52151EbEF6C7334FAD080c5704D77216b732	100.00 ETH	0	1
0×C5fdf4076b8F3A5357c5E395ab970B5B54098Fef	100.00 ETH	0	2
0×821aEa9a577a9b44299B9c15c88cf3087F3b5544	100.00 ETH	0	3
0×0d1d4e623D10F9FBA5Db95830F7d3839406C6AF2	100.00 ETH	0	4

Figure 13: Ganache Screenshot preloaded with 100 ether for testing purposes.

- **Metamask**. To use the blockchain, we required to connect to it since blockchain is a network as we learned from our study. Therefore, we installed a special browser extension to use the Ethereum blockchain. Using the Metamask, we connected to our local Ethereum blockchain with our account and interacted with our smart contract.



Ropsten Test Network

Chairman
0x4fB4...4033

2.9994 ETH

Deposit | Send

History

#2 - 4/27/2019 at 17:09
Sent Ether -3 ETH
CONFIRMED

#1 - 4/27/2019 at 17:09
Sent Ether -3 ETH
CONFIRMED

#0 - 4/27/2019 at 17:09
Sent Ether -3 ETH

Environment: Injected Web3 — Ropsten (3)
Account:
Gas limit: 3000000
Value: 0 wei

Ballot

Deploy
_ballotOfficialName: Chairman
_proposal: Should we re-elect Jack?
transact

At Address — Load contract from Address

Transactions recorded: 0

Deployed Contracts

Currently you have no contract instances to interact with.

Figure 14: Metamask screenshot

Off-chain computation using Ethereum signature

Signing and verifying messages

Signing a message with a private key does not require interacting with the Ethereum network. It can be done completely offline. In this section we demonstrate the off-chain computation of messages signed and then verified on chain using a smart contract. The message signing procedure involves two parties, a user that wants to obtain signatures on her messages, and a signer that owns his secret signing key. At the end of the process, the user obtains the signer's signature on message (m) without the signer learning anything about the message. The signature algorithm Ethereum has built-in support for is the Elliptic Curve Digital Signature Algorithm (EDCSA).

In our protocol, the sign method calculates an Ethereum specific signature with:

sign (keccak256("\x19Ethereum Signed Message:\n" + len(message) + message))).

By adding a prefix to the message makes the calculated signature recognizable as an Ethereum specific signature. This prevents misuse where a malicious decentralized application can sign arbitrary data (e.g., transaction) and use the signature to impersonate the victim.

Recovering the Message Signer in Solidity. Here we reviewed the ECDSA signature which consists of two parameters, ***r*** and ***s***. Signatures in Ethereum include a third parameter, ***v***, which provides additional information that can be used to recover which account's private key was used to sign the message. This same mechanism is how Ethereum determines which account sent a given transaction.

Solidity provides a built-in function ecrecover() that accepts a message along with the *r*, *s*, and *v* parameters and returns the address that was used to sign the message.

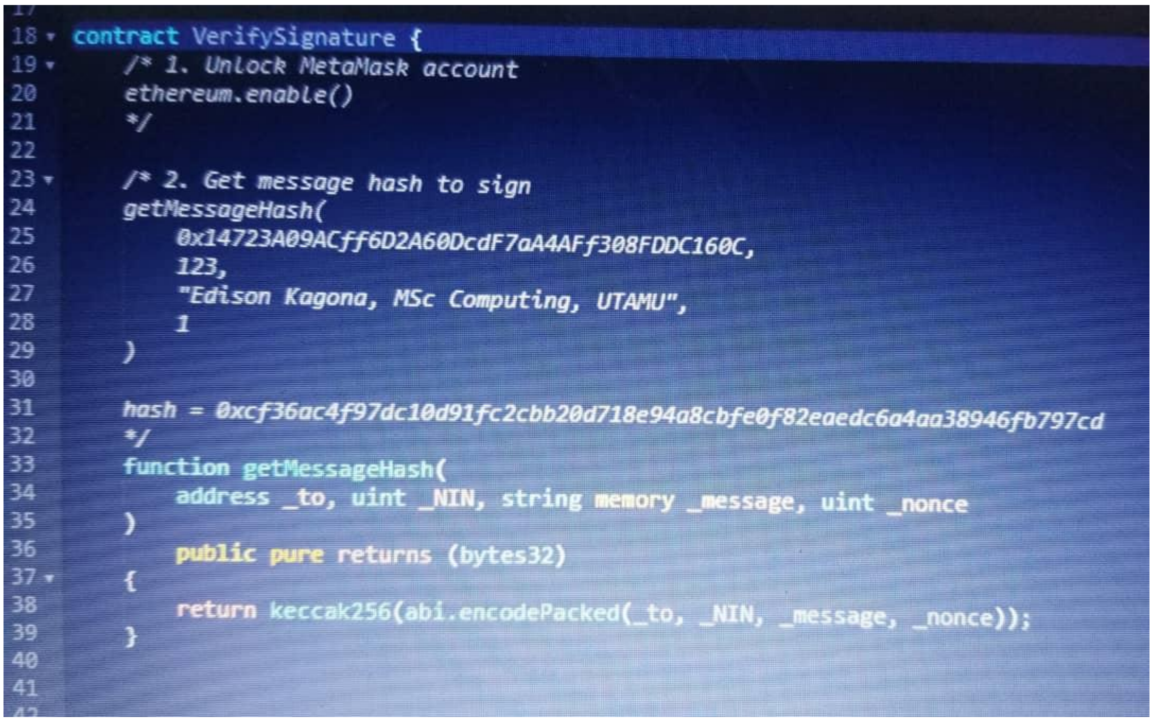
Verifying Signature

Steps taken in the signing and verification of the message.

First, we created a message to be signed

- We hash the message
- We sign the hash (off chain, keep your private key secret)
- Recreate the hash from the original message
- Recover signer from signature and hash
- Compare recovered signer to claimed signer

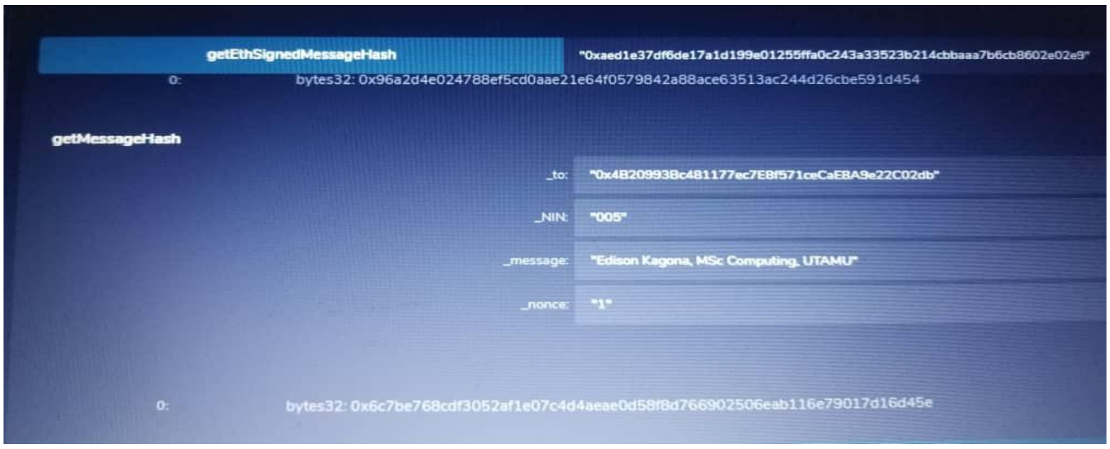
In the figures below, we demonstrate the message creation, hashing of the message, signing of the hash off-chain and verification process.



```
contract VerifySignature {
    /* 1. Unlock MetaMask account
    ethereum.enable()
    */

    /* 2. Get message hash to sign
    getMessageHash(
        0x14723A09ACff6D2A60DcdF7aA4AFf308FDDC160C,
        123,
        "Edison Kagona, MSc Computing, UTAMU",
        1
    )

    hash = 0xcf36ac4f97dc10d91fc2cbb20d718e94a8cbfe0f82eaedc6a4aa38946fb797cd
    */
    function getMessageHash(
        address _to, uint _NIN, string memory _message, uint _nonce
    )
        public pure returns (bytes32)
    {
        return keccak256(abi.encodePacked(_to, _NIN, _message, _nonce));
    }
```

Figure 15: Contract VerifySignature

In the above screenshot, we implemented a message with a receiver public key, the _NIN representing an ID of the sender, Message body of the sender and the _nonce. The _nonce is a number or bit string used only once.



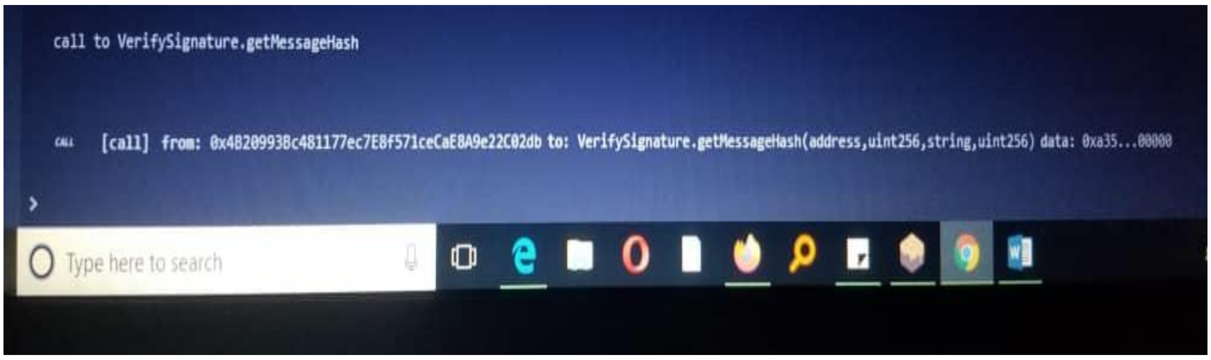
```
getEthSignedMessageHash    "0xaed1e37df6de17a1d199e01255ffa0c243a33523b214cbbaa7b6cb8602e02e9"
    0:    bytes32: 0x96a2d4e024788ef5cd0aae21e64f0579842a88ace63513ac244d26cbe591d454

getMessageHash
                  _to:  "0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db"
                 _NIN:  "005"
             _message:  "Edison Kagona, MSc Computing, UTAMU"
               _nonce:  "1"

    0:    bytes32: 0x6c7be768cdf3052af1e07c4d4aeae0d58f8d766902506eab116e79017d16d45e
```

Figure 16: getMessageHash function and message structure

The message to be signed in the above figure 25 of the message structure, the getMessageHash function represents the message to be signed. We obtained a hash of the message

bytes32: 0x6c7be768cdf3052af1e07c4d4aeae0d58f8d766902506eab116e79017d16d45e after encryption of the entire message structure.

