BroncoVote: Secure Voting System using Ethereum's Blockchain

Gaby G. Dagher[1], Praneeth Babu Marella[1], Matea Milojkovic[2] and Jordan Mohler[3]
[1]*Boise State University, Boise, Idaho, U.S.A.*
[2]*Winthrop University, Rock Hill, South Carolina, U.S.A.*
[3]*University of Denver, Denver, Colorado, U.S.A.*

Keywords: Blockchain, Ethereum, Smart Contracts, Voting, Privacy.

Abstract: Voting is a fundamental part of democratic systems; it gives individuals in a community the faculty to voice their opinion. In recent years, voter turnout has diminished while concerns regarding integrity, security, and accessibility of current voting systems have escalated. E-voting was introduced to address those concerns; however, it is not cost-effective and still requires full supervision by a central authority. The blockchain is an emerging, decentralized, and distributed technology that promises to enhance different aspects of many industries. Expanding e-voting into blockchain technology could be the solution to alleviate the present concerns in e-voting. In this paper, we propose a blockchain-based voting system, named BroncoVote, that preserves voter privacy and increases accessibility, while keeping the voting system transparent, secure, and cost-effective. BroncoVote implements a university-scaled voting framework that utilizes Ethereum's blockchain and smart contracts to achieve voter administration and auditable voting records. In addition, BroncoVote utilizes a few cryptographic techniques, including homomorphic encryption, to promote voter privacy. Our implementation was deployed on Ethereum's Testnet to demonstrate usability, scalability, and efficiency.

1 INTRODUCTION

The United States of America was founded upon beliefs in individual rights. The right to vote is inherent in the American way of life and, with improvements in technology and concepts such as smart cities becoming a reality, it could be assumed voting has become easily accessible for all individuals and votes are protected. However, even at the university level, voter fraud has been a continual adversary. A scam in 2016 at Kennesaw State University brought the issues of voter registration fraud to the forefront: individuals carried voter registration forms around the university's campus. Students believed they had signed up to vote in the 2016 Presidential Election without knowing their registration forms were simply trashed–until the day of the election when they were unable to cast a vote [1]. The same year city officials in Green Bay, Wisconsin refused to allow early voting on the University of Wisconsin's satellite campus. Instead, the nearest early voting location was a fifteen-minute drive from campus and was only open during regular business hours[2]. The difficulty of accessing a voting site excluded many students from having a voice in the 2016 Presidential Election. To add to this frustration, many locations across the country will not accept student identification cards as suitable IDs for voting. Voter registration fraud and lack of access to voting sites for university students are important issues that must be addressed.

Secure and privacy preserving voting systems are necessary for university-scale elections. For instance, at many universities, one of the major objectives of the student government organization Associated Students (AS) is to "advocate for the interests of students at the University". In order to achieve this purpose, students must have easy access to voting polls and the reassurance that their votes will not be tampered with or revealed. E-voting protocols have recently increased in popularity; these systems attempt to address the previously mentioned needs. One such system, TIVI, uses digital authentication of voters

[1]http://bettergeorgia.org/2016/09/11/a-different-kind-of-voter-fraud-one-to-actually-be-worried-about/

[2]https://www.thenation.com/article/city-clerk-opposed-early-voting-site-at-uw-green-bay-because-students-lean-more-toward-the-democrats/

Dagher, G., Marella, P., Milojkovic, M. and Mohler, J.
BroncoVote: Secure Voting System using Ethereum's Blockchain.
DOI: 10.5220/0006609700960107
In *Proceedings of the 4th International Conference on Information Systems Security and Privacy (ICISSP 2018)*, pages 96-107
ISBN: 978-989-758-282-0

through facial biometrics: specifically, selfies [3]. Although TIVI solves the accessibility issue previously mentioned, it does not completely stop fraudulent activity. Using public photos and 3-D rendering, malicious users are able to break into accounts [4]. Helios is the first online, open-audit voting protocol. Helios' first priority is data integrity; then it addresses voter privacy. To ensure data integrity, any observer may audit the election process at any time during the election. Although an individual's name is initially posted along with the individual's encrypted vote, after the election closes, the votes are shuffled and then the result is computed. Helios claims to be the optimal voting system for small groups where coercion is unlikely but private voting is necessary (Adida, 2008). Although Helios maintains data integrity, voter privacy is not preserved to the utmost. Another major limitation associated with current e-voting systems is voting fraud in the form of database/platform manipulation (Tarasov and Tewari,). Current e-voting protocols are prone to vote manipulation due to their centralized nature. Our solution to the security concerns of current e-voting systems uses the blockchain. Voting systems using the blockchain do not have a central point of failure due to blockchain being a distributed system (Atzori, 2015). Therefore, through the blockchain, users can confirm none of the votes were tampered with and the final count is valid (FMV, 2016).

The blockchain is a public ledger that operates without a central authority. To ensure data integrity, all the nodes on the blockchain verify and store every transaction. Users create transactions which are then gathered into blocks by "miners." In order for a miner to append his block to the blockchain, he must complete a proof such as a Proof-of-Work or Proof-of-Stake. Due to the append-only structure of the blockchain and the computational power needed to add a block to the chain, the majority of the computational power on the network (at least 51%) would need to collude in order to rewrite a part of the blockchain. Because of these properties, the blockchain is considered an immutable, secure data structure. The Ethereum Blockchain expands this functionality by implementing smart contracts (Buterin et al., 2013).

