Internet Voting Using Zcash

Pavel Tarasov
School of Computer Science
and Statistics
Trinity College Dublin,
University of Dublin
Email: tarasovp@tcd.ie

Hitesh Tewari
School of Computer Science
and Statistics
Trinity College Dublin,
University of Dublin
Email: hitesh.tewari@scss.tcd.ie

***Abstract*—Voting systems have been around for hundreds of years and despite different views on their integrity, have always been deemed secure with some fundamental security and anonymity principles. Numerous electronic systems have been proposed and implemented but some suspicion has been raised regarding the integrity of elections due to detected security vulnerabilities within these systems. Electronic voting, to be successful, requires a more transparent and secure approach, than is offered by current protocols. The approach presented in this paper involves a protocol developed on blockchain technology. The underlying technology used in the voting system is a payment scheme, which offers anonymity of transactions, a trait not seen in blockchain protocols to date. The proposed protocol offers anonymity of voter transactions, while keeping the transactions private, and the election transparent and secure. The underlying payment protocol has not been modified in any way, the voting protocol merely offers an alternative use case.**

Index Terms – **Blockchain, EVoting, Zcash, zk-SNARK.**

I. Introduction

With blockchain steadily striving towards becoming the new system for decentralized payment schemes, amongst other implementations, it is easy to imagine why this technology can be considered an ethical liberator in some senses. Blockchain, although a relatively new concept, has gained enough popularity for applications to emerge such as simplified methods for identification and authentication, the widely known decentralized payment scheme, Bitcoin, and domain systems which reside outside the control of the government or non-governmental organisations (NGOs) [1]. The number of blockchain systems is steadily increasing, however the electronic voting domain is slow to adapt with a relatively low number of systems devised so far, based on our observation of the state of the art.

Electronic voting has been a topic of active debate, with significant number of people believing that electronic voting cannot be trusted enough to be used for significant elections due to uncertainty in the authenticity and integrity of the machines and the votes that have been cast using them. On the other hand, people acknowledge that paper solutions are significantly outdated and can be subject to serious manipulation from a coercer. The emergence of blockchains has introduced a new way to construct secure systems which have less inherent security issues present. It is a belief that a successful voting system can be implemented using blockchains, or with a blockchain being one of the main elements present in a hybrid electronic voting scheme [2].

In our work, we investigate a new decentralized, anonymous payment scheme called Zcash [28] and create a voting system without altering the inner working of Zcash protocol.

II. State of the Art

Electronic voting is a topic of much research and several viable schemes have been created in order to attempt and solve the problem. Here, we present some influential voting protocols and other viable voting schemes as well as the techniques they implement at the core of vote processing, their security issues and analysis that have been done on some of the protocols in this domain. Blockchain voting technologies that have emerged recently are also discussed here, with particular attention to *Ethereum* [3].

A. Influential Electronic Voting Protocols

Electronic voting protocols have been implemented in different elections, ranging from university to government based elections. Many viable protocols have been created since Chaum [4] first proposed *Votegrity*, one of the first *end-to-end* (E2E) verifiable voting schemes. E2E verifiability means that the voter can verify that their own vote has been cast as intended. the voter would be the assured that their vote has been counted correctly and included in the final tally and that the public members can verify an election externally without being involved in an election. These voting protocols, also provide a way to audit the voter's votes and the ballots prior to picking the candidate and casting the ballot.

Some of the most prominent examples that have stemmed from Chaums *Votegrity*, which also provide E2E verifiability, are Neffs *Markpledge* [5], *Prêt à Voter* [6], *Helios* [8], *Scantegrity* [9] and *STAR-Vote* [10]. Markpledge was one of the first E2E voting protocols which has been proposed alongside Votegrity, influencing the development of the other schemes mentioned above and more. Helios, a university voting scheme, has undergone security analysis, which uncovered security vulnerabilities with a potential to affect the outcome of the elections. This led to the development of Helios 2.0 [12] and Helios 3.0 versions, attempting to fix the vulnerabilities posted by Estehghari and Desmedt [13]. This is a good example of a security vulnerabilities in a voting protocol. A possible attack on Helios 2.0 included cross-site scripting (XSS) through the usage of a browser rootkit, a script capable of monitoring user traffic, capturing passwords entered by the user and get access to the DOM tree of the web page.

Some E2E protocols use public *web bulletin board* (WBB) for posting all of the cast ballots for the public to see. Web bulletin boards are used as an authenticated public broadcast channels which, display the cast ballots to the public in an encrypted form, and serve as an important stage for any E2E protocol. Typically, after the voter has cast their vote and received a receipt encrypting their choice in a way that is dependent of what voting protocol used, the encrypted vote is propagated to the WBB [15] [7] [11].

The receipt is an important part of the voting protocol, as it allows the user to prove their vote to an authority in case the voter wishes to dispute their vote or prove that they have voted contrary to what the system has recorded. The receipt also allows the user to find their vote and view how the system recorded their vote. These receipts vary from system to system, but typically these receipts are the summary of how the voter voted, which can be presented to the voter in an encrypted or obfuscated manner. As an example, Votegrity summarises the vote in a print out which prompts the voter to pick the top or the bottom layer of the receipt. The receipt is a laminated piece of paper, which is separated into two layers, which are only readable when these layers are combined and never on their own. The mutual relationship of the pixels on the translucent layers is how the vote becomes readable [4].

