Developing a Permissioned Blockchain Voting Application

Juan Matiz, Liz Tucker

Department of Computer Science, Eastern Connecticut State University, USA

[matizj@my.easternct.edu](mailto:matizj@my.easternct.edu)

[tuckerel@my.easternct.edu](mailto:tuckerel@my.easternct.edu)

**Abstract**

While we continue to move into an increasingly digital age, voting remains to be one of the least researched fields in terms of technology utilized. While the slow adoption of technology for such an important and protected process is understandable, the technology that currently exists is known to be very insecure and out of date. In the interest of continuing to protect this precious democratic process, and with the intent of increasing the amount of people this process sees, we attempted to leverage an opensource blockchain framework in order to develop a prototype blockchain system. The blockchain network will be secure, reliable, distributed, and have high availability. We have also developed a Python implementation of a Merkle tree, which forms the cryptographic basis for blockchain technology, to confirm proof of concept for how the blockchain would prevent the tampering of votes.

1. **Introduction**

Currently, there are only four states that support e-voting through the use of an internet portal. As the majority of US citizens have access to some form of internet, providing easy access to a secure voting technology would significantly increase the number of citizens that are able to vote on election days. Many registered voters currently refrain from voting because they cannot get to their respective polling place, or the wait time is simply too long [1]. One main concern around current and prospective voting technology is security.

Despite the security concerns currently seen in the United States, other countries such as Estonia seem to have successfully implemented an electronic voting system in the form of internet voting. With the countless data breaches and exploits that occur every year, one would think that the internet would be ruled out as an avenue for implementing voting systems. However, since 2005, Estonia has allowed its citizens to vote via the internet if they so choose [2]. Of course, the success behind the roll out stems from three key aspects: Estonia has built the legal and bureaucratic framework to address any concerns that might arise with this form of voting [2]. It has also invested heavily in the creation of a digital form of authentication that allows for Estonians to verify their identity securely [2]. Finally, due to it being a parliamentary representative democratic republic, national elections are less contentious for the losing party can still be part of the government [2].

The United States lacks many of the aspects that allowed for Estonia's internet voting system to be such a success, researchers have begun looking into other potential solutions to the current problem with electronic voting machines, one of them being blockchain technology. A blockchain is essentially a distributed database of records, or public ledger of all transactions or digital events that have been executed and shared among participating parties. Each transaction in the public ledger is verified by consensus of a majority of the participants in the system [3]. Each block in the chain is made up of transaction data, a unique hash, and the hash of the previous block in the chain. The hash is made from a digital timestamp [4] and the transaction data contained within a block. A blockchain network is made up of peers who each contain a copy of the entire chain, thus making it hard – although not impossible – to alter the chain [3].

Satoshi Nakamoto's invention, Bitcoin, utilizes a host of advancements in cryptography, such as the utilization of one-way hash functions to verify the authenticity of each new block in the chain [5], as well as Merkle trees, which are a form of binary trees that are capable of creating and allowing for the verification of an infinite amount of cryptographic signatures [6]. Furthermore, the implementation of a distributed ledger between all participating nodes allows for Bitcoin to address an important issue known as the Byzantine Generals Problem [7]. This problem is a hypothetical scenario crafted to address the issue that will arise if a node fails unexpectedly. The authors of this research paper wanted to know whether the rest of the nodes, including the faulty ones, can reach some sort of agreement. It was discovered that if the number of non-faulty nodes is greater than three times the number of faulty nodes plus one, the network will always reach an agreement [7]. Since a node in the Bitcoin can always return to the network, whether it left due to an attack or failure, the node will synchronize itself with the rest of the network due to all possessing a copy of the distributed ledger [3].

Another key aspect that makes Satoshi Nakamoto’s invention very useful at tackling the security concerns behind electronic voting systems is the fact that it seems to fulfill all three aspects of Brewer’s Conjecture [8]. Brewer’s Conjecture stipulates that a web service can't be consistent, available, and partition-tolerant [8]. The authors of this paper argue that a web service can be two out of three, depending on whether it is an asynchronous or partly synchronous system, but never all three at the same time [8]. However, due to the distributed ledger, and the way the blocks in the chain are calculated, Bitcoin can prevent catastrophic partitions from forming by allowing nodes to resync themselves with the rest of the network if they go offline. Also, the cryptographic algorithms used to calculate hashes as well as the Merkle tree algorithm have been proven mathematically to always be reliable [6], thus blockchains, if they are implemented correctly, can be a powerful solution that can tackle the security issues with electronic voting systems.

