Student Number: 170197098
Christopher Walker

Reducing Trust Requirements for e-Voting Schemes

Supervisor: Professor Kostas Markantonakis

Submitted as part of the requirements for the award of the
MSc in Information Security at
Royal Holloway, University of London.

ANTI-PLAGIARISM DECLARATION

I declare that this dissertation is all my own work, and that I have acknowledged all quotations from the published or unpublished works of other people.

I declare that I have also read the statement on plagiarism in the General Regulations for Awards at Graduate and Masters Levels for the MSc in Information Security and in accordance with it I submit this project report as my own work.

Christopher Walker
April 30, 2021

Acknowledgments

This project would not have been possible without the incredible support of my wife, Rosie, who has been a star during the worst possible time to ask so much of her.

Contents

List of Tables

List of Figures

List of Abbreviations

DLP Discrete Logarithm Problem.

DLT Distributed Ledger Technology.

DNS Domain Name System.

DoS Denial of Service.

EVM Ethereum Virtual Machine.

LRS Linkable Ring Signature.

LWE Learning With Errors.

NAT Network Address Translation.

PII Personally Identifiable Information.

PKI Public Key Infrastructure.

PoA Proof of Authority.

PoS Proof of Stake.

PoW Proof of Work.

TLS Transport Layer Security.

WASM WebAssembly.

ZKP Zero-Knowledge Proof.

Executive Summary

Voting is the core of democracy however the electorate must trust the authorities running any election will act with integrity and that the electorate will not be coerced into voting against their own will. The amount of trust that must be placed in the hands of the authorities and the feasibility of coercion depends on the methods used to run the election including how the voting is performed and how the ballots are tallied. With the advancement of technology we now have electronic voting schemes which variously prioritize security, verifiability or practicality.

This project looks at the level of trust these schemes require of both the electorate and the authorities and identifies areas where the level of trust can be reduced or eliminated by the considered use of technology. It then proposes an e-voting scheme which uses the discussed ideas to minimize the amount of trust required from users and provides a software implementation to demonstrate the feasibility of such a system.

1. Introduction

This project investigates three fields of technology and their combined intersection. It considers the role of trust in software systems, the promise of secure remote voting and the blossoming field of Distributed Ledger Technology (DLT) — a.k.a. Blockchain Technology.

Trust in technology is becoming more relevant in a world where security breaches and data leaks make it into the general media. The public is more aware than ever that their use of technology comes with the potential for their information to be used in ways they did not intend. They make a decision —consciously or not— to trust a service provider in order to acquire the benefits of a service. For some, those who give it serious consideration, the decision will be made by weighing the benefit of using the service, the sensitivity of the data they give to the service against the trust they have in both the technology powering the service and the reputation of the service provider. Others may only consider the benefits and give the trust freely. The author believes that the former group, while in the minority now, will only grow as technology continues to advance and increasingly permeate our lives.

Voting is the core of the democratic process. The promise of remote voting is the ease in which the electorate can perform this process, combined with the reduced cost for the organizing authorities of providing such a service remotely. Remote voting, provided it is at least as trustworthy as in person voting, appears to provide many benefits with few downsides. The potential benefits of convenience, cost, reliability and non-repudiation seem for far outweigh the potential downsides of requiring voters have access to the technology to participate — a downside that exists in an analogous form for in-person voting in that voters are required to be able to visit a voting booth in a timely manner. Despite these potential benefits, very few large scale remote voting systems are in use.

DLT is a technology which at its core provides a distributed, immutable ledger that mutually distrusting parties can agree on. The design for mutual distrust counter-intuitively creates a highly trustworthy system, as zero trust is required to use it. The surge of popularity in the last few years of cryptocurrencies —such as Bitcoin, Ethereum and Zcash— backed by DLT has lead to much real-word testing of the reliability and practicality of various forms of the technology. Although the long-term future of cryptocurrencies is still to be seen, the technology itself has more to offer than just cryptocurrencies. Of note is its ability to create audit trails in which a single entity may add to the chain and many others —possibly even the general public— can validate the data to ensure consistency. As the data are shared as created, all entities with read access will be able to detect attempts at mutation. DLT however suffers its own popularity with the "*when all you have is a hammer, everything looks like a nail*" issue of a solution in search of a problem. "*Blockchain*" has become a technology buzzword and many projects are emerging that use DLT in places where it is simply unnecessary and an over-complication. We will need a convincing argument that the use of DLT is bringing benefit before choosing to use it.

Taking these three fields together, can we utilize DLT to help provide a remote voting system

with low trust requirements? This report will investigate the process of remote voting, looking at various systems proposed and implemented to see how they address the issues of trust. It will describe DLT and its various usages and properties to see how it might be best used in the context of remote voting systems. Indeed, we should be able to show that using DLT in such a system actively provides benefit. With this information, we will be in a position to propose and implement a remote voting system which uses the best of this knowledge to provide a voting platform with minimal trust requirements.

1.1. Motivation

The author is a software engineer and very interested in both cryptography and the role of trust in software systems. Trust in the democratic process and hence election systems is also of great importance, as evidenced by the amount of contestation around the results of the 2020 US presidential elections. There has been a large amount of prior research into voting systems [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13] and properties of voting systems yet very few national level deployments exist. Could a lack of trust in the technology be the reason for the low adoption? Or are there more fundamental technical or societal reasons that these systems are not in place. It seems that a voting system with *universal verifiability* (see section 2.1) would make any contestation to the outcome provably valid or invalid.

The trust aspect seems especially relevant. How many entities must we trust? How many must be honest before the system falls down? What are the probabilities of an adversary being able to manipulate the voters, the votes or the tally? Although the average voter may not ask themselves these exact questions, they will likely make an intuition-based judgement on how much they trust the voting process. Whether this lack of trust in the system will lead them to protest, abstain, or simply accept and continue is another question, but not one technology can answer. However, the author would speculate that for an electronic voting system to be acceptable for an electorate its trustworthiness must be sufficiently high and reducing trust requirements works towards that goal.

Therefore, this report is an opportunity to investigate the trust requirements in various electronic voting systems and how those requirements might be minimized.

1.2. Objectives

This report aims to:

1. Describe the security properties desirable in e-voting schemes.
2. Analyse existing e-voting schemes, their security properties and identify areas that require significant trust from the user.
3. Introduce the technologies and cryptographic systems we can leverage to design a better system.

4. Propose and provide a software implementation of a scheme that is minimizes trust requirements while maximizing coverage of the desirable security properties.
5. Analyse the protocol and the success of the implementation against the objectives set out for it.
6. Form a conclusion whether we **can** reach a minimum level of trust requirement while maintaining the desirable security properties.

1.3. Structure of the Report and Methodology

The compilation of this report was done in a number of stages. The first was a literature review to ascertain the status quo. As mentioned, there is a lot of work published regarding voting systems and the first step was to find relevant material to get a picture of the current state of the art. The literature review also surfaced the common vocabulary for discussing the security properties of voting and e-voting systems and highlighted the common threads in the different technologies used.

The second stage was to take all the data from the literature and evaluate the different systems with respect to the security properties and also with respect to the trust requirements they imposed on their users. A variety of schemes were chosen to provide a broad view of the different approaches to the same problems. This stage led to an understanding of the core of electronic voting systems, how the security properties relate to trust and how the technologies involved, despite being different in methodology and efficacy, all operate towards similar goals.

The third stage was to build a picture for how a system with minimal trust requirements might be constructed. This stage introduced the cryptographic primitives that would be useful and the exploration blockchain technology as a tool for operating with transparency to reduce trust requirements. Once the system had shape, a software implementation of the system was created to demonstrate its feasibility. The process of creating a software implementation in turn provided feedback into the design and the final proposal and implementation where created concurrently.

The fourth stage was to analyse the success of the protocol, given the aims of providing desired security properties whilst minimizing trust requirements in a practical solution that could scale up to state-level elections. The conclusion then covers the main objectives of the report in light of the analysis, proposal, and success of implementation.

The report is intended to be read in order, and as such introduces concepts as needed. It covers the principles behind the voting systems, moves on to the technologies, introduces DLT and finally moves on to the proposal of a voting system.

- **Introduction:** This section (chapter 1) introduces the concepts, methodology and motivation for the report.
- **Voting Systems:** The systems, their security properties and trust requirements are

covered by chapter 2 and chapter 3.

- **Distributed Ledger Technology:** Blockchain technology and its applications in voting systems is covered by chapter 4.
- **Proposal:** A proposal for a voting system, its implementation and the analysis are covered by chapter 5, chapter 6 and chapter 7.
- **Conclusion:** The concise summary of findings is performed in chapter 8.

2. Requirements of Voting Systems

2.1. Security Requirements

There are a number of desirable properties for a voting system to have, some of which could be considered required for a voting system given the assumption that the system aspires to be fair and honest. There have been many papers outlining the security requirements specific to voting and e-voting [14, 15, 16, 17], and they tend to agree on the majority. First we make some definitions of the primary entities involved:

Entity	Definition
Candidate	An entity that is entering into the election as a potential choice for the voters. All possible candidates will be listed for the voter to choose from when voting.
Voter	An entity that is eligible to vote in the election. Proof of eligibility is not defined here but must be possible within the framework of an election.
Authority	The entity holding the election. The authority will define the parameters of the election, for example which candidates there will be, the timeframe voting will take place in and the voting scheme that will be used.
Registrar	An entity that will be held responsible for authenticating the voters, and hence determining eligibility. This entity may be the same as the authority, or it may be separate.
Trustee	An entity or one of a group of entities responsible for counting the votes and computing the final tally. This may be the original Authority or an external entity.
Bulletin Board	Refers to a publicly available record of the information regarding the election. Depending on the voting scheme, this is not always made available and may have differing amounts of information. This need not be electronic, or can be even if the voting scheme is otherwise non-electronic.
Auditor	Any entity that validates the election data is correct by inspecting the bulletin board, votes or tallies. In some schemes this is a separate, designated entity or group of entities and in others it can be performed by anyone.
Adversary	Any entity that is attempting to subvert the election in any way. This could be to attempt to have their preferred candidate elected or to stop the election from taking place at all.

Table 2.1.: Table of Definitions of Entities in Voting Systems

The following properties are basic requirements for any voting system to have practical use:

Correctness The outcome of the election should be a reflection of the votes cast. That is, if the protocol of the voting system is followed and the entities involved are honest to the degree required by the system then the outcome will be a true reflection of the votes cast.

Secrecy After casting a vote, no one can learn the choice of the voter.

Eligibility Only eligible voters should be able to cast a vote. The conditions for eligibility can vary, but these conditions must be met by all voters and the system must enforce this.

Fairness No information about the results —full or partial tallies— can be made available before the close of the election. That is, no party can gain information about the tally before all votes have been cast.

Of these properties, correctness and eligibility appear to me as self-evidently required for a voting system to work at all. If the result is not correct, then the system has failed. If ineligible entities are allowed to cast votes alongside the eligible voters then the result will be skewed.

The other two might not have such a fundamental importance, but I would still argue that they are required for a voting system to be practical. If information about the tally is revealed during the voting process it may give more power to the later voters, who would then be able to vote *tactically* to sway a vote. As an example we consider an election of three candidates: Alice, Bob and Charlie. Say that after 90% of votes have been cast, we know that Alice and Bob both have 40% of the vote and Charlie the remaining 20%. In this situation supporters of Charlie may decide that there is no way to win, and instead of voting for their preferred candidate, they instead vote for the *lesser of two evils*: attempting to prevent their *least* preferred candidate from winning. Therefore, the election would be skewed compared to one where all voters have *the same information* at the point of casting a vote.

Secrecy is an important requirement as it allows voters to cast votes as they wish without concern for the allegiance implied by the vote cast. Voters would be able to vote a certain way to let others know, for prestige, rather than because they think the candidate is the correct choice. Conversely, voters would be able to vote a certain way for fear of being persecuted for their real opinion. That is, secrecy is not such much a requirement for the election process as it is a requirement for voters to feel comfortable enough to use the system as intended — this ties in with the correctness property, that the vote reflects the voters' will. Without voters willing to use a system, or that system unable to represent their true will, it is of no practical value.

The next set of properties are desirable, and indeed should be present in any secure voting system.

Robustness The scheme should be resilient against an adversary attempting to influence the outcome. The influence could be arbitrary, simply attempting to create errors in the

process or to create a denial of service stopping the election from happening at all. The influence could be specific, attempting to sway the results in a pre-determined way. The scheme must also be resilient to errors occurring for any other reason; malignant or benign.

Receipt-Freeness After casting a vote, the voter should not be able to prove their choices to any entity. This prevents the voter from being able to prove their vote to a 3rd party in a vote buying or selling scheme, or in the case of coercion. This property is similar to *secrecy* but stronger, as not even the voter can prove how they voted.

Coercion-Resistance The scheme should prevent an adversary from being able to manipulate a voter into either abstention, or to coerce the voter into voting a certain way. The adversary should not be able to impersonate the user and place a vote on the voter's behalf.

Individual Verifiability Any voter may verify that their vote has be counted in the tally.

Universal Verifiability Anyone may verify that the final outcome is calculated correctly from the votes. The scheme provides enough public evidence to prove the truth of the outcome.

Auditability The scheme provides evidence of its behaviour during and after an election. That is the scheme provides enough public evidence throughout the course of the election that an auditor may discern any dishonest actions.

Robustness in a voting protocol can prevent continual postponement via general Denial of Service (DoS) or even election bias in the case of targetted DoS. An adversary attempting to prevent the election taking place could, if the process was not robust, could sabotage the process to the extent that the result was invalidated. This could serve the candidate that would benefit most from the effective outcome of the election should the process fail — e.g. the current elected candidate may remain in power in the event of the election process not completing successfully. Similarly, if the will of a group of voters is supposed to be statistically biased towards a specific candidate (perhaps as a result of opinion polls) then an adversary might perform a targetted DoS attack on those voters attempting to prevent their votes from being cast. The outcome if the attack was successful would be that the group's preferred candidate would be affected most by the lost votes.