Some electronic voting protocols implement a challenge system, which helps a voter to establish trust in the system. Apollo [16] is an extension of the Helios protocol, however, it avoids some security issues that are inherent in Helios by having voter assistants to verify, lock and audit the vote. The assistants are external to the voting protocol devices that can interact with the election and can be laptops, tablets, or any other external devices. These interact with the session by fetching the personalised string, input by the voter during the start of the session, to fetch the session. The voter that wishes to audit their vote sends the audit code through the voting booth, which in turn opens the encryption of the ballot by posting the randomness encrypted with the session key. Each voting assistant checks the bulletin board and displays the plaintext value of the vote. This procedure may be repeated as many times as the voter wants [16].

Mixing is one of the two predominant techniques that are used in electronic voting protocols and utilizes mix networks (*mixnet*), a protocol that takes in multiple input messages from the users and shuffles these messages in random order before passing them to the next destination [18]. Mixnets, in the context of voting, are used to provide a degree of anonymity to the user by obfuscating where the message came from. For example, Zeus [17] implements mixing after the election has been closed to break the linkability between the encrypted ballots and the voters who cast them. This is a multi-round procedure which depends solely on the number of mixing proxies available to the system. Each stage of the mixing provides a proof of correct mixing, which can be used to verify that the mixing server is not corrupt.

The second widely used technique is *homomorphic tally*. Cohen and Fischer [14] describe how this can be applied to a voting protocol. Homomorphic tally involves modifications, usually additions and multiplications, to the ciphertext which are preserved upon decryption to reveal the operations that have been done on the ciphertext while recovering the modified decrypted value. Protocols such as Helios 2.0 [12], STAR-Vote [10] and several others implement this technique for tallying the votes due to its simplicity both in application and for verification by the public, though the efficiency of these protocols, over mixnets, have been different through the papers where these methods are used.

Protocols such as Zeus [17] and Apollo [16] use the basis of Helios to build their own voting protocol on, while attempting to tackle some of the security issues that are inherent to Helios. For instance, Apollo tackles the issues of XSS, cross-site forgery, clickjacking and clash attacks with the help of the voting assistants. For example, XSS was possible due to the unchecked URL parameters that meant to obtain the election URL, but if compromised could have pointed to a proxy with malicious script forced to execute on the target machine by the attacker. Ultimately, the attacker could encrypt each choice of the voter correctly, but submit their own ballot instead of the voters when the voter continued to submit their vote. This attack is impossible to detect server-side, but can be detected by the voter if the voter checks the WBB later to find their vote. XSS is in the third place of the top vulnerabilities of web applications as found by OWASP in 2013 [19] and remains in the same position in OWASPs "Top 10 Application Security Risks" draft of 2017.

B. Blockchain For Voting

The conclusion can be made that an electronic voting system must be secure, while allowing for as much transparency as possible to be a working E2E verifiable. Blockchains [21] help to achieve this level of security and transparency, while maintaining privacy and non-malleability of the transactions [22] [23].

Although different, some elements from the above mentioned protocols may apply to the concept of blockchain voting. The notion of WBB, where the encrypted votes can be seen by the public members, can persist in blockchain in the form similar to [24]. Here the blocks of transactions can be observed as well as the height of the blockchain with any other relevant information. Although blockchain is a promising technology, we have not found any relevant papers to date that present a protocol for online voting with blockchains. Examples such as Follow My Vote [25] or TIVI [43] present a seemingly sound voting protocol, however they are presented without any in-depth specification to verify the security of the protocol.

One other noteworthy blockchain technology that could revolutionise electronic voting is Ethereum [3]. Ethereum differs from Bitcoin [21] as it serves as a generic platform

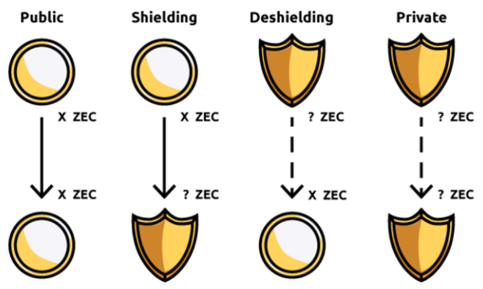

Fig. 1. Types of Transactions in Zcash [29]

for creation of custom functionality in the form of contracts. The currency used by Ethereum is *ether* and *gas*. However, the main difference is the fact that the contracts allow for different functionality using the Ethereum Virtual Machine (EVM), while being enforced by the peer-to-peer, decentralized way, inherent to the core structure of blockchain. Ethereum possesses two types of accounts, which is another way of specifying types of users. Accounts are used by human entities, whereas contracts are accounts which are operated by code on the EVM. Contracts are the agents that bring about the generic functionality of Ethereum and allow one to create custom behaviour for one's blockchain application. These applications include, and are not limited to, automatic payments or creation of custom currency, which is worthless outside of the context of the contract application [3] [26] [27].