Smart contracts are blocks of code that are stored on the blockchain. Smart contracts consists of functions or events that allow contracts to interact with each other and users. Since these smart contracts are stored on the blockchain, the code is not modifiable and is available for use by nodes connected to the blockchain. To protect the system against malicious users and compensate miners for computational power usage, the execution of every transaction includes a transaction fee, referred to as "gas" in Ethereum. Gas is the unit of measure for the amount of work that is accomplished for an operation and the gas price is measured in terms of ether in Etherem (Buterin et al., 2013). Smart contracts also extend the use of private blockchains; as opposed to public blockchains, private blockchains are only accessible by one organization. While this sacrifices part of the blockchain's decentralization property, it enhances the privacy of the blockchain (Buterin, 2015). Our system for handling university voting, BroncoVote, implements a private blockchain. We believe a private blockchain is suitable based on the needs for the integrity and privacy of ballots.

Our proposed system uses similar concepts as (McCorry et al., 2017), (Andrew Barnes and Perry, 2016), and (Ernest, 2014), specifically in the areas of privacy and smart contracts. All three of the systems in these articles encrypt ballots stored on the blockchain to ensure voter privacy. The systems also utilize hashing to ensure strong data integrity. In (McCorry et al., 2017), the voting system may have an optional round in which voters hash and post their encrypted vote to the blockchain. Transactions consisting of votes are hashed before being stored on the blockchain in the system described in (Andrew Barnes and Perry, 2016). Furthermore, (McCorry et al., 2017) employs smart contracts to ease the voting process: one to oversee the election process and one to aid in the cryptography process.

1.1 Contributions

Our implemented system, BroncoVote, provides a secure and private e-voting system that is also easily accessible. BroncoVote is a university scale voting system that utilizes smart contracts in Ethereum and Paillier Homomorphic Encryption to achieve our goals. Our system also allows for different types of ballots: users have the freedom to create polls or elections as well as have the option to choose who can vote on their ballot. BroncoVote provides voter privacy on all our ballots by encrypting every vote, homomorphically tallying, and revealing the vote count using Paillier cryptosystem decryption process. To maintain data integrity, all ballot and voting data is publicly available as part of the smart contracts or blockchain in our system. Congruent to the goals of smart cities, this implementation of a blockchain-based voting sys-

[3] https://eandt.theiet.org/content/articles/2016/10/voting-online-made-possible-with-selfie-recognition-technology/

[4] https://www.wired.com/2016/08/hackers-trick-facial-recognition-logins-photos-facebook-thanks-zuck/

tem further integrates technology into the daily lives of individuals.

2 PRELIMINARIES

2.1 Blockchain Mining

To reach consensus on the state of the blockchain in a trust-less network, a concept known as 'mining' is employed (Nakamoto, 2008). The role of a miner node is to verify transactions, group transactions into blocks, and append them to the blockchain. To append a new block to the blockchain, the hash of the block must begin with a certain number of zeros. To achieve this, a number called a 'nonce' is included in each block; each time miners hash the block without solving the computational problem, they increment the nonce and rehash the block (Nakamoto, 2008). The difficulty of solving the hashing problem is described as 'Proof of Work,' signifying the computational power and difficulty needed to append a new block to the blockchain (Nakamoto, 2008). Because of the computational power needed to mine the blockchain, miners are rewarded: for instance, in BitCoin, when miners successfully append new blocks to the blockchain, they are rewarded with the current payout rate in bitcoins (Nakamoto, 2008).

2.2 *Eth.calls*

Every valid transaction executed is stored on the blockchain (Buterin et al., 2013). Due to this, blockchains can suffer from scalability issues. Valid transactions sent to smart contracts in the Ethereum blockchain are considered state changeable calls and consume gas. To reduce gas consumption and the number of transactions on the blockchain, the Ethereum blockchain allows *eth.calls* to be utilized in addition to transactions. *Eth.calls* allow nodes to send messages to other nodes or smart contracts to retrieve its current state without storing the message on the blockchain [5]. Therefore, *eth.calls* are similar to simulations of transactions. By executing *eth.calls* to send notifications/messages or to retrieve current states, the size of the blockchain can be greatly reduced.

2.3 Paillier Encryption

Full homomorphic encryption enables users to perform computations on encrypted data that can be decrypted and yield the same result as if the computation had been originally performed on decrypted data (Xun Yi, 2014). However, doing fully modular multiplication in fully homomorphic encryption is computationally intensive and very slow (Wang et al., 2015). Nonetheless, because of the advantages of homomorphic encryption, partial homomorphic encryption is a prominent encryption scheme. One such scheme is Paillier Encryption. This probabilistic public-key encryption method supports addition and multiplication (Xun Yi, 2014). Paillier system can homomorphically add two ciphertexts but it can only multiply a ciphertext with a plaintext integer. Since the Paillier system cannot homomorphically multiply two ciphertexts, it is considered partially homomorphic. The process of encryption is not completely intuitive: multiplying ciphertexts is equivalent to adding the plaintexts and raising a ciphertext to the power of another ciphertext is equivalent to multiplying the plaintexts (OKeeffe, 2008). To achieve the advantages of homomorphic encryption without the substantial reduction in processing speed, Paillier Encryption is one of the ideal encryption schemes.

2.4 MetaMask

MetaMask was created to increase the accessibility of the Ethereum blockchain to the average user. A plug-in for Chrome, MetaMask acts as an Ethereum browser, allowing users to manage their Ethereum wallet and interact with decentralized applications and smart contracts without running a full node. Through MetaMask, users are able to manage multiple accounts and easily switch between different networks [5] . In order to allow users the flexibility of using the Ethereum blockchain without running a full node, MetaMask relies on trusted nodes to broadcast the transactions of MetaMask users in order to be mined. Since transactions are signed using the sender's private key, which is stored locally on the user's machine, MetaMask cannot impersonate the user and send transactions on the user's behalf. Acting as an intermediary between Chrome and the Ethereum blockchain, MetaMask allows users the convenience and security of the blockchain within a popular browser.