Receipt-Freeness has crossover with the secrecy property, except in this case we want the voter to be *unable* to prove who they voted for, rather than for an outside observer to be able to learn who they voted for. This means that the temptation to vote a certain way to be able to show people you voted that way, in an attempt to curry favour or the like, is removed as there is no way to show how you voted. It also helps towards coercion-resistance as the voter is unable to prove to a third party the way they voted. Similarly, vote buying schemes become more difficult as it becomes impossible to prove to the buyer the way in which you voted. Coercion-resistance takes this a step further to mandate that a voter is not able to be coerced to vote in a specific way. That implies receipt-freeness but the converse is not true as we will see when we get to e-voting specifically (in section 3.1). The reason that coercion-resistance is

desirable is that without it, the correctness property, if taken literally as the result representing the voters' will, is at risk.

Verifiability is desirable at it will increase voter confidence in the system. Individual verifiability allows voters to be confident that their vote has been cast as they intended and that their vote has been included in the final tally. Without this assurance, the election and its result is subject to the electorate trusting all the entities involved in the process. Once their vote is cast, they would have no way to check anything and must simply hope that their vote has been cast and counted correctly. Universal verifiability is a stronger property that ensures there is enough public evidence for anyone (including ineligible entities) to check all the votes and that they were counted correctly. This level of transparency again aids in increase confidence in the correct outcome of the election. Auditability is the final step along this path, ensuring that the level of transparency before, during and after the election is such that the outcome cannot be contested and the honesty of the participant entities confirmed.

Auditability is a stronger property than universal verifiability, and requires that the system can not only prove that is has behaved correctly, but also that is has not behaved dis-honestly. For example universal verifiability only requires that all the published votes are tallied correctly but does not enforce that we can detect if votes are missing or have been added post facto, such as in the case of ballot-stuffing. Auditability requires we can tell the difference between a vote cast correctly and one added to fill a gap from an absentee voter. It also requires that we can tell if any tampering occurs such as removing cast votes from the published results.

2.2. Trust Requirements

The Oxford English Dictionary defines trust as follows:

> **noun**: Firm belief in the reliability, truth, or ability of someone or something.

This is a difficult thing to quantify. Romano [18] defines a scale for measuring trust, but it is very subjective. We want to remove as much subjectivity from the question of trust and move it into something definite that we can answer — is it *easier* to trust X or Y?

Trust can be considered on a binary scale either you trust something or you don't.

Trust can be considered on a continuous scale you can trust something a little or a lot.

The author believes the two concepts are related, in that the amount of continuous trust you have in something must exceed the trust requirements it imposes on you for you to consider that you have the binary trust in it.

Often the thing you have continuous trust in —to whatever extent— will only be converted into the binary trust at the point of deciding to rely on that thing. At this point the value of the thing and the impact of the trust being subverted come into play to make the final decision.

With respect to voting systems trust is the belief that the system will work fairly and correctly and that it will be resistant to attempts to subvert it. Whether we trust this is the case or not for any given election is subjective and personal, however knowledge that the system is designed in a way to resist subversion attempts will greatly reduce the amount of trust we must extend to the system in order to maintain that belief.

Let us take a simple example based on a well known trust exercise. You stand on a chair and there are N people behind you. You fall back trusting that they will catch you and let us say that if any single person reaches to catch you then you will be safe. Clearly, if you know the people behind you, you will be better placed to make a judgement on each one's individual likelihood of letting you fall. Assuming you have no prior knowledge of the individuals behind you cannot make such a judgement. We do know N and the greater the value of N —the number of people behind you— the more likely it is that one of them will act in good faith. We can make our own judgement on the probability of a single person being a bad actor and call it P_b then the probability of all actors being bad —our failure condition— will be $(P_b)^N$. For any value of $P_b < 1$ then $(P_b)^N$ becomes small very quickly as N rises.

As we increase the N value of the system of this exercise, we lower the trust requirements. We could make a reasonable assumption that in most cases the individuals will likely not have motivation to subvert the exercise —people generally have enough empathy and would not want to see anyone fall— therefore we would estimate our P_b to be low and so even with only a few people we could easily exceed the trust requirements placed on us and fall back confidently.

It seems likely that this concept can extrapolate in most situations. That is to say, where trust is placed in a single entity, spreading that trust amongst a group and only requiring a subset to be trustworthy is likely to lower the overall trust requirements.

In the context of a voting scheme there will always be some requirements for trust, the biggest being that the Registrar confirms eligibility and therefore is able to deny eligibility as well. Furthermore, the desirable properties from section 2.1 add little to the election process except to enhance the public confidence that the election process is correct and honest. This implies an increase in trust that the system will behave itself, therefore lowering trust requirements for participating.

3. E-Voting

Electronic Voting by definition covers any electronically assisted form of voting. This could be electronically counted paper votes, or electronic voting machines at a voting booth. However, we will be considering another option, of fully electronic voting where the systems are accessible over the Internet — similar to the general usage of the term *email*.

3.1. Specific Requirements for E-Voting

The main difference between in-person voting and remote voting from the point of view of security is the lack of a secure voting booth. Such a booth is often listed as a requirement of in-person voting systems, as an assumption to allow enforcement of the eligibility and secrecy properties. That is a secure voting booth is one where the voter is authenticated on the way in and then their privacy whilst inside casting their vote is guaranteed. While a remote voting system does not preclude the use of such a booth, such a system loses a lot of it's perceived benefits by mandating the use of one. Indeed, it may not be immediately obvious why such a system would have any benefits over a traditional in-person election.

It could be argued that a seemingly obvious benefit —that people can vote without having to travel to a secure booth— is not really a benefit at all as we lose the security properties guaranteed by such a booth and the trade-off of convenience over security is not worth it. On the other hand remote voting does indeed have some beneficial implications. It would lower the cost of an election as there would be no need to hire secure locations or the people to run them. There would be no need for transportation of the ballots for tallying and no manual tallying process — both lowering costs and reducing the likelihood of human errors in the tallying. Furthermore, the convenience of not having to travel to a booth is also an enabler for those who would otherwise be unable to make it to a booth, potentially increasing the election turnout.

These benefits need to be weighed against the potential security downsides. It is worth noting that there is one insurmountable problem with remote voting which is that there is no possibility to satisfy the coercion-resistance property. This has been addressed in [19] and under their definitions the property we are discussing here is *simulation attack* in which the adversary acquires the means to vote on behalf of the user. No matter how secure the voting scheme is from a technical point of view, allowing voting from an arbitrary location via the Internet means that a coercer could stand over the voter during the casting process, a situation which technically indistinguishable from the voter voting alone. In [19] the solution is based on the assumption that at some point the voter will be able to vote unhindered and that that vote will be the one counted, any coerced votes would not count and additionally that the unhindered vote could not be linked by the adversary to the voter. This would allow the voter to *pretend* to cooperate with the coercer and separately vote legitimately without the coercer's knowledge (and hence no fear of retribution). The scheme proposed imposes an

overhead which is quadratic with respect to the number of voters, which is impractical for large scale elections. Further work has been done to attempt to reduce this overhead to be linear [20], but even so, the overhard is burdensome for large scale national elections and the coercion resistance is still predicated on the fact that the voter will have an opportunity to act unhindered.

Due to these trade-offs we consider how some security properties mentioned in section 2.1 that take on more or less importance in a remote system and some new security properties that can enhance a remote voting system.

The **Secrecy** property in a remote voting system becomes more important as now the cast ballot must be sent over a public channel from the voter to the trustees counting the votes. In some respects this is the similar to a paper ballot where, despite the ballot being placed in the ballot box by the voter, the ballot itself must not reveal the voter's identity or the trustees could match votes to voters. However, given that the vote is created by the voter remotely, there is also the transmission of the vote from the voter to the trustees —analogous to putting a paper vote into a ballot box— which is now performed over the public Internet, rather than within a secure voting booth.

The **Robustness** property in a remote voting system becomes more important as by definition the system will have computer systems at various boundaries which by necessity will be accessible over the public Internet. DoS attacks against parts of the system will become a valid threat against the system continuing to function correctly. The more entities are in play in the system —e.g. the Authority, Registrar, Trustees, Voter, Bulletin Board— the more opportunity for an adversary to disrupt the communication channels between them. This works both ways as with fewer single entities responsible for each function in the system, the more work an adversary will have to disrupt *all* of them. Therefore, remote voting systems must consider single points of failure and the public accessible nature of the system very carefully.

The **Eligibility** property is vital in all voting systems. Great care must be taken in remote voting systems to ensure eligibility, as we must authenticate the voters remotely and the protocol used must be secure. A failure in the authentication protocol would undermine the integrity of the election. This area is a major cause for concern as eligibility and the one-voter-one-vote semantics that make the election function are at risk. The method of authentication in specifications of votings system is mostly unspecified but defined to be secure as an assumption. I believe this to be a reasonable assumption as the exact technology used to provide the authentication may change without affecting the core properties of the voting system. However, all the systems looked at in this report fall into two categories: those that authenticate early with a "registration" phase and those that authenticate *just-in-time* — at the point of casting a vote.

Those that authenticate early predominantly use the registration phase to use the authentication process to create election specific credentials. This has the benefit that the credentials are shorter-lived, as they only apply to a single election, and that the Registrar has no further involvement during the rest of the election process. By no longer participating in the election

after the initial phase, an adversarial Registrar could not create fictional voters to ballot stuff, nor register on behalf of users who did not register themselves to *steal* their vote. This process provides eligibility verifiability, where we (the public) can tell that a voter, and the vote, is eligible.

The **Receipt-Freeness** property is primarily a factor adding to *coercion-resistance* and a protection against vote-buying. With the fact that coercion-resistance is practically impossible for remote voting systems, it might be tempting to write-off receipt-freeness as unimportant. However, we can still counter all forms of vote-buying and coercion that do not involve physical presence —which is expensive and logistically difficult on a large scale— by ensuring receipt-freeness. If there was no receipt-freeness then a voter could use that receipt to prove to a coercer or vote-buyer the way they voted, or make such a proof public. With receipt-freeness this is not possible and without proof, it is unlikely that a coercer or vote-buyer would be willing to believe the voter. Due to this I believe that receipt-freeness is just as important in a remote system as in an in-person system.

Real-Time Universal Verifiability is the property of a system that allows the property of universal verifiability to be conducted in real-time during the election. That is the bulletin board can be monitored in real-time and modifications observed. This would allow an external Auditor to be sure that the data on the bulletin board are not changed in a way that the voting system disallows, and such illegal modifications could be proven to have been made. This property enhances the universal verifiability property with time constraints. No party would be able to, with sufficient ability to break any cryptographic protocols, rewrite the bulletin board data to meet their goals while still validating under the rules of the voting system. They could only attack the system during the election itself, with all data making it onto the bulletin board.

Eternal Secrecy is a property that states that the connection between the voters identity and the vote they cast should never be able to be revealed. This means that any data released on the bulletin board, or during the voting process, should never reveal the link between the voter and their vote even if all the cryptographic protocols used are broken. This means that, for example, a voting system relying on a cryptosystem that can be broken with a quantum computing algorithm —while secure today— would not be secure in a future where such quantum computers exist. In that future, all the protections from the cryptosystem would be void. We need to ensure that the link between voter and vote is not compromised even then. Note that such a break before or during an election would break the *fairness* property, as the broken cryptosystem would reveal the state of the election before all votes were cast. So we will assume that this property is concerned with the future, after the end of the election.

Software Independence is a property that enforces that any undetected changes or errors in the software cannot lead to undetected changes or errors in the result. This property is discussed extensively in [21] and is extremely important from a trust point of view. The concept is that a bug —or indeed an actively created malicious piece of code— in the software for any stage in the election should not be able to produce data that any other correct software will

accept as valid. This is important for trust because it means that even the software running the election cannot violate the principles of the protocol knowingly or unknowingly without the error be noticed.

3.2. Existing Systems

There are many practical and theoretical remote voting systems — more than could be covered here. Therefore, we look at a selection that use a variety of technologies and techniques to achieve their goals, investigate the security properties they have and discuss the trust implication of their structure. See Table 3.1 for a table of comparison of the various systems and the properties they achieve. This is not as binary as it may appear, as many of the properties are satisfied with a given assumption of trust. Therefore, the table shows an indication of the level of trust required to achieve the property, if it can be achieved at all.

3.2.1. Helios

Helios Voting is an open-source software implementation of a voting system designed by Ben Adida [22]. It has progressed significantly since the conference in 2008, dropping the initial mix-net approach in favour of the use of homomorphic encryption. The project has always focussed on correctness and auditability and openly claims that is only suitable for *low-coercion* elections [23]. That is, Helios makes no attempt to provide coercion-resistance.

The current version of the Helios protocol is V3 and this is the version we shall consider. The system is created as a Python program accessible via a web interface using the PostgreSQL database for data storage. The system is shared tenancy, so all users' data are stored in a single, shared database; except of course that being open-source, anyone is free to run their own private server.

The basic flow of an election is that the creator lists candidates and defines the number of candidates that may be voted for. Additionally, a list of eligible voters may be created or the system may allow anyone to vote. A public/secret keypair is generated for the election and the election can have a number of trustees who each receive a portion of a private key enabling the decryption of votes to be a group operation. Voters cast a vote for each candidate where each is either a 0 or 1 and there are only a predetermined number of 1's allowed. The vote is encrypted with the public key, and the cryptosystem used (Exponential ElGamal) is additively homomorphic. In this way votes may be tallied while still encrypted and the final tally decrypted separately by the group of trustees. This is a simplification for brevity, the full protocol spec is available [24]. Notably this misses out the digital signatures and ZKPs used in the protocol.

Helios provides individual and universal verifiability through the publishing of the encrypted votes, along with enough proof to show that voters have voted within the bounds set in the election. The use of ZKPs ensures that the votes conform to the correct structure and the

homomorphic nature allows them to be tallied without decryption. This allows the encrypted vote data to be published in such a way that anyone can verify a specific vote is present and that the tally is correct. The vote data is available throughout the election process, so real-time universal verifiability is possible. The decryption of the tally also uses proofs based on verifiable decryption, so without the key we can still be sure the tally was decrypted correctly providing the full universal verifiability. Note that this does not satisfy our definition of auditability which enforces that we must be able to discern the correct behaviour during and after the election and detect tampering. During the election the cast votes may be listed, but the onus is on the observer to prove that the list has been tampered with, Helios does not provide such. Once the result is published, the individual and universal verifiability properties combined allow any voter to check that their vote was included in that tally. Therefore, it stands to reason that an adversary, even with full access to the server running the election, could not remove votes without chance of detection. However, the Helios protocol does not forbid such an adversary the ability to create simulated votes for the absentee voters before calculating the totals. This implies that a high level of trust is required in the authority running the Helios server.