III. Zcash Overview

Zcash is a decentralised blockchain payment scheme, which aims to provide anonymity of transactions. One of the biggest differences between Zcash and Bitcoin is the proof-of-work system, where Zcash relies on zero-knowledge proofs [28]. We present a brief overview of the important concepts of Zcash prior to describing details of the proposed voting protocol.

A. Addresses and Transactions

Zcash supports both anonymous and transparent transactions as it has two types of addresses, which differs from the Bitcoins single address. These addresses are, namely, z-address and t-address, where z-address is the address which preserves anonymity in transactions, and t-address resembles the Bitcoins addresses in structure and allows for transparent transactions. The transactions between different addresses ensures the conversion of transparent value into a *shielded* value and vice versa. The details of shielded values cannot be observed by the public. The transactions between addresses is illustrated by Fig.1. Private transactions occur when both, the sender and receiver use z-addresses, which ensures that no entity, outside the entities involved, can view the details and the value of Zcash (ZEC) that are exchanged in the transaction.

The private, z-addresses are generated with the combination of the keys, of which there are a total of 4 keys, which allow for spending, viewing, paying and transmission of secret values between the parties. These keys are namely:

- *Paying key* (a_{pk}): Used as a part to generate payment address.
- *Transmission key* (pk_{enc}): Used to encrypt and decrypt secret values to be passed between the parties involved a in transaction.
- *Spending key* (a_{sk}): Allows spending of ZEC.
- *Viewing key* (sk_{enc}): Establishing keys for viewing the private transaction between involved parties.

The combination of paying key a_{pk} and transmission key pk_{enc} is what makes up a z-address.

Part of spending a ZEC involves revealing a *nullifier* for a ZEC which has a *commitment* in a Merkle tree [21]. The commitment is placed on such tree in Zcash, whenever a new ZEC is generated. A nullifier is a serial value for each ZEC which prevents double-spending of the same ZEC. The spending procedure involves locating a commitment on the Merkle tree and ensuring that the nullifier has not yet been revealed, as once the nullifier is revealed, the ZEC is considered spent.0 The nullifier set is maintained at every full node and newly revealed nullifiers are inserted into the set with each transaction.

A secret pair of keys is established and known as the *ephemeral keys*. The ephemeral keys are established for the transmission of the secret values in private transactions to ensure that only the sender and the recipient can view the transaction. The possession of the private ephemeral key (esk) and the recipient's address is what allows the sender to view the transaction. At the same time, the receiver uses their viewing key (sk_{enc}) and the ephemeral public key (epk) to view the transaction from their end. The ephemeral public key is sent with the transaction, which is the way that the receiver obtains it. Even if a third party obtained this key, they do not have the other keys to view the transaction or derive a key to decrypt the secret values in the transaction.

Part of the transaction, named *JoinSplit Transfer* in Zcash [28], is for the sender to spend their ZEC, which reveals the nullifier for the input ZEC, and generation of the commitments for the new ZEC which will be passed to the receiver as part of the transaction. The values used to generate the ZEC are passed to the receiver after being encrypted with a key established via transmission key (pk_{enc}). These values in a transaction are accompanied, by a zero-knowledge proof to ensure that the transaction is legitimate and follows the rules for a transaction. Finally, the JoinSplit Transfer supports both shielded and transparent values in the same transaction as the transparent value pool in each transaction is dedicated for transparent transactions as well as to hold the miners reward for processing the transaction.

B. Zero-Knowledge Proving System

The key to the private transactions is the zero-knowledge proving system. This is because there is a need to transfer the secret values between the involved parties without disclosing these values to each other. To facilitate this transfer, Zcash implements zk-SNARKs (Zero-Knowledge Succinct

Non-Interactive Arguments of Knowledge) devised into *lib-snark* [30] library from the designs of Ben-Sasson et al. [31] [32]. This construct allows for generation of zero-knowledge proofs given an arbitrary program. The proofs are generated using several steps which are shown in Fig. 2.

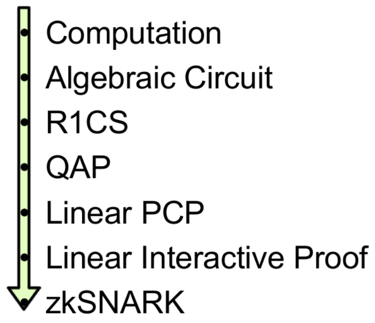

Fig. 2. Overview of zk-SNARK Proof Creation [37]

For this paper, we will not be discussing these steps in details, rather we provide an overview of the zk-SNARK functionality for better understanding of its application in Zcash. The purpose of supplying a proof is to verify the legitimacy of secret values, which are used to generate a ZEC, exchanged during a transaction. Libsnark allows the conversion of programs into proofs of knowledge. The program utilizes a port for a GCC compiler to create a circuit based on monitoring the execution of a program.

The core idea is that some universal program circuit is used as input to a generator function, with secret values acting as *toxic waste* to generate a pair of public keys, which can be distributed to both the proving and the verifying parties. The prover is to use the proving key, public value to be proven and a secret value (*witness*) as input to a function which generates a proof from the above information.