[5]https://github.com/ethereum/wiki/wiki/JSON-RPC

[5]https://github.com/MetaMask/metamask-extension

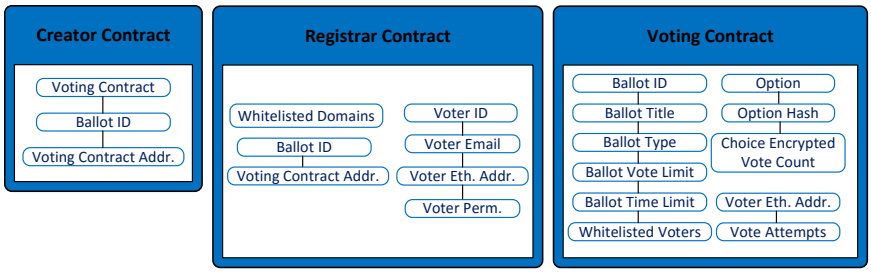

Figure 1: Memory field structure of smart contracts in BroncoVote, where lines between fields represent relational data.

3 PROPOSED SOLUTION: BroncoVote

3.1 Overview

Preceding the introduction to our voting system, it merits mentioning that the Ethereum protocol utilized as part of our system has not been modified in any way. Our system, BroncoVote, uses existing functionality and features provided by Ethereum to provide the ability for creating and voting on ballots. Our implementation consists of three smart contracts coded in Ethereum's Solidity language, two scripts written in JavaScript, and one HTML page. BroncoVote is an open source project and the entirety of the code is available for public use [6].

We assume the administrator, creators, and voters have the MetaMask plugin downloaded in their browser or running an Ethereum node to create and manage Ethereum accounts as well as interact with our system. We utilize Ethereum's Web3 framework internally, this allows our users to easily manage signed transactions and interactions with the Ethereum blockchain. Using MetaMask and Web3 eliminates the need for users to download full or even partial Ethereum blockchains on their local machines in order to broadcast transactions. The only action required of users when registering, voting, or creating ballots is to use their passwords to unlock their Ethereum accounts in the MetaMask plugin and securely interact with the blockchain. If the user decides not to utilize the Metamask plugin then they are responsible for running a node on their local machine and syncing it with the blockchain to interact with our system using Web3.

A brief description of all the user parts of BroncoVote follows:

- **Administrator** is responsible for deploying the initial Registrar and Creator smart contracts. The administrator also has the ability to grant or revoke ballot creation permission for registered voters/creators.
- **Voter** registers in our system with a valid student/employee ID and e-mail address to vote on given ballot ID numbers.
- **Creator** is a voter with ballot creation permission.

A brief description of the front/back-end pages implemented in BroncoVote follows:

- **VoteUI.html** page is the user interface for our users. This page allows users to enter necessary information for each of the different use cases. Once the user enters the necessary information, the corresponding click buttons will invoke functions in *App.js*.
- **VotingApp.js** gathers information from *VoteUI.html* and interacts with *Crypto.js* and the Ethereum Blockchain. For each corresponding request from *VoteUI.html*, it utilizes *eth.calls*, *Crypto.js* server calls, and Ethereum transactions to verify, encrypt/decrypt votes, and store ballot/vote information.
- **Crypto.js** acts as a cryptographic server. All votes are encrypted, homomorphically added, and decrypted using the Paillier homomorphic encryption system key pair in this server.

A brief overview of the smart contracts implemented in BroncoVote follows:

- **Registrar.sol** acts as the record and gate keeper. It keeps track of all registered voters and creators, ballot IDs, voting contract addresses, and whitelisted e-mail domains. As we can see in Figure 1, information regarding the voter and different ballots are linked together in the contract. This allows the contract to perform voter verification, permission modification, and *Voting.sol* address retrieval. The owner of this contract is the administrator.
- **Creator.sol** acts as a spawner for different *Voting.sol* contracts. The Creator defines the voting

[6] https://goo.gl/nqBpzM

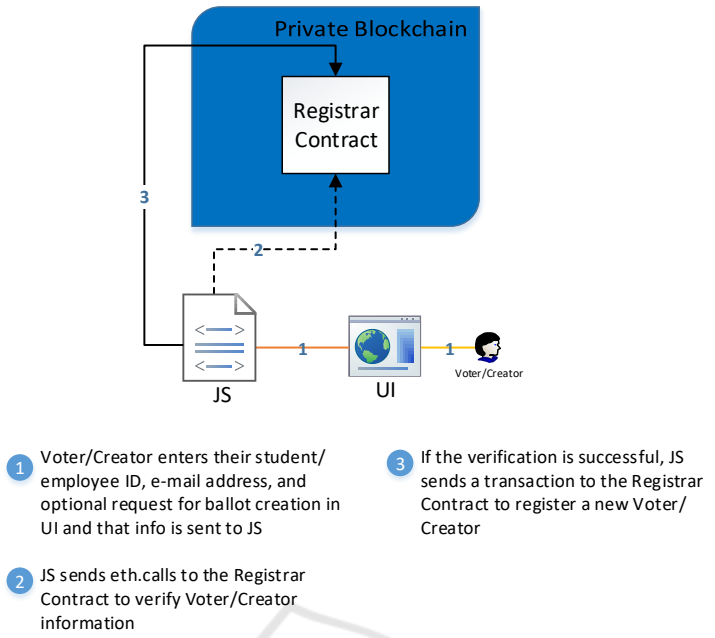

Figure 2: The process for registering a voter in BroncoVote, where black dotted line represent eth.calls and solid line represent transactions to the blockchain.

contract's details when filling out the required information in *VoteUI.html*. The owner of this contract is the administrator.

