Internet Voting Using Zcash

by

Pavel Tarasov, BA (Mod) Computer Science

Supervisor: Hitesh Tewari

Thesis

Presented to the

University of Dublin, Trinity College

in fulfillment

of the requirements

for the Degree of

Master of Computer Science

University of Dublin, Trinity College

May 2017

Declaration

I, the undersigned, declare that the following dissertation, except where otherwise stated, is entirely my own work; that it has not previously been submitted as an exercise for a degree, either in Trinity College Dublin, or in any other University; and that the library may lend or copy it or any part thereof on request.

Pavel Tarasov

May 11, 2017

Summary

The work describes a theoretical approach to an internet voting protocol with the use of blockchain technology. The blockchain technology which has been used as the bases for implementing the voting protocol is called ZCash, and is a new, decentralised, anonymous payment scheme. The voting protocol does not change any aspect of the ZCash platform, thus the system transactions are proven by the proofs of Zcash transactions. The work describes the operations and transactions of ZCash and outlines the complex mechanisms behind their work. The work also describes the steps taken in the voting system to hold an election and means for verifying the results of said election, the security considerations of the voting scheme and possible future directions the work can take. This paper also outlines extensive research done in the domain of existing electronic voting schemes and blockchain technology, while describing security vulnerabilities of both domains. The main focus of the work is to establish a theoretical system where election can be done with low likelihood of success from coercers, while assuming the presence of some features of a secure online system, such as authentication of the users.

The primary methods of investigation involved research and study of the existing voting schemes for voting protocols and blockchain specifications as well as interactions with the developers of various blockchain platforms. The investigation done into the existing blockchain voting protocols currently implemented, yielded very few results and fewer publications, leading us to believe that this work is among the first works covering this topic. The main conclusion that can be drawn from this work is that it is possible to run construct a voting system using a blockchain payment scheme which will provide reasonable election integrity and verifications as discussed by the work.

Permission to Lend and/or Copy

Pavel Tarasov

May 11, 2017

Acknowledgments

I would first like to thank Dr. Hitesh Tewari of the School of Computer Science and Statistics in Trinity College Dublin. Prof. Tewari presented me with an opportunity to journey into the world of blockchain technology and his door was always open whenever I ran into trouble. I would like to thank him for the support on all aspects of this work, from planning an approach to the problem to execution and documentation of the solution. He consistently steered me in the right direction and provided great advice that I will carry with me on the journey ahead.

I would also like to acknowledge everyone in the School of Computer Science and Statistics in Trinity College Dublin and beyond for giving me constructive feedback on my ideas and work.

Pavel Tarasov

University of Dublin, Trinity College
May 2017

Internet Voting Using Zcash

Pavel Tarasov, MSc.
University of Dublin, Trinity College, 2017

Supervisor: Hitesh Tewari

Voting systems have been around for hundreds of years and despite different views on their integrity, have always been deemed secure systems with some fundamental security and anonymity principles. Numerous electronic systems have been proposed and implemented, however many of these systems have been rejected, while creating further suspicion about 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 the approach that is offered by current electronic voting protocols. The approach presented in this paper involves a protocol developed on blockchain technology. The particular technology that is used as basis for the voting system is a new electronic currency protocol and offers a factor of anonymity in transactions, which has not been observed in blockchain technologies to date. The proposed voting protocol offers anonymity of voter transactions, while keeping the transactions private and the election transparent and secure. The underlying blockchain protocol has not been modified in any way, the voting scheme proposed merely offers and alternative use case of the protocol at hand, which could be presented as the basis for voting systems on blockchains with further development of underlying blockchain protocols.

Contents

List of Figures

Chapter 1

Introduction

With blockchain steadily striving towards becoming the new system for decentralized payment schemes, amongst a wide array of 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 already managed to gain 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 govenrment or non-govenmental organisations (NGOs). The number of emerging and existing systems that migrate to blockchain is steadily increasing, however a largely researched topic of electronic voting has still not been influenced properly by the blockchain domain [1].

Electronic voting has been a topic of large debate, with significant number of people believing that electronic voting cannot be trusted enough to be used for significant elections, which in turn gave rise to the debates about 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 more serious manipulation from a coercer if the need arose. The appearance 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 electornic voting scheme [2].

Many electronic voting protocols, such as [11], could be suitable candidates for major elections, but some of these protocols have had security issues rendering them unfit for such purposes or security vulnerabilities which could be exploited to coerce an election [12]. Efforts in this domain continuously present protocols, based on the concepts of the founding electronic voting systems, and therefore do not make strong advances in the topic of research. Research into the blockchain voting systems has seen little progress with a very small number of blockchain-based voting systems available as prototypes. The documentation about these systems is much more scarce, making it difficult push the idea of blockchain voting past it's initial stages.

The motivation for this research is to find the ultimate solution for electronic voting which will pose the smallest security risk and provide basis for a system which can be used for any election or electronic voting more frequently with less costs involved.

This work presents a conceptual protocol for a transparent, private and anonymous electronic voting scheme on the blockchain technology. The proposed system utilises a recent development in cryptographic currency on blockchain, namely ZCash [3], as the basis for voting system. The voting system does not make any changes to the original ZCash specification and allows for anonymization of the identities of the users who participate in an election. The scope of this work assumes that the identity of a potential voter can be verified. The underlying ZCash protocol inherently ensures that every vote is valid and no same vote can be cast twice. The rest of the paper is organised in the following way: we start by providing a high level overview of the protocol, then Section 2 reviews the existing electronic voting protocols and the mechanics that are common amongst these systems, as well as a look at some of the blockchain voting systems that exist today. Section 3 will discuss ZCash, an anonymous, decentralised, cryptographic currency, which is the basis for the voting protocol. Section 4 will provide in-depth view of *zero-knowledge Succinct Non-Interactive Arguments of Knowledge* (*zk-SNARK*) and some cryptographic primitives that play a significant role in the ZCash protocol and future blockchain systems. The subsequent section will outline the proposed voting protocol on the ZCash platform and it's variations. In section 6, some security considerations will be outlined for the use of the proposed system. Finally, section 7 will discuss the future work that is being done on the blockchain technology and how it can influence the proposed voting protocol and other voting protocols on the blockchain.

1.1 High Level Overview

The protocol uses ZCash [3] as it's foundation. ZCash supports two distinct types of transactions, transparent and private transactions. Transparent transactions resemble that of Bitcoin [28] and even use the same types of addresses as generated by Bitcoin. The private transactions are focus of ZCash. These are more complicated than the transparent transactions and require additional setup and values. The transaction in ZCash supports both private and transparent value passing. One other difference is that ZCash possesses two distinct types of addresses in order to allow the different transactions to take place. Once again, the transparent addresses (t-addresses) have the same structure as in Bitcoin, and the private addresses (z-addresses) are longer and more complex as to offer the privacy of the transactions and anonymity for the sender and receiver.

Zero-knowledge proving system is an important part of ZCash as private transactions must not disclose the secret values used to generate the zero coins or (ZECs) but a proof must be supplied for each transaction of their validity and of the sender's holding of the required values to generate these coins. The zero-knowledge proof operates on the premises of proving a value to another party without disclosing the said value, and the construct which is used for this problem is a zero-knowledge Succinct Non-interactive Argument of Knowledge or zk-SNARK. This construct allows for generation of a proof given a program. The program in this case can be the verification of a sender having the generation values for a ZEC that they own.

To summarize ZCash, a user has the ability to send transactions both privately and transparently, while having two distinct addresses, where each transaction requires a zero-knowledge proof as input to the transaction to verify it's integrity and the correctness of the supplied values within the transaction.

The voting protocol does not make any changes to the underlying ZCash protocol and implements the transactions provided by ZCash as means of passing votes between the voters and the candidates. A potential voter registers for a poll or an election of their choice via the provided registration page, which authorizes the voter and ensures the said voter is who they claim to be. The aim of the registration step is to obtain an email address of the voter, where an invitation to the participate in an election will be sent to. Once a poll is created by the administration of the system, the emails stored by the system are used for invitation forwarding, where each voter receives a unique invitation to participate in the chosen poll. The link brings the voter to a unique ballot for a particular election.

The voter has to provide a receiving address of their ZCash wallet in order to receive a ZEC vote token from the system. The voter, then, picks a candidate of their choice and proceeds through an agreement, holding them legally liable for the transaction. The transaction cannot be reverted at any point after the voter has accepted the terms and conditions of the vote transaction. The system continues by incrementing the system counters which represent the total number of voters, amongst others, and sends a ZEC which serves the purpose of a vote token to the provided address by the voter. As soon as the transaction is approved and arrives at the candidate wallet, a new transaction sends the received token to the candidate of voter's choice. A confirmation can be optionally configured to notify the voter of the transaction.

Upon the termination of the election timer, set initially by the administrators of the election or a poll, the candidate wallets forward their ZEC vote tokens to a system wallet which tracks the number of tokens it had before the transaction and after in order to calculate the total number of votes received by the candidates. These numbers are important and are required to balance in order to verify the integrity of an election. There needs to be exactly equal amount of tokens granted to the voters as voters who have cast a vote and the total number of candidate votes must be less than or equal to the above mentioned values. The security considerations section will outline the reasons for the last equality.

The nature of ZCash transactions allows the voters to track view their votes on the blockchain. A system, similar to Bitcoin's [36], which allows the viewing of the blocks of transactions, can be used for the public members to view the available transaction details for an election.

This has provided a very general overview of important steps of the protocol and will be explained further by the following chapters. Prior to deeper investigation of both ZCash and the voting protocol, it is important to review the state of the art in the electronic voting domain, which will be covered by the next section.

Chapter 2

State of the Art

Electronic voting is a topic of much research and a number of 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, with particular attention to a protocol called *Ethereum* [34], can be used as basis for the future work.

2.1 Influential Electronic Voting Protocols

Electronic voting protocols have been around and implemented in different elections, ranging from university to government based elections. Many viable protocols have been created since Chaum [4] first proposed Votegrity, which is one of the first *end-to-end* (E2E) verifiable voting solutions. End-to-end verifiability means that the voter is able to verify that their own vote has been cast as intended based on the receipt provided by the electronic voting booth, and by verifying that their vote has propagated to the public web bulletin board correctly. The second part of the E2E verifiability would be the assurance that the voter's vote has been counted correctly and included in the final tally. The final aspect of E2E is the ability of the election to be verified by all public members, who may not be involved in the election or voting, in order to ensure that no coercion took place and that the votes have not been compromised.

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* [5], *Helios* [7], *Scantegrity* [8] and *STAR-Vote* [9]. 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. Prêt à Voter is a modified version of the vVote protocol [10], and was used in the Australian state elections. Despite Helios iterations, the protocol is solely internet based and has been used in internal university elections. Helios 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 [11] and Helios 3.0 versions, attempting to fix the vulnerabilities posted by Estehghari and Desmedt [12], and will be discussed further in this section. Other protocols, have used Helios as a base for developing their own implementation of online voting, which aim to solve Helios vulnerabilities in a different way. Examples of these protocols include *Apollo* [13] and *Zeus* [14], and their handling of Helios vulnerabilities will be discussed further in this section. Finally, Scantegrity and STAR-Vote have both been proposed for elections in the USA [15].

Upon closer examination of these protocols and others such as [16] proposed by Heather in an STV modification of the Prêt à Voter protocol, vVote and the proposed improvement for the currently implemented Estonian voting protocol presented by Parsovs in [17], indicate that these protocols use public *web bulletin board* (WBB) for posing all of the cast ballots for the public to see. vVote implements an additional, private WBB in order to validate ballots and provide an indisputable signature of validity of the ballot.

Web bulletin boards in these protocols are used as an authenticated public broadcast channel which, as mentioned above, displays 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.

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 and so does the way the voter verifies these receipts, 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. When the layers are overlaid, the pixels which are different form an opaque result which is unreadable, whereas the pixels that are the same, form the letters and the summary of the vote of the voter. Furthermore, these protocols preserve an electronic copy of the vote, and Votegrity is no different. The electronic version of the receipt can be compared to a Russian doll. A trustee can be merely compared to opening the outer layer of the Russian doll with their secret key and passing the next doll to the next trustee, until the vote is completely readable. To be more specific, the trustees who decrypt the ballot have a key for each stage of the decryption as well as a coded sheet. Once decrypted, the trustees sheet is applied to the next stage of the decryption of the ballot thus decrypting the ballot at each stage until the ballot is in a readable form. The readable ballots can then be served as input into a script which reads their content and tabulates the votes. This process of decryption of the ballots uses a technique called mixing [4]. This technique will be outlined further in this section.

Another approach to handing receipts to the voters, while providing a backup of electronic votes in a physical form, is done by STAR-Vote. STAR-Vote prints two distinct receipts, one in human readable form with a unique serial number and a voters personal receipt which they can take home, which includes the terminal used, the time of the vote as well as the hash of the vote commitment. These two pieces are printed for the voter to take, while the electronic version of the vote is sent from the voting booth to database. What is interesting is the fact that, the voter must scan the serial number on the ballot (presented as a one-dimensional bar code) and then cast this receipt into the ballot box. The rule that STAR-Vote imposes is that the electronic vote must be supplied by the physical copy in the ballot box in order for the vote to be included in the tally [9].

STAR-Vote is also an example of a protocol where the voter can challenge the system. This is a common feature amongst protocols and a similar example is present in vVote [10] where the voter can verify the correctness of the ballot form prior to the casting the final vote to ensure that the system is not compromised. This involves the voter bringing back the ballot to the printer and getting the printer to produce the proof of correct ballot formation along with the signature from WBB, where the WBB must invalidate that vote as it must not be allowed to be counted in the tally process. The voter also can decrypt the vote to confirm that the ballot was formed correctly. The voter also has the ability and is encouraged to verify that the ballot possesses a valid WBB signature which includes the ballots serial number and the district where the vote was cast. This is one of the measures to help prevent a possible coercion of the votes. There is a surprisingly low number of ballots that are subject to voter verification. It was found by [8] that in the 2009 elections of the City of Takoma Park, MD, less than 4% of the ballots cast have been verified by the voters themselves after they cast their vote. While this is not a large number, this would be sufficient to detect any coercion significant enough to alter the course of the election.

STAR-Votes challenge of the system involves the ability of the voter to spoil the ballot that they have cast. A voter may spoil the ballot erroneously or on purpose, thus challenging the system. The spoiled ballot is returned to the poll worker who marks the ballot serial number, for the system to record and publish spoiled ballots at the end of election. "The original printed paper ballot thus corresponds to a commitment by the voting machine, before it ever *knew* it might be challenged". If the voting machine cannot produce a suitable proof that the ballot encryption matches the plaintext, then it has been caught cheating" [9].

The final example of such challenge, referred to as Benaloh challenge by several protocols including the Apollo [13]. In Apollo, this is done in the form of audit and resembles the verification steps taken by vVote. Apollo 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. Without these assistants, the Apollo protocol is identical to Helios. The assistants are external to the voting protocol devices that can interact with the election, by verifying that the vote has been cast properly. These assistants can be laptops, tablets, or any other external devices. The protocol has the notion of a voting session which is created by the voter as soon as the voting starts, but is customised with a personal string which is appended to the session ID to form a session key. This allows the voter to find and continue their voting session simply by entering their personal. Human readable string 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 [13].

As mentioned above, mixing is one of the two predominant techniques that are used in electronic voting protocols. Mixing utilizes mix networks, which is 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 which can be another mix network, or *mixnet*, or the destination node. Mixnets, in the context of voting and other applications such as the onion routing, are used to provide a degree of anonymity to the user by obfuscating where the message came from. This concept was first derived by David Chaum in [18] and is used in some voting protocols such as Votegrity, Helios 1.0 and Markpledge to name but a few.

In the case of Votegrity the collection of electronic ballots is assembled together to be decrypted and converted into the form where the ballots can be tallied. As mentioned above, mixing is a technique used for anonymization of votes, which can be observed from the Zeus protocol [14]. Zeus 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.

In the case of vVote, similar shuffle occurs to break the linkability between the voter and the vote. This is accompanied by a noninteractive, universally verifiable proof of a shuffle and decryption (of encrypted votes) and posts it to the Public WBB. [10]. This proof can be verified by any member of the public while guaranteeing that the source of votes is valid and that the vote has not been modified which adds to the aspect of end-to-end verifiability. As mentioned by [6], mixnets can be combined to perform encryption and decryption or, decryption and tallying but all of the steps in the E2E protocols have to be supplied with proof of correct tally or decryption and any other extra steps taken during the protocol.

Despite the different voting mechanism of [16], mixes still serve the same purpose as in the other protocols, i.e. anonymizing the votes. A concept of re-encryption mixes [19] is applied here, as well as in the number of other protocols, in order to accommodate the STV voting that is described within [16]. The particular details of the protocol can be found within [16], however the concept of re-encryption is given by an ElGamal encryption $< m, z >_{PK}$, it is possible to change this value of z even without knowing m or z. Thus, given just the ciphertext and the public key, one can produce a ciphertext that looks different to anyone not having the corresponding secret key, but decrypts to the same value. Interestingly, this is used within the mixes to encrypt each entry of the vote as the ballot does not comprise of a single vote, as is the nature of First-Pass-The-Post system. During the tally, at each stage of the voting, if a candidate becomes eliminated, the votes go to the subsequent candidate which is found on the voters ballot. However, each subsequent candidate on the ballot is only decrypted when the first candidate is removed from the race. This occurs due to the nature of the encryption that occurs in [16].