The verifier obtains the verifier key as well as the proof generated by the prover and using a verification function, whose inputs are verification key, the same public value to be proven and the proof provided by the prover to verify the legitimacy of the proof. The verification function is a simple function which returns either *true* or *false* depending on whether the proof has been successfully proven. [33]

The generator function is part of a setup procedure and uses toxic waste values as part of the setup stage. The setup phase for ZCash is done once to establish the proving and verification keys. If the toxic waste is not deleted and a party was able to obtain these keys, then the said party would be able to generate fake proofs.

The JoinSplit transfer also provides some proofs as part of the transfer is to generate new ZECs. Some of the things that the proof is used to prove are:

- the total values of input ZECs and output ZECs matches.
- the commitments exist and are valid for the input ZECs.
- the nullifier and the commitment have been calculated correctly.

The proofs are not limited to these three items and the total size of the resulting proof is 296-bytes [28].

IV. System Overview

Prior to describing our voting protocol, it is worth mentioning that the underlying Zcash protocol [28] has not been changed in any way. The protocol utilizes basic functions offered by Zcash and creates a platform with the ability to cast votes. The following things are assumed by the work: assumption of confirmed identity, where the protocol assumes that the identity of a potential voter can be verified, such as employing X.509 certificates [34] and Certificate Authorities (CA) to verify those identities. This is to facilitate legal authorisation of the vote transaction on behalf of the voter. The voting protocol can be separated into four distinct steps: registration, notification, voting, count/audit.

A. Registration

Registration is the first step of the protocol and is required as part of the identity verification step and for audit purposes, to keep track of which voters have cast a ballot, and is a control mechanism to disallow unregistered people to participate in the vote. A potential voter who wishes to participate in an election or a poll visits the registration page, where communication with the server is established transparently. The system needs to authenticate a potential voter and can do so by following the *Challange-Handshake Authentication Protocol* (CHAP) [35] and exchange challenge information and solution.

After successful registration, the voter's email address or an X.509 certificate containing their email address is stored in the database used by the voting system. The overall registration step can be seen in Fig. 3.

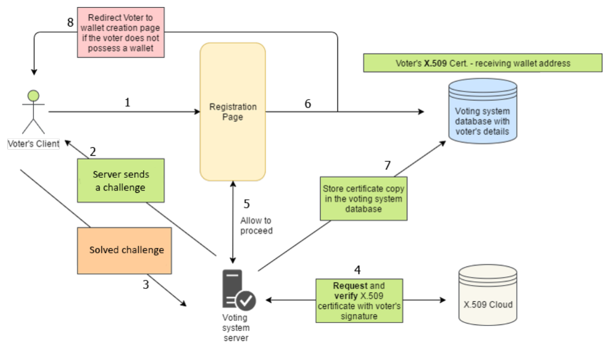
8
Redirect Voter to wallet creation page if the voter does not possess a wallet
Voter's X.509 Cert. - receiving wallet address
1
Registration Page
6
Voting system database with voter's details
Voter's Client
2
Server sends a challenge
7
Store certificate copy in the voting system database
5
Allow to proceed
Solved challenge
3
4
Request and verify X.509 certificate with voter's signature
X.509 Cloud
Voting system server

Fig. 3. Voter Registration

B. Invitation

Invitation step is a small step which sends a one-time unique link to the voter's email to redirect the voter to a unique ballot assigned to them. The voting server issues the invitations only when the administrator of the election issues the details for the election. This is similar to the Zeus protocol [17] where the administrator too, inputs the details of the poll as well as the list of the registered users.

The system server obtains the voter details from copies of stored user verification data, such as X.509 certificates, in the server's database and issues the vote links to all the registered voters. The links are active for as long as the election and expire as soon as the election timer has expired. The invitation step can be seen in Fig. 4.

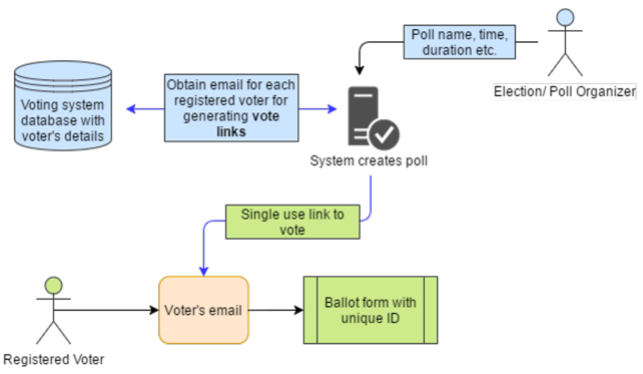
Poll name, time, duration etc.
Election/ Poll Organizer
Voting system database with voter's details
Obtain email for each registered voter for generating vote links
System creates poll
Single use link to vote
Voter's email
Ballot form with unique ID
Registered Voter

Fig. 4. Invitation to Participate in Ballot

C. Voting

Once the voter has followed the ballot link, they are redirected to the ballot page. The ballot is a simple interface which contains candidates names and a checkbox next to the names. The top of the ballot contains a field which requires the voter to input their receiving t-address. These addresses are generated by the voter to send and receive the tokens. To maintain anonymity, but at the same time adhere to the transparency of the vote, the voter is required to provide a receiving t-address and is required to send the vote with a z-address.