By default, the Helios elections identify voters with Personally Identifiable Information (PII) — in this case their email address. While this does not break the secrecy property directly, eternal secrecy is broken. Helios does have the ability to only identity voters by an alias which is a unique identifier not related to the individual and unique in each election. Provided the link between the voter and the alias is removed after the tallying, the eventual breaking of the cryptography would still not reveal the identity of the voter. Again, a dishonest server administrator could obtain that information from the server before it was removed, which implies a level of trust in the authority running the server, but not more so than already existed.

The voters are authenticated either by a password which the Helios system knows, or via an external authentication solution such as OpenID Connect. The exact method of authentication is configurable per election. This enforces the eligibility property with the exceptions of dishonest authentication providers —registrars— or dishonest server administrators. The result of the authentication of the user is not propagated into the vote, so with direct access to the database spurious votes could be cast.

The votes in Helios are encrypted by the client, before posting to the server. This means we do not need to trust the server with unencrypted votes. The encryption process itself however requires the client to generate some randomness to use during encryption and subsequently destroy it. This randomness could be used to prove which candidate the vote was for and so breaks the receipt-freeness property.

In summary, Helios' main weakness from a trust perspective is that much of its security is based on the security of the server running the software. This is often an acceptable assumption, given the specific threat model for the election being held.

3.2.2. Belenios

Belenios was heavy influenced by Helios, and the original implementation was indeed a fork of the Helios source code adding a distributed key setup for the trustees [25]. It was subsequently re-implemented from scratch, but the protocol shows many similarities.

A variant, BeleniosRF, enhances the protocol to provide receipt-freeness by use of a re-randomization step. The votes are encrypted and signed using the ElGamal cryptosystem, and a random number is required for each encryption. The knowledge of that number can prove your vote and so provide a receipt. The randomness is generated by the client, and so could be kept to provide that receipt to the voter. In BeleniosRF the encrypted vote and signature can be re-randomized at the voting server and a new valid signature created without the signing key. The voting server does not know the original vote as it is encrypted and the final cast vote is encrypted with randomness that the voter does not know, hence the protocol is receipt-free.

A second variant, BeleniosVS goes further and provides a mechanism by which the vote does not even need to trust their own client. This is achieved by the vote being provided with a voting sheet ahead of time with the encrypted and signed votes already prepared. This sheet would also contain the randomness used for each encryption, so the voter can verify the encrypted values are correct. Combined with the BeleniosRF system for re-randomization, the vote is receipt-free and the user does not have to trust their own device. Given a compromised device used for voting, the device could prevent submission of the vote but the voter would be able to confirm that. However, the device would not be able to create any vote on behalf of the user as the user never provides any secrets to it. This system seems like it would be an excellent fit for an election with a secure voting booth. The booth provides the guaranteed isolation and the use of pre-prepared encryptions of votes means the voter need not trust the devices used for the voting.

We will consider this second variant, BeleniosVS, as it appears to be an evolution of the original. Being based on Helios, we will focus on the differences. BeleniosVS uses distributed key setup for the trustee keys that ensure that no entity ever sees the full decryption key at any point during the process. The key setup is also a threshold based system, meaning that only a subset of M out of the N trustees are required to perform the decryption. This has two effects. Firstly, given that $x < M$ trustees are dishonest, the cast votes will remain secret, irrespective of the security of the server running the software or its database. Secondly, it means that $N - M$ trustees can decide not to participate in the tallying and the result can still be calculated. There is a natural trade-off here between the additional robustness of having $N - M$ large, the additional confidence of secrecy by having M large and the practical aspects of having N large.

3.2.3. Estonian i-Voting: IVXV

Estonia has embraced remote voting and the i-Voting system known as IVXV has been used to run state level parliamentary elections since 2005. The description of the system and a limited amount of the source code used to run it is publicly available online[i]. The system has had much attention since its inception as the primary example of binding governmental elections held remotely, such as [26, 27]. Four reports from the Office for Democratic Institutions and Human Rights (OSCE/ODIHR), most recently in 2019 have observed the election processes and commented on potential security issues. These reports are designed to continually assess the current state of the system and its reaction to previous recommendations. Judging from the statistics regarding the proportion of votes that come from remote voters, the system is widely popular amongst the electorate: in the 2019 election 43.8% of votes were cast remotely according to [26].

The system allows remote votes to be cast for a time period before the physical casts. Voters may cast as many electronic votes as they wish, with a "last-vote-counts" mechanism for differentiating the valid vote. Finally, voters may also vote via the in-person methods with this vote overriding any previous electronic votes.

The system is based on nationally run Public Key Infrastructure (PKI) for authentication of voters — a system which is widely used in Estonia to authenticate citizens and create legally binding signatures. The identity system provides the voter with a private key that can be used to create such signatures. There is also a key generated specifically for the election. There are three distinct entities involved in the process, the Collector, the Processor and the Tallier.

Votes are encrypted with the election-specific key, then the encrypted vote is signed with the voter's personal key and passed to the Collector, who does not have access to the private key for decrypting the votes. Once all votes have been cast, the Collector passes all votes to the Processor who verifies all the signatures and selects the correct single vote from any voter that has cast multiple, ensuring the "last-vote-counts" semantics. The Processor cannot decrypt the votes, so although it can match encrypted votes to voters, it cannot infer the voters' choices. The Processor then strips the signatures from the valid votes and passes them in bulk to the Tallier. As such, the Tallier does not have the ability to match encrypted votes to the voter that cast them, but the Tallier can decrypt the votes and so produce a final tally.

The observations of the ODIHR in [26] stated that the system was improved compared to the time of the previous report, however certain internal data —considered *private* but available to those with privileged access to the internal systems— contained enough information to break secrecy, when a voter had made their *receipt* public. This receipt is intended to be solely prove that a vote cast was counted, satisfying *individual verifiability*, but in this case would allow the entity with internal access the ability to reveal the voter's vote. This class of threat, that of internal users with elevated privileges accessing raw data presumed secret, exists in other remote voting systems and its severity may only be ascertained by the threat model and

[i]Source code at `https://www.valimised.ee/en/internet-voting/documents-about-internet-voting`

risk assessment of the specific election. In the case of the IVXV, these are binding national governmental elections and so the stakes are high.

On the other hand, the role of Collector, Processor and Tallier are provided by independent entities which reduces the chances of collaboration which would also defeat the secrecy of the election. This is only a reduction of chances, rather than a full mitigation as an adversary would simply need to subvert all three entities, rather than just one to be in a position to attack the election. This would objectively be more difficult, but not impossible.

Another area in which the report was recommended change was that the system was not *software independent* stating: "meaning that errors in its components may cause undetected errors in the election results, and it is potentially vulnerable to internal attacks and to allegations of cyber attacks". This last point is an interesting one, and corroborates a claim that the author would make: that the potential for vulnerability, whether exploited or not, leaves room for plausible allegations which can undermine trust.

Despite the criticism and the potential security flaws, the published statistics from the Estonian elections show that many voters can and do overcome the trust requirements for submitting their votes remotely and that the number of voters willing to do so is increasing over time. It is the author's belief that the widespread use of digital identity in Estonian governmental processes which has been happening for many years— has given the citizens enough confidence in the use of this identity for other legally binding commitments that they are willing to extend the same level of trust to the national voting system. As long as the system continues to work effective without major cyberattacks or public failings, the author expect that confidence to rise. However, a major public failing of the system would likely have severe consequences on public trust of the digital identities. This make the continued success of the system balance on a knife edge, with public support predicated on the lack of incident, but incident always possible.

3.2.4. Bitcoin Voting Proposal – Zhao and Chan

This section looks at the system proposed in [28] which creates a voting system on top of the Bitcoin cryptocurrency. The system is limited to a two candidate election, with each voter able to vote for one candidate or the other, without the possibility of abstention. There are further practically issues in this system that make it unsuitable for real-world usage, such as the fact that it requires real bitcoin transactions that are not free. Whilst the proposal itself used 1 Bitcoin as the base pricing unit —which would be an inconceivable amount with the current price of Bitcoin— but even using a smaller base, the real-world costs would be problematic. To core of the proposal is that the voters transfer bitcoin to the winner.

The system describes two possible voting methods. The first behaves in such a way that should the protocol fail and the election not be satisfactorily concluded by a certain time, then the voter can receive the deposited amount back by creating a refund transaction. This first method however requires that each subsequent voter commits a larger amount of bitcoin in

their deposit transaction. The second allows the voter to get back a deposit, but not the total cost of voting. Indeed, in either situation the cost of transactions themselves is not taken into account and every transaction has a non-zero cost in and of itself that goes to the miner of the block. While both these methods incentivize good behaviour of voters by imposing real-world costs for deviating from the protocol, the costs themselves impose a barrier to entry.

Another practicality issue is that the all the voters must contribute to the protocol in the correct stages and in the correct order, or the protocol fails and the refunds can be claimed but no winner decided. This make the process fragile, breaking the *robustness* property.

Therefore, while much of the algorithmic processes within the scheme are sound, there is an extremely high opportunity for any single voter to completely void the entire election by withholding a vote. This places an extremely high level trust required for each participant to give to each other participant.

3.2.5. ZCash Voting Proposal - Tarasov and Tewari

This section looks at the system proposed in [29] which uses the ZCash cryptocurrency as a base for a voting system. ZCash is a cryptocurrency system which supports two address types. The first: a *z-address* is which is anonymous and transactions to or from them cannot be linked back to any other transaction. The second is a *t-address* which is functionally similar to a bitcoin wallet address, and as such the movement of coins to or from these addresses can be publicly discerned.

The proposal uses a registration phase and the Registrar entity also plays the roles of the Election Authority. Once a voter has registered, their data is held in an unspecified datastore by the Registrar/Authority entity. On the *opening* of the vote, the Registrar/Authority sends invitation links by email to the voters. The voter's then provide a *t-address* to which a "vote token" is provided. This must come from some pre-existing source of coins and the state of which voters have been issued tokens must be tracked by the Registrar/Authority entity. This source is called the *ZEC pool*.

Once the voter has their "vote-token" there are two variants described for the voting process. The first variant sees the candidate using a *z-address*. This means that the transaction is anonymous in that no one can introspect the details of the vote. However, it also —by design— means that the vote is now unable to verify that their token is part of the system; they cannot trace their own vote has been counted.

The second variant has the candidate using a *t-address*. In this variant the token balance is discernable to all parties at all times, and the link between token and vote is preserved. While this latter property would allow the voter to verify their vote, it would also allow the authority to match the vote to the voter.

The final tally and audit stage of the process sees the candidates transfer the balance of the tokens they have back to the *ZEC pool*. The number of tokens that came from each candidate are counted and made public. The Registrar/Authority is tasked —and *trusted*

with performing the task of verification that the count of votes balances with all previous data.

The proposal itself contains a section on the security considerations, which reveals some areas of trust that exist within the protocol. The first voting variant, using candidate *z-addresses* have no way of tracking the balance of votes that a candidate has received. This enables the *fairness* property as no indication of the ongoing status of the election is revealed until the end. However, it does mean that we rely on the candidate to transfer the full contents of the address back to the pool for audit. An untrustworthy candidate may decide to wait until the other candidates have transferred their tokens and if they have not won the election —a fact that they can verify— they could underreport their votes with two consequences. Firstly, they could keep the other tokens, which have a non-negligible real-world monetary value and secondly, the election token balances would not add up and the election would be void. This latter situation could mean a re-vote which may be in the untrustworthy candidate's favour, or simply a denial of the result for the otherwise victorious candidate.

In the second voting variant, the underreporting of vote tokens would be noticeable as the candidate wallet is a *t-address* and could be directly linked to the guilty candidate. Whether this would void the election or simply disqualify the candidate is not specified. However, in the second system the *fairness* property has been broken, which in the author's opinion is enough to disqualify this proposal as a valid voting system.

A great deal of the state of the system is kept private by the Registrar/Authority yet required to be used in later stages of the election process. The integrity and authenticity of this state is not considered and there are no details to how it may be stored and validated. This would break the property of software independence, as the vote tokens and transactions may be on the blockchain the auxiliary state is not.

In summary, this proposal despite running on a blockchain requires an extremely high level of trust from its participants. The number of ways an adversarial entity —whether voter, registrar, authority, or candidate— could disrupt the system means that it is not *robust* unless all parties are honest. While that does not prevent the system from functioning, it does make it impractical and imposes high trust requirements from voters who know that they must trust many entities to behave correctly.

3.2.6. Scalable Open Vote Network

This section looks at the system proposed in [11] which itself is an evolution of [2] and uses smart contracts on the Ethereum blockchain as a basis for a voting. The system is limited to a binary vote i.e. the decision between one of two options. The smart contracts used ensure eligibility, the timing of the stages of the election and verify the Zero-Knowledge Proof (ZKP) that the vote is indeed a 0 or 1. The system addresses the issue of cost in the complexity of the smart contracts by making heavy use of Merkle trees over much of the computation, which is actually performed off chain. The computation must be *disputed* if deemed invalid

and the dispute process involves trigger a smart contract which performs the disputed step and will detect the dishonest behaviour or reject the transaction if the computation was actually valid. Note that this step will cost the disputer, as they must pay for the gas to perform the transaction.

To start an election the organizer creates a Merkle tree of the eligible voters which is used later to assess eligibility. The root of the tree will be part of the smart contract. Along with the timing data for the election and a deposit amount, the smart contract is deployed.

Next all voters register a voting keypair and ZKP of knowledge of the secret key. A Merkle proof of membership is constructed (which is not a fixed size, and grows with logarithmically with the number of leaves). These data along with the deposit amount are added to the chain via the smart contract. The contract will accept the voter if the timing is correct, and that the ZKP of knowledge and Merkle proof of membership are both valid.