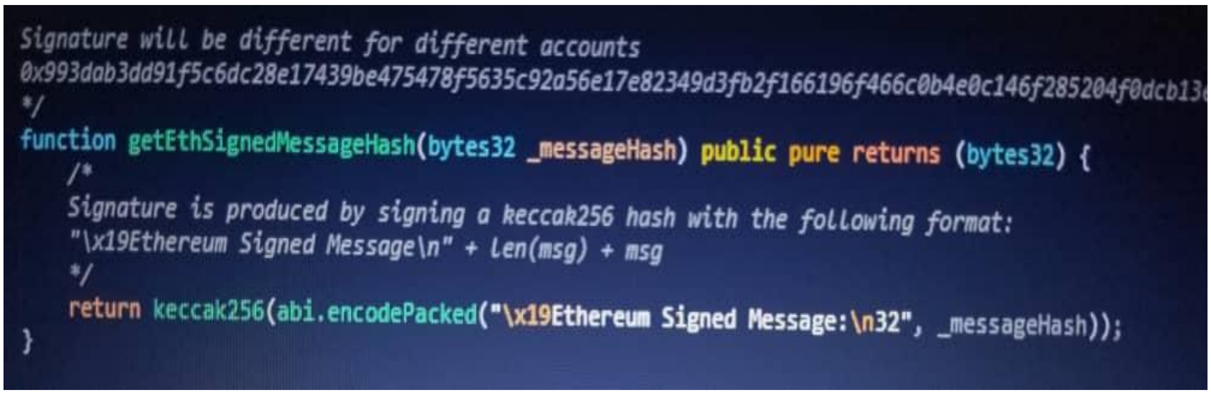
getEthSignedMessageHash function shows the _messageHash generated in the above figure to be signed by the Signer. After the signing process, we obtained a signed message bytes32: 0x88bfa4b1ff6af3465b71e59a3ed9b1e89a68c2f10cfc179863896c3f31f8134c of the above hash.



```
call to VerifySignature.getMessageHash

CALL   [call] from: 0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db to: VerifySignature.getMessageHash(address,uint256,string,uint256) data: 0xa35...00000

>
```

Figure 17: Signature verification call

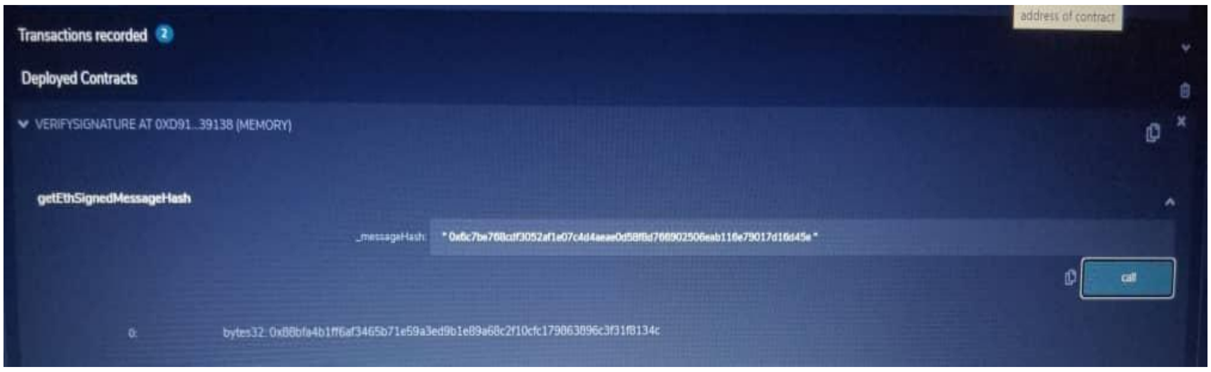
The above figure 26 describes the signature verification call. We ran this call to verifySignature.getMessageHash function. The signature of the messages of the different accounts that represent the voters will be different.



```
Signature will be different for different accounts
0x993dab3dd91f5c6dc28e17439be475478f5635c92a56e17e82349d3fb2f166196f466c0b4e0c146f285204f0dcb13
*/
function getEthSignedMessageHash(bytes32 _messageHash) public pure returns (bytes32) {
    /*
    Signature is produced by signing a keccak256 hash with the following format:
    "\x19Ethereum Signed Message\n" + len(msg) + msg
    */
    return keccak256(abi.encodePacked("\x19Ethereum Signed Message:\n32", _messageHash));
}
```

Figure 18: Function to get the signed message hash

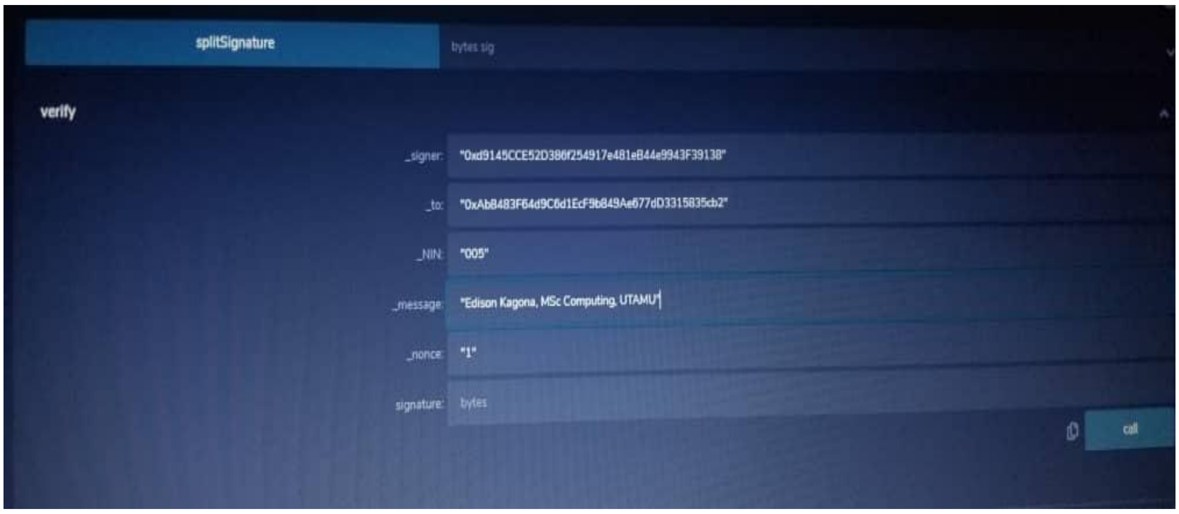
The result of the above function in figure 27 responsible for signing the message hash is shown below in figure 28 as bytes32: 0x88bfa4b1ff6af3465b71e59a3ed9b1e89a68c2f10cfc179863896c3f31f8134c. The result is the signed message hash.



Transactions recorded 2
Deployed Contracts
VERIFYSIGNATURE AT 0XD91...39138 (MEMORY)
getEthSignedMessageHash
_messageHash: "0x6c7be768cdf3052af1e07c4d4aeae0d58f8d766902506eab116e79017d16d45e"
call
0: bytes32: 0x88bfa4b1ff6af3465b71e59a3ed9b1e89a68c2f10cfc179863896c3f31f8134c

Figure 19: getEthSignedMessageHash - signed message hash

The above signed message hash is written to the blockchain and later published on the entire network chain with the message owner public key. The signed message can be verified as shown in the figure below before it can be added to the blockchain network.



splitSignature bytes sig
verify
_signer: "0xd9145CCE52D386f254917e481eB44e9943F39138"
_to: "0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2"
_NIN: "005"
_message: "Edison Kagona, MSc Computing, UTAMU"
_nonce: "1"
signature: bytes
call

Figure 20: Signature verification

First 32 bytes stores the length of the signature

add (sig, 32) = pointer of sig + 32

effectively, skips first 32 bytes of signature

mload(p) loads next 32 bytes starting at the memory address p into memory

// first 32 bytes, after the length prefix

r := mload(add(sig, 32))

// second 32 bytes

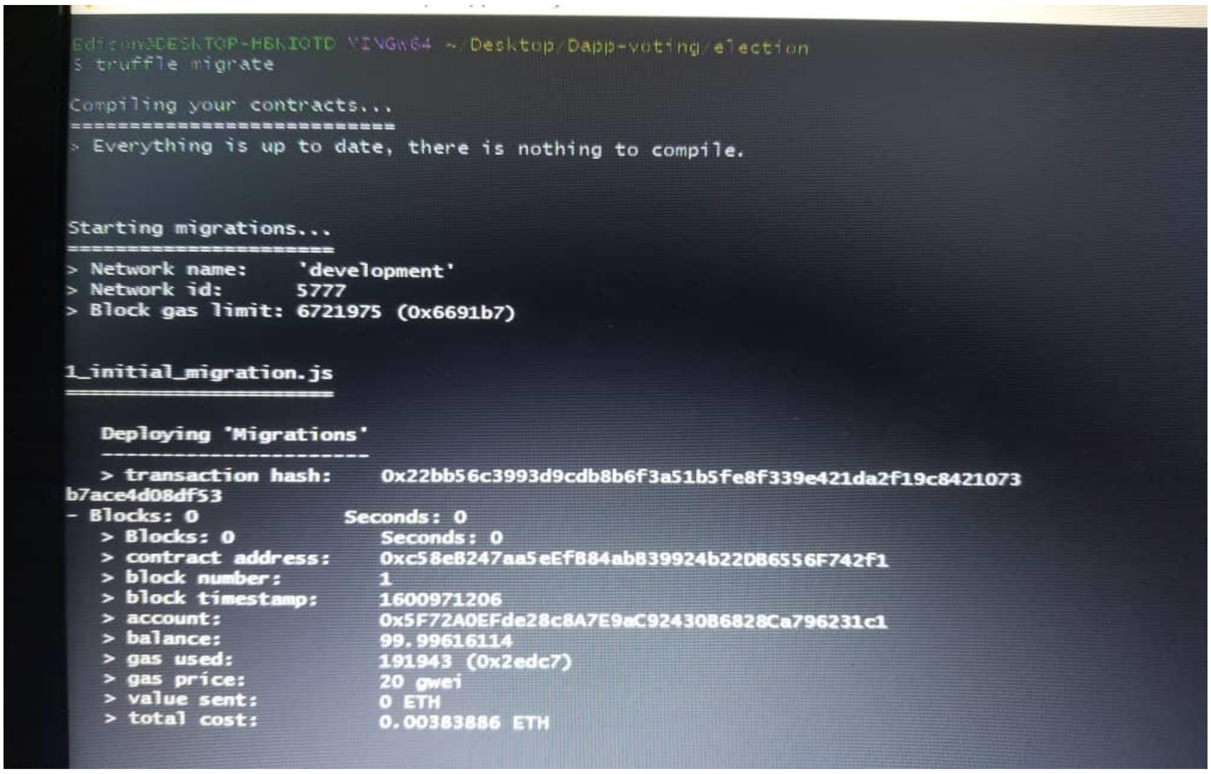
s := mload(add(sig, 64))

// final byte (first byte of the next 32 bytes)

v := byte(0, mload(add(sig, 96)))

// implicitly return (r, s, v)

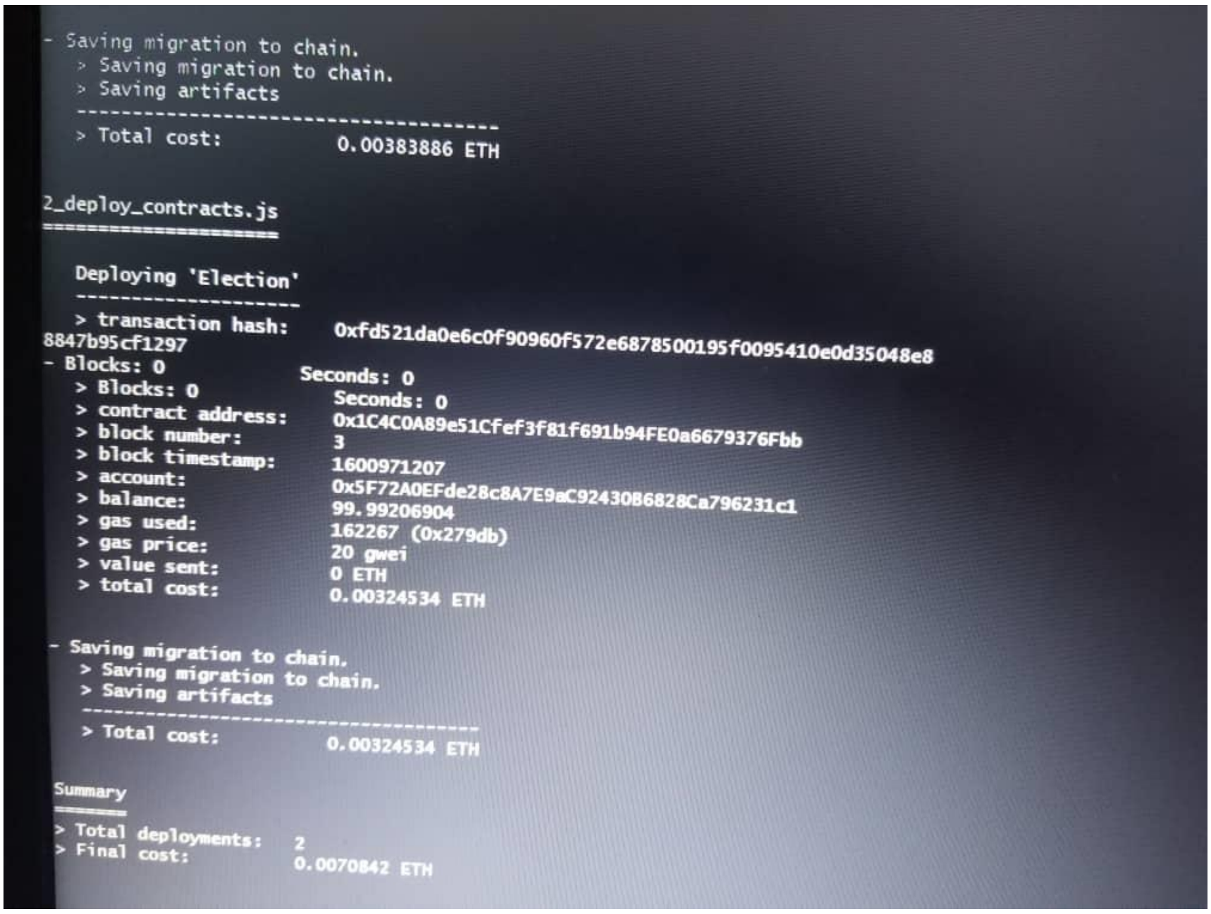
Smoke Tests



```
Edison@DESKTOP-HBKIOTD MINGW64 ~/Desktop/Dapp-voting/election
$ truffle migrate

Compiling your contracts...
===========================
> Everything is up to date, there is nothing to compile.


Starting migrations...
======================
> Network name:    'development'
> Network id:      5777
> Block gas limit: 6721975 (0x6691b7)


1_initial_migration.js
======================

   Deploying 'Migrations'
   ----------------------
   > transaction hash:    0x22bb56c3993d9cdb8b6f3a51b5fe8f339e421da2f19c8421073
b7ace4d08df53
- Blocks: 0            Seconds: 0
   > Blocks: 0            Seconds: 0
   > contract address:    0xc58e8247aa5eEfB84abB39924b22DB6556F742f1
   > block number:        1
   > block timestamp:     1600971206
   > account:             0x5F72A0EFde28c8A7E9aC92430B6828Ca796231c1
   > balance:             99.99616114
   > gas used:            191943 (0x2edc7)
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.00383886 ETH
```