The receiving t-address is provided by the voter on the top of the ballot and can be changed as many times as required by the voter. This is the address which will ensure that the voter receives the vote token which is redirected to the candidate wallet. The z-address is used by the voter to ensure that their vote is anonymous.

Once the authorisation takes place, the vote tokens can be generated by the system faucet to send to a ZEC pool, or if there are enough ZECs available, issue them straight from the ZEC pool. The ZEC pool is a system wallet which issues ZECs to the voters once the voter has authorised the vote. Once the voter has authorised the vote, the system changes the voter's status in the database, as well as incrementing the system count of the total votes for the current election. At the same time a token is sent to the receiving address specified by the voter. The number of issued tokens is tracked by the system and is compared to the total number of votes to ensure that no extra votes have been added into the tally. Fig. 5 outlines the steps taken when the voter casts their vote.

D. System Variants

The transaction between the candidate and the voter becomes private if the candidate uses z-address. Inherent to the

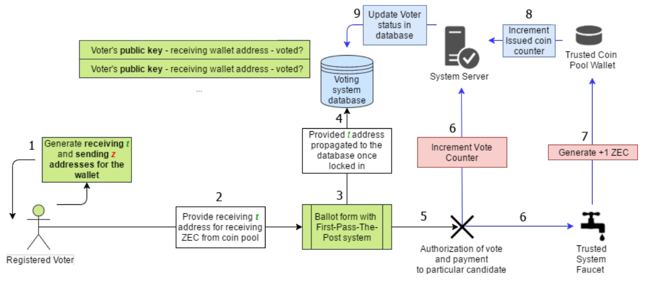
9
Update Voter status in database
8
Increment Issued coin counter
Trusted Coin Pool Wallet
System Server
Voter's public key - receiving wallet address - voted?
Voter's public key - receiving wallet address - voted?
...
Voting system database
4
Provided t address propagated to the database once locked in
6
Increment Vote Counter
7
Generate +1 ZEC
1
Generate receiving t and sending z addresses for the wallet
2
Provide receiving t address for receiving ZEC from coin pool
3
Ballot form with First-Pass-The-Post system
5
6
Registered Voter
Authorization of vote and payment to particular candidate
Trusted System Faucet

Fig. 5. Overview of the Voting Process

Zcash protocol, z-addresses break the linkability between the ZECs and previous transaction. This means that when the candidate empties their wallet into the ZEC pool, the voter may no longer trace their vote to the ZEC pool. This scheme requires more trust in the system, however it guarantees the privacy of the system i.e. no one can see the details and amounts of the transaction sent to the candidate.

Since private transactions require more complicated setup, there are more internal steps involved in making these transactions. One of the most important pieces of information is the establishment and sharing of *ephemeral keys*. These keys allow the voter and the candidate both to view the transaction, which is exclusive to the two parties. These keys are established as per key agreement function of Zcash. Internally, the transaction remains the same.

The second variant of the system involves the candidate's receiving with their t-addresses. This is an example of a deshielding transaction and would mean that the candidate's token balance can be observed by the public in real-time. The linkability of the tokens would also be preserved. Linkability in this case means that a token can be traced back to the sender to the ZEC pool, where the tally occurs after an election timer has expired. This can help users determine if their vote has been counted in the tally.

Regardless of the variant used, the JoinSplit transfers of vote tokens from voters to candidates are stored on the blockchain with the appropriate data for each JoinSplit transfer.

E. The Tally/Audit

The final stage of the voting protocol is the vote count and the audit which takes place after the count to review the election process and ensure that the integrity of the election has not been compromised. The candidate wallets send all the ZEC vote tokens to the ZEC pool which has ZEC balance of 0 ZEC vote tokens. This requires some trust in the system, however the assumption is that the candidate wallets and a ZEC pool have 0 ZEC vote tokens in the beginning and that candidate wallets send all the collected ZEC vote tokens into the coin pool. The transactions may be more difficult to verify as these are private and the details are only available to the voters and the candidates only. However, if the same party who starts an election holds the ownership of the candidate wallets and may implement verification systems to check each transaction destined for each candidate.

The candidate wallets send all their acquired ZEC vote tokens to the ZEC pool using sending t-address on the candidate's side and a receiving t-address on the side of the ZEC pool. The system declares the end of the election or a poll as soon as the expiry time has been met. After that, no votes are accepted into the count and the system, the unique vote links expire and the ballot forms do not allow to proceed with the submission of the votes.

At this point the number of total votes cast becomes public as well as the number of tokens issued for the voters. The total number of transactions may also be displayed with the total number of voters who participated in the election. It is not in the interest of the candidates to not empty their wallets upon conclusion of the election, or to send an incorrect number of ZEC vote tokens as the system equations will not balance and the election will be considered forfeit. Fig. 6 provides an example election with 100 total votes being cast between candidate X and candidate Y.