There have been some attempts at creating voting systems based on blockchain technology, Bistarelli et. al crafted a voting scheme around the process of sending a bitcoin to another individual on the network [9]. Candidates would obtain a public address that could be disseminated to everyone in the country. Then, once the voter has verified their eligibility to vote via a Kerberos authentication system, they are dispensed with a token that can be used to “place” their vote [9]. The main issue with this solution is the fact that they had to purchase some form of bitcoin, more specifically in the form of satoshi (the smallest value of bitcoin that can be purchased), to utilize the blockchain. This means that if their solution were to be implemented in the real world, a government would have to pay the value of satoshi, which is 6 x 10-6, times the number of registered voters. Also, the Kerberos authentication system can be an attractive single point of failure to hackers, thus placing people’s personal information at risk.

Finally, Ben Ayed has proposed a conceptual model for developing an electronic voting system based on blockchain technology. In his conceptual model, three blockchains were developed for each of the candidates participating in this theoretical election [10]. He did not provide a way of counting the number of votes received. He also addresses some concerns around the Estonia example, where the code used to implement the internet voting system has never been released, thus lacking the transparency aspect that one would like to achieve with a voting system [10]. Unlike the example created by Bistarelli et. al, Ben Ayed does not provide a way to determine whether the user is eligible to vote or not, to which he acknowledges can be quite challenging as centralization of this process can lead to a single point of failure [10].

Whether it be from outside forces or an inside job, the question remains for how we secure these systems to make them reliable enough for public use. The objective of this research was to take the conceptual model proposed by Ben Ayed, and develop a permissioned based blockchain that we used to tackle the following two scenarios [10]. First, to test whether the blockchain is capable of preventing a user who attempts to vote twice, either for the same candidate or for a different set of candidates. And Second, whether we can return to the user either the location of their transaction in the chain or a hash that can be used by the user to search for their transaction, thus fulfilling the accountability aspect that we seek in a voting system.

Furthermore, in many current systems, accountability is a questionable topic because for many states, there is no way to verify who a voter has voted for after the voting process has ended. Since the majority of the United States still relies on paper ballots, and due to the sheer mass of voters and that we do not record identifiable information about the voter on the ballot, we cannot simply have someone retrieve a certain vote and read off the selected candidate [15]. Even with several electronic systems, once our vote is cast, there is likely no way to view it again.

In order to conduct our experiments, we built a blockchain application that uses Hyperledger Fabric [11]. Hyperledger Fabric is a permissioned blockchain framework developed and maintained by the Hyperledger organization which is a part of The Linux Foundation projects. Fabric is a multipurpose distributed ledger written in Golang that supports the use of smart contracts. Smart contracts are essentially like real world contracts in that the system enforces the terms of the contract. Furthermore, Fabric makes the creation of peer nodes very easy. The customization of architecture is also simple to design and implement for distributing applications.

1. **Methodology**

The system is populated with one admin to issue votes to at least ten users who will have two candidates to vote for. The admin will issue 10 votes, one for each user, each with its own unique number. Of these ten users, five will vote normally, only attempting to redeem their vote once. The other five will attempt to resubmit their votes for the same candidate, twice. At least one user will attempt to redeem a vote that does not belong to them. Populating the admin and users will be done through a template file that Hyperledger uses - see crypto-config.yml. Further documentation is available on the README. You may find the documentation and the code repository at: <https://ltucker284.github.io/basic-chain/>.

Due to some network issues that we faced while setting up the Hyperledger Fabric network on a Linux virtual-machine, we decided to take the basic components of a blockchain such as a linked list and a Merkle tree, and developed a proof of concept that showcased these key components in Python. The first script that was developed, Create\_voters.py, was designed to create the vote and voter instance of the application. In total, 20 voter instances were created, each instance containing a voter\_id, vote\_number, candidate\_hash, and a vote\_cast\_time, that were then stored in a list that would eventually be parsed by the other script contained in the repository, merkle\_tree.py. In order to simulate the time difference that one would expect to see between votes, we imported the random python standard library to create a range between the time that the polls are open and the time they are close. These two timestamp values are then passed into random.randrange() function that selects random item from a range, and stores it as a vote\_cast\_time variable.