The second widely used technique is *homomorphic tally*. Cohen and Fischer [20] describe how this can be applied to a voting protocol in one of the first papers which had applied this technique. 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 the proposal from Parsovs [17] to replace the current Estonian voting system, Helios 2.0 [11], STAR-Vote [9] 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.

Parsovs in [17] proposes to reform the currently implemented scheme to use homomorphic encryption. His example of the application can be used as the perfect example of how this technique can be applied to voting. It is worth noting that a lot of these implementations utilise the common exponential version of ElGamal encryption [21] for encryption of the vote. In [17], the voter casts their votes by producing a number of ElGamal ciphertext corresponding to the number of candidates running in the election. The candidate that gets the voter's vote is encrypted as the ciphertext of a digit 1, while the other candidates are represented by the encrypted version of 0. In this scheme the voter is only allowed one vote for a particular candidate, therefore all the other candidates have to be represented by the value 0. Prior to proceeding with counting of the vote, like other stages, a proof of the vote must be derived, proving that the sum of the plaintext values within the vote is indeed 1. Parsovs, in his protocol, attempts to leave as much of the existing system infrastructure of the Estonian voting system [22] as possible. The protocol offers several stages that the vote must pass prior to being tallied. These stages store the votes and remove the digital signature from these votes in order to anonymize each of the votes. The final stage multiplies the aggregated ciphertexts to obtain one single ciphertext of the number of votes that a candidate has received. The removable media is brought to a *hardware security module* (HSM) for decryption of these ciphertexts.

Helios 2.0 has a similar starting approach to encryption of the chosen candidate. This is supported by the presence of a zero-knowledge proof for proving that each of the ciphertexts includes an encrypted value of a 1 or a 0 and a proof of fact that the homomorphic sum of the voter's votes equates to the total of 1. These proofs are verified upon tally to ensure the votes are well formed.

All the schemes mentioned above have proofs at each step of the protocol and the encryption of the votes seemingly provides the security and integrity of the votes. However, many schemes have been compromised which amplifies the mistrust of the population in the electronic voting. EVoting protocols such as Helios 1.0 and Helios 2.0 have both been proven to contain vulnerabilities. Currently implemented Estonian eVoting protocol is yet another example of a compromised voting system. These are examples of just some of the systems that have been used in elections with Estonian protocol continuously gaining voters throughout the years [35]. Some vulnerabilities, as reported by Estehghari and Desmedt in [12] for Helios 2.0, Parsovs note on the issues of Estonian eVote and the work done by Springall et al. [23] on the same protocol, are just some of the security issues that appear in these protocols.

As mentioned earlier, protocols such as Zeus [14] and Apollo [13] 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 cross-site scripting (XSS), cross-site forgery, clickjacking and clash attacks with the help of the voting assistants as stated earlier. 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 has 1) remembers the tracking number of the ballot and 2) checks the WBB later to find their vote. This is in the third place of the top vulnerabilities of web applications as found by OWASP in 2013 [24], which means that this is still one of the top threats to security of web applications. OWASP's new "Top 10 Application Security Risks" draft of 2017 outlines that XSS remains in the third position as a top application risk, as well as cross-site forgery which remains in the eighth position on the list [25]. These issues have been reported to Helios development team and have been patched with the release of the following versions of Helios.

Apollo developers were not the only ones to discover the attacks mentioned above, Estehghari and Desmedt have supported their claims about XSS vulnerability possible on the Helios 2.0 protocol. They have chosen to utilize browser rootkits as an attack vector on Helios system. A browser rootkit is simply an extension to a browser with modified, by attacker, script to monitor browser activity, obtain passwords and gain access to the DOM tree of the web page. This extension can be activated upon visiting a specific web page, like the web page of a bank to capture user information. Helios utilizes candidate statements in PDF format, and this is exactly the attack vector that Estehghari and Desmedt use. They exploit the vulnerability in Adobe Reader in order to inject malicious JavaScript which activates upon the user opening the PDF file. The malicious payload is installed and the browser restarts with the activated payload, which grants control of the client-side voting application for the attacker. As mentioned before, the script only activates when the voter visits the voting page, which then changes the information of the ballot that is supplied by the voter to the system. The aim of [12] was to demonstrate the issues of other web-based voting protocols, and not merely exploit Helios, however they have shown very well the issues that need to be considered by the designers of voting protocols.

To stress the importance of security in electronic voting, NIST [26] has published a set of guidelines and considerations that need to be included into the design and development of voting systems. This involves a variety of seemingly basic considerations such as: confidentiality of the voter, integrity of the vote and data and identification/authentication to name but a few, while also covering the potential threats and other issues that may arise because of safeguard that are applied to deal with the above-mentioned issues.

The Estonian voting system's structure is different from most the protocols mentioned above. One particular instance, outlined by Parsovs [17] is an invalid vote that has been detected during one of the elections. This was not deemed as an attack by many, who though that this invalid vote was a result of a bug occurring in the code. Others saw it as a potential attack on the system. The vote involved invalid plaintext and therefore could not be counted as a vote. This was the only suspicious instance in the election that year, however it raised some suspicion on the security of votes in the protocol. Research published by [23], mentions variety of issues of the protocol from insufficient transparency to vulnerabilities found in the code, which certainly puts these protocols in question since a lot of trust is required in the system if the protocol is not transparent.

However, it is worth noting that no system is ever 100% secure. These attacks are plentiful and occur in many voting protocols, this is merely an example of the how these protocols can be compromised. Final example describes many attacks on the Welsh iVote system, written by Halderman and Teague in [27]. Their findings include poor server handling, in terms of TLS vulnerabilities which was determined merely two weeks prior to the start of the elections. This vulnerability allowed a man-in-the-middle attack to be performed for long enough to attack an "unlimited number of voter's TLS connections" [27]. The attack involved downgrading the connection by making the server user the export-grade RSA instead of requesting the normal RSA to be used. But the main point is that zero-days, attacks on the system which have not been seen before, are always possible and [27] describes exactly the case with the Welsh iVote protocol. This relates back to the export-grade RSA, where if the server supported this export-grade Diffie-Hellman (DH), the attacker could set up a man-in-the-middle attack while forcing the browsers to use it, where the attacker could obtain session keys and change the contents or intercept the connection as they pleased.

2.2 Blockchain For Voting

From this investigation into the protocols, 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. Blockchain's [28] help to achieve this level of security and transparency, while maintaining privacy and non-malleability of the transactions, which may indeed be the future of eVoting protocols. The benefits of using blockchain is in it's decentralised nature, relatively low cost of transactions and tamper-proof properties, which play an important role if a voting system was based on this technology [32] [33].

Although different, some elements from 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 [36]. 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 [37] present a seemingly sound voting protocol, however without any in-depth specification to verify the security of the protocol, there is only the website information to go by. The code is also open-sourced, which if compared to the notion of public cryptographic protocols over the history of private ones, indicates that it may be a secure, however, without official specification publication, or any other documentation, it is difficult to verify these claims. One other noteworthy blockchain technology that could revolutionise electronic voting, as well as give birth to many other forms of electronic protocols is Ethereum [34]. Ethereum differs from Bitcoin [28] as it serves as a generic platform for creation of custom functionality in the form of contracts, while also having a slightly complex structure of transactions. The currency used by Ethereum is ether and gas, which will be discussed in more detail later in this section. 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, and potentially by other smart entities, whereas contracts are also accounts, however they are operated by code on the EVM. Contracts are the agents that bring about the generic functionality of Ethereum mentioned above. Contacts 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. Ethereum operates on the context of states and state transitions which are brought about by the transactions. The contracts, on the other hand, have a failsafe feature which addresses the Turing Halting problem, in other words, to prevent contract code from infinite function execution, Ethereum introduces the concept of gas. Gas is consumed for each consumed resource of the contract computation. This can be each stage of contract execution or the memory used by the contract. Gas costs are dictated by the gas limit in the contract and the gas costs are deducted from the user account, who wishes to send transactions to a particular contact. This feature means that upon exceeding the gas limit for a transaction, the transaction can revert to last state and refund the user account in case of an unexpected gas limit overflow. The contracts themselves can be used to send ether between other contracts, or to other accounts. The transaction between user accounts is simply moving ether around, which resembles the functionality of Bitcoin [34] [38] [39].

The transactions issue receipts back to the user accounts which include information like post-transaction state, the cumulative gas used in the block containing the transaction receipt as of immediately after the transaction has happened and others. Entities such as the receipts and the world state, are stored in Merkle Patricia Trees or tries. Merkle tress are used in Bitcoin transactions and serve as a way of storing all transactions inside one hash, which can be traversed to find a particular transaction. The Merkle tree contains hashes of hashes of transactions until the root hash is obtained. In Ethereum, these tries are used as databases which stores the mappings between bytearrays of account information [34] [38] [39].

One may wonder about the differences between Bitcoin and Ethereum, and the common question of which one is a better platform arises very often. In reality, there is no better platform out of the two, it is simply the matter of preference and functionality. Ethereum offers a set of features that differs from that of Bitcoin. This gives a brief overview of the Ethereum protocol, and its main difference to Bitcoin. Ethereum allows the creation of powerful sub-systems, driven by custom currency and functionality with the immutable support of blockchain. This makes is a very good candidate for different protocols, including a voting protocol. The protocol proposed here does not use Ethereum, however with the announced *Metropolis* upgrade [70] to the Ethereum platform, which will be discussed in the future work section, the proposed scheme will likely cause the shift to Ethereum contract, which will remove and simplify some of the architectural designs of the current system.

Having discussed some of the available voting protocols and security issues which are faced by a number of these protocols, it is clear that an alternative solution is required to bring a new outlook on domain of electronic voting systems. Blockchains present themselves as suitable candidates due to their inherent security features. This investigation into the domain of electronic voting mechanisms clearly identifies a gap, which could be filled with a blockchain voting technology. One application which allows creation of anonymous voting protocols is ZCash, which is further outlined in the next chapter.

Chapter 3

ZCash - Decentralized Anonymous Payment Scheme

The concept of ZCash has been around for since 2014 in the form of Zerocash [40]. ZCash aims to provide anonymity of transactions, something Bitcoin does not have, since the concept is based around public verifiability. The ZCash systems, which differs from the Zerocash specification, conceptually offer an anonymous version of Bitcoin, since it allows for both public and private transactions. One of the biggest differences between ZCash and Bitcoin is the proof-of-work system, where ZCash relies on zero-knowledge proofs [3]. This section will outline core ZCash features which will be used as part of the eVoting protocol for this paper.

3.1 High-Level Overview

One of the biggest differences that is noticed when looking at ZCash is that it has two types of addresses. These addresses is what allows the users to make transactions anonymously or publicly. ZCash, therefore, has two types of values, namely *shielded* and *transparent*, that come to pass from the existence of these address types. Shielded value means that the details of the value cannot be observed by the public, but is available only to the sender or recipient. Transparent values can be influenced by the *shielding* or *de-shielding* transaction types, making transparent values shielded, which are two of the four possible types of transactions that can take place in ZCash. The other, public values, do not hide their details or values and the public transactions are viewable by anyone. This type of transaction is the exact transaction that is present in Bitcoin. Therefore, the ZCash transaction, simultaneously supports both transaction types [3].

ZCash currency is called *zero coins* or ZECs, however the shielded values are named *notes*. Shielded values require more complex setup and have a *commitment* and a *nullifier* associated with them cryptographically. These values are derived from the holder's knowledge of a spending key, which will be explained further in this section. To give a brief idea of the nullifier, it is a unique value that is revealed once the note has been spent in a transaction, which is a mechanism for prevention of double-spending of any note. Double-spending simply means that no one note or ZEC is spent twice. A commitment is a function that allows the sender to commit to the input value that they are sending in the transaction. These values are part of the proof that is generated in order to prove that the note is legitimate and has not yet been spent. The transparent inputs are less complicated as there is not the same level of set up and information required about the transparent inputs [3].

The commitment and the nullifier are included into the data that is included in each of the transactions. There are different names attributed to these transactions, but in general they are called *JoinSplit transfers* or *JoinSplit transactions*. The JoinSplit transfers include data, such as the commitments for the input notes, the nullifiers for the input notes, the notes themselves and some more data which will be covered in later sections. This data is called *JoinSplit description*, and describes the data that is inside JoinSplit transfer.

One of the most important parts of ZCash transactions, is the proof behind the owner's notes. In other words, since no knowledge about the notes can be observed by anyone else on the blockchain except the sender and the receiver, the proof-of-work, still must be completed on each of the transactions on the system. Proof-of-work is a concept introduced by Bitcoin, where independent entities, or miners verify that each transaction is legitimate prior to adding it to a block on the blockchain. This is the central security mechanism behind Bitcoin, which ensures all the transactions are legitimate and well formed. In return, the first miner to be find the solution to a proof of work problem, will be rewarded with x amount of Bitcoins, which depends on the state and height of the blockchain at the given instant [28]. Since the values in ZCash could potentially be shielded, the proof-of-work concept revolves around zero-knowledge proofs. For this *zero-knowledge Succinct Non-interactive Arguments of Knowledge* or *zk-SNARKs* are used. Zk-SNARKs are highly complicated and will have an entire section dedicated to them. For now, it is important to understand that the zk-SNARKs are used to prove certain facts about the transactions in order for the miners to be able to verify the correctness of each of the transactions. Some of the facts that are proven include:

- The total value of input notes is equal to the total value of the output notes.
- Each note has an associated commitment and a nullifier.
- The nullifiers and commitments have been generated correctly.

Of course, there are more parts to the proof in order to prove the validity of the transaction. The zk-SNARKs also create a proof that the nullifiers have not been revealed for the note aside from proving the sender's knowledge of values associated with a note.

3.2 Protocol Details

With the general idea of some of the ZCash principles and how ZCash operates, it is possible to go more in-depth into some of the important concepts of the protocol which will give more appreciation to the voting protocol built on top of ZCash. The complete specification for ZCash is available from Hopwood et al. at [3].

3.2.1 Transaction Types

As mentioned above, ZCash has two types of addresses incorporated in it's protocol. The addresses are namely *z-address* and *t-address*. The purpose of these addresses is to differentiate between the transaction types. As mentioned before, there are four types of transaction that can take place which are outlined in Figure 3.1. Transparent values are represented as *coins* and the shielded values are represented as *shields*. The public transactions are the transactions that are native to Bitcoin, where the values of the transactions are visible. Similar to Bitcoin, the coins are traceable to wallets that hold them. This is called *pseudonymity* as the identity is not truly anonymous and can be revealed with enough effort [41]. When a transaction is made from a t-address to a z-address the transaction *shields* the transparent value which becomes a note. The *linkability* of the ZEC is broken at this point as z-addresses are considered anonymous. The value of the note cannot be seen. The conversion of the transparent value to a shielded note is done within the transaction and will be explain in future sections. De-shielding transactions act opposite to the shielding transactions, which essentially convert a shielded note into a transparent ZEC. This occurs if the transaction is made from a sender's z-address to the receiver's t-address. The most interesting type of transaction is the private transaction. This transaction type occurs when both the sender and the receiver use z-addresses. This details of this transaction can only be observed by the sender and the receiver and by no-one else on the blockchain. This is due to the fact that the sender and receiver, both agree on a secret key to use to view the details of the transaction. The details of the transactions will be discussed in further sections.

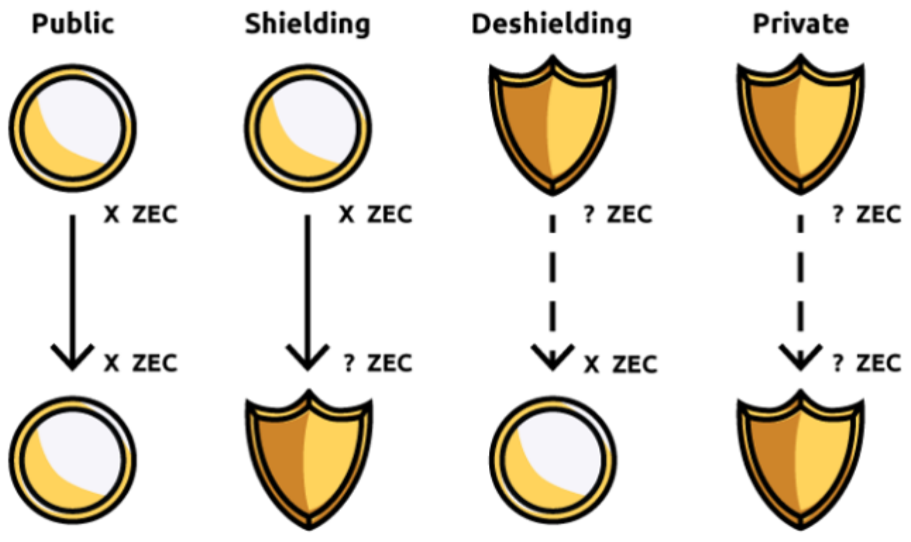

Figure 3.1: The four types of transactions possible in ZCash [43]

3.2.2 Payment Addresses

As established above, there are two types of addresses. The addresses are a combination of keys that, when combined, make up the payment address. These keys are what allow the users of ZCash to spend their ZECs, view private transactions and send ZECs to other addresses. There are four types of keys that are used in ZCash:

- *Paying key* (a_{pk}): Used as part of the key to generate payment address and commitment for a ZEC.
- *Transmission key* (pk_{enc}): Used to encrypt and decrypt random values to generate ZEC, or ZEC *plaintexts*
- *Spending key* (a_{sk}): Allows spending ZEC while revealing the nullifier for the ZEC spent.
- *Viewing key* (sk_{enc}): Established key for viewing the private transaction for the sender and the receiver.