The encryption technique used for concealing the votes is interesting. It requires using all the public keys of all registered voters, so that when *all* the results are multiplied, the final result p reveals the sum of the votes (it is actually the exponential, but we can brute force $log_g(p)$). At each stage of the multiplication, a leaf on a Merkle tree is created, at the end the final result and the Merkle root and stored in the contract by the organizer. The full content of the Merkle tree must be published somewhere accessible to the potential disputers. The encryption also allows creation of a ZKP to prove correct form of the plaintext, a 0 or 1.

As the result is only a single number, it is the number of *1* votes and the count of voters registered minus the result will give the number of *0* votes.

The disputers can verify the Merkle tree, and on finding an error can trigger a contract that will check the calculation and judge the dispute. If in favour of the disputer, the disputer is rewarded with the value of the deposit. After the dispute, irrespective of the outcome, all honest participants may reclaim their deposit (note a successful disputer ends up ahead one deposit, less the cost of the dispute). The organizer is only able to retrieve their deposit if no successful dispute was made.

We can inspect source code of the smart contract to ensure it will behave legitimately, so all aspects relating to the execution of smart contracts can be considered trustworthy.

The organizer in this protocol has no clear benefit to tallying dishonestly, in all likelihood it will cost them the deposit amount. Even if no-one disputes in the allotted timeframe, the tally can be recomputed at any stage and the calculation show incorrect, likely nullifying any outcome from the election. The organizer is involved with the initial data, and is required to publish the Merkle trees for which the roots are part of the contract state. The organizer could withhold the eligibility Merkle tree from voters they want to prevent from taking part, but later wish to claim *could have*. Denied access to the tree, the voter would not be able to create the Merkle proof of eligibility.

The other tree is the one for the computation of the tally. Restricting access to this would not prevent voters from checking the final outcome as the Merkle root is indeed present. So

there is little to gain here. Therefore, if the organizer has created favourable setup data for the election and published the eligibility Merkle tree data then there is no need to further trust the organizer.

However, a voter may throw election very easily by simply not casting a vote. The encryption and tallying only work if all registered voters actually cast a vote. A single abstention would render the election void. This implies that we must trust the other voters not to destroy the election.

3.2.7. Post-Quantum LWE Based Proposal - Chillotti

This section looks at the system proposed in [13] which uses a *post-quantum* homomorphic encryption system using a form of lattice encryption know as Learning With Errors (LWE). The cryptography in this system is significantly more complex than classical cryptography, but has parallels in function to that which we have seen in many of the other systems. The author cannot claim a full understanding of the mathematics behind the lattice-based system in use, but can understand the principles enough to see the functional similarities to classical cryptosystems.

The proposal is similar in structure to Belenios. There is a distributed key-generation phase to bootstrap a public key cryptosystem capable of providing a signature scheme, a non-malleable and a fully homomorphic encryption scheme and alternative solutions to ZKPs to provide verifiable decryption and proof of vote validity (that the vote correctly encrypts the appropriate number of 0s and 1s).

The entities involved are an Authority, a Bulletin Board and a set of Trustees. The Authority acts as Registrar and maintains a public list of legitimate voters. The Bulletin Board also acts as an Auditor in that it will only publish votes after ensuring validity and can reject them otherwise. The Bulletin Board has a keypair which is used during the election to perform the validation, and the secret key may be revealed after the election without compromising any security properties. Finally, the Trustees help in the initial cryptosystem setup and then during the tally phase to perform verifiable decryption of the homomorphic sum of the votes. The major concession, outlined in the proposal, is that the system is reliant on an honest Bulletin Board. It does not address the case where the board denies legitimate votes.

The scheme runs in a familiar order, with parameter setup, followed by voter registration, voting and then tallying. In fact despite the differences in cryptography used in this scheme versus Belenios, each phase, action and security property has an analogous post-quantum alternative here. Note that this is the plain version of Belenios, not BeleniosRF nor BeleniosVS. As such, the only property that is enhanced in this scheme is eternal secrecy, This is enhanced under the advent of practical quantum computing because the *difficult problem* posed by ElGamal is broken in this situation, but LWE currently has no known solution quantum or otherwise.

Therefore, while the trust requirements are objectively lower due to an enhancement in

eternal secrecy, the author would argue that the reduction would be is minimal and that this proposal has very similar trust requirements to Belenios.

3.2.8. Trustless - Gajek

This section looks at the system proposed in [12] which focuses on producing a zero-trust protocol.

This system does indeed have extremely low trust requirements. It uses DLT and smart contracts to perform most of the validation, meaning all the data required for verification is in on-chain and therefore public. In the system, voters have an individual identity and there is a shared anonymous identity also used for casting votes anonymously. In order to cast a vote using the anonymous identity, the voter must obtain authorization in the form of a signature from a *Signing Authority*.

The protocol has the familiar setup, registration, voting and tally phases. During the setup phase the organizer chooses a group of entities to be Signing Authorities and a group of entities to be Trustees (which the system calls Key Authorities). The organizer also defines here the set of voters and the choices they may vote for. These data are committed to the chain.

The Trustees now add their public key material to the chain, which checks to ensure the organizer allowed them to. Once all Trustees have added their keys, the voting may begin. In order to cast a vote, the voter first creates the data themselves and encrypts it will all the public keys registered by the Trustees. The vote must be signed by a threshold of the Signing Authorities. Each Signing Authority checks that the voter's true identity is eligible to vote and has not previously requested a signature, and provides a partial blind signature over the vote. Note that none of the Signing Authorities learn anything about the vote during this process, but at the end to voter is able to construct a valid signature over their vote. They now use the *anonymous* identity to submit the encrypted and signed vote to the chain, which validates the signature before accepting the data.

When the voting phase is complete, the tallying begins with all Trustees generate partial decryption keys for the election and publish them on the blockchain along with a signature to prove eligibility to do so. Once all keys are published the decryption of the votes is done on-chain, however it can be performed by anyone as the data are all public. Note that despite the fact that all votes are decrypted, they are also all anonymous and so no link between voter and vote is revealed, preserving the secrecy property.

We note that a dishonest set of Signing Authorities could enable double voting, or allow ineligible voters to vote. Either of these actions could be detected as the number of votes cast should be less than or equal to the number of eligible voters, make the outcome both more and less desirable depending on the intent of the dishonest group. If they wish to sway the election, then this is a risky method as it could invalidate the election. If the intent is to void the election then this may be an excellent avenue of attack. The protocol itself does not count that the number of cast votes is sane, so that must be performed by an external auditor.

A group of dishonest Trustees could —without detection— decrypt the all currently cast votes which would break the fairness property.

The voting system is designed to be run on the HyperLedger Fabric (`https://www.hyperledger.org/projects/fabric`) blockchain. This system is a permissioned blockchain network, where the public will not have direct access to the ledger data however the voting protocol requires the all organizations involved in the network make their ledger data available which satisfies the auditability property.

We also observe that receipt-freeness and therefore coercion-resistance is not provided by this system under any trust assumptions. The vote encryption process is probabilistic, so produces the "toxic waste" of the randomness used. This waste can be used to prove that the encrypted value matches a given plaintext without the decryption key, providing a receipt. Even if such waste were not available, the encrypted vote is available and the voter has knowledge of it. Once the election is complete the decryption key is released and therefore the known encrypted vote can be decrypted.

In this system we do not need to trust the organizer; we only need to agree with the initial parameters. We only need to trust that a sufficient subset of the Signing Authorities ate honest, and that a sufficient subset of the Trustees are honest. There are no single points of trust in the system, however there are practical downsides. For example, the entire list of eligible voters must be provided in the initial parameters for the election, which in a large-scale election could range into the 100s of millions[ii].

3.2.9. Trust Comparison

The following table list the discussed voting systems along with the entities which require trust. It also shows the level of trust required with a few broad terms.

- "Full" indicates that the entity must be trusted, as it could act in a way to jeopardize the election, breaking its otherwise held security properties.
- "Partial" indicates that the trust in this entity is not so strong as for the full requirement. This could be due to trust reducing factors such as distribution of trust amongst a group, or due to the influence or role of another entity in the protocol.
- "Minimal" indicates low trust requirements, breach would be possible but unlikely or with little benefit to the perpetrator.
- "None" indicates that there is no trust requirement here. This means that either the work this entity does it public and verifiable, or that it is possible for the user to perform it directly themselves.
- "N/A" indicates that this particular entity is not required within the protocol, or its purpose is subsumed by another entity.

[ii] According to `https://www.presidency.ucsb.edu/statistics/data/voter-turnout-in-presidential-elections` the US had over 200 million eligible voters in 2016

The table shows quite clearly that some systems provide very low trust requirements, however it is often the case that practical or security properties have been traded off against the low trust, making the systems sub-optimal for real world use.

Voting System	Authority	Registrar	Trustee	Auditor	Bulletin Board	Candidate
Helios	Full	Full	Partial	None	Full	None
BeleniosVS	Full	Full	Partial	None	Full	None
i-Voting	Full	Partial	Partial	Partial	Full	None
Zhao-Bitcoin	None	N/A	None	None	None	None
Tarasov-ZCash	Full	Full	N/A	N/A	N/A	Partial
Scalable Open Vote	Minimal	N/A	Partial	None	None	Partial
Chilloti-LWE	Full	Full	Partial	None	Full	None
Trustless-Gajek	None	Partial	Minimal	None	Minimal	None

Table 3.1.: Table of Voting Systems and Trust Requirements

4. Distributed Ledger Technology (a.k.a Blockchains)

4.1. Introduction to Blockchains and the Problems they can solve

Blockchains are fundamentally an immutable ledger: they allow data to be recorded in an append-only manner and such that the order and content of the ledger cannot be disputed in the future. Combined with a common, shared set of rules —known as *consensus*— for how to add data to the chain and validate those data, a blockchain allows for mutually untrusting parties to agree on the content and state of a shared immutable ledger.

This forms the basis of cryptocurrencies where the transactions in the currency are entries into this ledger. The transactions themselves use other cryptographic functions to ensure their own rules —which all parties can agree on— and the chain is what allows a public system where no node is required to trust any other node, yet all nodes can agree on the content of the ledger.

I believe it was this lack of trust requirements that really allowed DLT to become popular. The first well known use of a blockchain was in the cryptocurrency Bitcoin which has gone from interesting experiment to a global buzzword. One of the key factors in its success was the lack of central authority. There was no way to regulate trade in the coins and no restrictions on who could send or receive them. This would not have been possible without the trust-less distributed immutable ledger provided by the blockchain backbone of Bitcoin.

Blockchains can also be used without the public consensus rules in what is known as a Permissioned Blockchain. In this case the data are public, but only authorized entities may write to it. The key feature here is that the public data are tamper-evident. Such systems can be used as public audit logs with a high trust from the public as entries cannot be retroactively altered.

Whilst cryptocurrencies were the first major application of blockchain technology there are other problems —such as reputation systems [30, 31], smart grid energy usage reporting [32], private data sharing [33] and business process management [34]— that can benefit from the decentralized nature and immutability. The two properties together imply a form of censorship resistance that centralized systems can never have — that is, there is no authority that can intercede to perform the censorship, every node in the network has equal rites. For any system attempting to provide a global lookup table that currently uses centralized authorities, this is a desirable property. As such DLT has been used to create decentralized alternatives to the most common Domain Name System (DNS) used on the Internet [35, 36, 37] and the central PKI root(s) of trust used in many Operating Systems for certificate validation in Transport Layer Security (TLS) [38, 39, 40].

Blockchains have a major disadvantage to centralized ledgers in that the speed of trans-

actions is often prohibitively slow for many applications. Using the Ethereum blockchain network as an example, at the time of writing the largest transaction volume in a single day was 1,406,016 on Thursday, September 17 2020 [i] which equates to 16.2 transactions per second compared to the Visa payment network which claims greater than 56,000 messages per second [ii].

4.2. Structure of a Blockchain

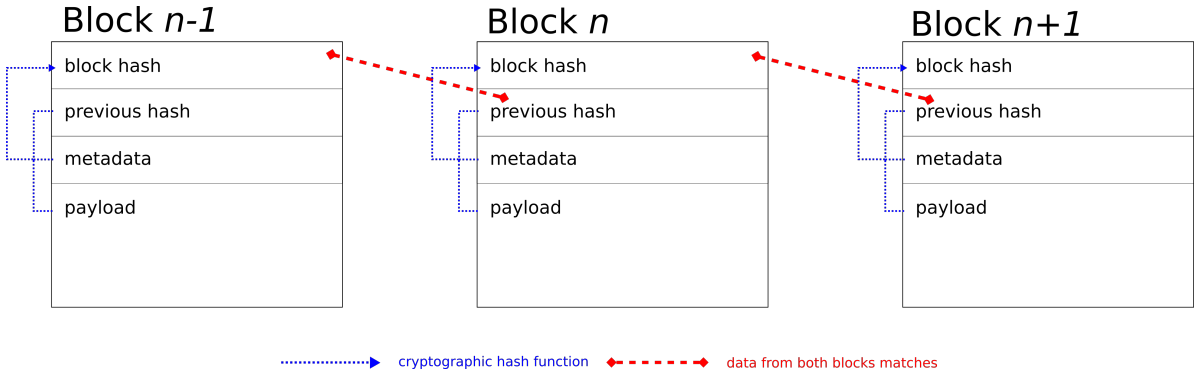

Figure 4.1.: Basic Blockchain Structure

Blockchains are at their core built from two simple principles. These two principles provide a foundation of the technology and number of useful properties. They are:

1. Every block is linked to the previous block in a deterministic manner.
2. Every honest node in the network must be able to agree on the same chain.

Let us look at the first principle. The important word is *deterministic*. Consider Figure 4.1; each block has its identifier as the output of a cryptographic hash function over data from the payload of the block as well as the hash of the previous block. This way we cannot retrospectively change the contents of a block, as its hash would no longer validate, nor would the hash of any subsequent blocks. To rewrite a block in the chain, we would need to rewrite the entire thing from that block onwards, so that all the new hashes would be correct.

The second principle ensures that while this may not be impossible in principle there must be a method of picking the *true* chain that all honest nodes in the network can agree on — despite the possible presence of adversarial nodes. This is a property is known as consensus. Firstly, nodes take the view that the longest valid chain is the true chain. The second part varies among chains (discussed later in subsection 4.2.2) but the original and most common method is a Proof of Work (PoW) scheme. This prevents re-writing the chain by enforcing that to create a valid block, the *miner* must perform some non-negligible amount of computation, requiring time and energy. In order to rewrite a block, the cost in time and energy would need to be paid again for that block and for all subsequent blocks. The consensus algorithm makes

[i] Ethereum transaction rate data from `https://etherscan.io`
[ii] Visa transaction rates as advertised at `https://visanet.co.uk`

it increasingly difficult to change the data in a block the deeper it gets into the chain and at some point the data becomes almost certainly immutable.