Finally, two candidates were created by hashing the following two strings: “Candidate\_one” and “Candidate\_two”. The hashes obtained from the candidate strings were then passed as a tuple into create\_vote() function. The way we simulated a voter voting for a particular candidate was by using another function from the random python standard library called randint(). The *randint()* function from the *random* library was used to randomly select a candidate for each voter.

Once the vote instance has been created, the contents of these functions are stored in a list called temp\_vote that would then be passed to the other script contained within our repository. The script in question, called merkle\_tree.py, was not developed by us, it was retrieved from the following GitHub repository: <https://github.com/JaeDukSeo/Simple-Merkle-Tree-in-Python/blob/master/MerkleTrees.py>. In his solution, Jae Duk Seo takes a list of items and hashes them using the SHA-256 cryptographic function. Once the hashing is completed, Jae Duk Seo’s Merkle\_tree class returns the Merkle root and an ordered dictionary of the items that were hashed.

Each Merkle root returned by the Merkle\_tree class is then stored in an OrderedDictionary that simulates the single linked list functionality that we see in a real-world blockchain application. The only slight difference is that OrderedDictionary is a double linked list data structure so whenever a key is removed from the dictionary and a new one is inserted the new item appears at the end of the dictionary. The key/value pair created by this OrderedDictionary allows us to theoretically traverse the chain all the way back to the first root, which in our case is the Merkle root generated by the genesis block.

In order to test whether Merkle Trees can be used to test the integrity of block chains, we passed the vote blocks through a function called create\_tampered\_block() that modifies the value of the candidate hash for the opposite candidate’s hash. Since a Merkle root is a one-way function, the digestion of tampered data by merkle\_tree.py will generate a new Merkle root. This new Merkle root will cause the blocks to no longer point to the next block in the chain, thus demonstrating that the chain has been tampered with. This same theory of one-way functions can be used to allow the voter to query for their vote in the chain.

A series of prompts will instruct the user to type the data that was returned to them in the form of a receipt. This data is then appended to a list that is then digested by merkle\_tree.py which returns the Merkle root. The Merkle root that is returned will then be used to query the OrderedDict to see whether the Merkle root is a key in the dictionary. If it is that means the key is in, and the vote has been recorded properly. If the script returns a Key Not Found Error then the key is not in the dictionary, and there was an issue recording the user’s vote.

1. **Results**

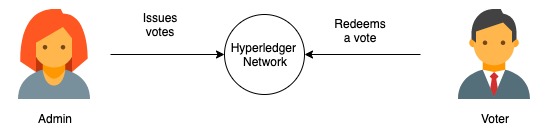
 In order to test whether the system is able to prevent an individual from voting twice, we developed a solution, in the form of an application that enforces smart contracts, that prevented users from voting twice or redeeming votes that did not belong to them. The application that was

Figure 1 - Overview of how votes are issued and redeemed.

built for the Hyperledger network is used to issue one “vote” - in the form of a smart contract - to each user, by admin users on the network. The smart contract is only designed to be issued, and redeemed. Once it is redeemed, it may not be redeemed again, and without an issued “vote”, a user may not redeem a vote. A user also may not redeem a vote that does not belong to them. This process is detailed in **Figure 1**.

A screenshot of a cell phone

Description automatically generated Each transaction takes up to five parameters. These include: voter, paperNumber, candidate, redeemingUsername, and issueDate. Each contract can only be redeemed by the user it’s issued to as the system checks the owner of the vote before setting the vote’s state as redeemed. This is shown in **Figure 2.**

Figure 2 – Code for redeeming a vote

Upon testing the final product, using the methodology described earlier, the first five users redeemed their votes without failure. The second set of users redeemed their vote their votes the first time without failure. When attempting to redeem each one a second time, the system produced the appropriate error code saying their votes had already been redeemed. This error code is depicted in **Figure 3**. One user also attempted to redeem a vote that belonged to another user which resulted in failure. It **must** be noted that because our system does not enforce unique vote numbers, it is currently possible to issue a vote with the same number twice therefore making it possible to redeem twice. This is a major flaw within the system’s current state.