The keys,however, are used to perform more complex functions. The combination of paying key (a_{pk}) and transmission key (pk_{enc}) is what makes up the payment address. In order to participate in the scheme, the user generates a key tuple ($a_{sk}, sk_{enc}, addr_{pk}$) where $addr_{pk}$ is the combination (a_{pk}, pk_{enc}). The spending key (a_{sk}) is used to derive other keys, and therefore is kept secret at all times. Figure 3.2 shows the key derivation that takes place to generate the payment address and the other associated keys. The functions used to derive the keys are outlined by Hopwood et al. in [3]. The spending key (a_{sk}) is a sequence of 252-bits joined with two sets of 8-bit versions of raw encoding of ZCash version. The part represented by [0] in Figure 3.2 is 4 zero bit padding. The key breakdown can be observed in Figure 3.3. Having this key, the pseudo-random function is used to derive the paying key (a_{pk}), which is 256-bit in length. The other two keys use the key agreement function in order to be derived.

The key agreement function is a protocol where two parties agree on a shared secret which involves their private and the second party's public key. In essence, the key agreement function creates public and private keys, as well as a shared key which can be used to derive further keys for encryption of a secret. The private key derives a public key, which when multiplied, derives a shared key for the key agreement scheme.

The derivation function, which will be used multiple times in this protocol, used to establish a key for encryption of a secret value to be passed between the sender and the receiver. On a high level, a private key esk is derived from the agreed private key from the key agreement scheme. The public key epk is derived from esk. The derivation function returns the public key epk and a set of agreed keys for encryption of random values used to generate ZEC.

The viewing key (sk_{enc}) is derived using the pseudo-random function and the transmission key (a_{sk}) is derived from the viewing key (sk_{enc}) with the key derivation function. This may be complicated by the main idea to obtain from this is that ZCash implements several pseudo-random functions and a single key derivation function to generate and obtain keys to encrypt secret values for ZECs and pass these on in a transaction. The details of the transaction will be explained in further sections.

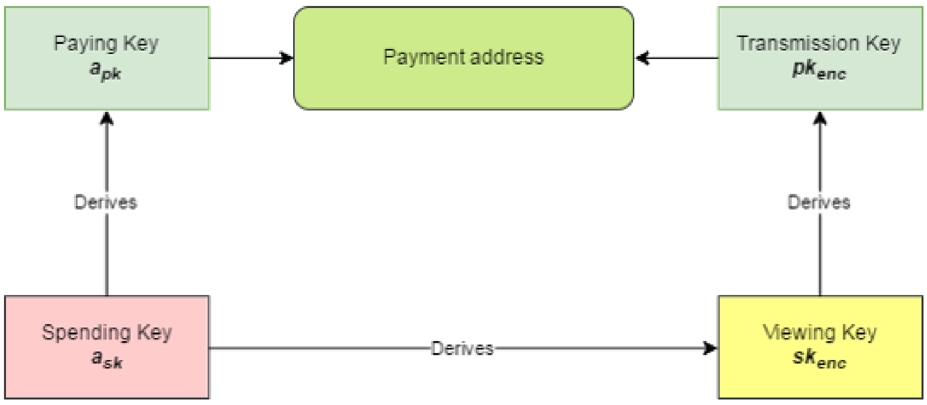

Figure 3.2: The keys generated as part of ZCash address generation routine

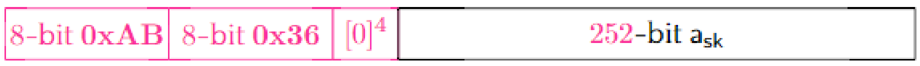

Figure 3.3: Values used to generate spending key [3]

As already mentioned, the t-address is exactly the same as the Bitcoin address and it's structure is exactly the same. Z-addresses on the other hand are much larger due to more complexity involved in the operations involving them. For comparison sake, the t-address may look something like [44]:

```
t14oHp2v54vfmdgQ3v3SNuQga8JKHTNi2a1
```

On the other hand, a z-address would be much longer and would look like:

```
zcBqWB8VDjVER7uLKb4oHp2v54v2a1jKd9o4FY7mdgQ3gDfG8MiZLvdQga8JK3t58y
jXGjQHzMzkGUxSguSs6ZzqpgTNiZG
```

Going back to the t-address, there are two ways of obtaining the address. These are *pay to script hash* (P2SH) or *pay to public key hash* (P2PKH). There is no difference in terms of the size as both of these schemes use 160-bit addresses. For P2SH, a script is hashed using the SHA256 message digest scheme. The script that is included in the hash outlines the conditions to be fulfilled for the Bitcoins to be spent. In other words the script specifies how the person receiving the Bitcoins can spend them [46]. The other type of address is the P2PKH. This is a public key hash obtained from a derived public key from the private key of the potential recipient. This, too uses SHA256 message digest. The key pair itself is generated Elliptic Curve Digital Signature Algorithm (ECDSA) to generate the private keys. Figure 3.4 shows the structure of a typical t-address in P2PKH form. The P2SH address would have exactly the same structure in terms of size. The first two sets of 8-bits as per the spending key, indicate the version of the raw encoding of P2SH address. This gives a high level overview of the t-addresses and how they are generated. In essence this is a much simpler address than the z-address. More details on the ECDSA and the mechanisms involved in creation of the t-addresses can be found in [45] [3].

8-bit **0x1C**	8-bit **0xB8**	160-bit public key hash

Figure 3.4: Bit structure of t-address in ZCash [3]

The z-address consists the paying key (a_{pk}) and the transmission key (pk_{enc}) coupled together and compressed with SHA256 message digest. From the earlier outline of the keys, the transmission key is the public key from the key agreement function. The address includes two 8-bit version raw encodings of ZCash payment address. Figure 3.5 shows the structure of z-address for comparison purposes [3].

8-bit **0x16**	8-bit **0x9A**	256-bit a_{pk}	256-bit pk_{enc}

Figure 3.5: Bit structure of z-address in ZCash [3]

3.2.3 Zerocoin (ZEC) Structure

So far a lot of detail has been given describing an entity that is being sent over the network, namely ZECs or notes. There has been a mention of the keys used to transfer these but no mention of what these entities really are. On a high level, these ZECs are simply values that are passed from sender to a receiver. Due to the natural complexity of the protocol, the ZEC has a slightly complex structure. Tromer in [49] outlines the makings of a Zerocoin's coin, and the structure has largely persisted over in ZCash. A ZEC is of the form: $(a_{pk}, v\rho, r)$, where a_{pk} is the paying key which has been described above, v is the value of the ZEC which the sender wishes to send to the receiver, this value is described in zatoshis. The value ρ is used as part of a pseudo random function in order to generate the nullifier for the ZEC and r is used as the commitment trapdoor for the commitment function. It is important to clarify the commitments and nullifiers for each shielded ZEC or a note prior to proceeding to talk about the transfer mechanics of ZCash.

A commitment is the result of a commitment function that commits an input to the transfer. This is the value that is generated for each new generated ZEC. The structure of a commitment is $(a_{pk}, v\rho)$. The nullifier is a generated value that acts as a serial number for each ZEC. This value must be unique and is stored in nullifier set, which is a set which keeps track of all the spent ZECs. A spent ZEC is the one for which the nullifier has been revealed, therefore for any unspent note, a nullifier has not been seen by any entity.

The note commitments are stored in a data structure called a Merkle tree [28] which is a binary tree that simplifies the process of finding a particular item. The Merkle tree is a data structure which takes two hashes of a transaction and combines them into a new hash. This operation is done for all of the transactions in a particular block until only one hash remains, which summarizes the block and can be used to find any transaction inside the Merkle tree in a few number of look-ups from the *root hash* which is the single hash remaining after all transactions have beene hashed into the Merkle Tree [31] [28]. Figure 3.6 shows a basic example of the Merkle tree. The Merkle tree is used to store note commitments in ZCash which keeps track of each state of the transaction. The nullifier set is merely a set which keeps track of each revealed nullifier for each of the notes to ensure that no note can be spent twice. Each computer, or a node, which keeps the full record of the transactions from the beginning to the latest transaction is called a *full node*. A note commitment tree is a Merkle tree where the JoinSplit transfers append the generated note commitments within the transfer. The concept of spending a particular ZEC is to generate a zero-knowledge proof, which proves that the spender knows the existence of a note commitment in the note commitment tree, and thus capable of spending the ZEC. The note commitment tree and the nullifier set are part of a *treestate* for each transaction. The treestates are connected to each other, where the output treestate for each transaction is the input treestate for the current transaction [3].

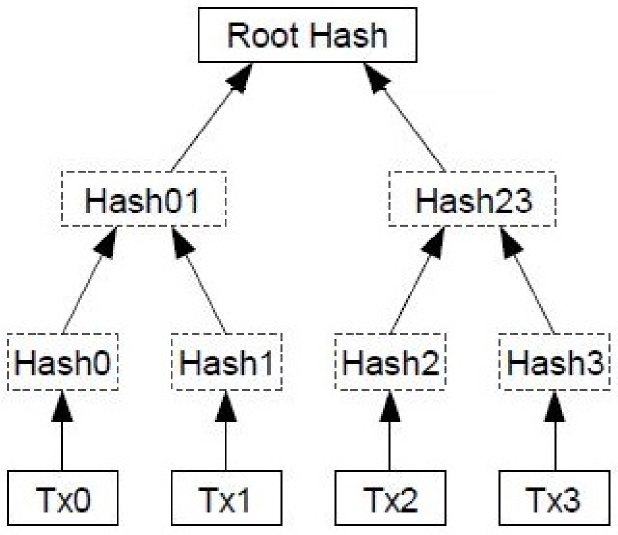

Figure 3.6: Simplified Merkle tree example [29]

One of the last pieces of very important information is the data that was used to generate each shielded ZEC or a note. This information is required to be passed on to the receiver upon transaction. This is highly important because, the receiver will not be able to spend the newly acquired ZECs unless they possess the randomness values ρ and r which have been used for note generation, as they are required to be supplied for the zero-knowledge proof that is passed into the transaction. This information is called *note plaintext* and the *note ciphertext* is the encrypted note plaintext which is passed into a transaction. The plaintext has the form $(v, \rho, r, memo)$. The newly mentioned variable *memo* represents a variable whose meaning is specific to the context between a sender and a receiver. More exactly, "the usage of *memo* field is by agreement between the sender and recipient of the note" [3]. The note ciphertexts are part of the JoinSplit description which will be discussed in the next section.

Naturally, the note plaintexts cannot simply be included into a transaction as there is a high possibility the ZECs would be stolen. The transmission key (pk_{enc}) play an important role when encrypting the note plaintexts. There are two operations, before the transaction the note plaintexts have to be encrypted by the sender, and then upon receiving the note by the receiver, the ciphertexts will have to be decrypted. The next two sections will give a high level overveiw of these steps. For a more in-depth description of the steps, one can investigate the ZCash specification [3].

3.2.4 Note Plaintext Encryption

The general procedure to encrypting the note plaintexts starts with generating a new key agreement key pair *epk* and *esk*. These are called the *ephemeral keys*. Each of the plaintexts then follows the next procedure:

- Obtain raw encoding of each of the note plaintexts.
- Establish the shared secret key via the key agreement function using the *ephemeral secret key esk* and the transmission key pk_{enc}.
- Derive a new key for symmetric encryption of the raw encoding of a note plaintext

The note plaintext, then become note ciphertexts and are added to the JoinSplit description with the *ephemeral public key*. The importance of the ephemeral keys will be stated in the next section.

3.2.5 Note Ciphertext Decryption

When the recipient receives the ZEC, they need to decrypt the note ciphertexts in order to be able to spend it later. Since the ZEC has been sent to the recipient's address the address is made up of two keys, the paying key a_{pk} and the transmission key pk_{end}. The recipient is also in possession of the viewing key sk_{enc} in order to be able to view the transaction. The procedure for decrypting the ciphertexts involves similar procedure to the encryption. The first step is for the recipient to agree on a key pair of their viewing key sk_{enc} and the ephemeral public key epk. The key derivation function allows the generation of the same key used by the sender to encrypt the note plaintexts. The procedure is then to go through each of the note ciphertexts and do the opposite of the sender's operation. With correct key, the recipient is able to extract the raw encoding for the note plaintext. If the decryption operation results in the raw encoding being $\perp$ or $false$, the decryption has not been done successfully. Otherwise, the note plaintext can be obtained from the raw encoding. The note commitment plays an important role in this verification. The recipient generates a commitment based on the information that has just been received and compares it to the commitment for the note in the JoinSplit Transfer, which has been generated as a result of the transaction between the sender and the receiver. If the note commitments do not match, once again the information supplied, or the key that was used to decrypt the note ciphertexts has not been the correect key, otherwise the correct note plaintexts are returned as to verify the commitment comparison.

3.2.6 JoinSplit Transfer

The JoinSplit Transfer is the mechanism that allows sending ZECs from one wallet to another. A lot of important details have been outlined so far which is done as part of the set up on, both, the sender's and the receiver's sides, however JoinSplit Transfer is what brigs the protocol to life. As described before, transfer differentiates between the types of values that are passed to the transaction. So far it has been established that shielded ZECs require zero-knowledge proofs, input and output ZECs to be processed by the transaction. One extra detail about the transparent value pool, which is held within the JoinSplit transfer, is that the transparent pool contains some *zatoshis*, a minor denomination of ZECs, to be paid to the miner for processing the transaction. This mechanism is present in Bitcoin and in Ethereum alike, where some percentage of the transaction is payed to the miner who processes it, which gives the miner a choice which transactions to process. This is the case at hand with Ethereum. Aside from holding miner's fee, the transparent value pool also withholds the transparent ZECs that are sent publicly or as part of de-shielding operation. Figure 3.7 shows a simplistic diagram for a JoinSplit transfer in ZCash.

Essentially, the JoinSplit transfer spends the shielded input ZECs and generates

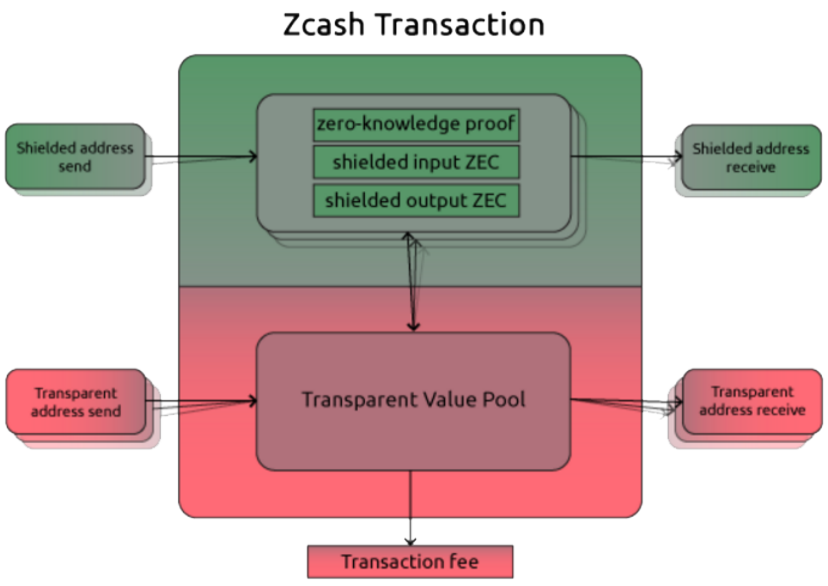

Figure 3.7: Simplified example of a JoinSplit Transfer in ZCash [3]

new output shielded ZECs. JoinSplit Descriptions have been described as the inner part of a JoinSplit Transfer, which holds the data which is to be passed as part of the transaction. JoinSplit descriptions include all of the above mentioned information, namely:

- Values of the input ZECs v_{old}.
- Values of the output ZECs v_{new}.
- Nullifiers for the input ZECs.
- Commitments *cm* for each of the newly generated ZECS.
- Ephemeral key *epk* which is the key agreement public key derived for encryption of transmitted note ciphertexts.
- Sequence of note ciphertexts.
- Zero-knowledge proof of the JoinSplit transfer statement.

- Sequence of hash signatures h which bind the input ZECs to the spending key of the sender.

- Random *seed* value, that is attached to the JoinSplit description.

Majority of the above mentioned items, and their importance, has already been discussed. The JoinSplit transfer can be observed in more detail in Figure 3.8.