4.2.1. Blocks and Hashes

Blockchains use cryptographic hash functions for a number of their properties. Here is a quick overview of the desirable security properties of a cryptographic hash function H [41]:

1. **Preimage Resistance** Given a hash output h, it should be hard to find a message m such that $H(m) = h$.
2. **Second Preimage Resistance** Given a message m_1 and the output $H(m_1) = h$ it should be hard to find a message m_2 such that $H(m_2) = H(m_1) \equiv h$.
3. **Collision Resistance** It should be hard to find two messages m_1 and m_2 such that $H(m_1) = H(m_2)$.

A fourth useful property which implies and is implied by second preimage resistance is **Pseudo-Randomness**. This property is satisfied if the output from the function is hard to distinguish from a random oracle. That is, there should be no discernable relation between the input and output and similar inputs should not produce similar outputs. Without this property, second preimage resistance could not be satisfied as we would gain some knowledge of m_2 due to its relationship with m_1.

Note that second preimage resistance is implied by collision resistance, however it is subtly different as we are not attempting to find *any* collision, but a specific collision for a known message. The recent SHAttered [42] attack on the SHA1 hash function was used to break collision resistance, not second preimage resistance. Blockchains use cryptographic hashes to link blocks together precisely because of second preimage resistance, which implies that we cannot feasibly forge a block while keeping the hash identical — which would be the only way to insert a block without having to rewrite the entire following chain.

We see in Figure 4.1 the data we want to ensure are immutable —the block payload, the previous block's hash and metadata— are included in the block's own hash. We can validate the block has not been tampered with by recomputing the hash and checking it matches the hash we expect. This also means we cannot change any of the block data without also changing the hash and if we change the hash then the next block will have the incorrect previous hash and the chain is broken.

There is a single exception for the first block in the chain —known as a *genesis* block— which by definition cannot have a previous block to reference. Most blockchain implementations prefer to keep the genesis block as similar to any other block as possible and simply use a zero value for the previous block hash.

While this concept is simple, in practice it only defines a valid chain. It is quite possible to create a group of blocks that form a directed acyclic graph rather than a single linear chain. In this situation how do we decide which leaf is the "true" chain. This uncertainty leads directly

to the requirement for the consensus covered next.

4.2.2. Consensus

Consensus in a distributed system is the property that the honest nodes in the system will agree about the data in that system, even in the presence of adversarial nodes. This is the reason for the second principle mentioned in section 4.1 and the reader may have already drawn a parallel to the Byzantine General's Problem [43] which describes an equivalent problem.

A blockchain peer-to-peer network consists of a group of nodes in a gossip network. Each node searches for other nodes and then *gossips* with them. They swap knowledge of other nodes in the network to expand the number of peers they can attempt to connect to. They also swap knowledge of their view of the blockchain. If one node claims to know a longer chain it will tell its peers, and send the blocks to them. The receiving node must then validate that the blocks are valid. The node will use the consensus algorithm to check this. If the blocks are invalid, then the node will block the sender as a dishonest node, ensuring that eventually dishonest nodes are ejected from the network. If the blocks are valid, then the node can be sure that the creator of the block has indeed put in the cost to create it.

As more than one new block could be sent to a node at the same time, we need a second part to the consensus algorithm to ensure that the chain doesn't split into many chains. This is actually very simple and all blockchains I could find treat the longest valid chain as the *true* chain. If a peer node can produce a longer valid chain, then the network will treat that as the true chain. This happens with a reasonable probability and most blockchains do not consider blocks confirmed unless they have a number (six in the case of Bitcoin) of blocks on ahead of them. After the blocks have been added it is considered infeasible for an adversary to be able to produce a longer valid chain with that block altered.

The consensus method most often used in blockchains is known as a PoW and involves performing a computational task that takes a non-trivial amount of processing power to perform but a trivial amount to check it was performed correctly. The value invested is the computation work performed.

An early example of a PoW system is HashCash [44] initially devised as a spam prevention mechanism. By adding a PoW requirement to email it would no longer be cheaper for spammers to send mail at volume. The main principle behind HashCash is to give a proof that some work has been done, specifically for a given commitment. The commitment is some data we wish to show we knew before we did the work. The *work* involved is the computation time required to perform the hash function. A single hash requires a very small amount of work, so we want the prover to perform a very large number of hashes, however the verifier should not have to do much work at all. This may seem impossible as the pseudo-randomness property means that to be sure the prover has performed the hashes, the verifier would have to perform the same number of hashes.

The solution to this is to lower our requirement of proof from complete certainty to *within a*

given probability. Now we can take advantage of the pseudo-randomness of the hash function. As every input produces a random hash, each bit of the hash will be 0 or 1 with probability 1/2. So producing a hash with the first bit 0 has probability 1/2. A hash with the first 2 bits 0 has probability $1/2^2$. For a hash with the first n bits 0 the probability is $1/2^n$ and will on average take 2^n attempts to find. Now we can create our commitment c with a shared method and append some randomness r and calculate the hash $H(c||r) = h$. If the first n bits of the hash are 0, we consider the work done, if not we create a new random value r and try again. This will take on average 2^n attempts, but the verifier given r can create the commitment c and confirm that $H(c||r)$ starts with n 0-bits with a single hash. Pseudo-code for this algorithm is shown in Figure 4.2.

```
// right-shift by this amount will leave only the bits
// we want to check are 0
let shift = numberOfBitsInHashOutput - nBitsShouldBeZero

function hashcash_prove (commitment) {
    let r;
    do {
        // pick a random value for r
        // until it verifies
        r = random()
    } while (!hashcash_verify(commitment, r))

    return r
}

function hashcash_verify (commitment, r) {
    // concatenate and apply hash function
    let h = hash(commitment || r)
    // after shift the result should be zero
    return (h >> shift == 0)
}
```

Figure 4.2.: Pseudo-code for the HashCash algorithm

For a blockchain we can use a similar function using all the data that makes the block identifier hash as the commitment and trying values for the proof until a satisfactory value is found. What makes the value satisfactory will vary, and is determined by the shared rules of the system. In many implementations the *difficulty* is variable to allow the chain to react to keep the rate of addition of blocks roughly constant.

As the blockchain needs new blocks added, performing the work required for PoW usually results in a reward for the miner, incentivizing miners to continue to add blocks. This rewards good behaviour in the system, but bad actors are not punished.

The second method of consensus is known as Proof of Stake (PoS) and instead of performing work to "prove" the block has been mined correctly, instead the mining is delegated to a committee of stakeholders. This concept only works in a cryptocurrency where the coin itself is assumed to have value. The idea is that holders of the currency —that is, those with a *stake* in it and motivation to act in it's best interest— are chosen to mine blocks. The method of choosing varies among algorithms, but it is usually a group of coin holders where the coin is oldest, ties broken by "lowest hash" over the transactions which can be effectively considered a random, but deterministic, choice. Those stakeholders must then digitally sign

the block data to *vouch* for it. Only the correct stakeholder can sign the requisite data and the algorithm means all can determine who should sign the data, all parties can confirm the block's legitimacy. The incentive for the stakeholders to take part in the process is also varied, but can be a share in a block reward and the transaction fees in the block. PoS systems have more power to punish bad behaviour, as stakeholders that do not follow the rules can be penalized, either directly by taking their *stake*, or indirectly by reducing the chances of being chosen to mint new blocks in the future.

Algorithms based on PoS have the advantage that they do not excessively waste energy performing the *work* in PoW. There have been attacks proposed on PoS algorithms [45] and none have yet seen widespread real-world adoption at this time. Some blockchains use [46] or have proposed to switch to [47] a hybrid approach —the Ethereum blockchain being the most prominent— requiring a PoW scheme to valid some blocks (known as *checkpoints*) at certain pre-determined points in the chain.

Although consensus is mostly used to cover the core algorithm for ascertaining that a block has been mined *correctly*, the author would argue that the rules inherent in the chain for exactly how the data are laid out and what constitutes valid block payloads should be considered part of the consensus, as blocks will be rejected if they don't match these core rules just as if the main PoW or PoS fails to verify. This concept can be extended to cover a third consensus type known as Proof of Authority (PoA) which creates a "permissioned" blockchain. In this consensus mode the proof that an entity may add a block is governed by ownership of a key that allows the block to be added. For example, a digital signature may be created over the block header which only the owner of a secret key could produce. Provided the swarm agrees on the public key to verify this signature, the chain can grow but only a single entity can add blocks. The real difficulty in such a system is to ensure that all nodes agree on the validity of the key(s) to add new blocks. An adversary could claim their key is the correct one and swiftly rewrite the chain, producing a longer chain that the "real" one. One of the benefits of a blockchain is its inherent immutability, which could be broken in this situation. Therefore, a PoA consensus system must ensure the authority can be identified correctly. This lends itself to a system where the initial authority definition could be predetermined by data on a secondary, popular blockchain and then the PoA chain, is a new side-chain rooted by the data on the primary one. An adversary could create a new block on the primary chain and a new side-chain, but the side-chain would be identified by a different block in the primary chain and clients would notice the mismatch.

While these consensus mechanisms exist in isolation, as mentioned for We can layer these consensus mechanisms and create a PoA based chain but with a PoW mechanism to prevent modification of previous blocks even by the rightful authority. Such a consensus mechanism is useful for the remote voting use case as these are exactly the requirements we want from a blockchain in that scenario.

4.3. Transactions, Smart Contracts and other chains

Blockchains can have arbitrary data in the block payloads, each chain defines the rules for what is allowed and the significance of the data. Cryptocurrencies based on blockchains typically allow for transactions moving currency around the system. These transactions may be directly from one user to another, or they may have some more complex logic embedded. This logic is known as a *smart contract* and the functionality available in such contracts is defined by the rules of the blockchain. Some chains have a Turing-complete instruction set for the smart contracts —such as Ethereum via the Ethereum Virtual Machine (EVM)— and others a more restricted instruction set —such as Bitcoin.

Unless the blockchain has only a single transaction model, there will be some form of smart contract involved with every transaction. In Bitcoin (and most cryptocurrencies) the transactions can be split into inputs and outputs, where the funds are sourced from the inputs and moved to the outputs. These transactions also form a chain where outputs of one transaction must be used as the input for another transaction. The nodes keep an *unspent transaction* log to guard against double-spending. Other currencies, such as Zcash (`https://z.cash`), keep a *spent coin* log for the same purpose. While both prevent double-spending, a spent coin log necessarily grows as the chain lengthens, but an unspent transaction log size only depends on the number of unspent transactions at any given time. Thus, there may be some scaling issues with a currency using a spent coin log which do not exist with an unspent transaction log. Some cryptocurrencies have rules for the creation of new currency without a previous transaction/coin as a method of increasing the supply of currency available. The Bitcoin network rewards block miners with an amount of bitcoin in an input-less transaction known as a *coinbase* transaction for each new block mined.

In order to spend funds from a previous transaction output the smart contract comes into play. The logic in the contract will determine what needs to be done to use the funds in a new transaction. A simple contract type would be simply that the spender must prove access to the private key matching a given public key in the transaction. A more complex contract might stipulate that M of N keys must be proved in order to release the funds. Both these types of contract can be created on the Bitcoin network. The Bitcoin network has hard limits on the complexity of the contracts, but no cost associated with execution. On the other hand, contracts of theoretically unlimited complexity may be created on the Ethereum network, but the contracts require *gas* to run. The amount of gas is calculated by the EVM while executing the code and there is a hard limit on the maximum gas that can be consumed in the execution before it is aborted. The gas also has a price (in Ethereum) which must be paid whether the execution completes before the limit is reached or not, so there are practicality issues and costs involved with high complexity contracts.

In the Namecoin (`https://namecoin.org`) blockchain while fully functioning as a cryptocurrency —it started as a fork of the original Bitcoin code— also has a second type of output for transactions which allows *purchasing* a key-value pair publicized on the chain which then lasts a fixed period of time before it must be renewed. The main idea behind the chain was to provide

a censor-resistant platform for registering DNS records without a central authority. The cost in the currency would deter squatting or mass registering domains and the immutability and decentralized nature would make external censorship or seizure impossible.

These transactions give a blockchain value, either by simply allowing it to function as a cryptocurrency or as a platform for trust-less computing by the execution of arbitrary smart contracts. There are some blockchains that do not aim for this public utility and instead aim for the immutability for audit purposes. By making it infeasible to alter historic records, the blockchain allows entities to publicize the chain for external audit, even if the rules do not allow external entities to write to the chain. Known as a *permissioned* blockchain where participants have different permission levels, the blockchain can still provide value with no cryptocurrency involved.

4.4. Trust in DLT and their use in e-Voting

Trust is important in decentralized systems. With no authority to control access to the system, it is important to understand that nodes must be able to operate *without* trusting each other. All nodes in the system are considered untrusted, but the consensus algorithm will ensure that the rules are followed. This does mean that all honest nodes need to be adhering to the same rules, but the dishonest parties cannot break them or the honest nodes will notice and ignore them.

This does not mean that a blockchain network is implicitly secure in the presence of adversarial nodes. There are known attacks on blockchain networks and their consensus mechanisms [48, 49, 50, 51, 52, 45]. For example any peer to peer network is susceptible to an eclipse attack [53] in which the adversary gains control over which peers a node is able to connect to. As the targetted node can only connect to adversarial nodes, who can all lie about the state of the network, they can keep the target behind the main chain or trick it into accepting a shorter chain of their own devising. Similarly, a Sybil attack [54] is when the adversary floods the network with adversarial nodes to the point that they wield influence over the network.

Blockchains that use Proof of Work as a consensus mechanism are susceptible to a 51% attack although [55] proposes a possible solution. This attack is realized when an attacker gains more than 50% of the mining capacity in the blockchain network. This gives the attacker a level of control over the network as they have the highest chance of being able to mine new blocks. In a cryptocurrency ,the control extends to being able to block transactions at will or ensure specific transactions get mined. This centralization of power would likely cause the value of the coin to drop and would therefore possibly be self-defeating, but could be an effective method of destroying trust in a cryptocurrency.