Figure 21: Running Migrations

In this first smoke test after running migrations of the implemented contract called "Election", we notice an automatic generation of a "Network name, Network id, Block gas limit" as implemented and set up within the development environment. Deploying "Migrations," we notice the generation of "transaction hash, blocks, contract address, block number, block timestamp" as stated in the implementation of the smart contract.



```
- Saving migration to chain.
   > Saving migration to chain.
   > Saving artifacts
   -------------------------------------
   > Total cost:          0.00383886 ETH


2_deploy_contracts.js
=====================

   Deploying 'Election'
   --------------------
   > transaction hash:    0xfd521da0e6c0f90960f572e6878500195f0095410e0d35048e8
8847b95cf1297
- Blocks: 0            Seconds: 0
   > Blocks: 0            Seconds: 0
   > contract address:    0x1C4C0A89e51Cfef3f81f691b94FE0a6679376Fbb
   > block number:        3
   > block timestamp:     1600971207
   > account:             0x5F72A0EFde28c8A7E9aC92430B6828Ca796231c1
   > balance:             99.99206904
   > gas used:            162267 (0x279db)
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.00324534 ETH


- Saving migration to chain.
   > Saving migration to chain.
   > Saving artifacts
   -------------------------------------
   > Total cost:          0.00324534 ETH


Summary
=======
> Total deployments:   2
> Final cost:          0.0070842 ETH
```

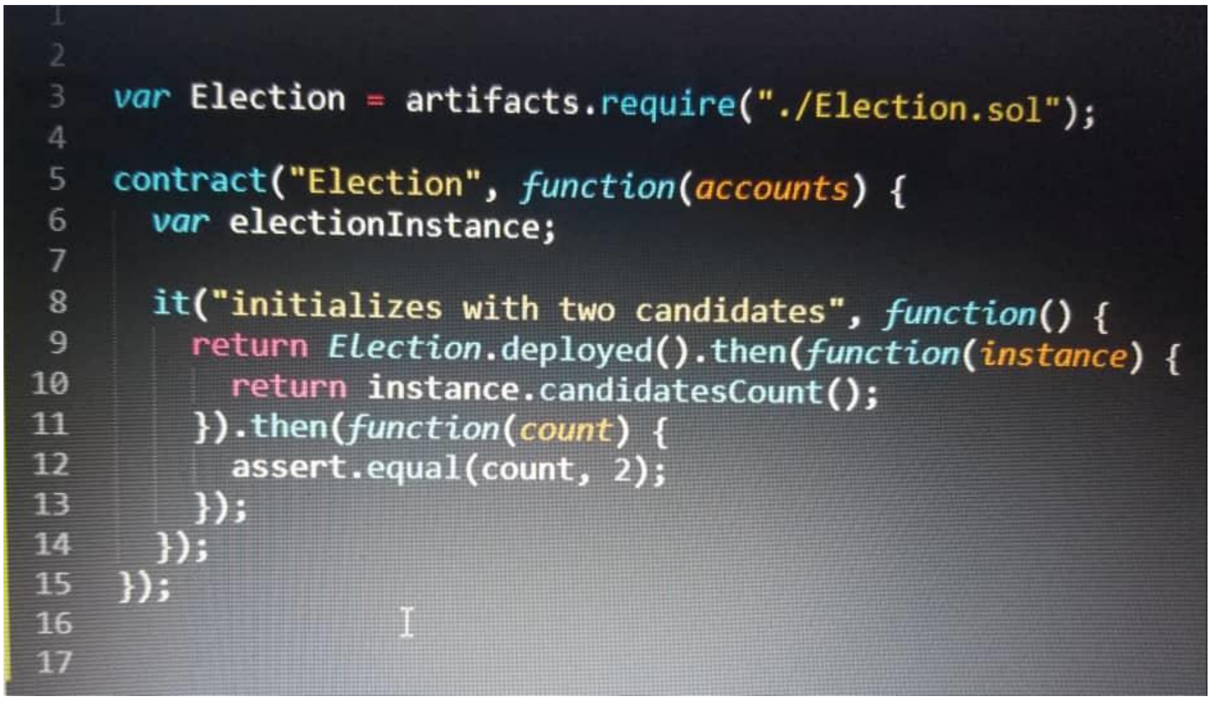
Figure 22: Deploying 'Election' Contract to the blockchain

The above screenshot shows the deploying of contracts (Election Contract), and the Saving of all migrations to the chain, as well as a generated summary of deployments.

Tests of the implemented prototype

Tests were written in JavaScript with a Mocha testing framework and the Chai assertion library. These come bundled with the Truffle framework. We wrote the tests in JavaScript to simulate client-side interaction with our smart contract.

British Journal of Computer, Networking and Information Technology
ISSN: 2689-5315
Volume 5, Issue 1, 2022 (pp. 56-115)
www.abjournals.org

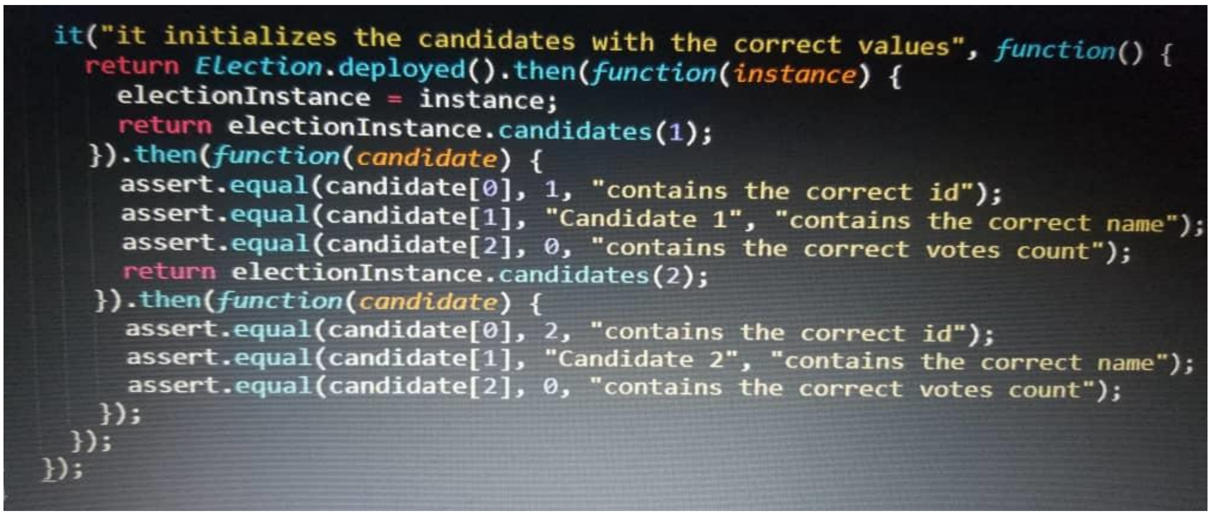


```
var Election = artifacts.require("./Election.sol");

contract("Election", function(accounts) {
  var electionInstance;

  it("initializes with two candidates", function() {
    return Election.deployed().then(function(instance) {
      return instance.candidatesCount();
    }).then(function(count) {
      assert.equal(count, 2);
    });
  });
});
```

Figure 23: Tests to simulate client-side interaction with our smart contract

In the above test, we first required the contract and assigned it to a variable. Next, we called the “contract” function and wrote all our tests within the callback function. This callback function provided an “accounts” variable that represented all the accounts on our blockchain, provided by Ganache. In the above screenshot, we check that the contract was initialized with the correct number of candidates by checking the candidate's count is equal to 2.

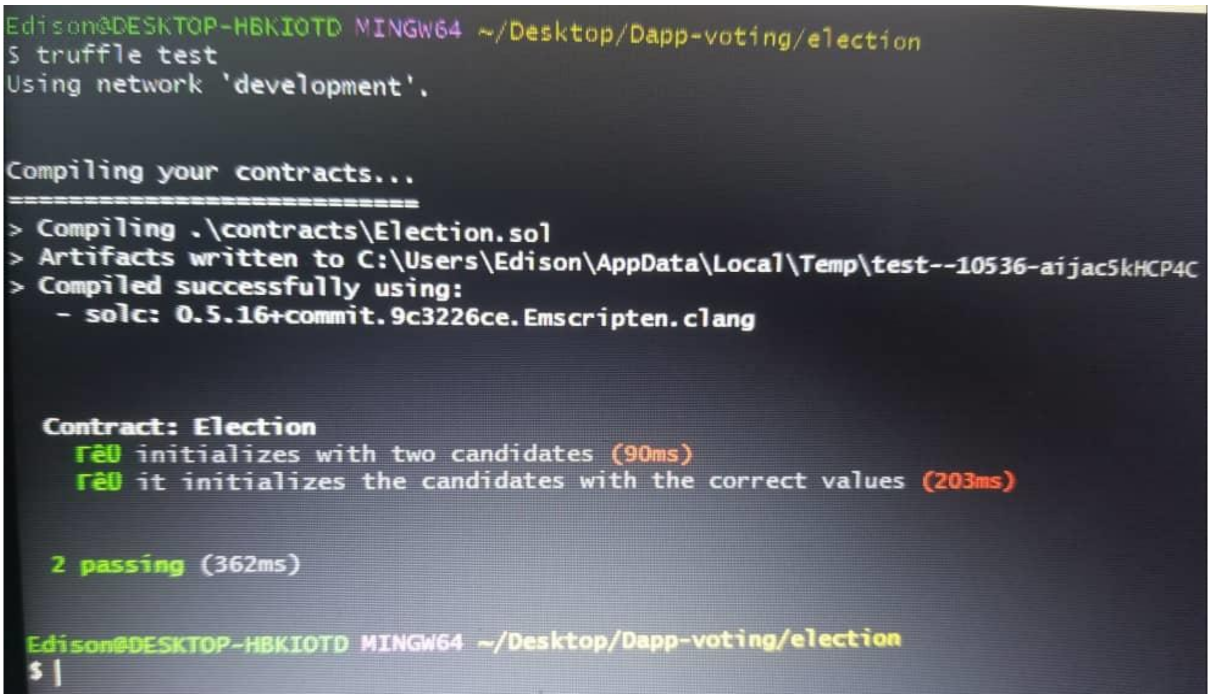
The next test screenshot below, inspects the values of each candidate in the election, ensuring that each candidate has the correct id, name, and vote count.



```
it("it initializes the candidates with the correct values", function() {
  return Election.deployed().then(function(instance) {
    electionInstance = instance;
    return electionInstance.candidates(1);
  }).then(function(candidate) {
    assert.equal(candidate[0], 1, "contains the correct id");
    assert.equal(candidate[1], "Candidate 1", "contains the correct name");
    assert.equal(candidate[2], 0, "contains the correct votes count");
    return electionInstance.candidates(2);
  }).then(function(candidate) {
    assert.equal(candidate[0], 2, "contains the correct id");
    assert.equal(candidate[1], "Candidate 2", "contains the correct name");
    assert.equal(candidate[2], 0, "contains the correct votes count");
  });
 });
});
```