A blurry image

Description automatically generated

**Figure 3 – Error code produced by the system when a user attempts to redeem the same vote twice**

Furthermore, for the network to run properly, new crypto-config material should be generated each time the network is booted up, using the generate.sh bash script. As is, generate.sh should only be run the **first time** the network is brought up, as each time the teardown script is run, it will run generate.sh as well. Much of the crypto-config materials are certificates needed by different members of the network. The start.sh script will make sure that the network isn’t currently running and then subsequently bring up the network, install the chaincode, and instantiate the chaincode. Also included in this project are stop.sh and teardown.sh. Stop.sh will pause the network and teardown.sh removes the network and its associated docker images and then regenerates crypto-config material. For more detailed documentation, please see the GitHub documentation.

As previously stated, due the issues experienced while trying to create the Hyperledger network, we decided to develop a proof of concept that would be capable of verifying whether a block chain has been tampered with, and allow the user to query the blockchain for their vote in Python. **Figure 4** demonstrates the result of adding the Merkle roots to the OrderedDictionary, while **Figure 5** demonstrates graphically how we were able to determine whether a block has been tampered with or not. Since the OrderedDictionary containing the Merkle roots contains

A screenshot of a cell phone

Description automatically generated**Figure 4 – Vote blocks and their Merkle roots pointing to the next vote block in the chain.**

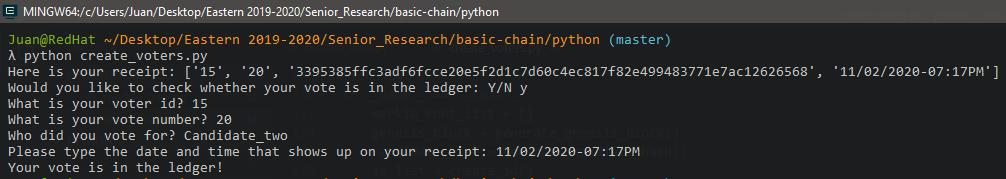
the key of the current root and the root before it, if a Merkle root is tampered the block will no longer contain the hash of said block, but instead a new hash. This new hash generated from the new data will no longer point to the next root in the actual chain, but instead to nothing. In theory, in a real-world blockchain application this block would have been removed by a consensus algorithm, but we did not have enough time to develop one.

**A screenshot of a social media post

Description automatically generatedFigure 5 – Structure of block chain after a vote block has been tampered with.**

In this implementation with an OrderedDictionary, whenever a key is popped from a dictionary,a new one is added and the new key/value pair goes to the end of the dictionary. Thus, **Figure 5** does not really showcase what would happen in a real-world block chain application, but instead shows what happens when a key is tampered and changed. A for loop went over every vote block instance and changed the value of the candidate hash for the opposite, thus every tampered block was moved to the end of the dictionary.

Regarding the querying of votes functionality of this proof of concept, a user was asked to submit the data that was included in the receipt that they received after voting. **Figure 6** shows the questions asked as well as the “receipt”. Since Merkle roots use the theory of one-way functions, every list of data, if correct, should return the same hash. Thus, no matter what, if the user inserts the correct data and the vote is in the chain, their Merkle root should return a confirmation that their vote has been added to the chain.

**Figure 6 – Receipt of vote and questions asked when the user tries to query the chain.**

1. **Discussion**

attempted to However, due to time and technical limitation the network currently configured to only have one node running on it. Although the network has the capability to bring up seven nodes, we ran into difficulties joining them to the channel the chaincode operates on. We left the code that beings up the other peers, but currently the network only uses peer0.org1.example.com.

We also chose the Hyperledger Fabric blockchain framework as opposed to something like Bitcoin because the Fabric framework gives us the flexibility of choosing what kind of consensus method we’d like to use. It also does not require the purchase of cryptocurrency in order to use the system like Bistarelli et al’s model does [9]. These two factors combined have the potential to reduce operating costs of a production system as well as prevent the need for unnecessary third parties participating in the system, i.e. miners. While the distributed ledger feature of blockchain is both attractive and vital in protecting the integrity of the system and its data, the introduction of unnecessary parties like miners is something that should be taken in with careful consideration.