The Eclipse and Sybil attacks are generic to any peer-to-peer network. In terms of trust, we as participants in a blockchain network must trust that the network is sufficiently robust against these attacks and that there exists a sufficient number of honest nodes. In a cryptocurrency, the block miners are financially incentivized to behave correctly which amplifies

the network effect. The more popular blockchain networks see the most miners and users and therefore become the most trustworthy.

How might we encourage good behaviour in a voting system without a reward mechanism system? The simple answer is that we do not need to as our requirements from the chain are quite different from that of a cryptocurrency. The requirements we have are that of a public audit log, very similar to a concept of a permissioned blockchain, where only certain entities can write to the chain.

Blockchains have been used in voting systems as we have seen in section 3.2. Open Vote Network [2] is a system running on a smart contract within the Ethereum blockchain. The protocol there is enforced by the smart contract, and indeed all computation is performed there also. This gave the protocol scalability issues, limiting it to just 40 voters. Subsequently, Scalable Open Vote Network [11] addresses the scalability issue by perform most computation off-chain and publishing evidence to the smart contract, which will then allow voters to contest the computation as they see fit. These two systems highlight the high cost of computation on a blockchain in a smart contract.

The Bitcoin Voting [28] system works with clever use of the Bitcoin transaction system and its smart contract system. While the smart contracts available in Bitcoin are not Turing-complete and in fact are extremely limited, they are still flexible enough to cover the needs of this voting system. However, in this system again, the cost of voting is non-trivial.

Both these previous paragraphs show systems that use blockchain-backed cryptocurrencies as a basis for their protocol. In my opinion the real financial cost of using such a system is prohibitive of it being viable as a national-level voting scheme. However, cryptocurrencies and smart contracts are not the only way to incorporate DLT into a voting system. The system proposed in [4] uses the blockchain purely a public, immutable ledger acting as the Bulletin Board aspect of a general voting system. The cryptographic techniques used in the data stored in the blocks ensures the privacy and correctness properties required by the system while keeping all data in the open.

5. Proposal for an Improved Protocol: Astris

This chapter looks at a proposal for new e-voting protocol. Given the existing protocols and their individual trade-offs, the author believes there is scope for a new proposal providing lower trust requirements with equivalent security properties. The focus of the new protocol will be minimizing the trust requirements of the user. It will achieve this by distributing single points of trust as far as practical and ensuring real-time universal verifiability.

> "There are only two hard things in Computer Science: cache invalidation and naming things."
>
> — Phil Karlton[i]

Fortunately our protocol description does not touch on cache invalidation. However, the protocol bears a distinct resemblance to the Helios v4 protocol. This is due to the fact that many constructs within Helios are perfectly suited for application in this protocol. There are also significant differences in both the way elections are set up, voters registered and how the public bulletin board is constructed. Therefore, our proposal is an evolution and *Astris* —one of the daughters of Helios in Greek mythology— seemed a fitting name.

5.1. Objectives and Non-Objectives

Astris aims to lower the trust requirements in comparison to other voting systems while maintaining the key security properties. This is achieved through a combination of processes working together.

Firstly, the system is designed for all working to be done in the open or to be provably correct via public data. This is the primary function of using DLT in the system: to provide a platform that is publicly verifiable and immutable. After data is on the chain, it cannot be rewritten without nodes on the network noticing.

Secondly, the system is designed to reduce the possible single points of trust to distributed ones. No single entity will be able to decrypt the data on the chain. The registrar has a large amount of power over the system, being the one making the decisions about who is eligible or not. This is an unavoidable focus of power, but we mitigate it by ensuring that the registrar may only wield this power during the voter registration phase and not during the voting phase.

The ability to decrypt the final tally is gifted to the trustees, and only when working in concert through threshold cryptography. This means we have a bound on the number of trustees that must collude before any data is revealed. There is a trade-off here between robustness and security in the choice of our threshold. If we require that all trustees be honest, and we require them all to decrypt the final tally, this will be most secure and only one need be honest to prevent any secret data being decrypted. However, it now only takes

[i] This quote was attributed to Phil Karlton by Tim Bray, first appearing on the Internet in 2005 in the post: `https://www.tbray.org/ongoing/When/200x/2005/12/23/UPI`.

a single dishonest party to break the system completely making the final result unknowable, as the withholding of a single decryption will reveal no information about the result. Same outcome would happen if any trustee lost their secret key before the vote decryption phase. On the other hand, if we lower the threshold too much, then fewer trustees need to be dishonest and collude before all the private votes can be decrypted.

There is no correct answer to this trade-off except to consider the circumstances of the election and the required security vs robustness. For example a national level election requires high security, but also must be robust. The electorate will not be happy if the election system fails at the end and has to be restarted, yet neither will they be happy if their votes are revealed. The way to increase both security and robustness is to increase the number of trustees, selecting them carefully to maximize the likelihood that they will not collude — that is, where there is incentive for them to act correctly, and no shared motivation to collude with other parties. Once the number of trustees is high enough, we can choose a threshold where the probability of lost keys causing failure and the probability of the required amount of trustees colluding are both acceptably low.

Finally, it is worth reiterating that an e-voting system can never attain coercion-resistance (see section 3.1), so we will not aim to cover that property.

We must make some assumptions about the environment in which this protocol will operate:

- We are not "programming Satan's computer" [56]. That is we must assume that each entity can trust the use of their own computer system and the integrity of the software running on it.
- We assume the existence of a private communication channel that can be established as needed between any two entities in the system.

With this in mind, the aims of the Astris protocol are defined in Table 5.1

#	**Objective**
1	Create a voting protocol that is correct, secret, fair, robust, receipt-free, individually and universally verifiable, universally auditable and honours eligibility.
2	Create a voting protocol with no single points of trust, minimizing trust requirements.
3	Create a voting protocol which maximizes auditability and transparency.

Table 5.1.: Table of Objectives for the Astris Protocol.

The non-goals of the protocol are:

- To define the exact authentication protocol to prove eligibility.
- To define the *correct* balance between security and robustness.
- To provide defence against a post-quantum adversary.

This last non-goal is an interesting one, that later work may wish to revisit. In order to

create the protocol in the manner proposed, we will require a public key encryption system with the following properties:

- There must be a distributed threshold key generation protocol without any party gaining knowledge of the secret key at any point during the protocol. This is important to ensure that there is no possibility to decrypt the data without a threshold of keyholders working together.
- It must be a probabilistic encryption. That is the same plaintext must not always encrypt to the same ciphertext. This is important as votes are going to have little variation in plaintext (i.e. they will be 1 or 0).
- It must support homomorphic addition. This is so we can add up the votes without decrypting them. There is an alternative to homomorphism in voting systems using mix-nets, but our protocol will homomorphism, and so requires an additive homomorphism.
- It must have a signature scheme, which can sign arbitrary messages.
- It must support verifiable encryption. We must be able to proof that our vote plaintext has a certain format, and the combined votes have a certain format. That is, we need to be able to prove that we voted for at most n candidates — the vote for each candidate is either a 0 or a 1, and the sum over the votes for all candidates the sum of votes v satisfies $0 < v \leq n$.
- It must support verifiable decryption. A trustee must be able to prove that their decryption of the tally is correct.

There are a number of quantum-resistant cryptosystems [57, 58, 59, 60] that fit some or all of these properties. ElGamal, while relying on the Discrete Logarithm Problem (DLP) which has shown to be insecure under Shor's algorithm [61], has all the required properties. Provided all else is equal, using a post-quantum algorithm would indeed lower trust requirements as it would be more likely that the publicly available audit data for the election could *never* be decrypted by an adversary, whereas under ElGamal a post-quantum attack could reveal data. But given that we have —under the reasonable trust assumption that the Registrar will destroy knowledge of the pairings of voter to voter ID— no way to try votes to individuals, the *eternal secrecy* property, this is of minimal added value. Therefore, should we be able to replace our quantum-vulnerable algorithm with a quantum-resistant one, we would further reduce trust requirements in our system, but this reduction would be orthogonal to any other aspect. It would merely be replacing one black box with another.

Given this and the fact that post-quantum algorithms are significantly more complex to implement in software, the author leaves this as future work and accepts that the solution will not be optimally minimized without post-quantum algorithms but that the core of this protocol would be unchanged by switching to one. Due to this, section 5.2 will not mention any specific cryptography and the exact nature of the implementation will be detailed only in section 5.3.

5.2. Protocol Overview

As mentioned at the start of this chapter, Astris is similar in some ways to the Helios v4 protocol, but also the Belenios additions to Helios. The features Astris shares with these existing protocols have been chosen due to their great fit for the problems at hand that all the systems are attempting to solve. It is self-evident that if we wish all our voting data to be public then the use of zero-knowledge proofs are essential, and both Helios and Belenios had already implemented the exact proofs required for Astris. By using the same ElGamal encryption methods, for which these proofs are available, Astris was able to keep almost all data public — more indeed than either Astris or Belenios. The previous section covered the requirements for encryption and zero-knowledge proofs in Astris and the overlap Astris has with the existing system is to cover those needs explicitly.

The Astris protocol is split into distinct stages. There is a setup stage where the election parameters are chosen, the trustees named and the distributed secret key created. This stage also identifies the Registrar, with details for authentication and key material for verifying signatures. Once this initial setup stage is complete, the data are put into a genesis block for the public audit chain for this election and the hash of the block becomes the election identifier used throughout the rest of the protocol. All further data are added to this chain, and any party wishing to verify the election as it happens can join the network and see all the blocks.

One section of the election parameters governs the timing of the phases, which will be used throughout the rest of the protocol.

The second stage is the parameter confirmation stage which is a timed stage. During this all the trustees contribute to the next stages of the distributed key setup. Each signed payload is added to the chain. All confirmations must be added within the time bounds or the protocol is deemed failed and must be restarted. We note that at this stage we require all trustees to confirm, and the protocol can collapse due to a single failure. However, we have not involved the public yet, except early auditors, and so there is little problem in restarting the process, generating a new election ID.

After the parameter confirmation, we have the voter registration phase which is also a timed phase. During this phase, the voters must authenticate with the registrar and have the registrar sign their public key and issue them a voter identifier. The voter then adds this block to the chain, to show that their vote may be allowed and that it will be signed with their secret key. If eligible voters do not register to vote during this period, they may not cast a vote in the vote casting stage.

The next stage is the vote casting stage, also timed. Once voting opens, a voter may cast a vote by encrypting a series of 0's or 1's in the order of the listed candidates. Alongside the encrypted votes the voter will create a series of ZKPs showing that each ciphertext encrypts either a 0 or a 1. The voter will also create a proof that the overall sum of votes is less than or equal to the configured election parameter minimum. Finally, the voter creates a signature over the ballot data and voter identifier with their private key. This signature can then be

verified using the previously published public key.

The last stage is the tally decryption stage, also timed. This poses the most threat to the integrity of the election as unless a threshold of trustees submits partial decryption results to the chain within the time bound, then the election will be void. It seems a legal incentive here would be necessary to encourage co-operation, or at least discourage abstention. Note that any entity may perform the homomorphic summation to get to the final encrypted tally, and once the threshold of partial decryption results have been published any entity may calculate the final result.

This process is visualized in Figure 5.1.

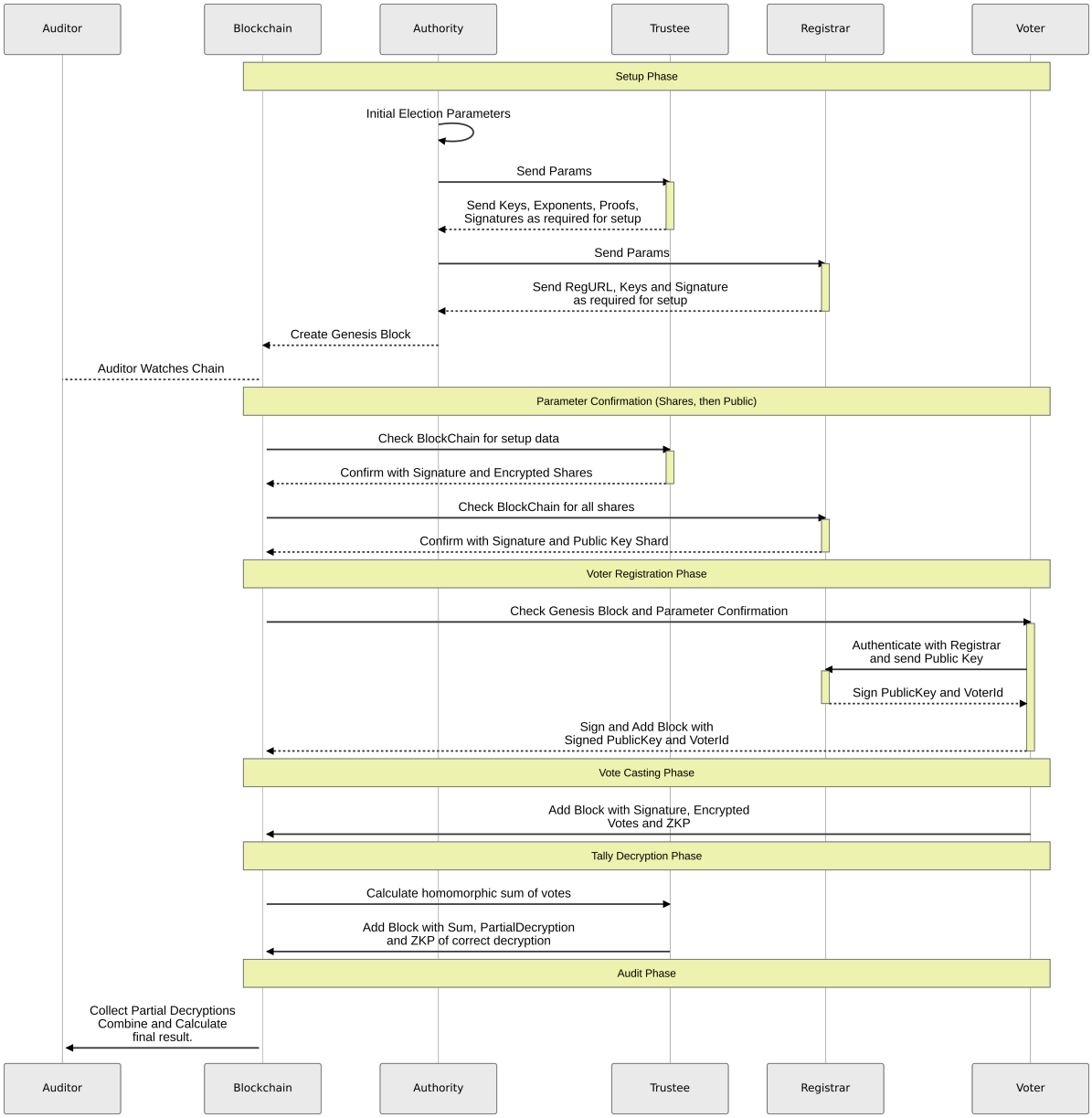

Figure 5.1.: Astris Protocol Sequence

5.3. Protocol Detail - V1.0

This section describes the functioning of the Astris protocol version `1.0`. Along with the appendices, there should be enough information to build a functioning client able to interoperate with the reference implementation. All the data types mentioned are fully described in Appendix A, along with information of how to sign various messages and details of the ZKPs used in the various payloads.