Figure 24: Tests to inspect the values of each candidate in the election

Screenshot of Test results

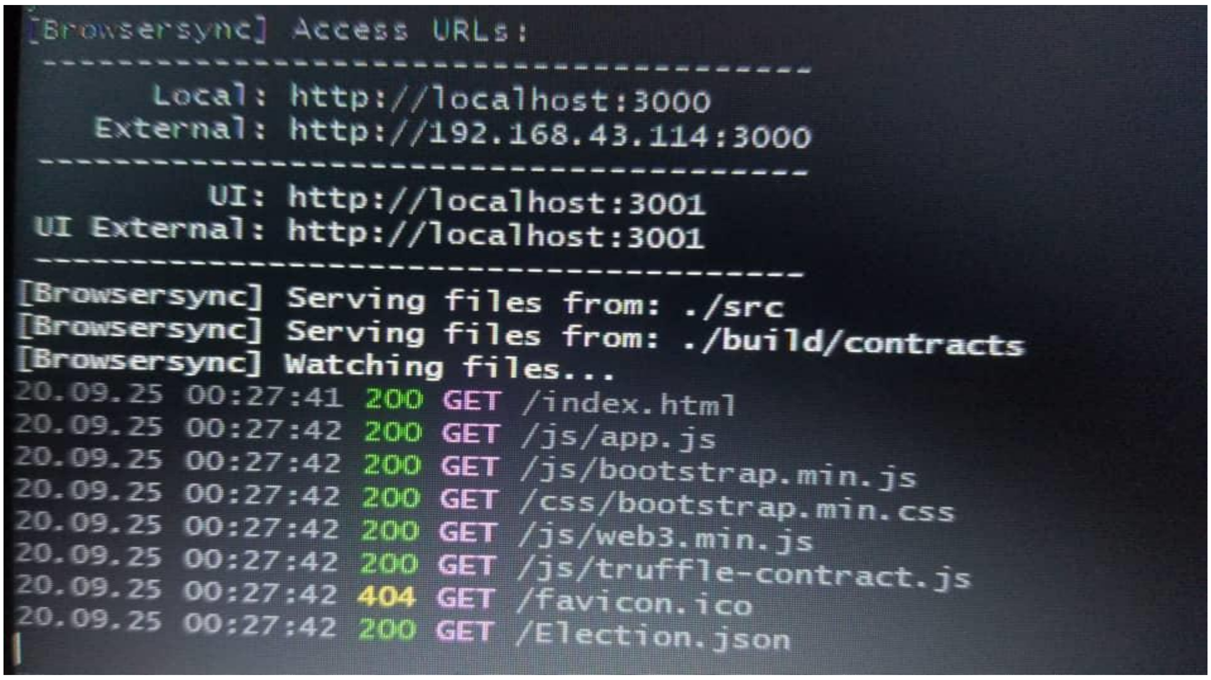


```
Edison@DESKTOP-HBKIOTD MINGW64 ~/Desktop/Dapp-voting/election
$ truffle test
Using network 'development'.


Compiling your contracts...
===========================
> Compiling .\contracts\Election.sol
> Artifacts written to C:\Users\Edison\AppData\Local\Temp\test--10536-aijac5kHCP4C
> Compiled successfully using:
   - solc: 0.5.16+commit.9c3226ce.Emscripten.clang



  Contract: Election
    rêü initializes with two candidates (90ms)
    rêü it initializes the candidates with the correct values (203ms)


  2 passing (362ms)


Edison@DESKTOP-HBKIOTD MINGW64 ~/Desktop/Dapp-voting/election
$ |
```

Figure 25: Test results

Running the Lite-server

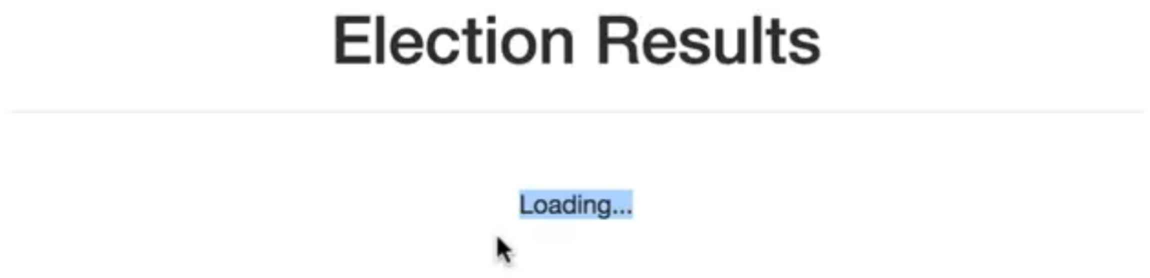
This test prototype can be run on the URLs below;



```
[Browsersync] Access URLs:
 --------------------------------------
       Local: http://localhost:3000
    External: http://192.168.43.114:3000
 --------------------------------------
          UI: http://localhost:3001
 UI External: http://localhost:3001
 --------------------------------------
[Browsersync] Serving files from: ./src
[Browsersync] Serving files from: ./build/contracts
[Browsersync] Watching files...
20.09.25 00:27:41 200 GET /index.html
20.09.25 00:27:42 200 GET /js/app.js
20.09.25 00:27:42 200 GET /js/bootstrap.min.js
20.09.25 00:27:42 200 GET /css/bootstrap.min.css
20.09.25 00:27:42 200 GET /js/web3.min.js
20.09.25 00:27:42 200 GET /js/truffle-contract.js
20.09.25 00:27:42 404 GET /favicon.ico
20.09.25 00:27:42 200 GET /Election.json
|
```

Figure 26: Running the Lite-Server

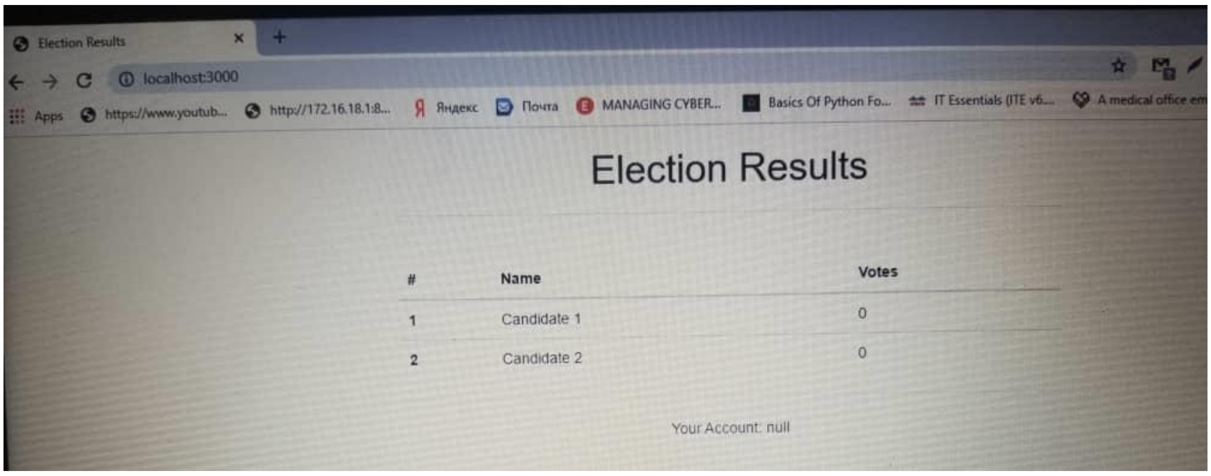
After starting the Lite-server on localhost:3000, a new browser window will be displayed with the client-side application. The client-side application displays “Loading …” meaning that the “Lite-server” is connected to the client-side application but not yet connected to the blockchain.


Election Results

Loading...

Figure 27: Lite-server connection to the Client-side application

After connecting to the blockchain, having imported one of the accounts from Ganache into Metamask, all of the contract and account data is loaded. If an account is not connected as yet, then the application will display “null”, meaning that the client-side application is running and connected to the Lite-server but no account is connected to it from the local blockchain.



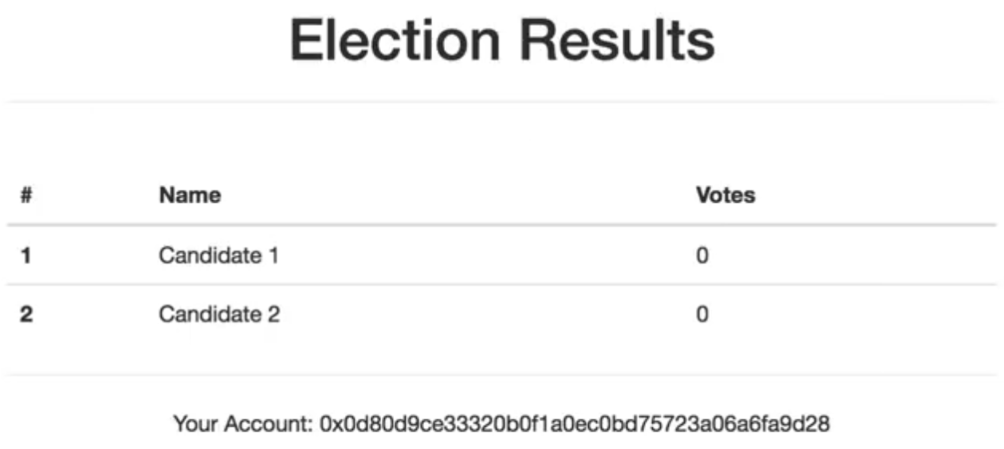
Election Results

#	Name	Votes
1	Candidate 1	0
2	Candidate 2	0

Your Account: null

Figure 28: Null account connected

After connecting an account from Ganache, the local blockchain. The screenshot below shows a connected account to the client-side application.



Election Results

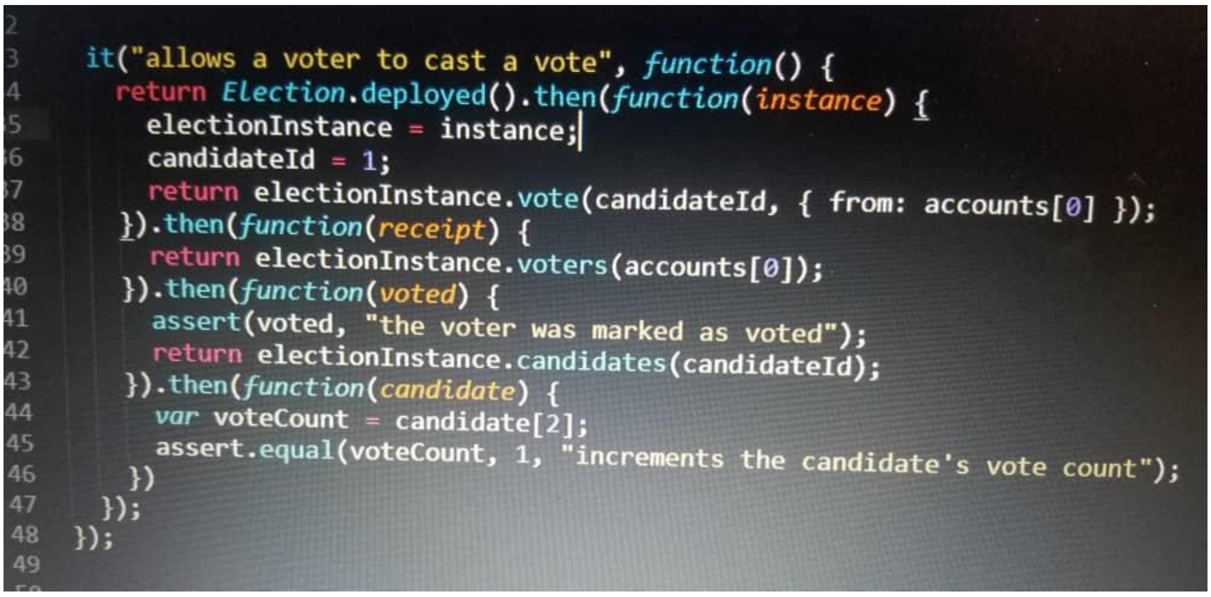
#	Name	Votes
1	Candidate 1	0
2	Candidate 2	0