- **Voting.sol** acts as a virtual ballot and regulates the voting on the ballot. Another set of voter verification, that includes vote attempts and ballot time limit, is also conducted in this contract. As we can see in Figure 1, ballot title and the choice encrypted votes are also stored here so that we can retrieve at later stages. The owner of this contract is the contract's creator.

3.2 Initial Setup

The administrator is responsible for the initial deployment of both the *Registrar* and *Creator* contracts to activate the system and enable users to start registering, voting, and creating new voting contracts. When deploying the Registrar Contract, the administrator is also responsible for whitelisting a set of e-mail domains that are allowed to register to be part of the voting system.

3.3 Register Voter

BroncoVote was created for a university setting. Therefore, anyone with a student/employee ID number and an e-mail with the whitelisted domain is allowed to register as a voter. When the voter completes the ID and e-mail field in *VoteUI.html*, then the information is sent to *VotingApp.js*. As we can see in Figure 2, the *VotingApp.js* makes *eth.calls* to the *registrar* contract to verify the domain provided is part of the whitelist and if the user has previously registered. If those checks are passed, then *VotingApp.js* sends a transaction to the *registrar* contract to store the new voter information, including the voter's ID, Ethereum address, and e-mail. It links the user's Ethereum address and e-mail address so that they cannot double register. Individuals can also request access to create ballots during the registration process; these requests are planned to be manually processed by the administrator but currently are granted automatically.

3.4 Create Ballot

If the user has permissions to create a ballot, the user is able to spawn a new voting contract by entering the required information in *VoteUI.html*. In order to create a ballot, the creator must provide their registered e-mail address then decide whether to create an election or poll, determine the title of the ballot, voting options, and number of votes allowed per voter. During this process, the creator can also elect to have a whitelisted ballot. If a whitelisted ballot is chosen, the creator enters the list of e-mail addresses allowed to vote on their ballot. If the creator chooses to not make a whitelisted ballot, everyone with a e-mail address that has the whitelisted domain will be allowed to vote. Lastly, the creator sets the end date and time

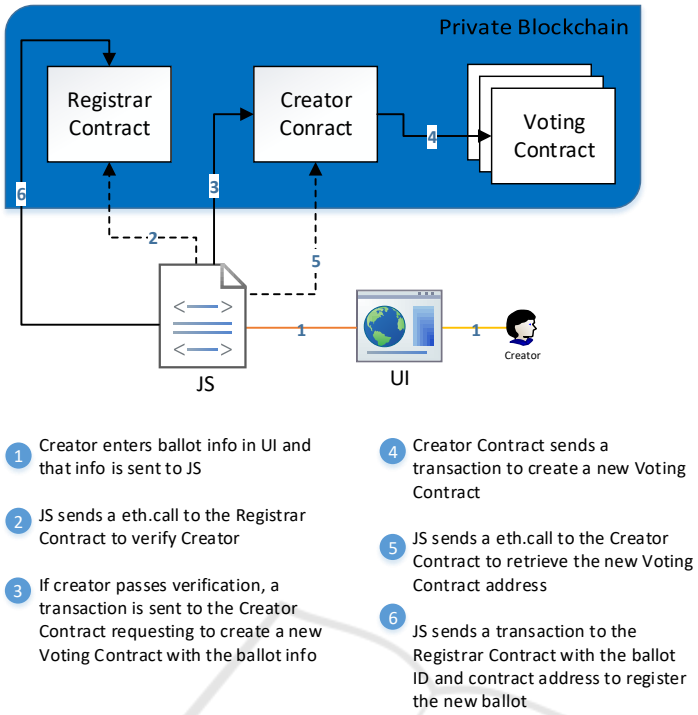

Figure 3: The process for creating a ballot as a creator in BroncoVote, where black dotted lines represent eth.calls and solid lines represent transactions to the blockchain.

of the election or poll.

After submitting this information, *VotingApp.js* utilizes three *eth.calls* to verify the Creator and they are condensed into one step in Figure 3. *VotingApp.js* sends the first two *eth.calls* to the *Registrar* Contract to verify the creator by checking if their e-mail address is registered and if the request originates from the registered Ethereum address. If those two checks are passed, then *VotingApp.js* sends the third *eth.call* to determine if the user has permission to create a ballot. Afterwards, if it was determined the user was allowed to create the ballot, *VotingApp.js* gathers the input data along with a randomly generated ballot ID number and sends a transaction to the *Creator* Contract with a request to create a new *Voting* Contract with the provided information. Once the new *Voting* Contract has been deployed, the contract's address is returned to the *Creator* Contract.

VotingApp.js then sends another *eth.call* to the *Creator* Contract to retrieve the new *Voting* contract address and sends it as a transaction to the *Registrar* Contract to store the new ballot ID and contract address. The ballot ID is then displayed afterwards and the *creator* must write down this ballot ID and pass it along to all the voters in order to let voters vote on the ballot.

3.5 Load Ballot

Using the ballot ID provided by the Creator of the *Voting* Contract, a voter can check the results or vote on the ballot, provided the voting period has not passed. Once the voter enters the ballot ID in *VoteUI.html*, *VotingApp.js* sends an *eth.call* to the *Registrar* Contract to determine the validity of the ballot ID. If the ballot ID is valid, the voting options, title, and encrypted vote count for each choice if the voting period has ended unless it is a poll. If the ballot type was a poll then the results are displayed live. Before the vote count can be displayed, there is another step involved, which can be seen in Figure 4 as step 4, that involves sending the encrypted vote count to the *Crypto.js* server so that we can display the tallied vote for each choice on *VoteUI.html*.