When developing the architecture of the system we also looked at the conceptual model developed by Ben Ayed [10]. This priority of this system was to design a simple architectural model agnostic of any existing frameworks. It’s a purely conceptual model discussing the application of blockchain in voting technology and the implications of blockchain concepts such as how to possibly set up the chain to easily count/query votes, what happens when concusses occur, etc. We chose not to follow the author’s example of using one chain per candidate because the framework we chose makes it easy to query the chain as well as easily produce a paper trail of votes in the case of a recount.

Furthermore, what we have done that expands upon these papers, is to build and implement our system on a small scale. While our implementation remains basic, and leaves much further work to be done, it proves that smart contracts in combination with the blockchain network, there is the potential to build a secure, distributed system such that we have the possibility expand access to voting technology safely. As current pieces of voting technology have been proven insecure, there is a need to improve technology that currently exists, and in order to ensure the legal right to every citizen to be able to vote, it is in the interest of the government to improve the technology instead of relying solely on paper ballots [18, 19].

As the focus of this project was to implement and test a permission based blockchain model, and due to the time scope in which this project was done, we acknowledge that there are major limitations in this model that conflict with real-life security limitations for voting practices. In this project we ignored the authentication portion beyond making a test admin that issues votes to users, and at least one user that can redeem a vote. It is also assumed that each vote is created with a unique ID – this is currently not enforced by the system. For the purposes of this study, it was ensured that each vote was created with a unique ID. We cannot claim that this system complies with all town/city, state and federal standards for authenticating a voter in an election.

The Hyperledger project uses one of Hyperledger’s tutorial examples as a base of the project [16]. It was modified to produce the architecture we were seeking to build. This simplified the development process greatly, although we must acknowledge that there is a lot of residual terminology from the original project that can make the code confusing to look at. As we are not blockchain or Hyperledger experts, it was better for us to take what was available and ensure it works as intended rather than to modify things without understanding any repercussions that may come with it,

In addition, also due to the scope of the project, we contained all of our peer nodes under one “organization. Currently, only one peer belongs to the network and that same peer is the only one endorsing transactions. This is due to time and technical limitation. For further work, it would be recommended to further distribute this application under multiple organizations and channels, begin working on a distributed authentication process, to implement more finely grained endorsement policies for the peers, or further examine and build upon the existing business logic contained in the application or smart contract.

In regard to the Python proof of concept, we were able to demonstrate the theory behind Merkle trees and their ability to create one-way hash functions [6]. These one-way hash functions can be used to check the integrity of blocks of data without having to parse through every data item used to create said data type. However, because of the nature of one-way hash functions the data that is passed through the one-way hash function the data is no longer recoverable. If a user is unable to reproduce the data that was used to create the hash for their vote, we would be unable to verify whether the vote is in the chain or not. Also, another problem with one-way functions is the ability of bad actors being able to brute force their way through the chain and be able to recreate every vote block in the chain. Since the data used to create a vote block contains data that does not possess any form of authentication, nothing is stopping bad actors from recreating every vote block in the chain.

Furthermore, the relative complexity of Merkle roots, although trivial for a Computer Scientist to recreate, makes this concept hard for the average user to understand and embrace. A success of the electronic voting systems seen in Estonia was the ability of users to use it as an alternative to traditional forms of voting, but not as the only way to vote [2]. The hope for our system would be for states and local governments to embrace it, and thus replace their very insecure voting machines. However, due to the lack of basic cybersecurity knowledge in local governments, the likelihood for this system to be implemented as an open source system is very unlikely. A truly open source solution would allow any Cybersecurity expert and citizen watchdog to verify the integrity of the voting system and confirm to the populace that their votes were counted properly.

After we finished with the testing for this project, we discovered that the state of West Virginia had commissioned a private company to develop a block-chain based application that would allow military personnel and people outside the United States to vote securely. The issue with this endeavor by a private company is that the integrity of the votes rest in the hands on a private entity that is unwilling to share their voting process with the public, citing intellectual property as the basis for this action. Even though they have every right to protect their intellectual property, it is this protection of IP and trade secrets that led us to the current security issues with the Optical Scan and DRE voting machines that are currently used in our voting system [17].