Your Account: 0x0d80d9ce33320b0f1a0ec0bd75723a06a6fa9d28

Figure 29: An account connected successfully to the client-side application

Cast Votes

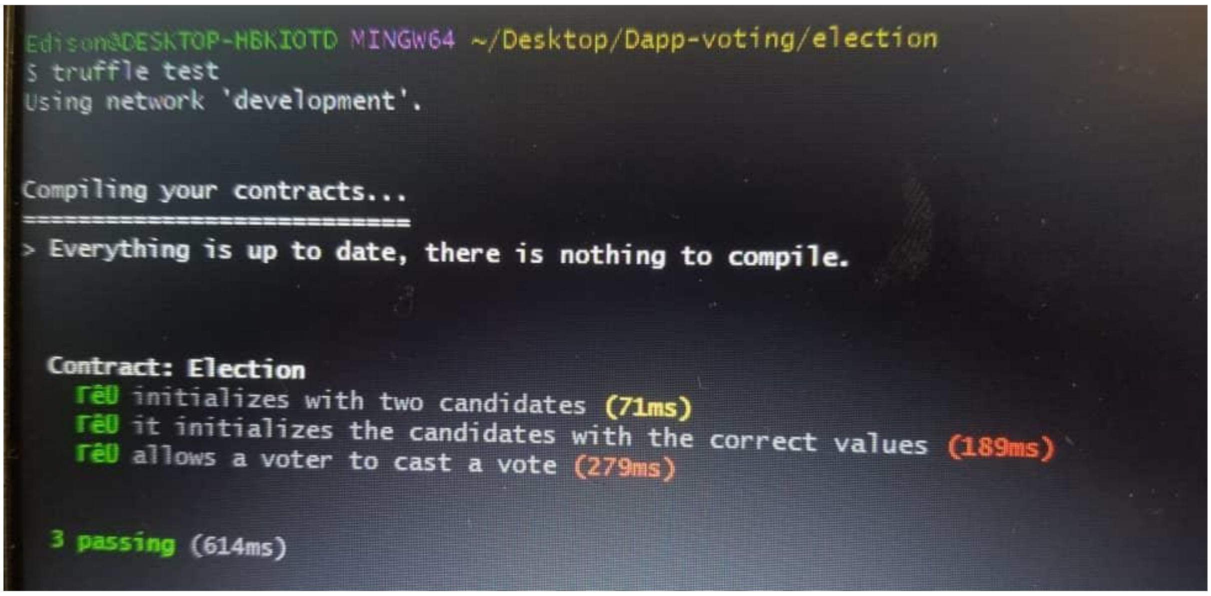
At this level, we added a way to cast votes in the election. We defined a "voters" mapping to the smart contract to keep track of the accounts that have voted in the election.



```
  it("allows a voter to cast a vote", function() {
    return Election.deployed().then(function(instance) {
      electionInstance = instance;
      candidateId = 1;
      return electionInstance.vote(candidateId, { from: accounts[0] });
    }).then(function(receipt) {
      return electionInstance.voters(accounts[0]);
    }).then(function(voted) {
      assert(voted, "the voter was marked as voted");
      return electionInstance.candidates(candidateId);
    }).then(function(candidate) {
      var voteCount = candidate[2];
      assert.equal(voteCount, 1, "increments the candidate's vote count");
    })
  });
});
```

Figure 30: Testing the voting function

In the above test function, we tested 2 things. We tested that the function increments the vote count for the candidate and that the voter is added to the mapping whenever they vote. The screenshot below shows the test function results.

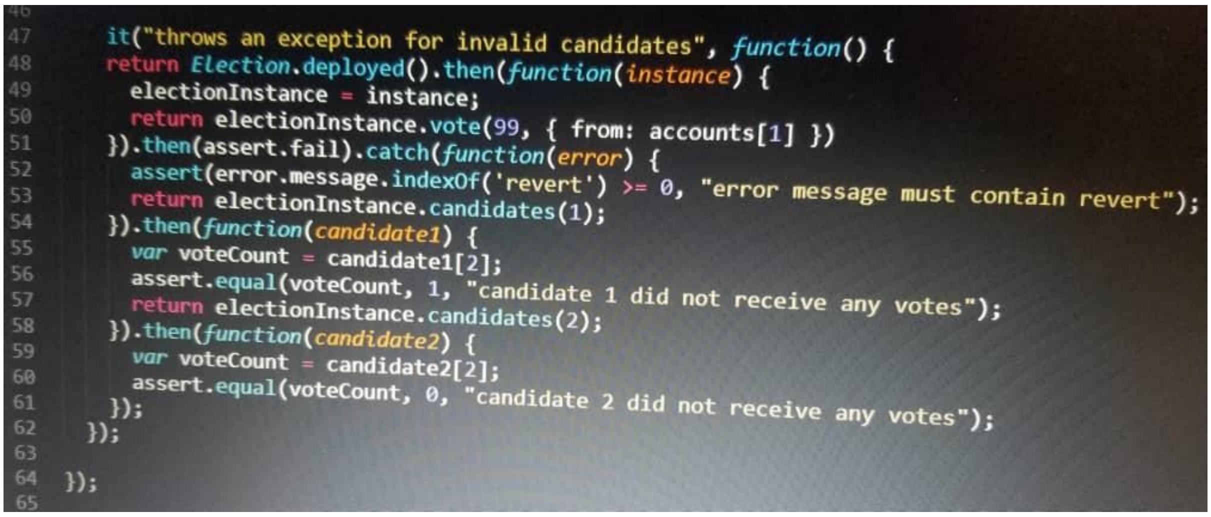


```
Edison@DESKTOP-HBKIOTD MINGW64 ~/Desktop/Dapp-voting/election
$ truffle test
Using network 'development'.


Compiling your contracts...
===========================
> Everything is up to date, there is nothing to compile.



  Contract: Election
    rê0 initializes with two candidates (71ms)
    rê0 it initializes the candidates with the correct values (189ms)
    rê0 allows a voter to cast a vote (279ms)


  3 passing (614ms)
```

Figure 31: Test function results

Function's requirements tests

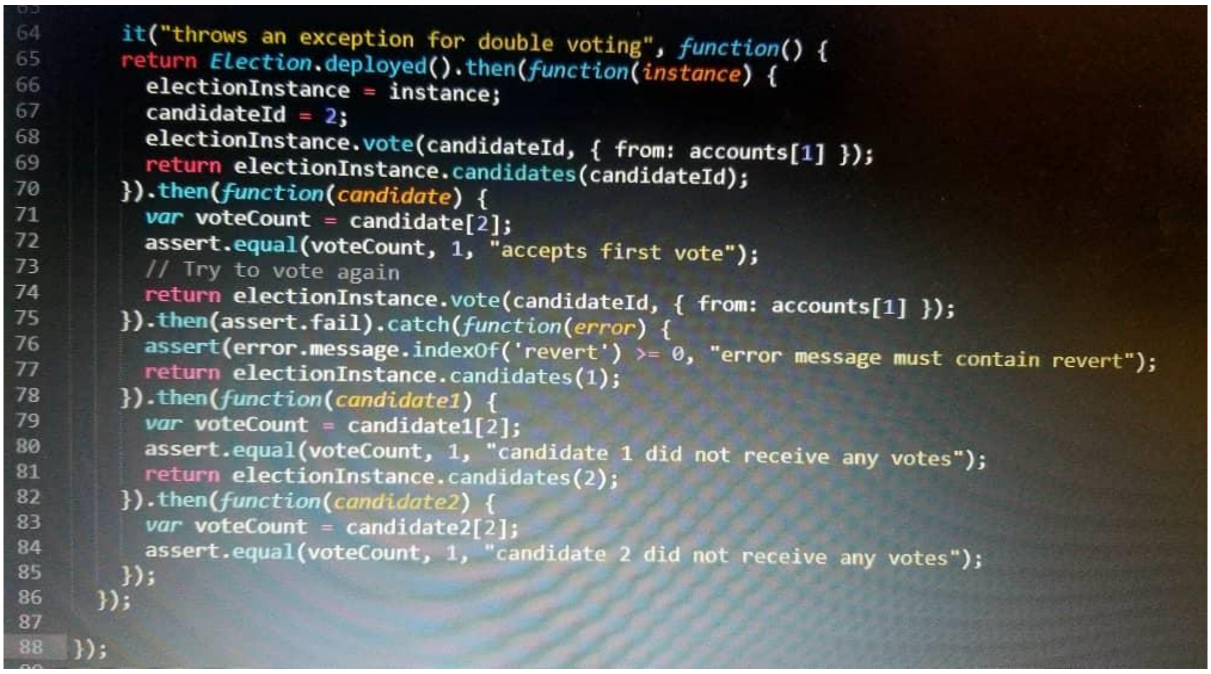
These tests ensured that our voter function throws an exception for double voting. We asserted that if the transaction fails, an error message is returned. We ensured that the error message contains the "revert" substring. Ensuring that our contract's state was unaltered, we ensured that the candidate did not receive any votes. The screenshot below describes the exceptions;



```
    it("throws an exception for invalid candidates", function() {
    return Election.deployed().then(function(instance) {
      electionInstance = instance;
      return electionInstance.vote(99, { from: accounts[1] })
    }).then(assert.fail).catch(function(error) {
      assert(error.message.indexOf('revert') >= 0, "error message must contain revert");
      return electionInstance.candidates(1);
    }).then(function(candidate1) {
      var voteCount = candidate1[2];
      assert.equal(voteCount, 1, "candidate 1 did not receive any votes");
      return electionInstance.candidates(2);
    }).then(function(candidate2) {
      var voteCount = candidate2[2];
      assert.equal(voteCount, 0, "candidate 2 did not receive any votes");
    });
  });

});
```

Figure 32:Exception handling for valid candidates

Tests to prevent double voting. The screenshot below describes the stated exceptions;

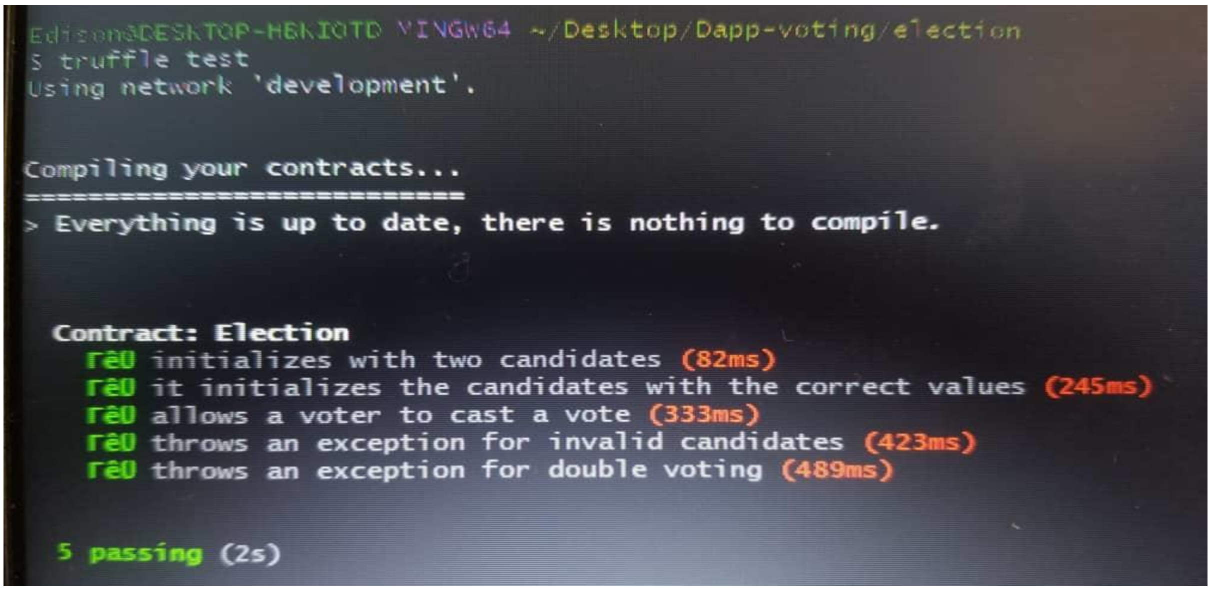


```
    it("throws an exception for double voting", function() {
    return Election.deployed().then(function(instance) {
      electionInstance = instance;
      candidateId = 2;
      electionInstance.vote(candidateId, { from: accounts[1] });
      return electionInstance.candidates(candidateId);
    }).then(function(candidate) {
      var voteCount = candidate[2];
      assert.equal(voteCount, 1, "accepts first vote");
      // Try to vote again
      return electionInstance.vote(candidateId, { from: accounts[1] });
    }).then(assert.fail).catch(function(error) {
      assert(error.message.indexOf('revert') >= 0, "error message must contain revert");
      return electionInstance.candidates(1);
    }).then(function(candidate1) {
      var voteCount = candidate1[2];
      assert.equal(voteCount, 1, "candidate 1 did not receive any votes");
      return electionInstance.candidates(2);
    }).then(function(candidate2) {
      var voteCount = candidate2[2];
      assert.equal(voteCount, 1, "candidate 2 did not receive any votes");
    });
  });

});
```