3.6 Vote

Once the ballot has been loaded, the user can vote for a particular choice on the ballot with his/her registered e-mail address. When the voter clicks *vote*, *VotingApp.js* receives the information and sends *eth.calls* to the *Registrar* Contract to verify the voter, it checks the voter registration and Ethereum address. If the voter is verified, an *eth.call* is sent to the *Voting* Contract to check whether the ballot is whitelisted or not.

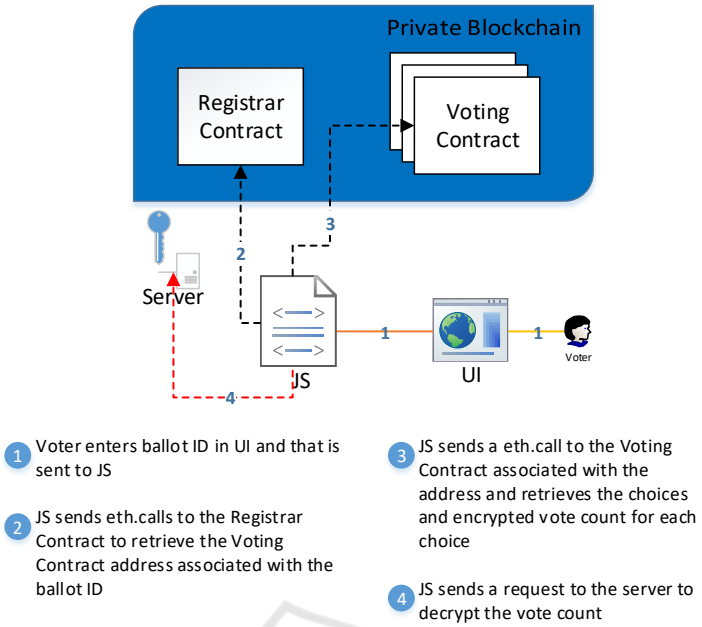

Figure 4: The process for loading a ballot as a voter in BroncoVote, where black dotted lines represent eth.calls to the blockchain and red dotted line represents decryption calls to the server.

If it is whitelisted ballot, then it will check to see if the voter's e-mail is part of the whitelisted voter list. Afterwards, the voting choice is verified and the number of vote attempts made by the voter as well as time limit on the ballot, which is checked by comparing the end time with the current block timestamp, are checked; if the user has not met the limit and the ballot voting period has not ended, then the vote is passed in as a 1 for the chosen vote option chosen and 0s for the rest of the voting options. As we can see in Figure 5, the current vote is encrypted first so that we can retrieve the encrypted vote count from the *Voting* Contract and homomorphically add them to get the new total vote count. So the vote is sent to *Crypto.js* server to be encrypted using the previously generated public key in *Crypto.js* server. Once all the votes have been encrypted, the previously encrypted vote count for every choice is retrieved using an *eth.call*. Then the current encrypted votes and previously retrieved vote count are sent to the *Crypto.js* server to be homomorphically added together. Then the new encrypted vote count for every choice is then sent as an array in a transaction to the *Voting* Contract.

3.7 Get Votes

getVotes acts as a data retrieval function. Whenever a user loads the ballot or successfully votes on a ballot, *getVotes* is invoked in *VotingApp.js*. *getVotes* sends an *eth.call* with the hashed choices to get the current total encrypted votes. Depending on the timelimit and election type, it would either decrypt the votes and display them or display the time when users can check back for the results. To decrypt the votes, *getVotes* sends the encrypted vote count to the *Crypto.js* server to be decrypted by the private key.

4 TESTNET EXPERIMENT ANALYSIS

Our implementation was deployed on the Ropsten Ethereum testnet to collect data on gas and time costs for each use case in our system. Though we envision our system to be part of a private blockchain, we chose to do the experiments on the testnet to simulate a mature blockchain. We focused on gathering gas costs for each process since it can be translated into performance data. This allows us to get estimated performance data in the later stages of our system deployment. We conducted experiments on varying styles of ballots and specified the gas and time costs for every user, including Administrator (A), Creator (C), and Voter (V), involved in our system.

The Administrator deploys the Registrar and Creator contracts as part of the initialization step and their deployment gas costs are shown in Table 6. Registrar Contract gas cost can vary depending on the size of the whitelisted domain list chosen by the Administrator.

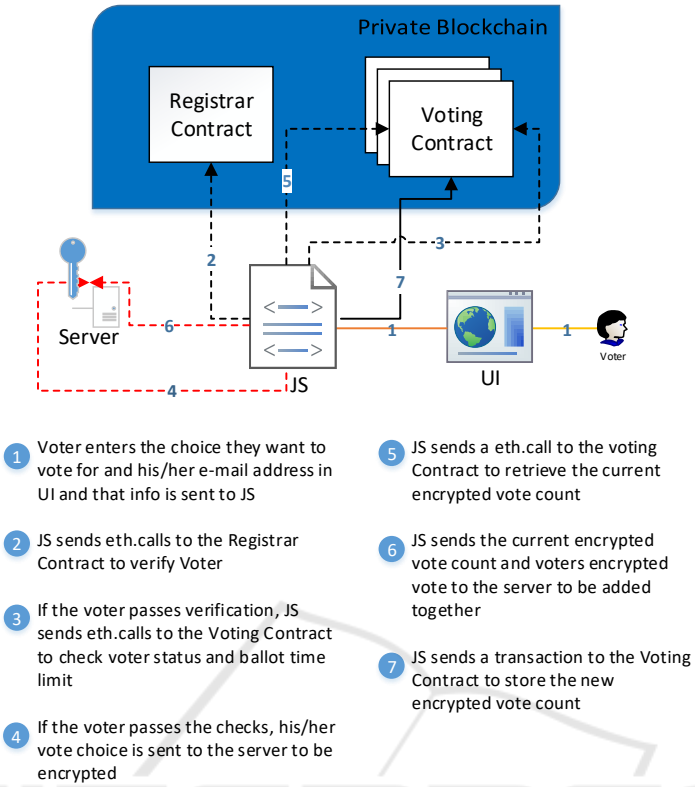

Figure 5: The process for voting on a ballot as a voter in BroncoVote, where black dotted lines represent eth.calls, black solid lines represent transactions to the blockchain, and red dotted lines represent encryption calls to the server.