One piece of information which is important with regard to the private transactions is how the transaction is exclusive to the sender and the recipient. This feature is not present in dealing with transparent values due to the fact that transparent values do not have secret values like ρ and r as part of their creation. The ephemeral keys that are established for the transmission of the secret values in private transactions 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 of derive a key for note ciphertext decryption.

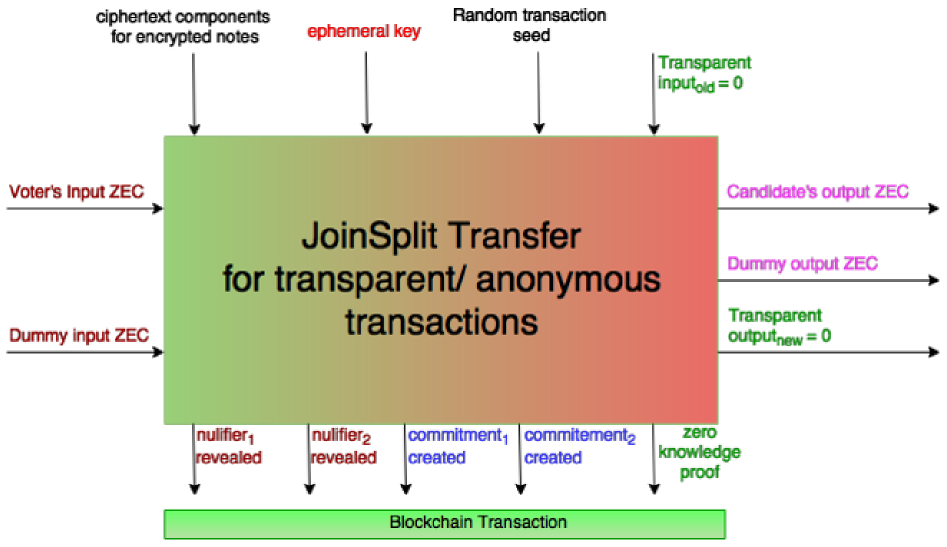

Figure 3.8: Detailed view of JoinSplit Transfer

3.2.7 Dummy ZECs

Having discussed the structure of the ZECs and the JoinSplit transfers a trivial, yet important point needs some attention. This is the concept for a dummy ZEC, or a ZEC which has no value, is important in ZCash. The reason for having such values is that the JoinSplit transfers take two ZECs as input and output. The high level reason is that this is done as means for splitting ZECs into denominations when people require a different ZEC amounting to the values that are not input values. This requires splitting the ZECs which follows the same principle as the split transaction in Bitcoin [30]. The idea here is to not split the coin in manipulate the inputs and outputs in order to transfer exact amount which is required to be sent to the recipient without unnecessary splitting on coins. Therefore, ZCash transactions require to have two inputs and outputs.

However, sometimes the transactions may have fewer than two inputs or outputs. This is the exact situation where the dummy ZECs are used. This feature of ZCash is also exploited for the voting protocol created on ZCash basis. The idea is to create a fake ZEC that will act as an input but will not be directed to any recipient as the destination address is randomly generated. There are two types of dummy coins: the input and the output dummy ZECs. The input ZECs requires the generation of a random spending key a_{sk} with the corresponding derived payment key a_{pk}. The value v is set to 0 and the generation values for the ZEC, ρ and r are picked at random. The dummy path is specified, which is where the note will be destined to go. This path is not checked and therefore the zero-knowledge proof around this note is more lenient as the ZEC is simply a place holder.

In the case of the output ZEC, things are much more simple. The dummy output ZEC is similarly constructed with a v value of 0 and is sent to a random payment address. These mechanisms are invaluable to the functionality of the voting protocol, which will be specified in detail in future sections.

3.2.8 Linkability

Linkability has been mentioned previously, however it plays an important role in privacy of the transactions. Linkability means that if one party, Alice, has sent a a number of ZECs to Bob, who wishes to send it to Carol, if they all used the t-addresses, you can link the transaction from Alice to Carol as the inputs and outputs of the transaction are publicly visible. The use of z-address by Bob, will break the linkability of the ZECs used in the transaction, thus making the transaction more private. This can be observed in Figure 3.9 [42].

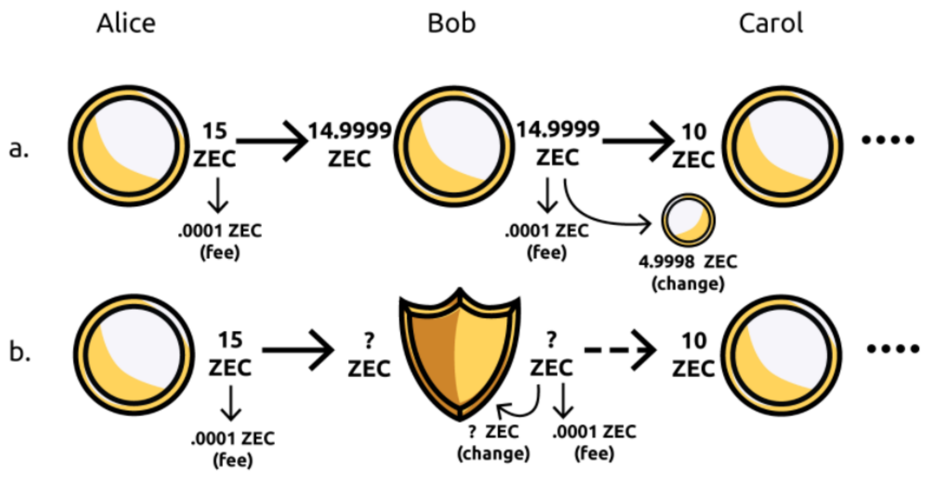

Figure 3.9: Breaking the linkability between t-addresses with z-addresses [42]

Linkability, however may not be as reliable in certain situations. For instance, if Alice sends x ZEC to Bob, Bob will receive $x - fee$ ZEC where the fee is the transaction fee granted to the miners for processing the transaction. Bob, subsequently sends the received $x - fee$ ZEC to Carol, where once again a fee is applied to the transaction. A public party may notice the link between these transactions because the transactions closely resemble each other in value. This is true, assuming Alice and Carol both use t-addresses for their transactions and Bob uses z-address. The transaction linkability may be compromised because the amount does not change between the transactions from Alice to Bob and from Bob to Carol, and the transactions may have happened in short succession from one another. In addition, Alice's input x ZEC for the shielding transaction and Carol's $x - 2(fee)$ ZEC output in de-shielding transaction are both transparent values and are publicly visible. For example, the likelihood of linkability compromised between the same value transactions between blocks 109233 and 109237 is more likely than if the transactions occurred in blocks 109233 and 145032. Figure 3.10 illustrates this example with assumption that the transactions happened in closee succession. [42].

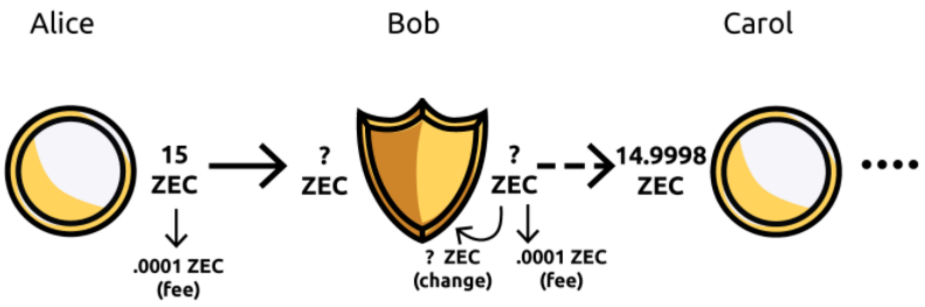

Figure 3.10: Potential for compromised linkability between two transactions [42]

3.2.9 Zero-Knowledge Proving System

Zero-knowledge proof has been has been mentioned several times already and it is of utmost importance to the ZCash protocol. Section 4 is dedicated entirely to explaining zk-SNARKs and how they are constructed for a particular problem. This section provides a look at the bigger picture by explaining the purpose of zero-knowledge proofs in ZCash and the things that it uses zk-SNARKs to prove.

For a zero-knowledge proof to be used effectively, three things must be satisfied, the proof must be:

- *Sound*: The prover can only convince the verifier if they are telling the truth
- *Complete*: If the prover is telling the truth, the verifier will be convinced eventually.
- *Zero-knowledge*: No information is leaked by the prover when trying to prove some arbitrary piece of information.

These are the three rules all zero-knowledge proofs must satisfy. One can find an example of zero-knowledge proving system from Green in [57] and [58]. The application of zero-knowledge proof in zk-SNARKs is outlined in section 4, however the main idea of the proof is that, when the sender transfers a particular ZEC to the receiver, the sender provides a proof that they know the secret values ρ and r that make up a legitimate ZEC. This is a required proof before the sender can spend the ZEC and since spending a ZEC is part of the transaction, the proofs are required for all ZCash transactions.

Since zero-knowledge proofs can be used to prove a particular argument of knowledge to the verifier, as stated above, it can be used to prove that the sender knows a value ρ that was part of generation function for the ZEC that the sender holds. The real question is how do these proofs apply to the concept of a ZCash program. This question is answered by zk-SNARK construct. An implementation of zk-SNARK called *libsnark* [50] allows the conversion of programs into proofs of knowledge. On a high level, the program utilizes the GCC compiler for C family programs to create a Boolean circuit, which is a mathematical model for logical circuits, from a piece of code which proves the knowledge of a value, such as ρ. The boolean circuit has a certain solution which will satisfy the circuit, or return *true* value for it's inputs. Any other value of ρ which is passed down to satisfy the proof will not satisfy the circuit and the prover will fail to provide a sound proof of knowledge to the verifier. The proofs at this stage of the process are simply mathematical equations which the prover knows the solutions to.

The zero-knowledge proof for the transactions can be explained treating the zk-SNARKs as a black box. The transaction proves that the sender knows some value in order to be able to spend the coin, thus giving the coin to the receiver. If one party is trying to prove to another that they know of some value, in typical protocol, the verifying party would request the secret value to obtain the hash, if dealing with hashes, to ensure that the hashed values match. In Zero-knowledge protocols, the purpose is to keep the secret value or *witness* undiscovered. If one wishes to do that, they would need zk-SNARKs and this is exactly what Zcash implements in its proofs.

On a high-level, explained by Lundkvist in [48] some universal program circuit is used as input to a generator function, with some secret values in order to generate a pair of public keys, which can be distributed to both the proving and the verifying parties. The program circuit is a Boolean circuit, which has been described above. The idea is for the prover to use the proving key, public value and a 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 and using a verification function, whose inputs are verification key, the same public value and the proof itself to verify the proof. The proof verification function is a simple function which returns either *true* or *false* depending on the whether the proof has been successfully proven.

The generator function is part of a setup procedure and uses highly secret values as part of the setup stage. These secret values are called *toxic waste* and must be deleted as soon as the setup phase has been completed. 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 cm 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 mentioned above. The resulting proof that is supplied for the transaction is 296-bytes long [3]. Libsnark is able to generate proofs that are relatively small in size.

This provides a relatively high level overview of the ZCash and its core features that will be used in the voting protocol. None of the outlined features have been changed in any way in order to facilitate the voting protocol. More detail on the ZCash protocol can be found in [3]. With understanding of ZCash protocol, it may become clear, that it possesses the required features for a successful voting mechanism. Before this mechanism is described in detail, a deeper knowledge of zk-SNARK construct is required to better appreciate the technology behind the voting protocol.

Chapter 4

Zero-Knowledge Proving System

As mentioned before, a fundamental part of ZCash protocol is the zero-knowledge proving system. This is done with the help of zk-SNARKs, which will be explained in-depth in this section. Zk-SNARKs are not only fundamental to ZCash, but also to the future releases of Ethereum. Zk-SNARKs are a very complex topic, but the general steps and procedures will be outlined in this section. To understand zk-SNARKs better, there is a need to understand the components that make it up. Before proceeding to explain the complex inner workings of zk-SNARKs, some a-priori knowledge of some of the cryptography is required. Other information gives a brief introduction into what type of problems zk-SNARKs are applicable to and the definition of zero-knowledge.

4.1 Application Domain For zk-SNARKs

Prior to discussing zero-knowledge, and indeed the SNARKs, themselves it is worth mentioning that zero-knowledge attempts to implement proofs for the NP-complete class problems, for which it is infeasible to derive solutions to a problem using brute force, however having the correct inputs, will assure the proof is satisfied. The NP-complete set of problems is a subset of a larger set of problems labelled NP problems. NP and NP-complete problems share the fact that there currently exists algorithm that can solve them in polynomial time. To more specific, the NP complexity class, represents a set of decision problems where the instances of proofs which are satisfied by the true answer, can be verified in polynomial time. The NP-Complete, or NPC, complexity class is interesting as there is an ability to reduce any NPC problem into an instance of another problem, which when solved will allow for the solution of the original problem. Therefore, all NPC problems can be reduced to a generic NPC problem. For the sake of completeness, the P complexity class problems can be solved in polynomial time, where given an instance of a problem, the answer true or false to a proof can derived in polynomial time [51] [52] [57] [58].

4.2 Zero-Knowledge Proofs

By definition, a zero-knowledge proof is a proof that convey no additional knowledge other than the correctness of the proposition in question [53]. This was derived by Goldwasser et al. [53] in the paper proposing interactive proof systems, where the "prover" could convince the "verifier" of the correctness of a statement. The question that has been raised by Goldwasser et al. is what would happen in the instance where the verifier could not be trusted the same way as the prover. Zero-knowledge proofs must satisfy three important properties, namely completeness, soundness and zero-knowledge. Completeness suggests that the prover will eventually convince an honest verifier, provided the prover is honest themselves and is telling the truth. Soundness indicates that the prover can only convince the verifier if the statement is true. Finally, the zero-knowledge, intuitively, indicates that the verifier does not learn any useful data by means of zero-knowledge proof, therefore the proof does not leak any data to the verifier. Mathew Green in [57] provides an easy to follow example of the zero-knowledge proof at work and the important properties at work. In short, [57] demonstrates a 3-colorable graph and the attempts of the prover to fool the verifier by unconventional means, such as the time machine, to violate the soundness and completeness property. For instance, to prove soundness, Green in [58] describes the presence of a knowledge extractor, which resembles the verifier and whose job is to extract the original secret from the prover. The soundness, and indeed the zero-knowledge, is proven by contradiction where it should not be possible to extract the secret value from the prover, however it is shown that such an agent exists if liberties were taken and provers secret was to be disclosed. The same approach is taken for zero-knowledge with an agent called the simulator, which serves the purpose of proving, by contradiction, that zero-knowledge holds in these types of proof. There is a small distinction to be made that separates the proofs from the proofs of knowledge. Proofs are simply statements about particular facts, a graph has a three colouring as mentioned by Green in [58]. Proofs of knowledge are proofs that involve an entitys statement of personal knowledge of some piece of data.

So far, the types of proofs that have been discussed are interactive proofs. These proofs require the verifier to be active and present to be able to challenge the prover. There exists a second type of proofs called the *non-interactive zero-knowledge proofs* (NIZK). The essential difference here is that instead of the verifier being present, the prover sends all relevant data for the verifier to be able to prove that the proof is sound and complete. This is achieved by the prover, who computes the challenge, which is sent to the verifier and includes a hash of a provers message as well as an arbitrary message string. This information should provide basic understanding of zero-knowledge, prior to going in-depth into SNARKs.

4.3 Elliptic Curve Cryptography

This is the final part of the pre-requisite knowledge before proceeding to zk-SNARKs. The verification and the operations that the prover requires to do, revolve around *elliptic curve cryptography* (ECC). ECC offers a better trap-door function than, for example RSA, while also providing much shorter keys than that of RSA. The difference between key sizes can be observed in [62], where a 228-bit ECC key is compared to the RSA 2380-bit key in terms of the effort which is required to break the key of equivalent strength. In general, ECC operates on curve functions, some of which have been picked as they serve a better purpose for ECC cryptography than others, and thus have been standardised. The curves are horizontally symmetric and any line can cut the curve in at most three points. The idea behind ECC is that having two points on the curve can yield the third point on the curve, which can then be used to find subsequent points traversing the graph as many times as required. The system has some interesting properties, there exists a point G which is considered a "generation point" which is standardised known value for ECC encryption. Another such interesting point is the "point of infinity" O, which is an equivalent of zero in point arithmetic. This means that a point $P + O = P$, for any point P. Most importantly, given a point P and a point Q it is possible to find a point R such that $P + Q = R$, this operation is known as dot product. Finally, the curve also has order which is a number n that when multiplied by P results in the point of infinity. An interesting point is that, the curve is not represented as the set of real numbers, primarily because the use of real numbers in cryptography would result in the ability to reverse logarithms and break the entire concept, as well as the fact that the amount of memory required to store the and represent numbers would increase by a significant amount. The curve is instead represented with whole numbers, and it is still possible to find a point with the same logic as above. The example curve can be seen in Figure 4.1. Using this technique, it is possible to take messages and represent them as points on the curve, although it is more complex in reality. The whole idea of ECC is that given a public starting point, and using the dot product operation a prime number of times on a known curve function. The prime number that was used for the dot product is the private key, and it is infeasible to compute the private key from the public key, or attempt to break the encryption via brute-force by guessing the number of times the starting point, has been used in the dot product function. This is what is called the elliptic curve discrete logarithm and creates a much better trapdoor function that RSAs prime number factorisation [60] [62].