5.3.1. Peer-to-Peer Communication

This specification includes the detail of the inter-node communication as it this integral to the creation of interoperable clients. There are a number of operations within the protocol that call for an *out-of-band* communication between one or more parties. The exact nature of these interactions is not specified, but the assumption is always that the data *could* be safely exchanged over a public network, without the need of a private channel.

All inter-node communication via this API is concerned with the exchange of data about the blockchain. No higher voting functions leak into the communication API.

The Astris inter-node communication is done via gRPC (`https://grpc.io`) which is not technically an acronym but is, in essence, a Google™Remote Procedure Call framework. While gRPC is distinctly client/server in nature, it can be used to build a peer-to-peer network. It's usage greatly simplifies the implementation versus using a much more complex —albeit more feature rich— peer-to-peer protocol such as the suite around `libp2p` (`https://libp2p.io`) or attempting to create a custom peer-to-peer networking solution. Instead, we use a client/server based system to connect a peer in each direction. This means two connections per connected peer, rather than one, but given the reduced complexity of implementation and the ability to use a well-defined data exchange format with existing implementations and code-generation for various programming languages seemed worth it. The only concession to this design is that the common usage is to listen for new data from remote nodes —hence the `RecvBlocks` method described below— but also to add a *single* block ourselves such as a voter registering or voting, or a trustee submitting a partial tally. As such the API is designed to allow most operations to *not* require full participation in the network and only to connect outwards to listening nodes.

As with all gRPC services, there is a file describing the protocol. The full file is described in Appendix C, but the methods that need describing are as follows:

`rpc PeerExchange(Peer) returns (stream Peer)` {} A node will call this method to exchange its own peer information —allowing the remote node to *call back* if wanted— and the remote node should send any peers it knows of back to the caller, as it discovers them. The number of peers a node should remember or send with this call is not specified but nodes should respond to this with as much useful information as possible.

`rpc RecvBlocks(Empty) returns (stream BlockHeader)` {} This is a request from the call-

ing node to ask the remote node to send new blocks as it discovers them. The remote node *should* send the block currently at the end of its chain immediately. This allows the local node to attempt to get itself in sync with the remote node, or ascertain whether the remote node is *behind* it.

`rpc GetBlock(BlockID) returns (FullBlock)` {} This is a request for a block with a specific block ID. If the remote node knows about this block it should return it.

`rpc FromBlock(BlockID) returns (stream FullBlock)` {} Ask the remote node to send an ordered stream of blocks, starting with the block with the given ID and continuing until the remote peer's current head of chain.

`rpc AtDepth(Depth) returns (BlockID)` {} Get the block ID of the block at a given depth according to the remote peer. This method allows us to use a binary search to find where our view of the chain differs from

`rpc PublishBlock(FullBlock) returns (Acceptance)` {} Publish a block to the chain.

5.3.2. Setup Phase

The setup phase involves the Authority, Registrar and Trustees and is done before the blockchain is created. The data created in the phase will form the genesis block for the chain for this election.

1. The Authority decides on the `ElGamalParams` EG, the number of Trustees l and the required number of Trustees to decrypt the votes t
2. Each Trustee is assigned an index i where $0 \leq i < l$,
3. The Authority sends each Trustee their initial data (EG, i, t)
4. Each Trustee Creates their `TrusteeSetup` from the initial data and returns it to the Authority.
   a) Let p, q, g be the parameters from EG.
   b) Generate two `KeyPair` objects from EG. Label one as signing keypair and one as encryption keypair.
   c) Let the public key of the signing keypair be P_{sig} and the public key of the encryption keypair be P_{enc}.
   d) Create a `ProofOfKnowledge` Z_{enc} of the secret key corresponding to P_{enc}.
   e) Create C an ordered list of $t + 1$ random integers modulo q, which represent the coefficients of a polynomial of order t.
   f) For each $c \in C$, calculate the exponent $e = g^c \bmod p$. Let E be the ordered set of all the exponents for each c. These exponents act as a commitment to the coefficients, which the trustee does not reveal.
   g) Create a `Signature` S over $(i, E, P_{sig}, P_{enc}, Z_{enc})$ by creating a message as described

in the apenndix for `TrusteeSetup`

h) The trustee returns $(i, E, P_{sig}, P_{enc}, Z_{enc}, S)$ to the Authority and keeps hold of the secret keys and coefficients.

5. The Registrar creates their `RegistrarSetup` data from the initial data and returns it to the Authority.

   a) Generate a `KeyPair` from EG to use as a signing key.

   b) The registrar must provide a registration URL U_r where the webpage must provide the facility to authenticate the voter and provide them with the information for them to complete the Voter Registration Phase.

   c) Create a `Signature` S over (P_{sig}, U_r)

   d) Return (P_{sig}, U_r, S) to the Authority.

6. The Authority verifies all the proofs and signatures.

7. The Authority then fills in the rest of the data required in the `RegistrarSetup` using the responses from the Trustees and the Registrar.

8. The Authority then creates a genesis block using the Canonical JSON Encoding of the setup data, starting the election.

Once the genesis block is created, the election ID is defined as the block ID of the genesis block. Now the Authority starts a peer-to-peer node service for the election and publishes the peer address as a `host:port` pair and the election ID.

At this point any node may connect and watch the growth of the chain by following the rules described in each section for adding new blocks to the chain.

From this point onwards the setup data are fixed provide a reference for all cryptographic and time-sensitive processes in the following steps.

5.3.3. Parameter Confirmation Phase

This phase involves the Trustees, and each must produce two blocks each for the chain. All trustees must add the first of these blocks before any can add the second. All trustees must add both blocks in the allotted time period specified in the setup data timing section using `setup.timing.parameterConfirmation` and `setup.timing.timeZone` to calculate the minimum and maximum times allowed for the blocks to be added.

This phase is split into two sections. The sharing of encrypted shares and the publishing of the public key shards.

1. For Each trustee with index i

   a) Create the secret shares for each other Trustee j as S_{ij} encrypting them with the Trustee's published public encryption key. The secret share is created by evaluating the polynomial chosen by the trustee's secret coefficients at the point j. The value

is then encrypted with the recipient's public encryption key.

b) Each share is signed individually with the trustee's private signing key using as described in the appendix.

c) Put the data into a `PayloadTrusteeShares` object, mine into a block and publish to the network.

2. Wait until all trustees have added their blocks to the chain.

The chain should only accept the blocks if the signatures are valid, using the trustee's verification key from the setup data. It should also verify that the shares cover every trustee except the publisher.

1. Each trustee i:

a) Validates the signature on the shares.

b) Decrypts all S_{ji} — shares for this trustee from the others

c) Validates each share by computing the product $\prod_{k=0}^{t} (E_{jk})^{i^k}$ where E_{jk} is the public exponent (commitment) from Trustee j, at position k. This should match the exponent $g^{S_{ji}}$, and if not the parameters are incorrect and the process should be considered failed, the Trustee will call out the failure to the Authority and the election can be aborted.

d) Given the shares are valid: computes the key pair shard of the distributed key. This is the sum of all the decrypted shares (along with a calculation of the trustee's own share S_{ii}) into a value x_i. The value x_i is now the secret key for the shard, so we calculate the public key y_i.

e) Creates a `ProofOfKnowledge` of the secret key x_i.

f) Creates a `Signature` over the data as described in the appendix.

g) Creates a `PayloadTrusteePublic` block and publishes to the chain.

The chain should only accept blocks where both the proof of knowledge and the signatures are valid, and we have not seen the trustee's block yet. Note that the share validation is proof the other trustees have behaved correctly in generating their commitments (the exponents). In order to further validate the block the node should also calculate the expected $g^{S_{ji}}$ values using the public exponents. Then the expected public key value returned by the Trustee y_i, can also be calculated using the public information as $y_i = g^{x_i}$ and $x_i = \sum_{j=1}^{n} S_{ji}$ (where n is the number of trustees), therefore $y_i = \prod_{j=1}^{n} g^{S_{ji}}$. As the ZKP proves that the trustee does indeed have knowledge of the correct private key, we can be sure that all the shares have been generated correctly.

After this all participants have enough data to combine the public key shards into a combined public key to use for encrypting votes, provided all the trustees have correctly submitted both

blocks and if not, then the election cannot continue.

5.3.4. Voter Registration Phase

This phase involves the Voters and the Registrar. Timings for the phase should be taken from `setup.timing.voterRegistration`.

1. Each eligible voter creates a keypair for the election using the ElGamal parameters in the setup data. The secret key for the election must be kept safe by the voter.
2. They then use a browser to navigate to (a GET request) the registration URL with the addition of the extra URL parameters:
   `election=<electionid>&public_key=<voter_public_key_json>`
3. The remote URL authenticates them via a mechanism the approved by the registrar.
4. The registrar gives the voter a unique identifier for them for this election and creates a signature over the data as described in the appendix on `PayloadVoterRegistration`. This will be the `registrarSig`.
5. The registrar gives the voter the signature and identifier.
6. The voter then creates their own signature as `voterSig` using the method described in the appendix on `PayloadVoterRegistration`.
7. The voter then combines these data into the `PayloadVoterRegistration` and publishes the block.

The chain should only accept blocks were the voter ID has not been seen before and both the signatures are valid. The first restriction could be lifted to allow voters to re-register should they lose their keys, but for version `1.0` only a single registration per voter is allowed.

Note that we define the mechanism for propagating the election identifier and the voter's public key to the Registrar, but do not specify how the Registrar should return the data. This is deliberate as each implementation of the voting software may have a different mechanism for accepting the data back. We could use a redirection based approach similar to OAuth, with the opportunity for an *out-of-band* method for browser-based interfaces. However, that would complicate the protocol beyond the scope of this report.

By the time the registration phase closes, no more voters can register. Only those voters that have registered by this point will be able to cast a vote.

5.3.5. Vote Casting Phase

This phase involves the Voters only. The timings should be taken from `setup.timing.voteCasting`. In this phase the voters will require their secret signing key.

Once this phase is opened, voters may cast a ballot by creating a number of encrypted votes (one for each candidate), encrypted with the public key for the election. This public key is not

directly published on the chain, but instead is implicit and may be calculated by the client from the data that is on the chain.

These votes will be assured to each be representing 0 or 1 by means of a zero-knowledge proof. We then also combine them into an overall proof that the sum of the votes is less than a threshold given by x, taken from `setup.maxChoices` (probably 1 in most cases but can be $0 < x \leq num_{candidates}$). The ElGamal based system we will use for the encryption makes the proofs fairly simple and very compact.

1. The voter decides on up to x candidates they wish to vote for.
2. For each candidate, the voter creates a `CipherText` of the value 0 or 1 as the vote (or not) for that candidate. We use Exponential ElGamal for this as it is additively homomorphic, so the actual value enciphered is $n = g^v \bmod p$ where $v \in (0, 1)$. This encryption requires a random value r, which is available to the voter. This breaks the receipt-free property. See section 7.3 for a full discussion of the implications of this.
3. The voter now creates a set of `ZeroKnowledgeProof` data to show that each ciphertext indeed encrypts a 0 or 1 and a final one proving that the sum of the votes is at most x using the method in appendix B.
4. The voter creates a signature for the `PayloadCastVote` using their secret signing key generated in the registration phase.
5. The voter combines these data along with their voter identifier into a `PayloadCastVote` and submits it to the chain.

The chain should only accept votes from voter identifiers that are both registered during the registration phase and have a valid signature and whose ZKPs validate correctly.

In this version we will allow duplicate voting, with *last-vote-cast-wins* semantics. That means a voter can vote more than once, but only the final vote counts. Ordering is enforced by the chain depth, not the timestamp.

5.3.6. Tally Decryption Phase

This is the final interactive phase and involves the trustees only. The timings for this phase should be those from the `setup.timings.tallyDecryption`. We hope that all trustees participate in this phase, however only $t =$ `setup.trusteesRequired` trustees are necessary.

1. The trustee creates the homomorphic sum of valid votes. Remember that we have last-vote-counts, and only one vote per voter should be added. The Exponential ElGamal process means that homomorphic addition is done by multiplication of the `a` and `b` values of the ciphertext.
2. The trustee then uses the secret part of the shard of the key to produce a partial decryption of each tally.
3. The trustee produces a `ZeroKnowledgeProof` of correct decryption using the method in

appendix B.

4. The trustee produces a `Signature` over this `PayloadPartialTally`.
5. The trustee produces the final `PayloadPartialTally` with the signature and proofs and adds the block to the chain.

Blocks should only be accepted if not already seen for this trustee. The homomorphic sum of valid votes can be calculated externally and should match the values confirmed in this block. The ZKPs of decryption should be valid and the signature should also validate.

After at least t trustees have submitted these blocks, we can start to combine the partial decryption results to reveal the final plain text tallies. These will still be the exponentials of the final values, but can be converted back by means of a discrete logarithm lookup. Given we know the maximum possible value for any vote is the number of voters, we have an upper bound on the amount of exponentiations we must perform to calculate the logarithms. Even with 50 million voters, this is not a particularly taxing computation. These final adjusted tallies do not need to be added to the chain as they are directly computable by the state of the chain up to this point, with no external data required. Implementations are likely to cache such results.