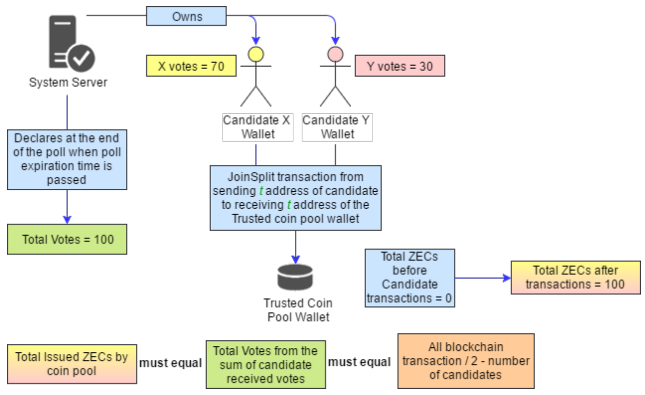
Owns
System Server
X votes = 70
Y votes = 30
Candidate X Wallet
Candidate Y Wallet
Declares at the end of the poll when poll expiration time is passed
Total Votes = 100
JoinSplit transaction from sending t address of candidate to receiving t address of the Trusted coin pool wallet
Total ZECs before Candidate transactions = 0
Total ZECs after transactions = 100
Trusted Coin Pool Wallet
Total Issued ZECs by coin pool
must equal
Total Votes from the sum of candidate received votes
must equal
All blockchain transaction / 2 - number of candidates

Fig. 6. Example of the count and audit of an election

V. Security Considerations

A significant issue with internet voting protocols are compromised voting machines. Since the target platform of the protocol would be user's end devices, such as computers and mobile devices, it is possible for a coercer to influence the outcome of the vote by compromising the voter's device as it would be much easier to achieve than compromising the entire electronic voting scheme. The coercer could infect the voter's machine and influence the voting software installed. The voting software will then be influenced by the coercer's candidate choice. One of the possible ways that a concerned voter can defend against such an attack is to obtain a checksum of the voting application [36]. A checksum can simply be a hashing of the voting software of a specific version which the voter has installed on their device. If the voter's device is compromised then the hash versions will not be the same and the voter can obtain a new copy of the software.

Since our proposed voting protocol does not make any changes to the underlying Zcash protocol, some problems, like double voting i.e. using the same granted vote token to vote for multiple candidates, is inherently absent in the voting protocol. However, since the unique ballot link is sent to a voter's email address, the issue of compromised machine can persist once again. A potential coercer could get access to the voter's email first and attempt to cast a vote on their behalf. Notification systems can be in place to send email confirmation when a vote has been issued by the voter and visually notify the voter of the number of times they have attempted to vote so far.

A major consideration in dealing with Zcash and the automated script assumption is that all the operations deal with ZECs, which have a non-negligible value on the market [38]. This gives a potential incentive for corrupt voters to attempt to hijack the vote token upon brief arrival to their wallet. The assumption is that the script can detect the specific transaction arriving into the wallet and redirecting it to a candidate immediately. There are several mitigations to avoid ZEC hijacking. First is to deal with the smallest denominations of ZEC (1 zatoshi) to reduce the incentive to steal a whole ZEC as 1 ZEC is 10^8 zatoshis [28]. Though the audit calculations for the end of the election may not fail, a rogue transaction to a wallet, not belonging to a candidate may be noticed by the public.

Having mentioned the required balance of values at the end of an election in order to verify it's integrity, a possible attack could be carried out on the system, where a losing candidate does not submit all of the received votes. This would cause the election to be forfeit as the total number of ZEC vote tokens does not balance with the total number of votes and ZEC vote tokens issued. This attack could be detected if the candidates were using t-addresses as all the receiving transactions would be visible, however it would pose a problem if the candidate used z-address as no public party, except the administrators of the voting system would know if a candidate is misbehaving. A possible mitigation for this attack can disregard the total number of ZEC vote tokens returned back to the counting pool, only if this number is less than or exactly equal to the number of total votes. In case of this occurrence, the voters and the administrators can be notified by the system that the votes returned did not match the total number of votes issued.

An alternative solution can implement internal system trackers, which count the number of votes cast for each candidate, and serve the purpose of controlling the amount of ZEC vote tokens returned by the candidates. A tracker for each candidate increments each time a vote has been cast for a candidate and the system expects to withdraw this amount of ZEC vote tokens from the candidate wallet, which would not let a malicious candidate trick the system. These trackers would function even if the candidates used z-addresses. It is also possible to make this tracker public, during the tally period, to notify the public what the expected vote count is.

A question may arise, of what would happen if the system counters have been modified by an attacker. According to the rules of the system, the integrity of the election will be considered compromised and the result will be forfeit. The reality of a decentralised system is that there may be more than one instance of the tracker initialised at a given time, and it may be required that they all need to agree at the end of an election.

One significant attack on the entire blockchain is called 51% attack [39]. This is one of the biggest flaws in blockchain technology. This attack allows an entity with the biggest contribution to block mining to be able to change the contents of the past blocks on the blockchain due to the sheer computer power available to the entity. Other activities would include prevention of some transactions from obtaining a required number of confirmations and preventing people from sending ZEC vote tokens to the candidate addresses. This attack would be difficult to prevent. On one hand, it is possible to pick out several trusted verifiers out of the public volunteers and allow them to confirm the transactions to be included in the blocks. On the other hand, there may be trust issues raised by the participating voters. One other option is to allow any willing public member to participate in a verification pool. This would mean that the voter adds their computing power to the pool of other voter's machines to verify the transactions, however this pool would need to be organised by a trusted party whose actions can be verified in case the party is considered rogue.