Some of the keys like the transaction key (pk_{enc}) and the key agreement function keys, along with some other not mentioned keys, utilise ECC for key generation and agreement. In particular Curve25519 is used for such purposes. Without going into too much detail Curve25519 is the state of the art curve for elliptic curve Diffie-Hellman (ECDH) for key agreement. Bernstein in [47] outlines the new speed record set by using this curve for high-security Diffie-Hellman computations.

4.4 The Structure Of zk-SNARK

The computation of any SNARK starts with a C program. Since ZCash is the main platform, that serves the basis of the electronic voting system, I will discuss zk-SNARKs with relation to ZCash values and what ZCash attempts to prove by use of the SNARK. ZCash makes use of a library called *libsnark*, which essentially helps to build SNARKs from any C program provided to it. A good high-level overview of the steps involved in the design of the zk-SNARKs is shown in Figure 4.2. Majority of the section will be based on the Ben-Sasson et al. work [54] where libsnark library and the derivation of

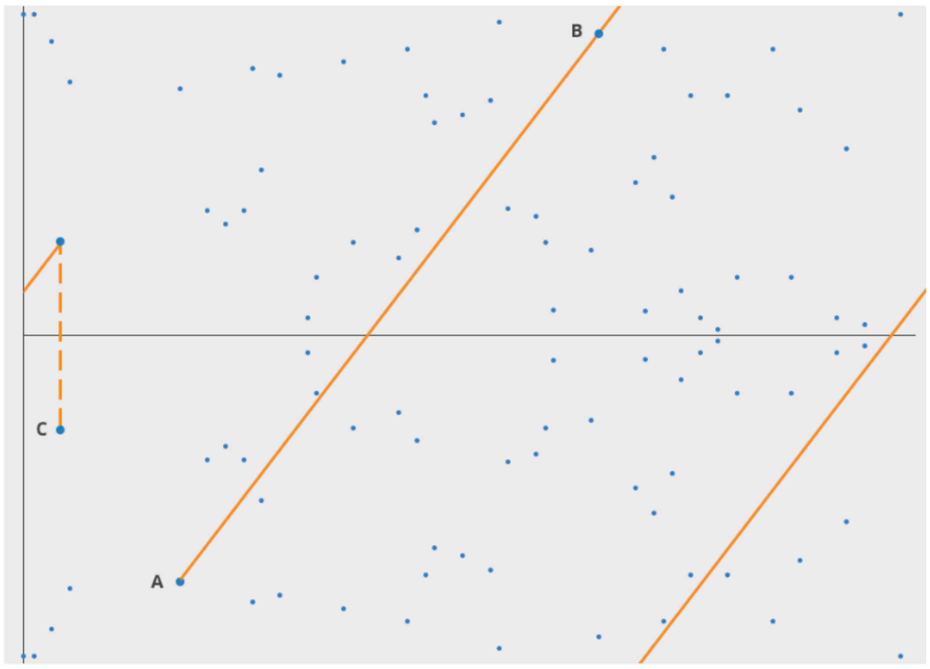

Figure 4.1: Elliptic curve represented with whole number points [62]

SNARKs is explained in great detail. In essence, libsnark as described by Ben-Sasson et al. [54] and by Madars Virza in [56] has three main parts, namely the compiler, the circuit generator and the generation of SNARK for CircuitSAT, or circuit satisfiability problem. The conversion of a C program, which in this case can be a part of the proof where the prover attempts to convince the verifier that they possess a value s such that the random value s has been used in the generation of their ZEC. This conversion is done with a compiler into a version of a TinyRAM program. TinyRAM, by definition, is a "minimalistic RISC random-access machine with Harvard architecture and non-deterministic random-access memorytailored for efficient computation of non-deterministic computations". To put more simply, TinyRAM is a port for the GCC compiler which allows efficient compilation of programs while having reasonably simple instructions to design Boolean circuits for verifying correctness of proofs. Although the code size overhead is up to four times bigger than a complex CISC architecture of x86, and the execution time is slower than the same x86 architecture by a factor of 2-6, the compiler compensates for these overheads by the fact that TinyRAM has a compact circuit for correctness verification [54][56].

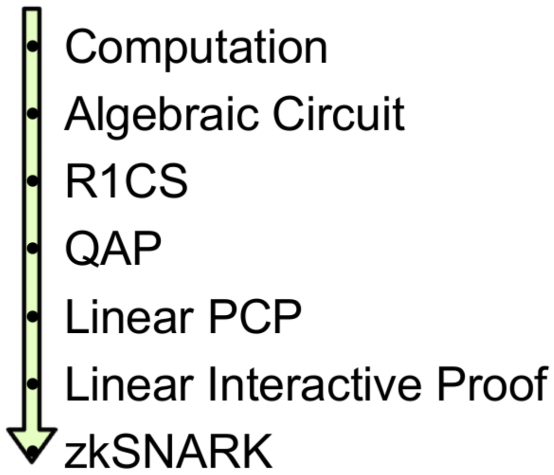

Figure 4.2: High-level overview of zk-SNARK [58]

Following from the compilation of the C program to verify the knowledge of s value for a ZEC, the TinyRAM program must be reduced to a Boolean circuit to be used for the circuit satisfiability problem, CircuitSAT. A Boolean circuit is simply a mathematical representation of digital logic gates for a problem. The Circuit satisfiability problem is merely a decision problem, where an assignment of inputs that returns *true* value, must be determined for a given Boolean circuit. In other words, if one could assign truth values to a mathematical formula such that the formula evaluates to truth, the formula is satisfied. With a basic example from Reitwiener in [52]: $((x_1 \wedge x_2) \wedge \neg x_2)$ is a Boolean formula which is not satisfiable and therefore does not lie in the set of problems considered to be satisfiable or simply SAT.

Now it is clear exactly what the circuit generator part of libsnark is attempting to create. On a high level, an execution trace of a TinyRAM program is obtained, where the execution trace is simply a time-sorted list of CPU states. In other words, each different state of the compiled TinyRAM program. The goal is to design a circuit such that the circuit at the end is valid in terms of code consistency and memory consistency. The creation of the output circuit revolves around the creation of three sub-circuits. Code consistency aspect of circuit generation is involved with the use of a transition function which is applied to each 2 consecutive states in order to determine whether the states can logically follow one another, which also ties in with memory consistency. This also is the most complex sub-circuit as it is largely dominated by the size inner sub-circuits for multiplexing bit strings and arithmetic logic unit (ALU). The consistency is ensured by applying the transition function to each of the states. The states at hand are simply the registers and the program counter values for each stage of the program execution. The transition function acts like any other CPU, which uses things like register-file multiplexing or ALU, amongst others. The code consistency step provides us with a sub-circuit to add to the main circuit function described later in this section. The reduction optimizations for this step lie in efficient implementation of ALU functions. For example, bitwise operations such as AND, OR and others, are computed with binary representations, where addition, subtraction and multiplication is implemented in integer representation [54] [56].

Memory consistency is a more difficult step. In order to verify the transition function's validity, memory consistency is important. Memory consistency is namely the verification "that every load operation from any address in memory retrieves the value of the last store to that address". On a high level, difficulty arises in storing memory correctly, as memory is external to the CPU, and the entries that are written to memory at any point in time, may prove to be totally unpredictable a time later. One potential solution would be to append a memory snapshot for every state in the trace, however this is not efficient and would result in a quadratic worst case execution time for the generated circuits, or $O(T^2)$. The more efficient solution is to create another sub-circuit as input to the circuit function. This memory consistent sub-circuit is the same execution trace sorted in terms of accessed memory addresses. The memory consistency circuits are like transition functions between the states, only here these circuits verify that:

- The items stored in a memory address are the items that are returned upon the load function
- The items stored in memory addresses are not altered upon loading of these values form memory
- Loaded values match after consecutive loads from the same memory address
- If an item is stored in memory after a load and a store operation, the memory circuit does not perform any verification [54] [56].

The reduction optimization that has been implemented here is the fact that by completing the transition function for code consistency on opposing end of the routing network, which is discussed below, the memory addresses being accessed by the operations and values stored within for a particular state, have already been computed. As described by [54], by making modifications to the routing network, the function of memory circuit, for memory consistency, is to merely check the ordering of these value and address pairs.

In order to check if the memory trace is correctly sorted, the routing network is used. Libsnark uses Benes routing network which has $O(logT)$ layers of T nodes, where each node in a layer is connected to other two nodes in the next layer. A T-packet routing network is a directed graph of T sources and T sinks with switches where T packets travel to T sinks according to a particular permutation without using any particular switch twice. As a bonus, this adds more non-determinism to the circuit generation algorithm. Constraints implemented into the protocol, can help ensure that a permutation has been implement some permutation to the inputs and outputs for the graph. Once again, the graph colouring problem can be observed here in a slightly different context. The check of the routing constraints is the most expensive sub-circuit out of the three because, as described in [54], there are $\Theta(TlogT)$ total nodes in the routing network, which are compared to the T copies from the first two sub-circuits each. On a high level, the optimization achieved here is the ability to package the value/address pair from memory consistency sub-circuit into small packets, which are then easy to transmit and compute the routing constraints [54].

Overall, the circuit generation step, designs a circuit of size T such that $T \cdot (|C_{TF}| + |C_{MC}| + O((logT)^2))$, where C_{TF} is the trace state transition functions, C_{MC} is the memory circuit for memory consistency step and $O((logT)^2)$ is the Benes routing network. In summary, the TinyRAM assembly code is reduced to satisfiability of a constraint-satisfaction problem on a routing network, where an arithmetic circuit verifies all the constraints of the problem given a correct assignment as input [54].

The above aspects describe the soundness as most of the discussion has been about the verification of the validity of execution trace of the given TinyRAM program. The completeness is achieved by an entity called a witness map. This is something that is run by the prover when the proof is generated. This consists of two main steps. The first step is taking the inputs for a TinyRAM program and outputting an execution trace for P, which has been described above. The second step takes in the execution trace as input and outputs a satisfying assignment for the generated circuit. The satisfying assignment is the input to the circuit which returns the value *true* for the particular circuit. This is done by finding the satisfying assignment for each transition function C_{TF} of the circuit, finding the satisfying assignment for each C_{MC} of the same circuit and, finally, finding the satisfying assignment for the routing network sub-circuit, which verifies that the sorting of transition functions is correct. Having completed the above steps, the witness map is derived [54].

So far, the process of deriving the circuit has been described in terms of the functions of the libsnark. An alternative step is to convert the assembly code into *rank-1 constraint system* (R1CS), which has similar concept to the circuit derivation from libsnark. Essentially, R1CS deals with the simplified operations of the complex code instructions to be able to convert them to groups of three vectors to define the constraints on the system. R1SC, as described in [59], is a sequence of groups of three vectors, a, b and c, and the solution vector s, where s must satisfy the equation: $s \cdot a \times s \cdot b - s \cdot c = 0$, where $\cdot$ is the dot product of the two vectors. In other words, by multiplying the values in same positions in s and a vectors, and adding these products, and doing the same for the b and s vectors, the third result, namely s and c vector, will be the product of the other two results. There may exists more than one constraint per system, and the number of variables in each vector depends on the number of variables in the whole system. To give a simple example, in a code sequence that evaluates x^3, the simplified operations would result in:

$$i = x \times x$$
$$result = i \times x$$

The variables that would be present in the s vector would be: *one*, x, *result*, i. The number *one* is simply a dummy variable representing the number one in case values in operations cannot be represented in terms of numbers stored in the vectors. Each of the gates is now summarised as the variables in the vectors, following the equation outlined above. To be more specific, the first and second vector for the first gate, in the equation presented above would look like: $[0, 1, 0, 0]$ and the third vector would be $[0, 0, 0, 1]$ as the variable i is stored in the last place in the vector. This step would be done for each of the logical operations of the compiled code. A much more comprehensive example can be found in Buterins article [59].

R1CS is then converted into *quadratic arithmetic program* (QAP), which is also created as part of libsnark for derivation of *linear probabilistically-checkable proof* (PCP), which will be discussed further in this section. Alternative approach is to use the *quadratic-span programs* (QSP). Libsnark builds on the QAP approach, even though both implementations are equally efficient. As described by Ben-Sasson et al. in [54], one of the reasons that QAPs are used over QSPs, is that the QAPs are significantly simpler to construct, which is crucial for practical applications. It also provides smaller reductions for *arithmetic* circuits. On a high level, the objective is to convert the vector group from the R1CS from into a QAP form, where the constraints are derived by evaluating polynomials at x coordinates. This is done using Lagrange interpolation as mentioned by Buterin in his article [59] with an example following the R1CS example. This helps to derive a group of coefficients, which when evaluated at x coordinates of the derived polynomials, will produce the constraints outlined by R1CS step produced in the above step. One of the advantages of this step is that the R1CS constraints do not have to checked individually. Instead, all constraints can be checked at the same time by doing dot product operation on the polynomials. The verification process of these polynomials involves doing additions and multiplications of polynomials, which will result in a polynomial, which result must be zero in order for the check to pass. Similarly, if the polynomial does not return a zero result, then one of the input values must be inconsistent and the check is not passed. The polynomial is not evaluated to find the non-zero result, instead the polynomial is divided by another polynomial, and if the division is done without remainder, the check is considered to be passed [54] [59].

Final part of computation before zk-SNARK derived, is the linear PCP. A PCP is a proof that is checkable by an algorithm which implements a degree of bounded randomness. Ben-Sasson et al. in [54] state that their linear PCP consists of 5 queries of 2 field elements each, where each of the queries is generated in linear time. With the help of PCP, it is possible to accept or reject proofs which do not pass this stage, with particularly high probability. The process of deriving a SNARK involves compilation of linear PCP into a 2-message *linear interactive proof* (IP). Here the purpose is that the verifier is only able to apply linear functions to the message. The high-level approach is to add consistency-check query, which is a random linear combination of linear PCP queries. The linear IP is compiled into the SNARK, by forcing a polynomial-size malicious prover to act if it were a linear function, given a cryptographic encoding function. The interaction between the prover and the verifier to verify the proof is as follows. Given an encoding function, the SNARK generator samples the verifier message for the linear IP and outputs an encoding of the messages. Based on the encrypted messages and linear PCP proof, the prover is able to homomorphically evaluate the inner products of the messages and return, as a proof, the encoded answers. The verifier checks the proof by running the linear IP decision on the encoded answers. It is also worth noting that if the linear PCP is honest-verifier zero-knowledge, meaning that the verifier is acting according to the protocol, then the SNARK also becomes zero-knowledge [54].

Ben-Sasson et al. have since made modifications to their TinyRAM architecture which is described in [55]. The main goal of the modifications was to improve security and efficiency of their architecture. Though a lot of the work has been based on [54], the updated version of TinyRAM improves the architecture by introducing the universal circuit generator, unlike the previous version which improves on the inefficiencies for things like prover and verifier key generation and does not hardcode a copy of the program into the circuit. The new circuit generator is universal and does not depend on the program, but merely on the program size. Universal circuits, when combined with CircuitSAT problems, such as zk-SNAKRS, allow for the universal parameters for the proof. It begins to follow the von Neumann paradigm, and supports the von Neumann RISC architecture with its modification called "vnTinyRAM". The main contribution is the upgraded efficiency of the circuit generator and the zk-SNARK for the circuits, which meant creating tailor made implementations of cryptographic libraries instead of using the off-the-shelf resources.

With the mention of the above techniques to obtain the SNARK some things may still be misunderstood. Here is some more clarification on the actions that take place from the QAP to the zk-SNARKs with the use of ECC. Elliptic curve pairings take the concept of elliptic curve cryptography one complex step further. The best way to think about pairings is to view individual curve points as one-way encrypted numbers. This means that instead of viewing the traditional elliptic curve math as one that lets one check linear constraints on the numbers, pairing lets one check quadratic constraints. For example, if $P = G \times p$, $Q = G \times q$ and $R = G \times r$, the pairing approach allows you to check $e(P, Q)$ for some point arithmetic. The operator $e(P, Q)$ represents the pairing operation as per [60]. This pairing can also be called "bilinear mapping", as it satisfies the following constraints:

$$e(P, Q + R) = e(P, Q) \times e(P, R)$$
$$e(P + S, Q) = e(P, Q) \times e(S, Q)$$

These pairings are later used in the proof of solution for a QAP by the prover to the verifier and essentially, allow for the zero-knowledge aspect of SNARK, which will be discussed later in this section. As already mentioned, the purpose of pairing is to combine two elliptic curves in such a way that they satisfy bilinear constraints and to obtain the output in the form of an element F_p^{12}. An entity called the divisor, which is an alternative method of representing functions on elliptic curve points, is required for pairing. The divisors job is to count zeros and infinity values of the function on the curve points. The points P, Q, R and any other point can then be represented as $[P]$, $[Q]$, $[R]$ as a standard notation for divisors, which includes the point P, Q and R along with the zeros and infinities as mentioned above. A basic example from [60] can be seen in the next example. Suppose that we have a point P such that, $P(Px, Py)$, let the function be $f(x, y) = x - Px$. The divisor for the following function is the $[P] + [-P] - 2 \times [O]$. Buterin in [60] outlines that since the function goes towards infinity, it equals to infinity at the point of infinity. The equation of the curve for pairing, as per example is $x^3 = y^2 + b$. The reason that the point of infinity is multiplied by negative 2 is that infinity is represented as negative entity, and because of the equation of the curve, y reaches infinity 1.5 times faster than x, in order for y^2 to keep up with x^3 [60]. Therefore, if a linear function includes x only, the point of infinity is multiplied by 2, otherwise if y is present, then it is multiplied by 3. The idea is that having functions defined via their divisors for point P and point Q, including the order for of the curve, it is possible to derive one divisor from the product of the above-mentioned divisors in order to combine them. This is important because separately, the divisors, $[P] + [Q]$ are not the same as $[P + Q]$. The final step is to exponentiate the resulting equation to a power of the multiplicative order in the output F_p^{12}. The result is a product of functions raised to a power, which is a limited form of one-way homomorphic encryption. This allows to perform equality checking in the last stages of the proof [60] [61].