Figure 33: Exception handling for double voting

Testing Client-Side Voting

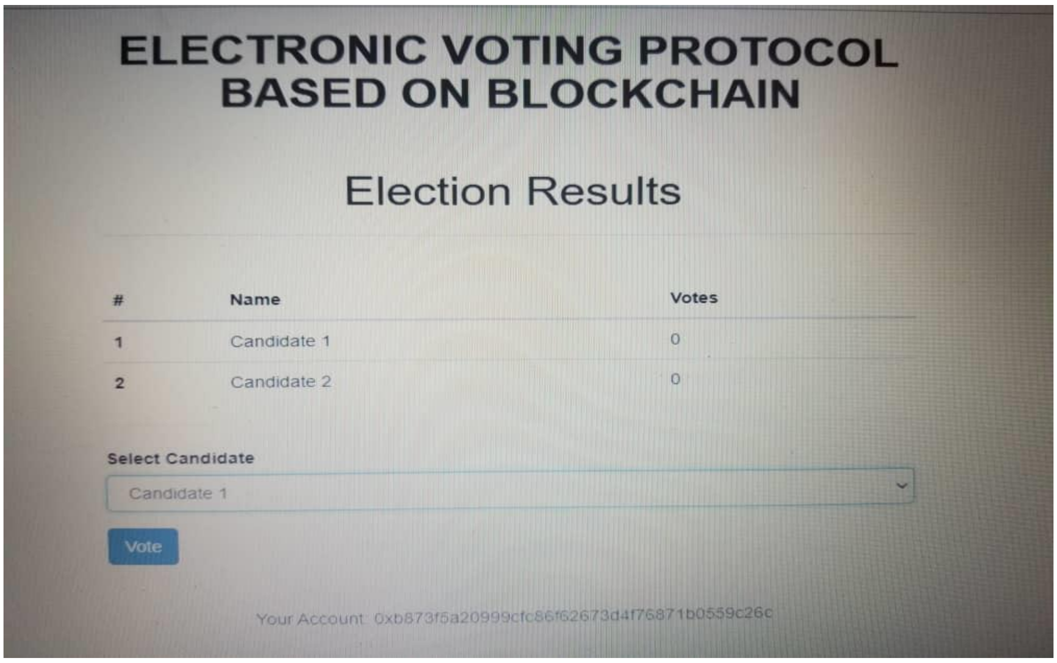
First, we set up a test scenario with a fresh account that has not voted yet. Then we cast a vote on their behalf. Then we try to vote again using the same account to test the validating exceptions. We assert that an error occurred here, and as well inspect the error message, and ensure that no candidate received votes. The screenshot below describes the test results of the exceptions set;



```
Edison@DESKTOP-HBKIOTD MINGW64 ~/Desktop/Dapp-voting/election
$ truffle test
Using network 'development'.


Compiling your contracts...
===========================
> Everything is up to date, there is nothing to compile.



  Contract: Election
    rêÜ initializes with two candidates (82ms)
    rêÜ it initializes the candidates with the correct values (245ms)
    rêÜ allows a voter to cast a vote (333ms)
    rêÜ throws an exception for invalid candidates (423ms)
    rêÜ throws an exception for double voting (489ms)


  5 passing (2s)
```

Figure 34: Testing client-side voting and exception handling

Client-Side front-end interface – Screenshots Capture



ELECTRONIC VOTING PROTOCOL BASED ON BLOCKCHAIN

Election Results

#	Name	Votes
1	Candidate 1	0
2	Candidate 2	0

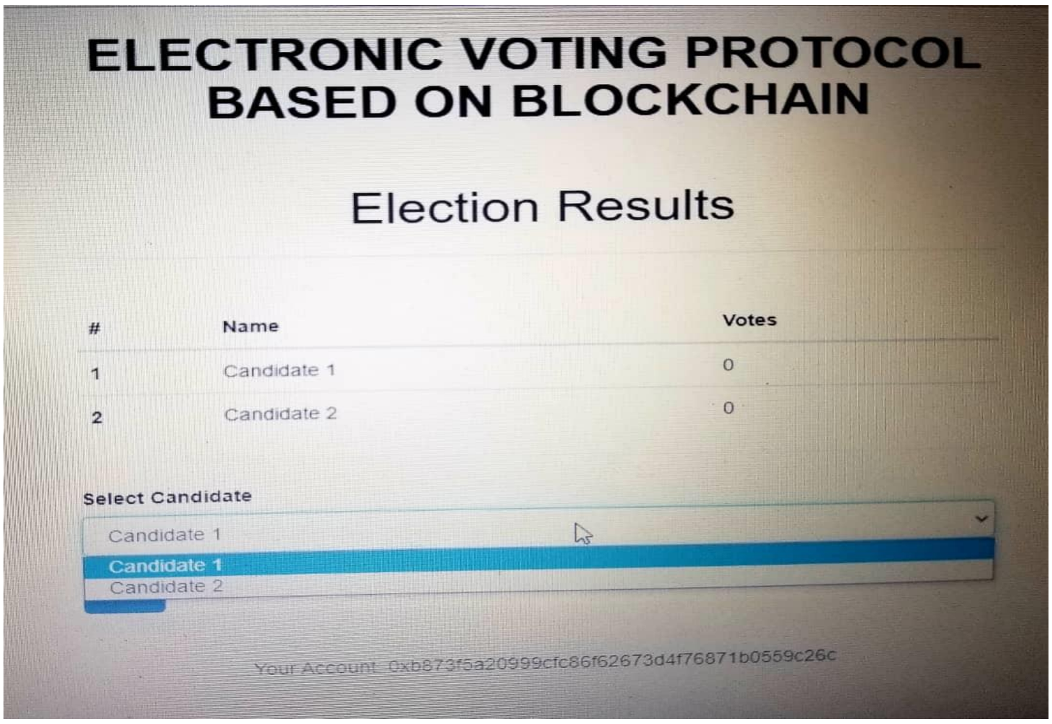
Select Candidate

Candidate 1

Vote

Your Account: 0xb873f5a20999cfc86f62673d4f76871b0559c26c

Figure 35: Single Candidate display – Select candidate and cast your vote



ELECTRONIC VOTING PROTOCOL BASED ON BLOCKCHAIN

Election Results

#	Name	Votes
1	Candidate 1	0
2	Candidate 2	0

Select Candidate

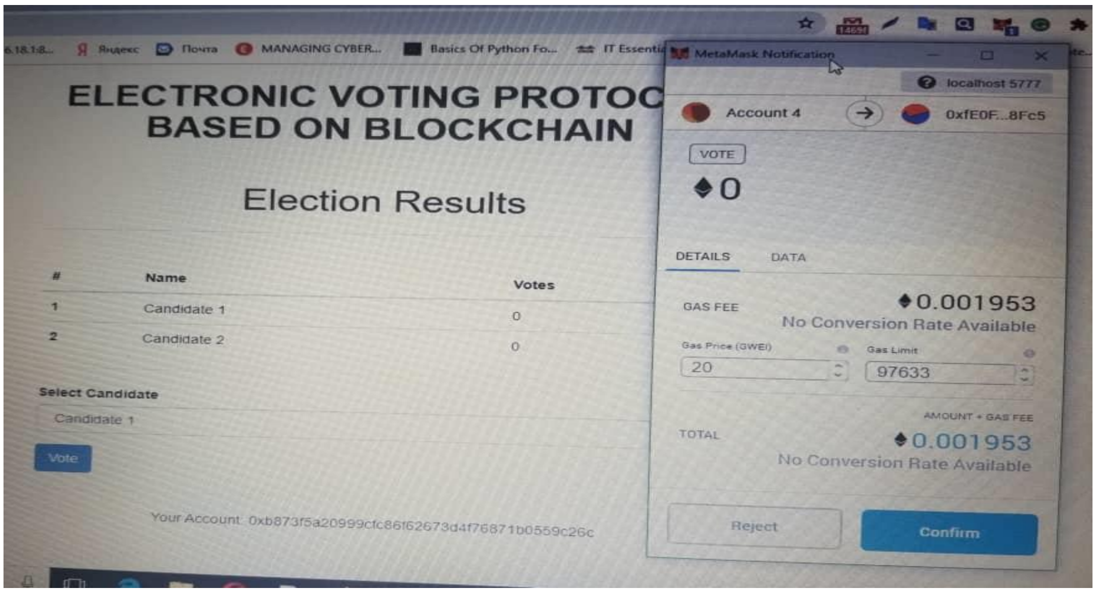
Candidate 1

Candidate 1
Candidate 2

Your Account: 0xb873f5a20999cfc86f62673d4f76871b0559c26c

Figure 36: Display of all Candidates in a drop-down

When the voting function is run, a Metamask confirmation pops up, requesting you to either accept the submission, reset or reject the submission. Once one-click submits, the vote gets cast successfully.



ELECTRONIC VOTING PROTOC
BASED ON BLOCKCHAIN

Election Results

#	Name	Votes
1	Candidate 1	0
2	Candidate 2	0

Select Candidate
Candidate 1
Vote

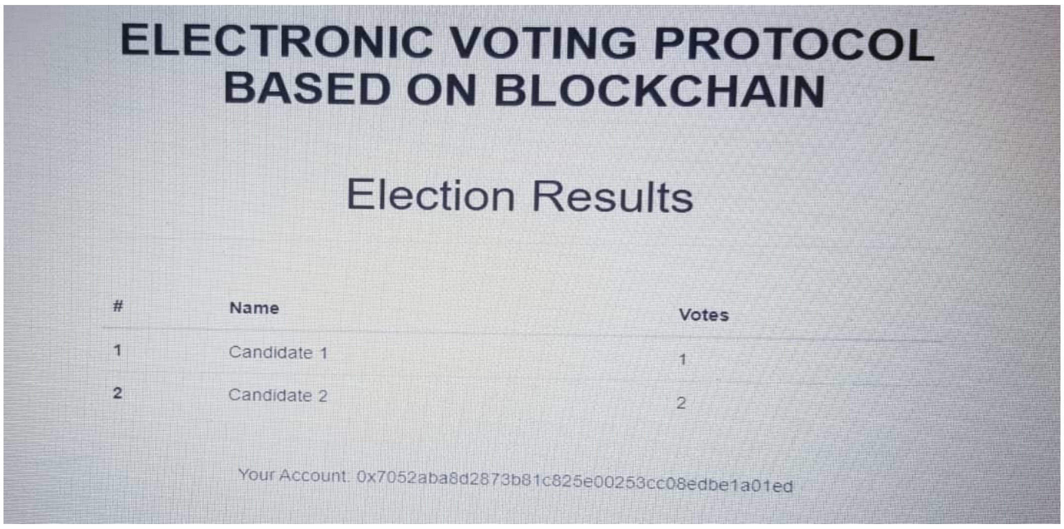
Your Account: 0xb873f5a20999cfc86f62673d4f76871b0559c26c

localhost 5777
Account 4 → 0xfE0F...8Fc5
VOTE
0
DETAILS DATA
GAS FEE 0.001953
No Conversion Rate Available
Gas Price (GWEI) 20
Gas Limit 97633
AMOUNT + GAS FEE
TOTAL 0.001953
No Conversion Rate Available
Reject Confirm

.