Contract (User):	Gas Cost:
Registrar (A)	473640
Creator (A)	2142263
Voting (C)	1201820*

Note: * = Base cost of creating a Voting Contract

Figure 6: Gas costs for initial contract deployment to the blockchain for system activation.

Following the initialization step, our next phase in experimentation was creating, loading, and voting on different types of ballots with varying sizes of voting options and whitelisted voters. All the ballots created in our experiments are elections but there isn't a noticeable difference present in cost for choosing polls. The gas costs for those tests can be viewed in Table 7. The number of ASCII characters in the data being passed into the contracts has a noticeable effect on the gas cost so we used an average ASCII character length of 10 for ID numbers, 12 for voting options, and 25 for e-mail addresses. Registering a voter is not dependent on the ballot; therefore, the cost only fluctuates slightly based on the length of the e-mail address as well as the student/employee ID. In the Table, we did not disclose the cost for registering a creator but we did gather the costs of being a creator, and it is approximately 10,000 gas with the same cost variation depending on the length of the e-mail address used. As can be seen in the Table 7, an increase in the number of voting options by two increases the cost for creating a non-whitelisted ballot by approximately 200,000 gas and the cost for voting increases by 10,000 gas per additional voting option.

We created a graph (Figure 8) to show the increase in gas cost for creating and voting on the ballot as voting options on the ballot increase. We used the calculated rate of change in cost as voting options increased to plot the graph and spot checked the calculated results at different voting option counts to verify that the results were close to the expected results. Currently the Ropsten Ethereum testnet has a block gas limit of 4,700,000 gas so we were able to achieve a ballot with max ballot options of 32 without whitelisted voters. If this system was deployed on a private blockchain with modified block gas limit then we could have larger ballots.

We also conducted a time cost analysis on how long, in seconds, each use case would take and the

	Register (V)	Create Ballot (C)	Load Ballot (C/V)	Vote (V)
Ballot Types (Election):		**Gas Cost:**		
1a. 8 Voting Options and 6 Whitelisted Voters	91733	2171399	0	813977
2a. 8 Voting Options and 8 Whitelisted Voters	91348	2220759	0	813913
3a. 8 Voting Options and 10 Whitelisted Voters	91344	2284159	0	811301
1b. 6 Voting Options and 0 Whitelisted Voters	90778	1698226	0	596665
2b. 8 Voting Options and 0 Whitelisted Voters	91302	1911928	0	808597
3b. 10 Voting Options and 0 Whitelisted Voters	91934	2089816	0	1039538
1c. 6 Voting Options and 5 Whitelisted Voters	91458	1942755	0	604221
2c. 8 Voting Options and 10 Whitelisted Voters	90288	2285954	0	811417
3c. 10 Voting Options and 15 Whitelisted Voters	91586	2675374	0	1044726

Note: (A) = Administrator, (C) = Creator, (V) = Voter

Figure 7: Gas costs collected from the conducted experiments on the testnet with varying types of ballots.

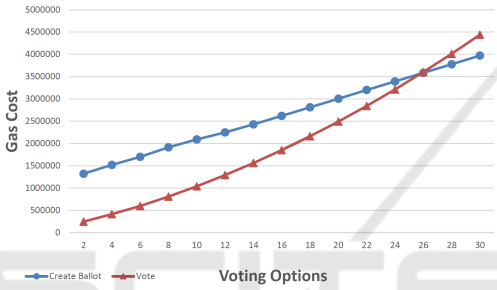

Figure 8: The average change in gas cost based on the number of voting options on the ballot.

results for that are shown in Table 3. We computed the time from when the initial button was clicked until the transaction was mined and verified. We conducted time analysis on the same set of ballots that we created for our gas cost analysis so we could get accurate estimated time costs. The results do vary significantly on each of the use cases that require sending transactions and requires mining due to the mining process. During the mining process the miners can choose which transactions to pick up and the block confirmation time may also vary depending on the hardware used for the mining process. As we can see from the results, the Load Ballot takes the least amount time due to it only using *eth.calls* to retrieve ballot information and not going through the mining process where as Create Ballot takes the longest on average since it requires sending a few transactions to the smart contracts to create and setup ballots.

5 TECHNICAL DIFFICULTIES

Due to the nature of the Solidity programming language, the implementation of BroncoVote encountered a few technical difficulties. One such difficulty is support for cryptography: the maximum data value in Solidity is unsigned int of 256 bit. The majority of encryption protocols require much larger integer numbers than are supported in Solidity. Therefore, BroncoVote' cryptography is completed through a server, which may adds a vulnerability. However, for our purposes, we assumed the server is secure and not compromised. Solidity also lacks proper debugging tools. In order to debug smart contracts, we utilize Remix, an integrated development environment for Solidity [7]. To debug a transaction, Remix uses either the transaction's hash or the transaction's block number and index. From there, Remix provides details regarding the transaction's execution, including local and state variables, storage changes, and return values [7] . Allowing a user to go through each step of the contract's execution expedites the process of debugging and, therefore, the completion of the contract and system.