VI. Future Work

Having outlined the voting protocol and the basis for it's operations, it is important to outline the direction this protocol can take. The Ethereum protocol [3], has been established early on in the work as a potential candidate to become the platform for our voting protocol. One of the reasons for this is that Ethereum supports creation of contracts, which are accounts which are operated by the EVM. These contracts can be used to implement a voting scheme. However, voters anonymity and privacy is an important piece of any voting protocol and is not yet handled by EVM transactions.

Steady advancements in development of the Ethereum platform bring the possibility of creation of this protocol closer. The future Ethereum aims to make use of zk-SNARKs to add privacy and anonymity of transactions. The zk-SNARKs are complex to implement efficiently due to the time taken to generate proofs, which is one of the issues in implementing these today. However, steps towards adoption of zk-SNARKs have already been taken by Ethereum [40].

Finally, steps have been made to integrate Zcash and Ethereum together in projects such as Zcash over Ethereum (ZoE), however these are still at very early stages [42] [41].

VII. Conclusion

A standardised electronic voting solution which would be widely adopted has not yet emerged, and although there are some good candidates, there are inherent security issues which make these protocols unsuitable for elections. The literature identifies a distinct gap in the domain which could be filled by a protocol, using a different technology then the previous protocols. Blockchain offers an inherently more secure platform, and with development of the recent anonymous transaction scheme, namely Zcash, it is finally possible to tackle the anonymity issues of blockchain transactions, which would open a possibility for blockchain voting. Ethereum has offered the smart contract functionality since it first came to pass, however the much-needed anonymity factor has not been present in the protocol so far. The rapid growth of the Ethereum protocol, and it's integration with Zcash will most likely come up with the protocol, suitable for wide-spread, cheap voting system. As indicated by the future work on these protocols, voting on blockchain has received the much-needed push in the right direction.

The applications for the proposed protocol are not limited to government elections only. These can be stretched to opinion polls or corporate elections providing a unified platform for voting regardless of the cost or circumstance. The drive behind a cheaper, unified, electronic voting system was the basis for the above protocol, which has potential to grow into a real wide-spread implementation, dealing with assumptions and concerns which limit the current system.

The standardisation or adoption of such protocol would be a step towards public approval of electronic voting schemes, provided that the said protocol is secure and has been tested and tried. The release of new protocols, with security issues does not take steps to progress in public approval and, ultimately, replacement of the paper elections.

A major effort has gone into development of a sound voting system and with the rapid developments of blockchain technology and it's implementations in various fields, one final push is required to bring a sound electronic solution to one of the humanities basic rights - to vote.

References

[1] M. Swan, "Blockchain: Blueprint for a New Economy", *Sebastopol. California. O'Reilly Media*, http://w2.blockchain-tec.net/blockchain/blockchain-by-melanie-swan.pdf, 2015.

[2] D. Bradbury, "How Block Chain Technology Could Usher in Digital Democracy", *CoinDesk*, http://www.coindesk.com/block-chain-technology-digital-democracy/, June 2014.

[3] G. Wood, "Ethereum: a Secure Decentralised Generalised Transaction Ledger", Sante Publique (Paris)., vol. 28, no. 3, pp. 391397, 2016.

[4] D. L. Chaum, "Secret-ballot receipts: True voter-verifiable elections", IEEE Secur. Priv., no. January/February, pp. 3847, 2004.

[5] C. Neff, "Practical high certainty intent verification for encrypted votes", VoteHere Doc., 2004.

[6] P. Y. A. Ryan, D. Bismark, J. Heather, S. Schneider, and Z. Xia, "Prêt à Voter: A voter-verifiable voting system", IEEE Trans. Inf. Forensics Secur., vol. 4, no. 4, pp. 662673, 2009.

[7] J. Heather, "Voter Implementing STV securely in Pr et a Liesl Gretl", 2007.

[8] B. Adida, "Helios: Web-based Open-Audit Voting.", USENIX Secur. Symp., pp. 335348, 2008.

[9] R. Carback et al., "Scantegrity II municipal election at Takoma Park: the first E2E binding governmental election with ballot privacy", Proc. 19th USENIX Conf. Secur., pp. 1935, 2010.

[10] J. Benaloh, M. Byrne, and P. Kortum, "STAR-Vote: A secure, transparent, auditable, and reliable voting system", arXiv Prepr. arXiv , vol. 1, no. 1, pp. 1837, 2012.

[11] C. Culnane, P. Y. a. Ryan, S. Schneider, and V. Teague, "vVote: a Verifiable Voting System", ACM Trans. Inf. Syst. Secur., vol. 18, no. 1, pp. 130, 2015.

[12] B. Adida, O. De Marneffe, O. Pereira, and J.-J. Quisquater, "Electing a university president using open-audit voting: analysis of real-world use of Helios", Electron. voting , no. i, pp. 115, 2009.