Here we have developed and tested our own opensource blockchain application and proof of concept. Network administrators have the ability to issue votes to registered users. Registered users have the ability to redeem votes. Users are only allowed to vote with votes issued to them. We have also addressed current system limitations that may be addressed in future research papers.

1. **Author Contributions**

LT designed, developed and tested the Hyperledger project. JM designed, developed

and tested the Merkle tree implementation in Python. LT and JM have both contributed to writing this paper as well as creating the diagrams contained within.

**References**

[1] A. Cohen, “No One in America Should Have to Wait 7 Hours to Vote,” *The Atlantic*, 05-Nov-2012. [Online]. Available: <https://www.theatlantic.com/politics/archive/2012/11/no-one-in-america-should-have-to-wait-7-hours-to-vote/264506/>. [Accessed: 03-Oct-2019].

[2] R. M. Alvarez, T. E. Hall, and A. H. Trechsel, “Internet Voting in Comparative Perspective: The Case of Estonia,” *APSC*, vol. 42, no. 03, pp. 497–505, Jul. 2009.

[3] S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System,” p. 9, 2009.

[4] S. Haber and W. S. Stornetta, “How to Time-Stamp a Digital Document,” p. 19, 1991.

[5] D. Bayer, S. Haber, and W. S. Stornetta, “Improving the Efficiency and Reliability of Digital Time-Stamping,” in Sequences II, R. Capocelli, A. De Santis, and U. Vaccaro, Eds. New York, NY: Springer New York, 1993, pp. 329–334.

[6] Merkle R.C., “A Digital Signature Based on a Conventional Encryption Function,” in Pomerance C. (eds) Advances in Cryptology – CRYPTO ’87. CRYPTO 1987, Springer, Berlin, Heidelberg, 1987.

[7] L. Lamport, R. Shostak, and M. Pease, “The Byzantine Generals Problem,” *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, pp. 382–401, Jul. 1982.

[8] S. Gilbert and N. Lynch, “Brewer’s conjecture and the feasibility of consistent, available, partition-tolerant web services,” SIGACT News, vol. 33, no. 2, p. 51, Jun. 2002.

[9] S. Bistarelli, M. Mantilacci, P. Santancini, and F. Santini, “An end-to-end voting-system based on bitcoin,” in *Proceedings of the Symposium on Applied Computing - SAC ’17*, Marrakech, Morocco, 2017, pp. 1836–1841.

[10] A. Ben Ayed, “A Conceptual Secure Blockchain Based Electronic Voting System,” *IJNSA*, vol. 9, no. 3, pp. 01–09, May 2017.

[11] “Hyperledger – Open Source Blockchain Technologies,” *Hyperledger*. [Online]. Available:<https://www.hyperledger.org/>. [Accessed: 03-Oct-2019].

[12] “The Go Programming Language.” [Online]. Available:<https://golang.org/>. [Accessed: 03-Oct-2019].

[13] “Welcome to Python.org,” *Python.org*. [Online]. Available:<https://www.python.org/>. [Accessed: 03-Oct-2019].

[14] “Enterprise Container Platform,” *Docker*. [Online]. Available:<https://www.docker.com/>. [Accessed: 03-Oct-2019].

[15] “Voting methods and equipment by state,” *Ballotpedia*. [Online]. Available:<https://ballotpedia.org/Voting_methods_and_equipment_by_state>. [Accessed: 14-Nov-2019].

[16] “Commercial paper tutorial — hyperledger-fabricdocs master documentation.” [Online]. Available: <https://hyperledger-fabric.readthedocs.io/en/release-1.4/tutorial/commercial_paper.html>. [Accessed: 05-Dec-2019].

[17] D. Balzarotti *et al.*, “Are Your Votes Really Counted?: Testing the Security of Real-world Electronic Voting Systems,” in *Proceedings of the 2008 International Symposium on Software Testing and Analysis*, New York, NY, USA, 2008, pp. 237–248.

[18] M. Piras, “LONG LINES AT THE POLLS VIOLATE EQUAL PROTECTION AND REQUIRE JUDICIAL AND LEGISLATIVE ACTION,” *University of St. Thomas Law Journal*, p. 24.