The possibility of such pairings also means that the types of elliptic curves must be considered, otherwise security issues may arise. Pairing on some curves is not possible due to the fact that if a value, called embedding degree is large enough, the computation of the pairing is not feasible. On the other hand, if the embedding factor is too small, the discrete logarithm problem for elliptic curves can be reduced and the private key can be obtained by an attacker, which is shown by the MOV attack [60] [63].

The final part of the zk-SNARK is possible due to the knowledge-of-exponent touched on by Buterin in [61]. This means that given two points P and Q, where $P \times k = Q$, it is not possible to come up with an alternative value $C \times k$ as the exponent is not known. This extends to the fact that given a pair (P, Q), a pair (R, S) can only be derived by multiplying both P and Q to give R and S points, which are multiplied by a factor that a prover knows. This roughly defines the *knowledge-of-exponent* (KoE). This goes back to the point made above, that it is not required to know the value k, merely the pairing between $e(R, Q) = e(P, S)$ can be checked to determine whether it holds. This similarly can be applied to a set of points (P, Q) but instead of the prover knows the coefficient to multiply each of the elements of the set by to reach the points (R, S). An important point, stressed by [52] and [61] is that the value k must be deleted after the setup step has been performed. This is called the *toxic waste* and includes five values including k, which if left alone, could mean that anyone could generate fake proofs [61].

The QAP, as mentioned above, is a set of polynomials which follow the constraint $A(x) \times B(x) - C(x) = H(x) \times Z(x)$, where A, B and C are linear combinations of sets of polynomials. Entities A, B, C and Z are part of the problem statement and it is not feasible to deal with large polynomials. Instead, a part of the *trusted setup* step is the evaluation of these polynomials at a secret value, which is also a part of toxic waste. The prover must not know these secret values, but must instead be able to compute the above from the set of points provided to them during the trusted setup. The sets of polynomials are all raised to the same coefficient in order to be able to prove that $A \times B - C = H \times Z$. A value used to bring the linear combinations to the same coefficient is part of the toxic waste of the trusted setup also. A pairing check can then be done between the pairings of each of the linear combinations at a secret point t. So far, the part H of the equation has not been mentioned. H is also a polynomial, and the task is to predict the coefficients for H for every QAP solution.

This is done using the generation point G, mentioned in the ECC section and obtaining a product of G with t at different exponent of t. Since ZCash is a primary protocol, the sequence goes up to around 2 million powers of t. This is to ensure that H at a value t can always be computed. The purpose of the proof is not to prove that a solution exists for a problem, but it is preferred that the prover proves the correctness of a solution or to a problem exists when some of the parameters are limited. The libsnark as described by Ben-Sasson in [55], outlines the procedure for verification of these proofs of knowledge of QAP solution. These roughly outline the checks required to verify the steps mentioned above, for example, the linear combination check for A, B and C and the fact that linear combinations have the same coefficients. As mentioned by Buterin in [61], the difficulty in making the proofs from simply instructions is that, from the single gate in a circuit, the operation must be processed through the elliptic curve operation and produce a zero-knowledge proof of it after. This alone adds an overhead of 20-40 seconds to every ZCash transaction.

To refer back to the high level description of the zero-knowledge proving scheme from Section 3, given a program circuit as input for the generator function, the linear combinations of sets of polynomials A, B, and C, are evaluated about a random point t, which is considered toxic waste. In the interaction between the verifier and the prover, the verifier sends the prover a set of linear PCP and consistency-check query to be sampled by the generator function to output the parameters for prover and verifier keys pk and vk.

The prover utilizes the proving key pk in order to compute the coefficients for the polynomial H, and use these coefficients, the secret witness and the prover key to generate the proof for the verifier. The verifier computes a "*custom verification key* which is specific to the instance" [61], and computes pairings to perform checks, such as the coefficient and division and others, outlined above [55].

One final aspect of zk-SNARKs is the compactness of the proof. According to [54], the proof length is merely 2576 bits, which can be fitted into any TCP packet size. With regards to ZCash, the proof of having the random generation value s as well as knowledge of entities such as the nullifier for the ZEC, amongst other proofs is incorporated into the zk-SNARK. The proofs are joined to make a total 296-bytes complete proof of knowledge for a particular ZEC as per [3].

It is, perhaps, clear that this zk-SNARK's inner workings are quite complex, but despite the initial proof generation, they are very efficient in terms of size and verification speeds. This chapter concludes the journey into the specification of ZCash and it's inner mechanisms. Having acquired some of the required knowledge, it is possible to proceed to the description of the voting protocol in greater detail.

Chapter 5

ZCash Based EVoting Protocol

Prior to describing the voting protocol, it is worth mentioning that the underlying ZCash protocol [3] 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.

5.1 Protocol assumptions

Before proceeding to explain the protocol, the assumptions of the work must be stated. These include the assumption of a confirmed identity, meaning that the protocol assumes that the identity of a potential voter can be verified by the dealing with the X.509 certificates [65] and a Certificate Authority (CA) to verify those identities. Any other verification mechanism can be in place to facilitate the verification function. This step is required in order to ensure that the person attempting to register for voting, is the person they claim to be. This does not, however, guarantee the protection against Sybil attack [64], which is one of the serious issues in digital identity domain. Sybil attack revolves around a person having several identities and using these to gain access to the system from multiple personalities.

Second assumption is that the system is built on top of ZCash platform, or that there is a script running on top of ZCash platform which allows for the legal authorisation of a vote transaction on behalf of the voter. As the voter does not get to handle the ZEC, or the vote token, the assumption of a script capable of transferring the transaction with the consent of the voter is in place.

One of the final assumptions of the work is that the first block in the blockchain for a particular election is known, so as to be able to do audit verifications later. This can be the genesis block on a new blockchain or a recorded block to be considered as a starting point of the election on the blockchain.

5.2 High-Level Overview

The voting protocol can be separated into four distinct steps: registration, notification, voting, count/audit. Voters register to participate in an election or poll of interest by visiting a registration page where their details are recorded and stored for audit purposes. As per verification assumption, the X.509 certificate is assumed to have their active email address stored as part of the certificate [65]. The certificate is saved by the voting system and upon commencement of the election a one-time unique link to a unique ballot is sent to the registered voter's email address. Despite ZCash being decentralised, one central authority exists to issue the details of the election and specify the wallets for each of the candidates. During the registration step, the user is queries whether they possess a ZCash wallet, as it is necessary to posses a ZCash wallet in order to proceed with voting. If the user does not have a ZCash wallet, then upon finishing registration, the user is redirected to a website which allows the creation of such a wallet.

Upon visiting the vote, the voter is able to pick which candidate receives their vote. Only one vote is allowed per person and the system allows the person to re-vote. The voter is also only able to cast a single vote for one candidate, resembling the First-Pass-The-Post (FPTP) voting scheme. The voter provides the details of their ZCash wallet, to which they will receive a vote token. Upon completing the ballot, the voter authorises the voting transaction for the system. The system records that the voter has voted, updates the database with the wallet details and the vote status as well as issuing exactly one ZEC as a vote token, representing the one vote from the voter for a particular candidate. The token first arrives in the candidate wallet and with the presence of the second assumption of possessing a redirection script, the token immediately gets packaged into a further transaction to be forwarded to the candidate specified by the voter on the ballot.

The voting procedure is repeated for each of the voters. There are two variants of the system which allows for improved linkability at the cost of privacy of the vote or vice versa. The two variants discussed in this section describe this trade-off. Regardless of the variant used, the rest of the protocol remains the unchanged. Upon conclusion of the election, the candidate wallets send all of the acquired vote tokens to a common pool, which allows the verification of the entire election. The candidate with most vote tokens wins the election and the audit procedure that follows the election validates it.

The audit procedure helps to ensure that the election has been ran properly without the system suffering from throw-in votes by coercers. The system ensures that the following equality holds:

$$\sum_{n=1}^{x} Tv = \sum_{n=1}^{x} Vt = \sum_{n=1}^{x} \frac{Tx}{2} - \sum_{n=1}^{x} Tc$$

Where Tv is the total number of votes authorised by the voters, Vt is the total number of vote tokens issued, Tx is the total number of transactions on the blockchain and Tc is the total number of candidates. In the above equations these values are a constant of 1 .

In other words, the total number of authorised votes must be equal to the total number of vote tokens issued which must be equal to the quotient of total number of transactions divided by 2, less the number of candidates present in the election. The division by 2 of the total number of transactions is to account for the fact that it takes 2 transactions for the voter to obtain their vote and then to issue it to a particular candidate. If this equation does not hold the election is considered forfeit and may require to start again.

Finally, the voter is able to view the their votes as these are the transaction on the blockchain that follow the same rule as transactions in ZCash. In case the voter voted incorrectly, they are able to re-vote, and the final equation has to be balanced accordingly as the the quotient of transactions will not equate the total number of the vote tokens issued and the number of total votes. In the case that this happens, the candidate wallet with the incorrect vote issues the token back to the voter, once the voter has authorised the re-vote. The high level overview of the voting system can be observed in Figure 5.1.

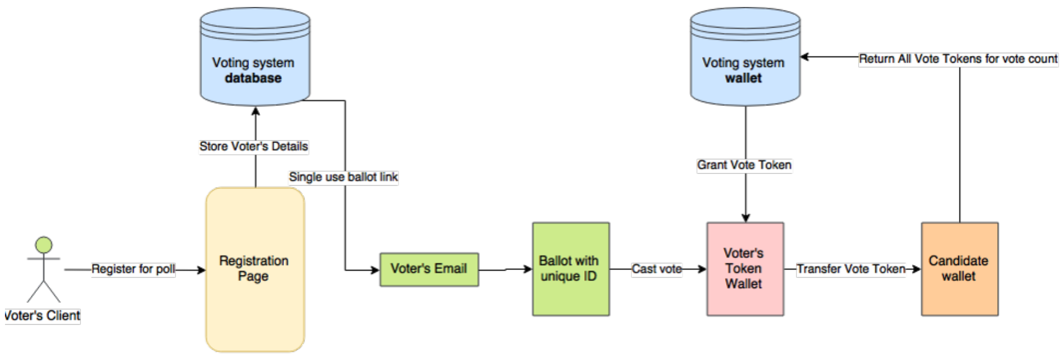

Figure 5.1: High level overview of the voting protocol

5.3 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 voter has voted, and is a control mechanism to disallow unregistered people to participate in the vote. Similar to Zeus voting protocol [14], the system at hand follows the same principle with a bit more leniency. Only the registered voter is able to vote, however, once the election has started, any registered voter after the commencement of election is still entitled to a vote. In theory, it is possible to enforce a similar scheme as in Zeus, where only registered voters prior to the election commencement time are able to issue a vote, and the latecomers are not able to vote despite attempting to register.

Registration can be viewed in Figure 5.2. A potential voter who wishes to proceed with to the registration page, transparently, communicates with the server which sends the voter a challenge signed with the server's private key. The potential voter is then required to solve the challenge and issue a response comprising of the solved challenge and a time stamp, signed with the voter's private key. A well known protocol called *Challange-Handshake Authentication Protocol* (CHAP) [66] describes this exact procedure. This step is required in order for the client to authenticate themselves to the server and while also providing periodic validation of said remote client.

Part of identity verification is dealing with the first assumption established earlier. The X.509 certificate is verified with CA's public key to ensure the issued certificate has not been tampered with. Once the voter is verified, authenticated and has proceeded with registration, the server stores a copy of X.509 certificate for the voter as part of proof of registration as well as further fields of information which will be filled in at later stages of the protocol, such as the receiving wallet address and the voting status. The active email address is confirmed with the voter during registration and in order for the voter to be able to proceed with the voting procedure.

As mentioned above, if the voter does not have access to a ZCash wallet, they would need to create one in order to participate in the election. Figure 5.2 outlines the steps taken for a voter to register for the election and the data that is processed in this step.

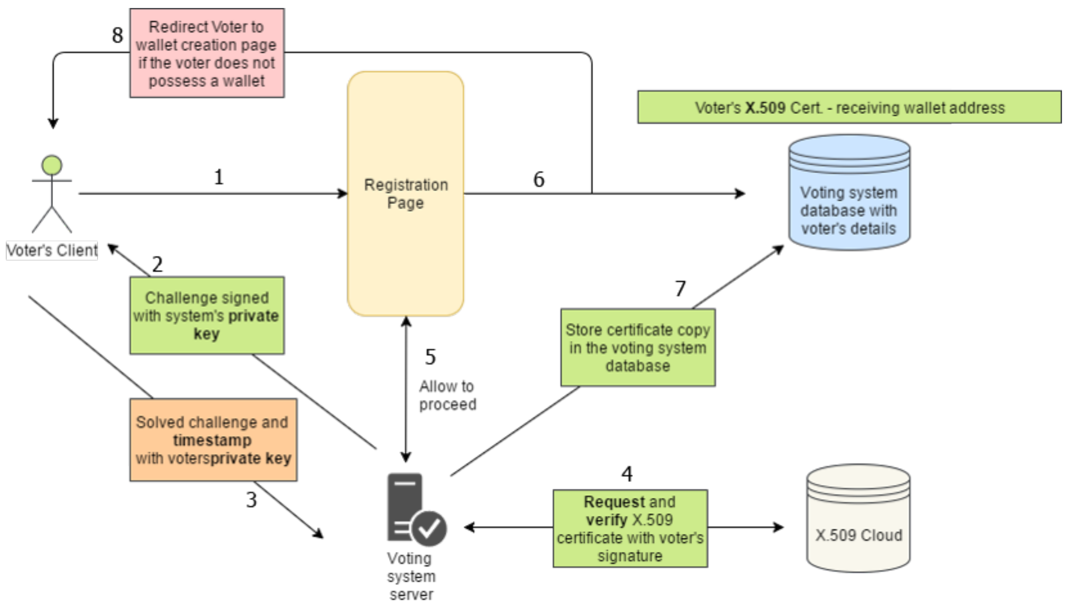
8
Redirect Voter to wallet creation page if the voter does not possess a wallet
Voter's X.509 Cert. - receiving wallet address
1
Registration Page
6
Voting system database with voter's details
Voter's Client
2
Challenge signed with system's private key
7
Store certificate copy in the voting system database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

Figure 5.2: Registration step of the voting protocol

5.4 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 registration details are assumed to be checked the same way they were checked during the registration step, using the CHAP protocol. The voting server issues the the invitations only when the administrator of the election issues the details for the election. This is similar to the Zeus protocol [14] where the administrator too, inputs the details of the poll as well as the list of the registered users.

The system server obtains the details from the copies of X.509 certificates stored in the server's database and issues the vote links to all of the registered bodies. Depending on the setup, if the voters are allowed to register and participate during the vote, the server periodically checks for new added voters and issues the links to the emails specified within the certificate. The links are active for as long as the election and expire as soon as the election timer has expired. Figure 5.3 outlines the invitation step for the voting protocol.