[13] S. Estehghari and Y. Desmedt, "Exploiting the client vulnerabilities in Internet e-voting systems: Hacking Helios 2.0 as an example", Proc. 2010 Electron. Voting , no. Section 4, pp. 027, 2010.

[14] J. D. Cohen and M. J. Fischer, “A robust and verifiable cryptographically secure election scheme”, 26th Annual Symposium on Foundations of Computer Science (sfcs 1985). pp. 372382, 1985.
[15] A. Parsovs, “Homomorphic Tallying for the Estonian Internet Voting System”, pp. 111, 2016.
[16] D. Gawel, M. Kosarzecki, P. L. Vora, and H. Wu, “Apollo End-to-end Verifiable Internet Voting with Recovery from Vote Manipulation”, pp. 119, 2013.
[17] G. Tsoukalas, K. Papadimitriou, P. Louridas, and P. Tsanakas, “From Helios to Zeus”, USENIX J. Elect. Technol. Syst., vol. 1, no. 1, pp. 117, 2013.
[18] D. L. Chaum, “Untraceable electronic mail, return addresses, and digital pseudonyms”, Commun. ACM, vol. 24, no. 2, pp. 8490, 1981.
[19] Owasp, “OWASP Top 10 - 2013”, OWASP Top 10, p. 22, 2013.
[20] “Top 10 2017-Top 10 - OWASP”, Owasp.org, 2017. https://www.owasp.org/index.php/Top_10_2017-Top_10
[21] S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System”, Www.Bitcoin.Org, p. 9, 2008.
[22] “Blockchain technology: 9 benefits & 7 challenges”, Deloitte Nederland. https://www2.deloitte.com/nl/nl/pages/innovatie/artikelen/blockchain-technology-9-benefits-and-7-challenges.html
[23] P. Glass, “How secure is blockchain?”, Taylorwessing.com, 2016. https://www.taylorwessing.com/download/article-how-secure-is-block-chain.html
[24] “Bitcoin Block Explorer Blockchain”, Blockchain.info, 2017. https://blockchain.info/.
[25] “The Online Voting Platform of The Future - Follow My Vote”, Follow My Vote, 2017. https://followmyvote.com/.
[26] V. Buterin, “Devcon2: Ethereum in 25 Minutes”, YouTube, 2017. https://www.youtube.com/watch?v=66SaEDzlmP4.
[27] G. Wood, “Ethereum Blockchain Mechanism (Proof of Work)”, I.stack.imgur.com, 2017. https://i.stack.imgur.com/afWDt.jpg.
[28] D. Hopwood, S. Bowe, T. Hornby, and N. Wilcox, “Zcash Protocol Specification”, https://github.com/zcash/zips/blob/master/protocol/protocol.pdf, January 2017.
[29] P. Peterson, “Anatomy of a Zcash Transaction”, z.cash, 2016. https://z.cash/blog/anatomy-of-zcash.html.
[30] libsnark: C++ library for zkSNARK proofs (Zcash fork). https://github.com/zcash/libsnark
[31] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza, “SNARKs for C: Verifying program executions succinctly and in zero knowledge”, Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 8043 LNCS, no. PART 2, pp. 90108, 2013.
[32] E. Ben-sasson, A. Chiesa, and E. Tromer, “Succinct Non-Interactive Arguments for a von Neumann Architecture”, USENIX Secur., pp. 135, 2013.
[33] C. Lundkvist, “Itroduction to zkSNARKs with Examples”, ConsenSys Media, 2017. https://media.consensys.net/introduction-to-zksnarks-with-examples-3283b554fc3b.
[34] L. Hazlewood, “What is an X.509 Certificate?”, Stormpath User Identity API, 2011. https://stormpath.com/blog/what-x509-certificate.
[35] W. Simpson, “RFC 1994 - PPP Challenge Handshake Authentication Protocol (CHAP)”, Tools.ietf.org, 1996. https://tools.ietf.org/html/rfc1994.
[36] “Checksum”, Ant.apache.org. https://ant.apache.org/manual/Tasks/checksum.html.
[37] V. Buterin, “Quadratic Arithmetic Programs: from Zero to Hero”, Medium, 2017. https://medium.com/@VitalikButerin/quadratic-arithmetic-programs-from-zero-to-hero-f6d558cea649#.c1fkogp41.
[38] “Zcash Price Chart (ZEC/EUR)”, CoinGecko, 2017. https://www.coingecko.com/en/price_charts/zcash/eur.
[39] “51% Attack”, Learncryptography.com. https://learncryptography.com/cryptocurrency/51-attack.
[40] “ethereum/EIPs”, GitHub, 2017. https://github.com/ethereum/EIPs.
[41] S. Bowe, “Zcash - zkSNARKs in Ethereum”, Z.cash, 2016. https://z.cash/blog/zksnarks-in-ethereum.html.
[42] C. Reitwiessner, “An Update on Integrating Zcash on Ethereum (ZoE) - Ethereum Blog”, Ethereum Blog, 2017. https://blog.ethereum.org/2017/01/19/update-integrating-zcash-ethereum/.
[43] TIVI, 2017 https://tivi.io/tivi/.