6 RELATED WORKS

Correctness and privacy has been a predominant concern in election processes. McCorry et. al (McCorry et al., 2017), Zyskind et al. (Zyskind et al., 2015), Barnes et al. (Andrew Barnes and Perry, 2016), Ernest (Ernest, 2014), and Varshneya (A.J. Varshneya, 2015) discuss various methods to utilize the blockchain to ensure data integrity. In (McCorry et al., 2017) zero knowledge proof is implemented to protect data privacy and authenticate users before determining the result of the election. (Andrew Barnes and Perry, 2016) and (Ernest, 2014) apply symmet-

[7]https://media.readthedocs.org/pdf/remix/latest/remix.pdf

	Register (V)	Create Ballot (C)	Load Ballot (C/V)	Vote (V)
Ballot Types (Election):		**Time Cost (seconds):**		
1c. 8 Voting Options and 6 Whitelisted Voters	31.70	48.43	14.71	62.35
2c. 8 Voting Options and 8 Whitelisted Voters	11.42	78.54	13.82	52.07
3c. 8 Voting Options and 10 Whitelisted Voters	39.02	91.88	13.15	77.89
1b. 6 Voting Options and 0 Whitelisted Voters	17.85	73.63	9.47	48.37
2b. 8 Voting Options and 0 Whitelisted Voters	9.90	88.59	11.91	58.35
3b. 10 Voting Options and 0 Whitelisted Voters	39.31	58.58	13.15	78.34
1a. 6 Voting Options and 5 Whitelisted Voters	18.78	55.34	10.59	45.98
2a. 8 Voting Options and 10 Whitelisted Voters	16.65	88.57	12.74	75.40
3a. 10 Voting Options and 15 Whitelisted Voters	33.16	51.32	14.36	68.57

Note: (A) = Administrator, (C) = Creator, (V) = Voter

Figure 9: Time costs (in seconds) collected from the conducted experiments on the testnet with varying types of ballots.

ric encryption to data; additionally, data is segmented on the blockchain in (Andrew Barnes and Perry, 2016). (A.J. Varshneya, 2015) analyzes two separate voting systems, Follow My Vote and BitCongress. Although the Follow My Vote protocol has not yet been integrated with the blockchain, this online voting platform encrypts data with symmetric encryption. In Follow My Vote, users are only identifiable on the blockchain through addresses and are therefore essentially anonymous; however, central authorities such as the government are able to identify users. BitCongress relies on proof of work and proof of tally to maintain data integrity. To authenticate a voter, users sign their votes with their digital signatures before encrypting the vote with their candidate's public key. BitCongress's Whitepaper (Rockwell, 2015) details the system's privacy features. For instance, although the action of sending a vote to a candidate is public, other nodes in the system are not able to trace the vote or election to any particular node. New key pairs are also generated for each election to increase the difficulty of data forensics. BroncoVote applies partial homomorphic encryption to secure the privacy of voters and their votes on the blockchain.

(Zyskind et al., 2015) introduces a peer-to-peer network called Enigma. This network connects to the blockchain and retrieves private and computationally intensive data from the blockchain and stores these records off-chain. A system utilizing Enigma has three decentralized databases: the blockchain's public ledger, a distributed hash table to store off-chain encrypted data, and multi-party computation that distributes randomly partitioned data among random nodes. Secure multi-party computation is applied to create data queries without revealing raw data to nodes in the network. In a multi-party computation, data is randomly split among a random set of nodes in the system and these nodes process their piece of data without sharing information among each other. Data integrity and privacy is enhanced by the Enigma network and its multi-party computations by dividing the processing of data among a random subset of nodes in the system, safeguarding access to the complete, raw information. In order for an information leak to occur, collusion among the majority of the nodes selected to process the data would be required (Zyskind et al., 2015). The private blockchain employed by BroncoVote establishes a closed voting system to protect voters from outside privacy breaches. The homomorphic encryption mentioned above safeguards voter privacy from attacks within the system.

Smart contracts automate the voting process in (McCorry et al., 2017) and (Zyskind et al., 2015). Two smart contracts are implemented in (McCorry et al., 2017): a Cryptography Contract and a Voting Contract. The Cryptography Contract creates the code necessary for zero knowledge proofs while the Voting Contract manages the election process and the verification of the zero knowledge proofs. Because every node in the system runs each smart contract to reach an agreement on the contract's output, voters can rely on this consensus to achieve data integrity instead of executing the code themselves. Similar to smart contracts, private contracts in (Zyskind et al., 2015) are applied to enhance the system's scalability. These contracts are designed to process the system's private information. Three smart contracts are utilized in BroncoVote: a Creation Contract, Voting Contract, and Registration Contract. The Creation Contract establishes the poll or election; once this contract is de-

ployed, it can be used to create multiple, different ballots. The Registration Contract lists the eligible voters; and the Voting Contract allows eligible voters to vote for a candidate.

User interfaces ease a voter's experience. For instance, (McCorry et al., 2017) created three potential HTML5/JavaScript pages voters can access through a browser. This ease of use and access increases the system's probability of adoption. BitCongress (A.J. Varshneya, 2015) utilizes an application called Axiomity as the graphical user interface through which users create elections and vote; Axiomity also keeps a voting record history for users to review on demand. Similarly, voters in the BroncoVote system cast their ballots through an HTML website.