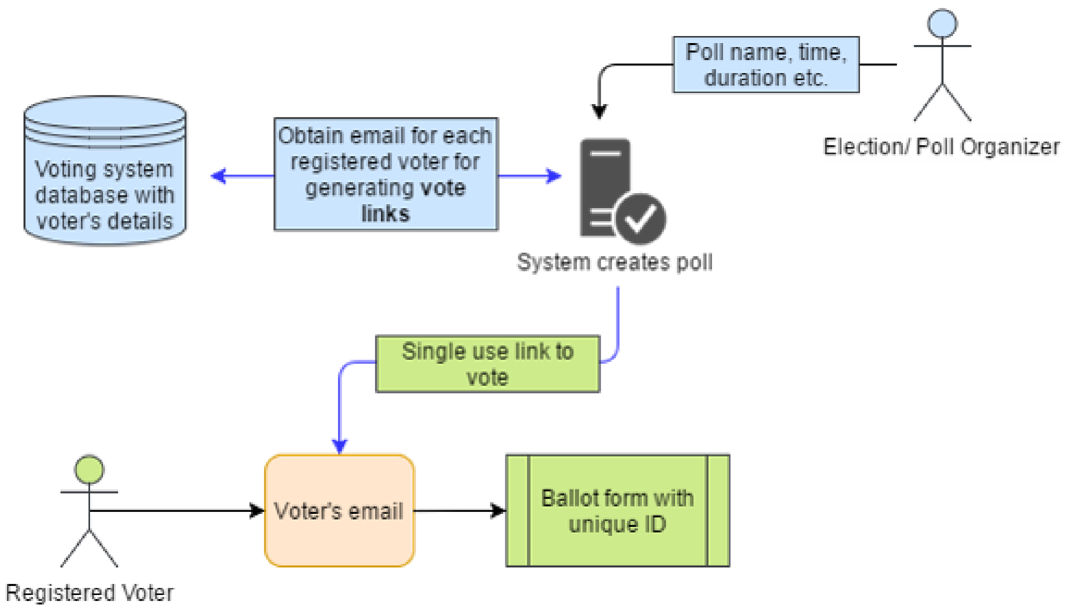
Poll name, time, duration etc.
Election/ Poll Organizer
Voting system database with voter's details
Obtain email for each registered voter for generating vote links
System creates poll
Single use link to vote
Voter's email
Ballot form with unique ID
Registered Voter

Figure 5.3: Steps taken prior to invitation of the voter to the ballot

5.5 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 in order to send and receive the payments. In order 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. The voter can use t address, but the anonymity of the vote is not guaranteed.

Once the voter has provided receiving address on the ballot and picked the candidate of their choice, the voter is able to proceed with passing of the vote. Prior to proceeding with the vote, the voter is presented with authorisation notification, which states the legal binding this voting system and that the voter authorises the transaction that will take place by the system on their behalf directing the transaction to the candidate of their choice. This functionality is possible with the assumption of the script which will execute these timed transactions.

This system is in place as there is no limit to how many ZECs can be sent in one transaction. The wallet of the voter may also not have any ZECs. Of course a script, could simply ensure that a transaction of no more than one ZEC is sent at any given time, but a voter who just created a wallet may not have any ZECs. The denomination of the ZECs can be from 1 ZEC to 1 Zatoshi, the smallest denomination of ZEC, in case a dishonest voter found a way to hijack the ZEC that arrived at his wallet. This is one of the risks of this system which will be discussed more in the next section. Some mitigations for this risk include, as mentioned above, the use of smaller denomination of the ZEC, or reducing the amount of confirmations prior to verifying a transaction, to speed up the transition between the system and the voter to grant the voter less time to hijack the ZEC.

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. At this point, the system tracks the number of voter who have authorised the vote, and the amount of ZECs that have been issued by the ZEC pool. Going back to above equation, these numbers have to balance.

Finally, once the voter has authorised the vote, the system changes the voter's status in the database. Figure 5.4 summarises the voting procedure from the point of view of the voter. The systems actions denoted with blue arrows happen transparently within the system.

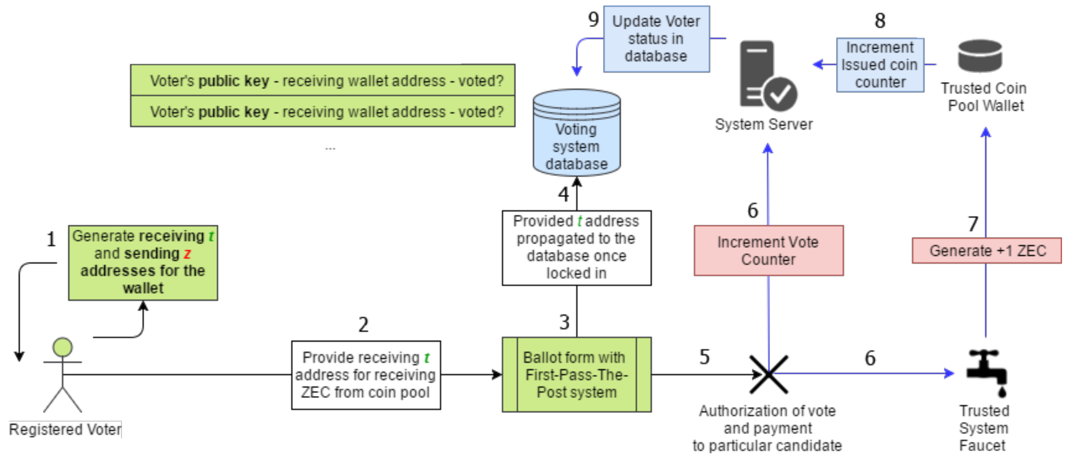
9
Update Voter status in database
8
Increment Issued coin counter
System Server
Trusted Coin Pool Wallet
Voter's public key - receiving wallet address - voted?
Voter's public key - receiving wallet address - voted?
...
Voting system database
4
Provided t address propagated to the database once locked 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

Figure 5.4: Overview of the voting step

5.6 Vote Transfer Variations

As mentioned previously, the system may have two variations and the trade-offs between them. The first variation comes as part of the candidate using a receiving t address. This means that the voter sends a vote token from their z address into the candidate's t address, resulting in a de-shielding transaction. This keeps the linkability of the vote token, meaning, when the candidate empties their wallet, the voter would be able to link the coin all the way to the final ZEC pool, which is used as part of the audit mechanism.

The second variant allows the candidate wallet to use a receiving z address. This makes the vote transaction completely private at the cost of linkability of the vote token. Regardless of the variation of the system, the ZEC pool issues a ZEC vote token to the candidate's wallet on the address that was specified on the ballot form. This token is then forwarded to the candidate's address as a blockchain transaction. More in-depth explanation of these variations is explained in the next sections.

5.6.1 Transparent Candidate Transaction Variant

As outlined above, the use of receiving t address by the candidate makes the candidate transparent. This means that at all times, the total number of votes that the candidate has received is visible by the public, resembling the Bitcoin [28] transactions. This happens as part of a de-shielding transaction, where the anonymous ZEC value is converted into the transparent ZEC value observable by all.

This may cause legal issues in some countries. The vote is transferred from the ZEC pool to the voter's wallet. The purpose of this transparent transaction is for public verifiability that, indeed, one ZEC vote token has been issued by the pool to the voter, verifying that the voter has not received a forbidden number of vote tokens from the start. The number of confirmations can be set to 0, as mentioned above, in order to not verify the transaction from the ZEC pool, as it is considered a trusted source. On the other hand, it is possible to leave the confirmations to guarantee the integrity of the protocol, though it would take longer to deliver the ZEC vote token to the voter.

This results in a less complicated setup, still guaranteeing the anonymity of the voter and further traceability of the election. This also means that everyone would be able to view the details of the transaction. Figure 5.5 demonstrates the general steps taken for the transparent candidate vote passing.

5.6.2 Private Candidate Transaction Variant

The transaction between the candidate and the voter becomes private if the candidate uses z address. Inherent to the 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, which has been outline in section 3, 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. Figure 5.6 shows a detailed diagram of the steps taken to create the transactions. As stated above, the first transaction can vary in the number of confirmations required before the transactions are accepted. Similarly, the first transaction is the same for both

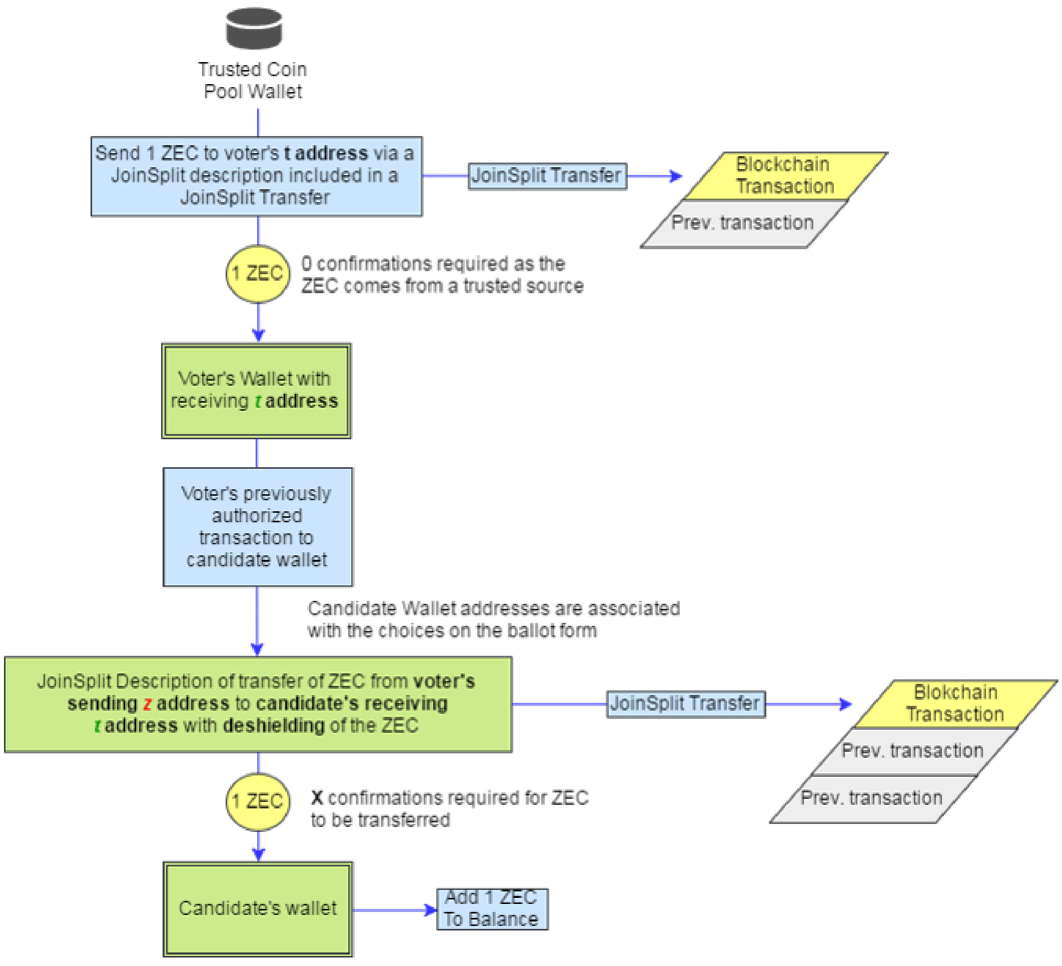
Trusted Coin Pool Wallet
Send 1 ZEC to voter's t address via a JoinSplit description included in a JoinSplit Transfer
JoinSplit Transfer
Blockchain Transaction
Prev. transaction
1 ZEC
0 confirmations required as the ZEC comes from a trusted source
Voter's Wallet with receiving t address
Voter's previously authorized transaction to candidate wallet
Candidate Wallet addresses are associated with the choices on the ballot form
JoinSplit Description of transfer of ZEC from voter's sending z address to candidate's receiving t address with deshielding of the ZEC
JoinSplit Transfer
Blokchain Transaction
Prev. transaction
Prev. transaction
1 ZEC
X confirmations required for ZEC to be transferred
Candidate's wallet
Add 1 ZEC To Balance

Figure 5.5: Steps taken for issuing the vote in **transparent** candidate transaction

diagrams.

5.7 Vote Count/Audit

The final stage of the voting protocol is the vote count and the audit which takes place after the count in order to review the election process and ensure that the integrity of the election has not been compromised. The candidate wallets send all of 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

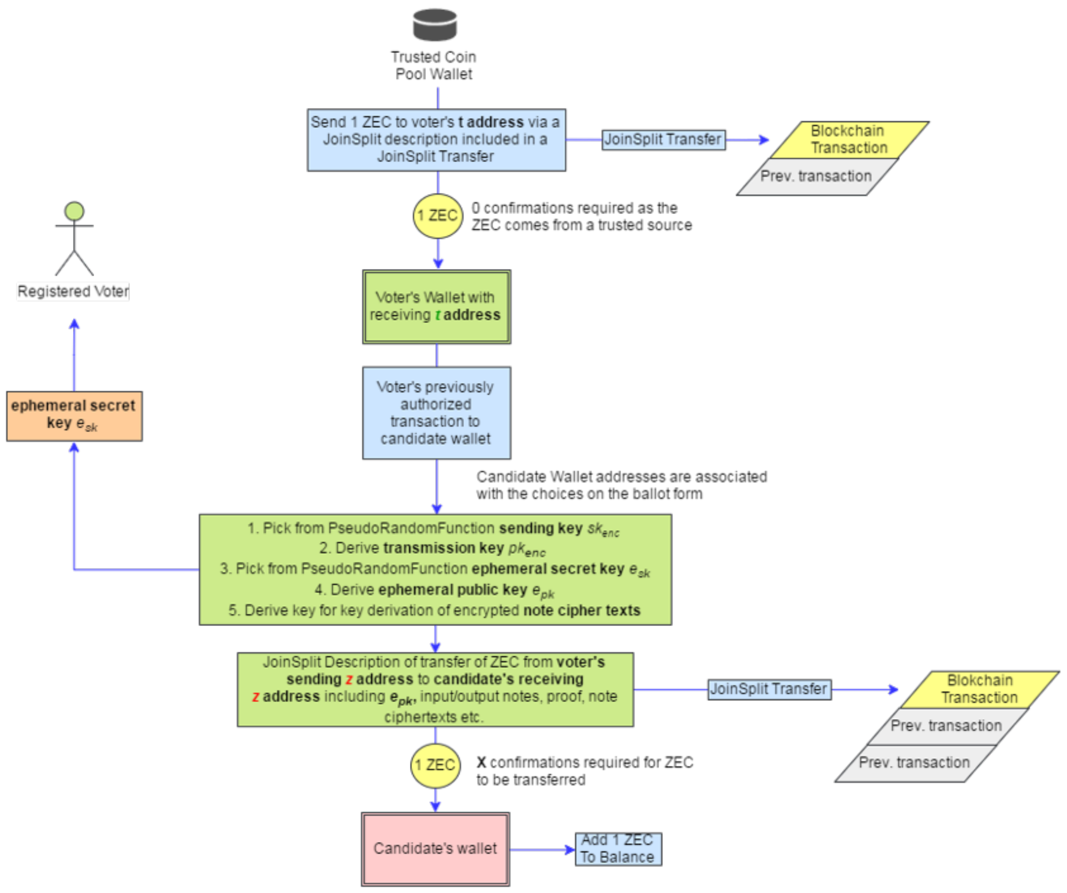
Trusted Coin Pool Wallet
Send 1 ZEC to voter's t address via a JoinSplit description included in a JoinSplit Transfer
JoinSplit Transfer
Blockchain Transaction
Prev. transaction
1 ZEC
0 confirmations required as the ZEC comes from a trusted source
Registered Voter
Voter's Wallet with receiving t address
ephemeral secret key e_{sk}
Voter's previously authorized transaction to candidate wallet
Candidate Wallet addresses are associated with the choices on the ballot form
1. Pick from PseudoRandomFunction sending key sk_{enc}
2. Derive transmission key pk_{enc}
3. Pick from PseudoRandomFunction ephemeral secret key e_{sk}
4. Derive ephemeral public key e_{pk}
5. Derive key for key derivation of encrypted note cipher texts
JoinSplit Description of transfer of ZEC from voter's sending z address to candidate's receiving z address including e_{pk}, input/output notes, proof, note ciphertexts etc.
JoinSplit Transfer
Blokchain Transaction
Prev. transaction
Prev. transaction
1 ZEC
X confirmations required for ZEC to be transferred
Candidate's wallet
Add 1 ZEC To Balance

Figure 5.6: Steps taken for issuing the vote in **private** candidate transaction

all of the collected ZEC vote tokens into the coin pool. If the system is using the transparent candidate transaction variant, then the balance of the candidate would be open to the public at all times. This fact means that if one of the wallets is dishonest, the public members would identify a corrupt candidate. The same logic applies to the transactions under the same variant. It would be easy to identify if a system was compromised if one of the transactions has sent more or less than exactly 1 ZEC vote token. This becomes more difficult in the private candidate transaction variant, where more trust is required in the system and it is up to the voter to verify their transaction as they are the only person who may view the transaction aside from the candidate.

Regardless of the variant, the candidate wallets send all of 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. Figure 5.7 demonstrates an example with 100 vote tokens enforcing the equation specified earlier. 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.