Figure 37: Metamask authentication

After the vote is taken, the “Vote” button disappears for that particular voter and he or she can see and verify his or her cast vote in real-time. In the screenshot below, we have tested with 3 voters accounts to cast the vote and observe the votes come in real-time.



ELECTRONIC VOTING PROTOCOL
BASED ON BLOCKCHAIN

Election Results

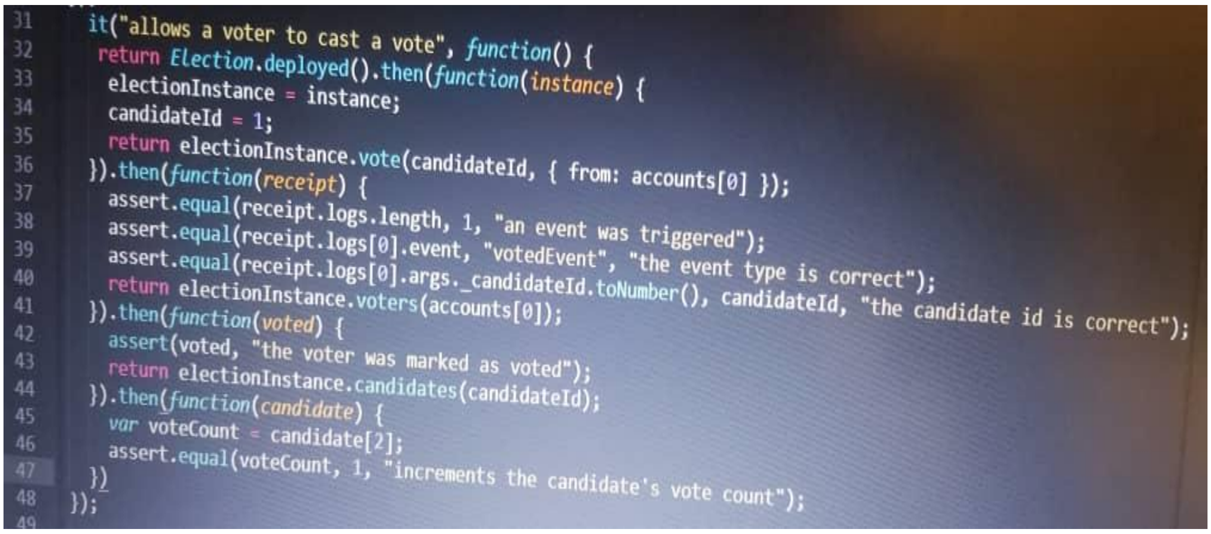
#	Name	Votes
1	Candidate 1	1
2	Candidate 2	2

Your Account: 0x7052aba8d2873b81c825e00253cc08edbe1a01ed

Figure 38: Real-time display of votes

Watch Events

In this last step, we modified to trigger an event whenever a vote is cast. This enabled us to update our client-side application when an account has voted.



```
  it("allows a voter to cast a vote", function() {
    return Election.deployed().then(function(instance) {
      electionInstance = instance;
      candidateId = 1;
      return electionInstance.vote(candidateId, { from: accounts[0] });
    }).then(function(receipt) {
      assert.equal(receipt.logs.length, 1, "an event was triggered");
      assert.equal(receipt.logs[0].event, "votedEvent", "the event type is correct");
      assert.equal(receipt.logs[0].args._candidateId.toNumber(), candidateId, "the candidate id is correct");
      return electionInstance.voters(accounts[0]);
    }).then(function(voted) {
      assert(voted, "the voter was marked as voted");
      return electionInstance.candidates(candidateId);
    }).then(function(candidate) {
      var voteCount = candidate[2];
      assert.equal(voteCount, 1, "increments the candidate's vote count");
    })
  });
```

Figure 39: Checking triggered “voted” event inside our “vote” function

The above test inspects the transaction receipt returned by the "vote" function to ensure that it has logs. These logs contain the event that was triggered. We check that the event is the correct type and that it has the correct candidate id.

To watch events in real-time, we updated the client-side application to listen for the voted event and fire a page refresh anytime that it was triggered. We did that with a “listenForEvents” function. The function does a few things; First, we subscribe to the voted event by calling the “votedEvent” function. We pass in some metadata that tells us to listen to all events on the blockchain. Then we “watch” this event. Inside here, we log to the console anytime a “votedEvent” is triggered. We also re-render all the content on the page. This will get rid of the loader after the vote has been recorded, and show the updated vote on the table.

CONCLUSION

In the implementation, testing, and validation section, we have implemented an application that employs part of the designed protocol. This application is a full-stack application that demonstrates real-time voting on a decentralized electronic voting system using blockchain. A voter can vote from the client-side application, and watch the votes recorded in real-time.

SUMMARY, FUTURE WORK, DISCUSSIONS, AND CONCLUSIONS.

Summary

Using Ethereum signatures implemented using blind RSA signatures and blockchain, we developed an e-voting protocol, which introduces a lot of desirable properties from blockchain, and meets the essential requirements of an electronic voting process. All votes in the blockchain are cryptographically linked block by block. The block with a higher value of signature is selected over others when they have the same timestamp. During elections, using the voter's choice in the election represented by CHj, a voter can make a choice that range from a set of predefined choices. In order to assure fairness, the voter reveals only a digital commitment (DCHj) over the said choice. DCHj (Digital commitment for CHj) only reveals the choice itself only during the counting stage of the election. The voter can vote following the list of candidates by voting for any other persons he/she prefers. Generally, the vote is public, thus the information of vote is not encrypted. The blockchain-based e-voting protocol can be applied to a variety of voting situations and other applications. Although blockchain is a secure technology, it uses Elliptic-curve-based (ECC) public-key encryption/decryption, which is not secure to quantum computer attacks.

Future work

The blockchain based electronic voting protocol still has a large room for improvement, such as improving the voter's anonymity while voting electronically on the internet.

In the proposed blockchain electronic voting protocol, functions like switching more networks between BitCoins, testnet and LiteCoin can be appended. In addition, accomplishment of multiple voting within one vote can be an ideal topic for further study, blockchain with counter-measures to quantum computer attacks, and securing electronic voting applications against client-side attacks.

Conclusion

Because of the properties such as transparency, decentralization, integrity, verifiability, eligibility, and robustness, blockchain is not only a fundamental technology of great interest in its own right but also has large potential when integrated into many other areas for example electronic voting. The blockchain-based e-voting protocol is decentralized and does not need to rely on human trust or a centralized server. The registered voter has the right to vote using their electronic devices connected to the Internet. All the vote records are publicly distributed and can be verified by any intended personnel. In this protocol, everyone involved in the election, including spectators, who can see the voting process (recorded on the blockchain), can verify the whole election's procedure and its outcome, each voter can verify individual voting procedures, e.g., whether his/her ballot has been cast and recorded successfully, counted in the final tally, dependability is guaranteed by the cryptographic algorithms and the practical consensus mechanisms of blockchain, the protocol protects the voting procedure against dishonest behaviors and attacks, consistency supported by the practical consensus mechanisms of blockchain, all participants involved in the election, hold the same record of the voting procedure, and thus accept the same outcome of the election, the whole voting procedure recorded on the blockchain is auditable after the election, only voters themselves know the information of their votes, and all ballots in the ballot box have no connection with their voters. Due to the transparency of blockchain, the whole procedure is open to the public and this leads to more fairness and validity.

The cryptographic signatures are used to validate the origin and integrity of the votes by preserving the voter's choices during the election process. The data is stored on multiple computers on a blockchain network. Attackers planning to get hold of vital data assets using SQL injection have to intrude into every system to reign the network, which is nearly impossible. Even if hackers happen to gain access to the network, the changes they make to the data through SQL injection will be reflected in the systems, notifying every participant about the same. All of these make blockchain unique and probably the best solution for any infiltrations, data tempering, and server-side SQL injection attacks.

In the implementation section, we have implemented an application that employs part of the designed protocol. This application is a full-stack application that demonstrates real-time voting on a decentralized electronic voting system using blockchain. A voter can vote from the client-side application, and watch the votes recorded in real-time.

Discussions

Based on the criticism made on the existing systems, server-side SQL injection attacks on ballot secrecy are particularly troubling, since preserving ballot secrecy is the main goal of the system's cryptographic double-envelope architecture. In our review of the literature, we found out that SQL injection attack is now the most common server-side attack in web applications whereby malicious codes are injected into the database through user inputs fields by unauthorized users and this could lead to data loss or in a worst case, to database hijacking. We further realized that the existing systems depend on centralized databases or servers in the storage of data. They place complete trust in the server that counts the votes at the end of the election process. Votes are decrypted and counted entirely within the unobservable black box of the counting server. This creates an opportunity for an attacker who compromises this server to modify the results of the vote counting.

Privacy of Data Transmission

In our protocol, the communication through the blockchain network may disclose voters' IP addresses, which may lead to the exposure of connections between voters and ballots via network analysis. To enhance voters' privacy, we recommend voters to use anonymity services like proxies or TOR, with which voters can hide their IP addresses.

Security Analysis

In general, the security of our e-voting protocol mainly relies on that of Ethereum signatures implemented using blind RSA signatures and blockchain. In the following, we discuss several security issues on this protocol.

Ballot Manipulation and Forgery

In our protocol, manipulated ballots will be rejected by the network due to wrong signatures and incorrect formats of ballots. Meanwhile, for potential dishonest EA officers, it is impossible to return a wrong signature to invalidate voters' ballots, since wrong signatures associated with the original messages can be detected on the blockchain. When the attack aims to forge ballots, it could not succeed if there exists at least one honest EA officer.

Network Attack

When there are enough honest nodes in the P2P network, the intercepted ballot will be resent, and then accepted by those honest nodes and recorded on the blockchain. Note that, in our proposed protocol, malicious nodes can hardly influence the voting procedure. We also note that replay attacks do not work to forge multiple ballots in this protocol, because if two ballots have the identical voting string, they will be counted only once.

Ballot Collision

Ballots are identified by the choice code and the random string in the voting string. If different voters produce the same string, a collision occurs and one of the two ballots will be invalid. According to the Birthday Attack, for 128-bit voting strings, the probability that collisions occur is less than 10−18. Therefore, we can ignore the existence of collisions provided that the random string is long enough.

In order to improve the quality of protections to the electronic voting systems, an electronic voting protocol based on blockchain is proposed that uses decentralized encrypted distributed databases on a linked network chain. If this protocol is used in the implementation of the electronic voting systems, we are confident that the protocol if well implemented will mitigate SQL injection attacks on the e-voting systems and as well create public and transparency in the voting process while protecting the anonymity of the voter's identity.

REFERENCES

A. Kiayias, M. K. (2006). *An internet voting system supporting user privacy.* in Proceedings - Annual Computer Security Applications Conference, ACSAC, 2006, pp. 165–174. .

Academy, L. (2019). *Cryptography*. Retrieved 09 Monday, 2019, from www.lisk.io/academy: https://lisk.io/academy/blockchain-basics/how-does-blockchain-work/blockchain-cryptography-explained

Alia M, S. A. (2007). *A New Public-Key Cryptosystem Based on Mandelbrot and Julia Fractal Sets.* Asian Journal of Information Technology, AJIT, 6(5), pp. 567-575.

Ayed, A. B. (May, 2017). *A CONCEPTUAL SECURE BLOCKCHAIN- BASED ELECTRONIC VOTING SYSTEM.* International Journal of Network Security & Its Applications (IJNSA) Vol.9, No.3.