The voting processes in (McCorry et al., 2017), (Rockwell, 2015), (Andrew Barnes and Perry, 2016), (Ernest, 2014), and (A.J. Varshneya, 2015) are described below. The system in (McCorry et al., 2017) executes in five stages. In the first stage, the election administrator creates a list of eligible voters and creates the election, including setting the election's applicable timers, registration deposit, and the optional commit stage. Next, voters register for the election. The next stage is the optional commit stage, where voters store a hash of their vote onto the blockchain before progressing to the stage where they publish their vote and a zero proof of knowledge onto the blockchain. Lastly, the result of the election is computed and revealed. It is important to note that in this system, voters can only vote for two options, typically "yes" or "no" (McCorry et al., 2017).

A similar process is followed in BitCongress (Rockwell, 2015). In BitCongress, every "yes" or "no" token and candidate has an address. Voters send their tokens to a candidate's address; once the election ends and the results are tallied, the tokens are returned to the voters. The system outlined in (Andrew Barnes and Perry, 2016) supports on-line and off-line voting; this system uses two separate blockchains: one to store registered voters and one to store the actual votes. By using two separate blockchains, (Andrew Barnes and Perry, 2016) ensures voter privacy and anonymity. Regardless of how a voter registers (whether on- or off-line), the same information is required such as their social security number and mailing address. Assuming the voter decides to vote on-line, the voter's registration is stored on the blockchain for government miners to verify the voter. Once verified, the voter is sent a ballot card and password to submit a vote, which is stored on the blockchain as a transaction. In both (Ernest, 2014) and Follow My Vote discussed in (A.J. Varshneya, 2015), voters can access and update their vote until the end of the election. Additionally, in Follow My Vote, voters can vote for multiple candidates. An election in BroncoVote is established when an administrator in the system deploys the Creation Contract in order to set up the ballot; this include defining the candidates of the election and the election timer. Next, the administrator defines within the Registrar Contract who is eligible to register. Lastly, the voters cast their ballots through the Voting Contract, which encrypts each ballot to provide security and privacy to the voters. Unlike the systems in (McCorry et al., 2017) and (Rockwell, 2015), BroncoVote allows users to vote for multiple candidates.

BroncoVote is currently a university-scaled voting system employed on the Ethereum Blockchain. Voter privacy is handled by homomorphic encryption; integrity of votes is ensured through passwords. To guarantee voter accessibility, voters cast their votes on an HTML website that can be accessed anywhere with Internet access. BroncoVote also has the ability to be used to conduct polls: similar to elections, polls allow individuals to voice opinions on matters. However, individuals are able to view poll statistics in real-time. BroncoVote is a secure, economical voting system with the potential to be expanded from a university scale to a larger scale.

7 CONCLUSIONS AND FURTHER WORK

In this paper, we have presented a proof of concept system for BroncoVote that utilized the Ethereum blockchain and Paillier homomorphic encryption. Our implementation was tested on the Ethereum testnet network with different types and sizes of ballots. BroncoVote used the smart contracts in Ethereum blockchain to keep a record of every user in our system as well as all the ballots and the information regarding them. We also utilized the smart contracts to achieve access control. We integrated Paillier homomorphic encryption into our system to preserve voter privacy. With the deployment of our system on the testnet for experiments we showed that our system can easily be deployed and setup to use as a voting system for universities or other similar settings.

In future work, we will investigate the possibility of implementing Paillier cryptosystem as a library in Solidity. With the system we currently have, moving the cryptography to a library in Solidity could largely improve our individual ballot verifiability. Having the Paillier library in Solidity would help us generate a

new private and public key for each ballot. This will help us achieve individual voter audit on different ballots without compromising the other ballots. To increase user accessibility, we will also look into integrating the Ethereum Lightwallet into our system will allow users to unlock their accounts in our UI without needing to run a node or plugin. Finally, to help with voter verification, we will try to integrate an API/process that will allow us to check the validity of all e-mails used to register into our system.

This research was supported by the US National Science Foundation (NSF) under grant CNS 1461133 and the Information Security, Privacy, and Mining (ISPM) Research Lab.

REFERENCES

(2016). Blockchain technology in online voting. Web.

Adida, B. (2008). Helios: Web-based open-audit voting. In *USENIX security symposium*, volume 17, pages 335–348.

A.J. Varshneya, Sugat Poudel, X. V. (2015). Blockchain voting.

Andrew Barnes, C. B. and Perry, T. (2016). Digital voting with the use of blockchain technology.

Atzori, M. (2015). Blockchain technology and decentralized governance: Is the state still necessary?

Buterin, V. (2015). On public and private blockchains. *Ethereum Blog*, 7.

Buterin, V. et al. (2013). Ethereum white paper.

Ernest, A. K. (2014). The key to unlocking the black box: Why the world needs a transparent voting dac.

McCorry, P., Shahandashti, S. F., and Hao, F. (2017). A smart contract for boardroom voting with maximum voter privacy. *IACR Cryptology ePrint Archive*, 2017:110.

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system.

OKeeffe, M. (2008). The paillier cryptosystem: a look into the cryptosystem and its potential application. *College of New Jersey*.

Rockwell, M. (2015). Bitcongress whitepaper.

Tarasov, P. and Tewari, H. Internet voting using zcash.

Wang, W., Hu, Y., Chen, L., Huang, X., and Sunar, B. (2015). Exploring the feasibility of fully homomorphic encryption. *IEEE Transactions on Computers*, 64(3):698–706.

Xun Yi, Russell Paulet, E. B. (2014). *Homomorphic Encryption and Applications*. Springer International Publishing.

Zyskind, G., Nathan, O., and Pentland, A. (2015). Enigma: Decentralized computation platform with guaranteed privacy. *arXiv preprint arXiv:1506.03471*.