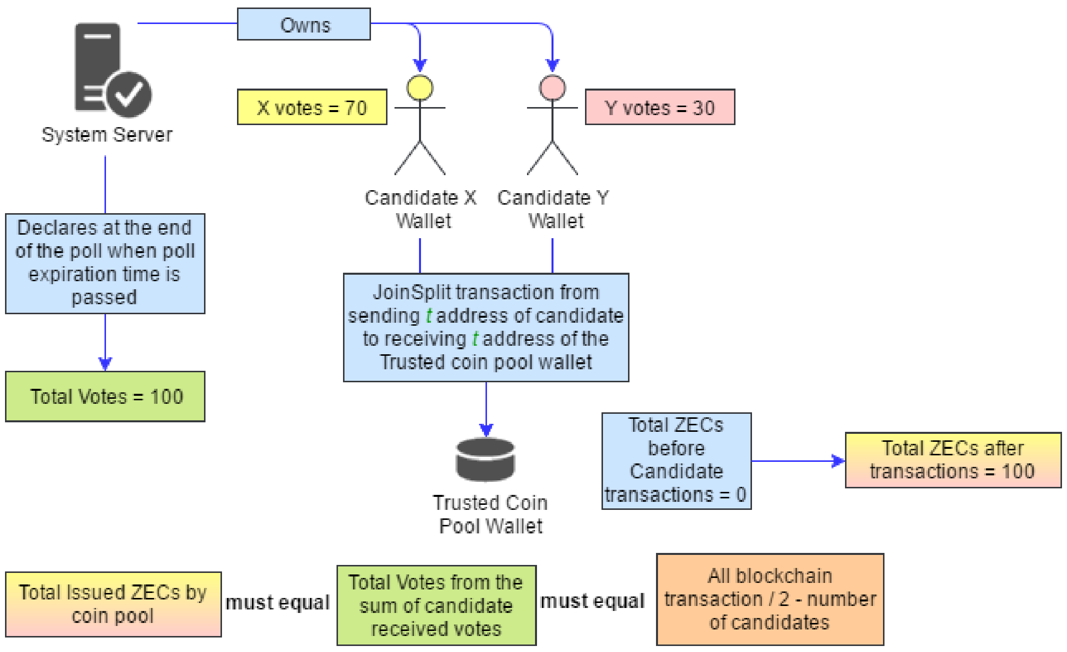
Owns
System Server
X votes = 70
Y votes = 30
Candidate X Wallet
Candidate Y Wallet
Declares at the end of the poll when poll expiration time is passed
JoinSplit transaction from sending t address of candidate to receiving t address of the Trusted coin pool wallet
Total Votes = 100
Trusted Coin Pool Wallet
Total ZECs before Candidate transactions = 0
Total ZECs after transactions = 100
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

Figure 5.7: Counting of the votes and audit rules for the voting protocol

The administrators of the elections decide on the necessary action, if the vote numbers have not balanced. The election may re-run on another date, with the entire process running from the registration step, to audit.

The protocol described by the paper builds on the premises of ZCash, and since the original protocol has not been manipulated in any way, it can be said that the proof of functionality of the voting scheme's transactions can be proven by the ZCash functionality. The protocol relies on some assumptions, and the security considerations of these assumptions and other aspects of the protocol will be reviewed in the next chapter in more detail.

Chapter 6

Security Considerations

Security of the voting protocols is a widely discussed topic. As mentioned previously, many protocols have emerged and found to contain security issues which would break the protocol. In some cases, these issues happened during the election, which means the protocol was compromised during a live election. There are issues that present a significant threat to all voting systems, regardless of the paper or electronic form. This issue is an external influence on the voter outside of the election. No voting system is capable of dealing with a voter who has been influenced by an outside coercer to vote in a particular way. For this reason this issue is out of scope of the current work.

A much more real issue is a compromised voting machine. 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 would be able to 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 [67]. 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.

Issues regarding *double-voting* i.e. using the same granted vote token to vote for multiple candidates is eliminated inherently due to the adaptation of the ZCash platform as basis for the voting protocol. However, since the unique vote link is sent to thee 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 [69]. This gives an incentive for corrupt voters to attempt to hijack the coin upon brief arrival to their wallet. The assumption is that the script is capable of detecting 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 [3]. Another mitigation is to create a specific wallet capable of only working with vote tokens, specifically for election and nothing else. 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. Specialized wallets, may be the solution for this problem.

An undetected coercer would be able to successfully manipulate votes without a notification system in place as even if the voter was able to vote after the vote manipulation, a significantly low number of voters verify their vote as found by [8].

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 of 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 the 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. The forfeit election can be investigated, while displaying the results to authorised people to determine the root cause of the issue. It could be possible that after an investigation, the tracker has been found to be malfunctioning, in which case the system could implement a recounting feature with involvement of authorised human parties. This could potentially be a self-healing mechanism which starts to re-count all the entities involved in an election. For example, if the total number of voters has been altered, the self-repair mechanism could traverse the database to fetch the recorded number of people who have voted successfully, in order to restore the tracker. Alternatively, the election could be rescheduled, which introduces extra cost.

One significant attack on the entire blockchain is called 51% attack [68]. This is considered to be 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 form sending ZEC vote tokens to the candidate addresses. This attack would be difficult to prevent. On one hand it is possible to pick out a number of 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 in order 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.

One of the final considerations would be the above mentioned Sybil attack. Since the assumption of the verified identity has been established, the Sybil attack still poses a problem. A user who wishes create multiple valid identities for voting could do so if they wished. Technically, it would not be considered as a violation of the voting protocol and the transactions would go through to the candidate wallets. Moreover, nobody will be able to distinguish these transactions from valid transactions. A possible mitigation for this attack is to request additional personal information during registration. Information such as PPSN could be requested as this information is issued by the government and additional internal checks of the PPSN can be done prior to registering a user.

Some assumptions of the work give rise to security questions and considerations which have been discussed above. The system's focus is on transparent voting, with a great degree of security to disallow as many coercion attempts as possible, which has been outlined by some of the design decisions of the protocol. The direction of some of the blockchain protocols, such as Ethereum, promote bright future for this protocol's improvements. The next chapter will outline the attempts of blockchain applications, such as Ethereum, to step in the direction, which allows implementation of zk-SNARKs, bringing blockchain voting protocols one step closer to establishing a strong foothold.

Chapter 7

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 [34], has been established early on in the work as a potential candidate to become the platform for the 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 accounts execute specific code which performs a certain function, for example a monthly rent payment to a particular address in Ether. These contracts can be used to implement a voting scheme. The benefits of this would cover some of the security considerations outlined in the previous sections with regards to the major concern for the above protocol and would make render some of the assumptions obsolete. The particular benefit is the creation of custom currency, which may not have any value outside the context of the contract within which it has been created. This means that even if a corrupt voter decided to steal a vote token made in Ethereum, they would not be able to use it anywhere else except of the voting contract context.

Since blockchain is decentralized, there is more incentive to make the voting protocol more decentralized as well, while not making a particular government a trusted central party issuing and monitoring the vote tokens and the votes. Complete decentralization of the protocol would be the desired outcome, however the lack of anonymity of Ethereum platform currently poses as an issue to the fundamental concepts of voting. Steady advancements in development of Ethereum platform bring the creation of this protocol closer. Buterin, the co-founder of Ethereum, has given some updates on the direction of the Ethereum and its pending upgrade named "Metropolis" [70]. Metropolis is an update which aims to improve Ethereum, and although thee exact details of the update are unconfirmed, the future Ethereum aims to make use of zk-SNARKs, the same technology as used in ZCash. 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 [71].

These steps have been done in the form of small projects like ZCash over Ethereum or "ZoE" [73] which has been an extension of "baby ZoE" [72]. Baby ZoE was the first attempt to merge the zk-SNARKs with Ethereum, through pre-compiled contracts, which can be seen as experimental contracts not added to the EVM. Baby ZoE attempted to add elliptic curve pairing operations to the pre-compiled contract in order to prove that a sender knows a commitment on a Merkle tree of the Ethereum contract with some authentication in order for other users not to withdraw from the contract spending the original receiver's deposit. The current issue with zk-SNARKs in Ethereum is that there the SNARKs take a long time to generate. The difference between baby ZoE and the updated ZoE is the fact that baby ZoE attempted to implement the older, simpler version of ZCash, Zerocoin [40], which has been outlined previously. The updated ZoE protocol, used the setup phase from ZCash on a pre-compiled contract in order to create shielded custom tokens, which can be used in voting, and have the same properties as shielded ZCash transactions [73].

The other potential for improvement of the proposed protocol is for the registration side. In accordance to the security considerations mentioned above, it is possible to request more private information during the registration stage, or create two-factor authentication process with user's mobile device of choice. However, with the Ethereum protocol, perhaps, this centralized aspect of the protocol can be removed completely by implementation of a contract. If a government body was to issue a voting contract with the candidates defined within, any person who wishes to vote would be able to vote, and the rules of the election will be enforced by how the contract is created.

Having outlined the developments which are being made on these protocols, indicates a good possibility that blockchain voting system will become the system which would fill the identified gap in the electronic voting domain. The voting protocol and ZCash payment technology have both been described in great detail and it is now possible to conclude the work.

Chapter 8

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.

Bibliography

[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] D. Hopwood, S. Bowe, T. Hornby, and N. Wilcox, "Zcash Protocol Specification", `https://github.com/zcash/zips/blob/master/protocol/protocol.pdf`, January 2017.

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

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] S. F. Shahandashti and F. Hao, DRE-ip: A Verifiable E-Voting Scheme without Tallying Authorities, 2016.

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

[17] A. Parsovs, Homomorphic Tallying for the Estonian Internet Voting System, pp. 111, 2016.

[18] D. L. Chaum, Untraceable electronic mail, return addresses, and digital pseudonyms, Commun. ACM, vol. 24, no. 2, pp. 8490, 1981.

[19] P. Golle, M. Jakobsson, A. Juels, and P. Syverson, Universal Re-encryption for Mixnets, Organization, vol. 2964 of LN, pp. 163178, 2004.

[20] 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.

[21] T. Elgamal, A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms, IEEE Trans. Inf. Theory, vol. 31, no. 4, pp. 469472, 1985.

[22] M. Kitsing, Internet voting in Estonia, ACM Int. Conf. Proceeding Ser., vol. 2014Novem, pp. 137144, 2014.

[23] D. Springall et al., Security Analysis of the Estonian Internet Voting System, Proc. 21st ACM Conf. Comput. Commun. Secur., no. May, p. 12, 2014.

[24] Owasp, OWASP Top 10 - 2013, OWASP Top 10, p. 22, 2013.

[25] "Top 10 2017-Top 10 - OWASP", Owasp.org, 2017. `https://www.owasp.org/index.php/Top_10_2017-Top_10`

[26] N. Hastings, R. Peralta, S. Popoveniuc, and A. Regenscheid, Security Considerations for Remote Electronic UOCAVA Voting, p. 71, 2011.

[27] J. A. Halderman and V. Teague, The New South Wales iVote system: Security failures and verification flaws in a live online election, Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 9269, no. March 2015, pp. 3553, 2015.

[28] S. Nakamoto, Bitcoin: A Peer-to-Peer Electronic Cash System, Www.Bitcoin.Org, p. 9, 2008.

[29] C. Pacia, "Merkle Tree", 2013. `https://chrispacia.files.wordpress.com/2013/09/merkle-tree.jpg`.

[30] "How do bitcoin transactions work?", CoinDesk, 2015. `http://www.coindesk.com/information/how-do-bitcoin-transactions-work/`.

[31] A. M. Antonopoulos, Mastering Bitcoin. Sebastopol: OReilly Media, 2014.

[32] "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`

[33] P. Glass, "How secure is blockchain?", Taylorwessing.com, 2016. https://www.taylorwessing.com/download/article-how-secure-is-block-chain.html

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

[35] Estonian National Electoral Committee. Statistics about Internet Voting in Estonia, 2015. http://vvk.ee/voting-methods-in-estonia/engindex/statistics/.

[36] Bitcoin Block Explorer Blockchain, Blockchain.info, 2017. https://blockchain.info/.

[37] The Online Voting Platform of The Future - Follow My Vote, Follow My Vote, 2017. https://followmyvote.com/.

[38] V. Buterin, "Devcon2: Ethereum in 25 Minutes", YouTube, 2017. https://www.youtube.com/watch?v=66SaEDzlmP4.

[39] G. Wood, "Ethereum Blockchain Mechanism (Proof of Work)", I.stack.imgur.com, 2017. https://i.stack.imgur.com/afWDt.jpg.

[40] E. Ben-Sasson et al., Zerocash: Decentralized anonymous payments from bitcoin, Proc. - IEEE Symp. Secur. Priv., pp. 459474, 2014.

[41] "Some things you need to know - Bitcoin", Bitcoin.org, 2017. https://bitcoin.org/en/you-need-to-know.

[42] P. Peterson, "Zcash - Transaction Linkability", Z.cash, 2017. https://z.cash/blog/transaction-linkability.html

[43] P. Peterson, "Anatomy of a Zcash Transaction", z.cash, 2016. https://z.cash/blog/anatomy-of-zcash.html.

[44] D. Hopwood, "Zcash 1.0 "Sprout" Guide", GitHub, 2016. https://github.com/zcash/zcash/wiki/1.0-User-Guide.

[45] "Developer Guide - Bitcoin", Bitcoin.org, 2017. https://bitcoin.org/en/developer-guide#p2sh-scripts.

[46] "Pubkey Script, ScriptPubKey - Bitcoin Glossary", Bitcoin.org, 2017. https://bitcoin.org/en/glossary/pubkey-script.

[47] D. J. Bernstein, Curve25519: New Diffie-Hellman speed records, Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 3958 LNCS, pp. 207228, 2006.

[48] C. Lundkvist, "Introduction to zkSNARKs with Examples", ConsenSys Media, 2017. https://media.consensys.net/introduction-to-zksnarks-with-examples-3283b554fc3b.

[49] E. Tromer, "Zerocash: improving Bitcoin using SNARKs", YouTube, 2014.https://www.youtube.com/watch?v=S6qOj9ap6RM.

[50] libsnark: C++ library for zkSNARK proofs (Zcash fork). https://github.com/zcash/libsnark

[51] M. G. D. Johnson, "Computers and Intractability." p. 175, 1979.

[52] C. Reitwiener, zkSNARKs in a Nutshell, pp. 115.

[53] S. Goldwasser, The knowledge complexity of interactive proof systems, vol. 38, no. 1, pp. 108128, 1989.

[54] 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.

[55] E. Ben-sasson, A. Chiesa, and E. Tromer, Succinct Non-Interactive Arguments for a von Neumann Architecture, USENIX Secur., pp. 135, 2013.

[56] M. Virza, "SNARKs for C: Verifying Program Executions Succinctly an ...", YouTube, 2017. https://www.youtube.com/watch?v=nS3smRAfUd8&index=3&list=PL8hxk6hqV_vk3gT79ydz-0fmUf3ZXNLCa.

[57] M. Green, ”Zero Knowledge Proofs: An illustrated primer”, A Few Thoughts on Cryptographic Engineering, 2017. https://blog.cryptographyengineering.com/2014/11/27/zero-knowledge-proofs-illustrated-primer/.

[58] M. Green, ”Zero Knowledge Proofs: An illustrated primer, Part 2”, A Few Thoughts on Cryptographic Engineering, 2017. https://blog.cryptographyengineering.com/2017/01/21/zero-knowledge-proofs-an-illustrated-primer-part-2/.

[59] V. Buterin, ”Quadratic Arithmetic Programs: from Zero to Hero”, Medium, 2017. https://medium.com/@VitalikButerin/quadratic-arithmetic-programs-from-zero-to-hero-f6d558cea649#.c1fkogp41.

[60] V. Buterin, ”Exploring Elliptic Curve Pairings”, Medium, 2017. https://medium.com/@VitalikButerin/exploring-elliptic-curve-pairings-c73c1864e627#.rmua7kxbl.

[61] V. Buterin, ”Zk-SNARKs: Under the Hood”, Medium, 2017. https://medium.com/@VitalikButerin/zk-snarks-under-the-hood-b33151a013f6#.nz0i9s3an.

[62] N. Sullivan, ”A (Relatively Easy To Understand) Primer on Elliptic Curve Cryptography”, Cloudflare Blog, 2017. https://blog.cloudflare.com/a-relatively-easy-to-understand-primer-on-elliptic-curve-cryptography/.

[63] ”Elliptic Curves - The MOV Attack”, Crypto.stanford.edu, 2017. https://crypto.stanford.edu/pbc/notes/elliptic/movattack.html.

[64] A. Howard, ”What is a Sybil Attack?”, Top Ten Reviews, 2011. http://www.toptenreviews.com/software/security/best-antivirus-software/what-is-a-sybil-attack-.html.

[65] L. Hazlewood, ”What is an X.509 Certificate?”, Stormpath User Identity API, 2011. https://stormpath.com/blog/what-x509-certificate.

[66] W. Simpson, "RFC 1994 - PPP Challenge Handshake Authentication Protocol (CHAP)", Tools.ietf.org, 1996. https://tools.ietf.org/html/rfc1994.

[67] "Checksum", Ant.apache.org. https://ant.apache.org/manual/Tasks/checksum.html.

[68] "51% Attack", Learncryptography.com. https://learncryptography.com/cryptocurrency/51-attack.

[69] "Zcash Price Chart (ZEC/EUR)", CoinGecko, 2017. https://www.coingecko.com/en/price_charts/zcash/eur.

[70] J. Manning, "Ethereums Vitalik Buterin Gives Keynote On Metropolis", ETHNews.com, 2017. https://www.ethnews.com/ethereums-vitalik-buterin-gives-keynote-on-metropolis.

[71] "ethereum/EIPs", GitHub, 2017. https://github.com/ethereum/EIPs.

[72] S. Bowe, "Zcash - zkSNARKs in Ethereum", Z.cash, 2016. https://z.cash/blog/zksnarks-in-ethereum.html.

[73] 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/.