Bagnoli, L. (2017, 10 13). *E-VOTING or E-RIGGING? Why technology cannot not solve Kenya's corruption problem*. Retrieved 09 23, 2019, from www.theelephant.info: https://www.theelephant.info/features/2017/10/13/e-voting-or-e-rigging-why-technology-cannot-not-solve-kenyas-corruption-problem/

Bankrate. (2020). *Blockchain*. Retrieved from www.bankrate.com: https://www.bankrate.com/glossary/b/blockchain/

Ben, A. A. (2017). *A Conceptual Secure Blockchain- based Electronic Voting System.* Department of Engineering and Computer Science, Colorado Technical University, Colorado Springs, Colorado, USA: International Journal of Computer Networks & Communications (IJCNC).

Branovic I, G. R. (2003). *Memory Performance of Public-Key Cryptography Methods in Mobile Environments.* New Orleans, LA, : ACM SIGARCH Workshop on Memory performance: Dealing with Applications, systems and architecture (MEDEA-03).

Caarls, S. (November 2010). *E-voting handbook.* France : Council of Europe Publishing.

Canada, G. o. (2017). *Online Voting: A Path Forward for Federal Elections.*

Chang, T.-Y. (2015). *Electronic Voting Protocol Using Identity-Based Cryptography.* The Scientific World Journal.

Chaum, D. L. (2019). Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Technical Note Programming Techniques and Data Structures* .

Clement Chan Zheng Wei, C. C. (2020). *336 Blockchain-Based Electronic Voting Protocol.* Faculty of Computer Science and Information Technology, University Tun Hussein Onn Malaysia, Malaysia.

CoinTelegraph. (2020). *Electronic Voting With Blockchain: An Experience From Naples, Italy*. Retrieved from www.cointelegraph.com: https://cointelegraph.com/news/electronic-voting-with-blockchain-an-experience-from-naples-italy

Commision, E. (1958 - 2012). *A Brief History of Elections in Uganda 1958-2012*. Retrieved 09 Monday, 2019, from www.ec.or.ug: https://www.ec.or.ug/?q=content/history-ec

Commission, E. (2019). *Establishment and Mandate of the Electoral Commission*. Retrieved 09 Monday, 2019, from www.ec.or.ug: https://www.ec.or.ug/?q=establishment-and-mandate-electoral-commission

Commission, U. L. (2011). *COUNTRY REPORT ON ELECTORAL.* Kampala: UGANDA LAW REFORM COMMISSION. Retrieved 11 16, 2019, from http://www.justice.gov.za/alraesa/conferences/nam2011/uganda.pdf

Curran, K. (2018). *E-Voting on the Blockchain.* ResearchGate.

D. Ashok Kumar, T. U. (2012). Electronic voting machine — A review. *International Conference on Pattern Recognition, Informatics and Medical Engineering (PRIME-2012).* Salem, Tamilnadu, India: IEEE.

Diffie W, H. M. (1976). *New Directions in Cryptography.* IEEE Transactions on Information Theory, IT-22, pp. 644-654.

Djina Pavlovic, Q. U. (2019, June 06). *Analyzing the Risks of Implementing Electronic Voting.* Retrieved July 04, 2019, from Inquire Publication: http://inquirepublication.com/analyzing-the-risks-of-implementing-electronic-voting/

Douceur, J. R. (2002). *The Sybil Attack.* Retrieved 10 11, 2019, from www.link.springer.com: https://link.springer.com/chapter/10.1007/3-540-45748-8_24

Drew Springall, T. F. (November 2014). *Security Analysis of the Estonian Internet Voting System.* ResearchGate.

Drew Springall, T. F. (2014). *Security Analysis of the Estonian Internet Voting System.* ResearchGate.

Drew Springall, T. F. (2019). *Security Analysis of the Estonian Internet Voting System.* University of Michigan, Ann Arbor, MI, U.S.A. ‡Open Rights Group, U.K. .

ElGamal. (1985). *A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms.* IEEE Transactions on Information Theory, IT-31(4), pp. 469–472.

Estonia. (2014, May). *Independedent report on E-voting in Estonia*. Retrieved from www.estoniaevoting.org: https://estoniaevoting.org/findings/summary/

Estonia, E. (2017, September). *Estonia's i-voting: more secure, more popular*. Retrieved July 23, 2019, from www.e-estonia.com: https://e-estonia.com/estonias-i-voting-more-popular-more-secure/

Francisco, J. (2020, May 12). *How Blockchain Technology Can Check Voter Fraud*. Retrieved from businessblockchainhq.com: Many organizations like Horizon State are already developing applications that can bring blockchain to the voting process and voters. In the case of Horizon State, the company is proposing that voters will be able to cast from the comfort of their smartph

G.O. Ofori-Dwumfuo, E. P. (September 30, 2011). The Design of an Electronic Voting System. *Research Journal of Information Technology 3(2): 91-98, 2011.*

Gaby G. Dagher, P. B. (1-1-2018). BroncoVote: Secure Voting System Using Ethereum's Blockchain. *4th International Conference on Information Systems Security and Privacy, 2018-January, 96-107.http://dx.doi.org/10.5220/0006609700960107.* Boise State University ScholarWorks.

Hardwin Spenkelink . (August 2014). *The adoption process of cryptocurrencies.* Amstelveen.

IDEA, I. (2011). *Introducing Electronic Voting.* PolicyPaper.

Institute, N. D. (2020). *The Important Uses of Cryptography in Electronic Voting and Counting*. Retrieved from www.ndi.org: https://www.ndi.org/e-voting-guide/examples/cryptography-in-e-voting

J. Hoffstein, J. P. (2008). *An Introduction to Cryptography, vol. XVI.*

Jonathan, R. (2020). *Electronic Voting With Blockchain: An Experience From Naples, Italy*. Retrieved from www.cointelegraph.com: https://cointelegraph.com/news/electronic-voting-with-blockchain-an-experience-from-naples-italy

Kalaisankaran, B. (2013). *Students Attendance Management System.* Pollachi: Dr. Mahalingam College Of Engineering And Technology.

Karspersky. (2020). *Cryptography Definition*. Retrieved from www.usa.kaspersky.com: https://usa.kaspersky.com/resource-center/definitions/what-is-cryptography

Kazi Sadia, M. M. (2020). *Blockchain Based Secured E-voting by Using the Assistance of Smart Contract.* Department of IT Convergence Engineering, Kumoh National Institute of Technology, Gumi, 39177, South Korea.

Kibin Lee, J. I. (2016). *Electronic Voting Service Using Block-Chai.* The Journel of Digital Forensics, Security and Law.

Koblitz. (1987). *Elliptic Curve Cryptosystems.* Mathematics of Computation, pp. 203–209.

Laboratories, R. (2007). *What is a Hard Problem.* RSA the Security Division of EMC.

Lake, J. (2019, April 18). *What are the risks of electronic voting and internet voting?* Retrieved July 04, 2019, from Comparitech: https://www.comparitech.com/blog/information-security/electronic-voting-risks/

Lewis, A. (2018). *The Basics of Bitcoins and Blockchains: An Introduction to Cryptocurrencies and the Technology that Powers Them.*

Marian Stoica, B. G.-M. (2016). E-Voting Solutions for Digital Democracy in Knowledge Society. *ResearchGate*.

Marko Hölbl, M. K. (2018). *A Systematic Review of the Use of Blockchain in Healthcare.* Symmetry.

Menezes A, P. V. (1996). *Handbook of Applied Cryptography.*

Michael Crosby, . . (2015, October 16). *Sutardja Center for Entrepreneurship & Technology Technical Report.* Retrieved July 04, 2019, from https://scet.berkeley.edu/: https://scet.berkeley.edu/wp-content/uploads/BlockchainPaper.pdf

MLSDev. (2019, May 07). *Blockchain Architecture Basics: Components, Structure, Benefits & Creation*. Retrieved 10 11, 2019, from www.medium.com:

https://medium.com/@MLSDevCom/blockchain-architecture-basics-components-structure-benefits-creation-beace17c8e77

Mulligan, G. (2017, 05). *Technology of business reporter*. Retrieved 09 Monday, 2019, from www.bbc.com: https://www.bbc.com/news/business-39955468

news24. (2019, 09 23). *Kenya shows pitfalls of digital elections*. Retrieved 09 Monday, 2019, from www.news24.com: https://www.news24.com/Africa/News/kenya-shows-pitfalls-of-digital-elections-20170907

NIST. (2009). *Digital Signature Standard (DSS).* FIPS 186-3.

Orhan Cetinkaya, D. C. (2019). *Verification and Validation Issues in Electronic Voting.* Ankara, Turkey.

Plantera, F. (2019, 08). *when-will-other-countries-join-estonia-in-voting-on-the-internet*. Retrieved 09 Monday, 2019, from e-estonia.com: https://e-estonia.com/when-will-other-countries-join-estonia-in-voting-on-the-internet/

R. A. Rivest, A. S. (1978). *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems.* Communications of the ACM, 21(2), pp.120-126.

Rhodes, D. (2020, April 27). *SHA-256: An Overview & Guide of the SHA-256 Algorithm*. Retrieved from komodoplatform.com: https://komodoplatform.com/sha-256-algorithm/

Rifa Hanifatunnisa, B. R. (2017). *Blockchain based e-voting recording system design.* Lombok, Indonesia : IEEE.

Rivest R A, S. A. (1978). *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems.* Communications of the ACM, 21(2), pp.120-126.

Ronald L. Rivest, A. S. (2001). *How to Leak a Secret.* Springer-Verlag Berlin Heidelberg 2001.

Ronald L. Rivest, A. S. (2019). *How to Leak a Secret.* 1 Laboratory for Computer Science, Massachusetts Institute of Technology, 2 Computer Science department, The Weizmann Institute, Rehovot 76100, Israel.

SANSON, L. D. (October 16 - 19,2001). Security Technology . *35th ANNUAL 2001 International Carnahan Conference on Security Technology* . LONDON, ENGLAND : THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS .

Schwartz, J. (2018, November 1st). *The Vulnerabilities of Our Voting Machines*. Retrieved from www.scientificamerican.com: https://www.scientificamerican.com/article/the-vulnerabilities-of-our-voting-machines/

Scott Wolchok, E. W. (Feb. 2012). Attacking the Washington, D.C. Internet Voting System. *Proc.16th Conference on Financial Cryptography & Data Security.* The University of Michigan, Ann Arbor .

Scott Wolchok, E. W. (2012). Attacking the Washington, D.C. Internet Voting System. Michigan: 16th Conference on Financial Cryptography & Data Security.

Scripts, M. T. (2020). *SHA-256 Cryptographic Hash Algorithm*. Retrieved from www.movable-type.co.uk: https://www.movable-type.co.uk/scripts/sha256.html

Setor de Administração Federal Sul- (SAFS), Q. 7. (2019). *The History of Electronic Voting*. Retrieved July 04, 2019, from electronic-voting : http://english.tse.jus.br/news/the-history-of-voting

Sidhu, P. I. (2015). *BlockChain Technology Beyond Bitcoin.* UC Berkeley: Sutardja Center for Entrepreneurship & Technology.

Springall, T. F. (2014). Security Analysis of the Estonian Internet Voting System. *In Proceedings of the 21st ACM Conference on Computer and Communications Security (CCS ’14), November 2014.* https://estoniaevoting.org/findings/paper.

Standards, N. B. (1977). *Data Encryption Standard.* Washington D.C.: FIPS-Pub.46. National Bureau of Standards, U.S. Department of Commerce.

Standards, P. (2009). *"The Digital Signature Standard (DSS).* Processing, pp. 1–119.

TAHER ELGAMAL. (JULY 1985). *A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms.* IEEE TRANSACTIONS ON INFORMATION THEORY, VOL. IT-31, NO. 4, .

TechTerms. (2020). *Cryptography*. Retrieved from www.techterms.com: https://techterms.com/definition/cryptography

Types of Software Testing: Different Testing Types With Details. (2019, November 10). Retrieved from Software Testing Help: https://www.google.com/amp/s/www.softwaretestinghelp.com/types-of-software-testing/amp.

Union, I.-P. (1994, March 26). DECLARATION ON CRITERIA FOR FREE AND FAIR ELECTIONS. Paris, Le Grand-Saconnex/Geneva, Switzerland.

Wang, Y. L. (2017). *An E-voting Protocol Based on Blockchain.*

Xukai Zou, H. L. (2017). Transparent, Auditable, and Stepwise Verifiable Online E-Voting Enabling an Open and Fair Election. *Open Access*.