We can now still allow all the remaining trustees to add their blocks. Each additional block after the first t will also be able to be combined with any $t-1$ other partial decryption results to produce the *same* tally. Trustees could withhold their decryption at this point, but cannot have a dishonest decryption validated due to the proofs supplied in this phase and during the parameter confirmation phase.

5.3.7. Verification Phase

This is the simplest possible phase as any observer can retrace the chain, applying the exact same checks that were performed by each node as the blocks were added in the first place. If the chain is valid, then the election is also valid and the result can be confirmed.

6. Implementation

6.1. Objectives of the Implementation

The implementation of the Astris voting system will be utility-first. It is intended to prove the feasibility of creating such a system and provide an insight into the practical application of the protocol. User-friendliness will not be a priority, nor efficiency, but instead correctness will be the focus. This extends to some aspects of the peer-to-peer networking such as automatic peer-discovery and black/white listing capability (although these features may be implemented in the fullness of time).

The objective of the implementation is to create a software service that can adhere to the protocol and can perform the roles of:

- **Authority:** the election organizer role. Creating the initial election parameters and setup information, the pieces required to allow the trustees and registrar to perform their setup duties and to create the genesis block.
- **Trustee:** one of the decryption trustees. Participating in the distributed key generation algorithm in the setup phase, and performing the partial decryption in the tally phase.
- **Registrar:** the eligibility authority. Creating the setup parameters, keys and voter list. Running a web-server to provide a dummy authentication module to allow voters to have their voting keys signed.
- **Voter:** a potential voter. Creating a voting key-pair, creating the registration URL for the voter to authenticate with the Registrar and have their key signed. Adding the signed key to the chain.
- **Auditor:** a member of the network simply validating all blocks on the chain, and therefore all workings of the election.

The **Auditor** is actually performed by any of the other roles, as when they connect to the blockchain they will all validate all the blocks.

The software should show that these functions can each be run independently, with no collusion and that the privacy of the voter is maintained.

6.2. Architecture of the Software

The author chose Go (`https://golang.org/`) as the implementation language. This choice was for a few main reasons not least of which is that the author is familiar with the language and have implemented several small projects in it. Another good reason for the choice is that the simplicity and enforced formatting make most Go projects easily accessible to new contributors. Finally, another reason was that Go code can be trivially compiled to WebAssembly (WASM)

easily, meaning the same cryptographic code could be used in the browser as natively in the future.

The Astris specification uses gRPC (`https://grpc.io`) as the inter-node communication protocol and Go is well-supported by the gRPC toolchains.

The core blockchain in the Go implementation will be backed by an SQLite database for simplicity. Only a simple key-value store is required, but SQLite will aid in ad-hoc debugging and will function happily as a key-value store. It is one of the most widely deployed pieces of software in existence [62] and so reliability should not be an issue.

The software is designed to run on a command line, but provide an HTML user interface where appropriate. This *local* HTML UI will help perform the tasks necessary for each role the software wishes to accomplish. The UI uses a JavaScript framework: ReactJS (`https://reactjs.org`) and CSS framework: Bulma (`https://bulma.io`) to accelerate development. The command line interface can pick a random local port to run it's web-server and optionally open the user's default web browser to the correct URL. This design decision has a number of benefits. Firstly, creating usable interfaces in HTML is far easier than the same attempt at a text-based interface and secondly the communications between the tool and the browser are all local, and therefore the chances of interference or snooping on them is limited. This second property allows us to forgo the work of re-creating all the cryptographic code for the browser. As mentioned, Go can compile to WASM, but there is a significant amount of *wiring* to ensure type-safe communication between the WASM module and the JavaScript code.

The code for this project is versioned with the Git version control software and a copy of the repository is located on GitHub: `https://github.com/thechriswalker/go-astris`.

6.3. Notes on Implementation

As with any software project, pinning down the scope of the work and the requirements is essential. However, in a project such as this, the implementation itself fed back practicality information to the design and the final output of both have been mutually influential. Initially, it was intended to be a more complete software application, but as the implementation itself took a considerable amount of time, some of the more user-experience focused features where dropped in favour of providing the core functionality correctly.

A notable example of a feature that would be almost essential in a production-ready implementation of Astris is the automated peer-to-peer discovery methods (as mentioned earlier) and Network Address Translation (NAT) traversal. Both of these problems have common solutions: DHT (e.g. Kademlia [63], Chord [64]) for discovery and STUN/TURN (RFC8656[i]) for NAT traversal or a solution encompassing both issues such as the `libp2p` (`https://libp2p.io`) could be added to this implementation in the future. The lack of these feature does not prevent the software from working, but instead the software requires more complex configuration

[i] `https://tools.ietf.org/html/rfc8656`

in the case that the user wishes to receive incoming connections. To counter the issues the STUN/TURN aim to solve, instead a carefully designed API lowers the impact of the issue, mitigating it completely for the common use cases.

A large amount of time sunk into the implementation went on user interface, such interfaces are necessary to work the system work, but require a lot of small detail to make functional. A web UI was the most efficient method of creating a usable interface, but there was still a great deal of iteration required. In the end, the web-based interface was dropped in favour of a command-line only approach to save time. In contrast, the cryptographic code —which the author had expected to be more complex— was fairly straightforward to implement from existing knowledge describing the constructs. Similarly, once the design of the peer-to-peer API was fixed, the implementation of the gRPC service was relatively straightforward. This could well be explained by the authors lack of skill in the UI arena compared to *systems programming*.

7. Analysis

7.1. Analysis of the Proposed Protocol

Does the proposed protocol achieve the objectives intended? Table 5.1 listed the objectives of the protocol. The first objective was concerned with the security properties of the protocol. It aimed to achieve correctness, secrecy, fairness, robustness, receipt-freeness, individual verifiability, universal verifiability, auditability and eligibility.

We will discover as we cover these properties that many rely on the existence of sufficient honest Trustees to make it impossible for a group of dishonest Trustees large enough to be capable of decryption of vote data independently. This in fact was the purpose of requiring a threshold of Trustees to participate in decryption and the exact number of Trustees and the threshold required for decryption should be chosen with care according to the threat model of the specific election. As we look at specific properties we will note if they are broken by a failure of this threshold, but not consider that a failing of the protocol. Similarly, we will assume any adversary cannot break or subvert the encryption directly.

The proposal does achieve correctness: all votes cast are cryptographically ensured to be cast according to the rules of the election. The tallying process can be performed by any party and the decryption of the tally is also similarly verifiable. As such any incorrect tally or invalid vote can be detected.

The proposal achieves secrecy to a point. There are two ways that an attacker might be able to link the encrypted vote to the voter who cast it. The Registrar holds the information for this link during the registration phase of the process, but a dishonest entity could keep the data afterwards. Another threat comes from the public nature of the Internet, over which the blocks on our chain are shared. While a large scale attack would not be feasible: a targeted Sybil attack —discussed in section 4.4— could isolate a voter from enough peers that the adversary could intercept their submitted vote and link it to them with high confidence. However, neither of these reveal the plaintext vote.

An improvement to secrecy aspect of the protocol could be to use Linkable Ring Signature (LRS) instead of including the voter identifier sign the votes, which would provide K-anonymity while retaining the ability to detect duplicate votes. This is possible as all public keys of voters are registered, but would require some consideration to how to choose the set of public keys to use for each signature. It would seem that in large elections it would not be practical to construct a signature from the keys of all voters as the signature size (and time to compute) grows with the number of signatures involved, so it would be necessary to use only a subset. This subset would need identifying without revealing any more data about the voter identifier than the set it belongs to. Given that we do not know all the voters that will register before the election we cannot assign cohorts at the Registrar, so a different method must be chosen. Provided such a method could be found, and voters are split into groups of G voters, then the

chance of correctly guessing the link between voter and voter would be $1/G$, but the *linkable* nature of the signatures would guarantee that we can detect repeat votes. This in turn would lower the trust requirements for the Registrar, as a leak of data allowing the link between voter and identifier would not directly compromise the link between voter and vote.

The fairness property is provided under our assumption of enough honest Trustees, as no individual votes can be decrypted without a large enough group of Trustees and the ciphertexts and proofs in the vote do not reveal any information about the candidates voted for.

The robustness property is provided to a degree. An adversary cannot introduce errors into the system as they would be detected by all honest participants. The Adversary could not alter votes in the system for the same reason. The peer-to-peer nature of the protocol has advantages and disadvantages when it comes to robustness in the face of adversarial nodes. Theoretically the adversary could mount a DoS attack against the network, flooding it with malicious nodes or preventing legitimate communication between nodes. No distributed system is free from this issue, which is well known as the Byzantine Generals Problem [43]. The usual discussion of the problem dismisses the pathological case where the enemy intercepts and destroys all messages, which is possibly the simplest attack if DoS is the goal. Our protocol does not deal with this, either in the pathological case, or in a takeover attempt (a 51% attack as described in section 4.4). If the intent is to stop the election and DoS attack is *enough*, but if it is to sway the election then the "rules" of the system are embedded in the genesis block, and the genesis block gives the election its identifier. A change of the rules to favour an adversary would change the election identifier, and could not go unnoticed.

The proposal does not provide receipt-freeness. There is "toxic waste" created in the voting process, that is the randomness required in the encryption and ZKP creation. The voter has to have this information to correctly encrypt the vote and prove its validity. If the voter keeps this information, combined with the knowledge they have of which encrypted vote is theirs, they would be able to prove they voted a specific way. As the protocol is *software-independent*, the protocol can be implemented by anyone, including an adversary. We explicitly dismiss the trustworthiness of the software the voter uses, but should the voter wish to publicize their vote (or is being coerced), this waste data is all that is needed to prove how they voted. Some similar systems —e.g. BeleniosRF [25]— have managed to achieve this property using a re-randomization step, such that a 3-rd party can re-encrypt the data with new randomness, whilst retaining the validity of the ZKPs used to validate conformance to election rules. In the case of our proposal, achieving this would have introduced another entity —let us call the entity a *Sealer*— that would handle all the votes and do the re-randomization. This could be provided in such a way that the Sealer would not know anything but the already encrypted vote, except via the implicit information in the communication between the voter and the Sealer, which without a secure and anonymous channel to communicate is significant. The Sealer would also be able to identify the sealed vote on the blockchain and so the introduction of this entity provides another vector to break secrecy. Overcoming this is a particularly tricky problem when all data must be kept public and the sealer must provide proof that the vote has been re-randomized. The author would very much have liked to create a novel solution

to this problem whilst maintaining the other properties but was unable to do so during the course of this report — perhaps future versions of the Astris protocol will be able to plug this hole.

The proposal is both individually and universally verifiable. All votes are public on the blockchain and any entity can verify the entire election process from start to finish. Individuals can be sure their cast vote is included in the tally by computing the tallies themselves. Universal verifiability is satisfied in the same way, the election data is all public except the tally decryption steps which instead are proved in zero-knowledge with publicly verifiable proofs.

The proposal is universally auditable — this feature was the primary driver of the design of the system, all entities can confirm each step of the election from start to finish, including in real-time during the event itself. All data has been designed to be publicly released without compromising the other security properties. This led to the compromise that lost receipt-freeness, in a considered trade-off.

The second main objective was to create a protocol with no single point of trust, minimizing trust requirements. Let us consider all the entities involved in the protocol and how much trust we must put in each one.

The Authority in the protocol is the election organizer, the creator the original genesis block to start the election. That is the full extent of the participation of the Authority. They will likely publish and distribute the election identifier and run an Auditor node endpoint that other user can reliably connect to, but given we have the election identifier, they can no longer influence any aspect of the election or mutate the original election parameters. However, those original election parameters include all the details of the Registrar and the Trustees and all data in the genesis block *is* under the full control of the Authority. We cannot be sure that the data were collected from the entities they claim to be. Ideally the Registrar and the Trustees will be entities independent enough from the Authority that we can trust them to confirm the correctness of the situation. It does remain that in the end we have to trust the election organizer has correctly acquired the details of the entities it claims to have, and not just created keys to pretend to be them. In the author's opinion, the matter of trust here is actually not resting on the Authority, but on the Registrar or Trustee. They must be suitably publicly trustworthy or accountable to provide enough trust that they would contest the Authority if any dishonesty in the setup took place.

The Registrar is a single entity. This implies a single point of trust, which the protocol aimed to alleviate. We could have chosen to have multiple registrar entities, requiring a quorum of valid signatures to validate the voter's eligibility. This method would have put excessive burden on the voter to authenticate with a quorum of the Registrar's individually and perform the same registration step at each one. It was the author's opinion that the eligibility of a potential user to vote is likely decided in the end by a single entity. For example, in a governmental election, a voter would need to prove some form of citizenship — a process likely controlled by a government agency. Any Registrar would have to go through that same agency to validate the eligibility of the voter, so distributing the Registrar role would be simply a shift of the

single point of trust back to the entity that is really doing the validation. Therefore, a single Registrar is used in Astris. The registrar must provide key material and an authentication URL to the Authority for the genesis block in the setup phase. During the voter registration phase the Registrar must be trusted to:

1. Allow eligible voters to register.
2. Generate an opaque voter identifier for the voter which reveals no information about the voter's identity.
3. Keep the link between the voter and the identifier secret, ideally destroying the information at the earliest moment — the end of the registration phase.

All of these items are required of any eligibility system, the Registrar must by definition be able to identify the voter. The data from the voter and the data the Registrar is required to provide must allow the Registrar to restrict the user to a single registration only — preventing double voting. Therefore, the Registrar must be able to link the published data from a voter with the real voter identity irrespective of the protocol here, with the constraints of all data being public. So despite the fact we have three trust requirements here in a very important role, they are fully minimal.

The Trustees are indeed appropriately named, as they hold together the ability to reveal all votes. Due to this fact, we deliberately distribute this trust amongst a group of entities. A threshold of Trustees must come together to decrypt any of the votes or tallies. This threshold is a configurable number, as is the total number of Trustees and will require that at least that number of Trustees must be dishonest before the encryption breaks down. The more Trustees an election has and the higher proportion of them are required for decryption the easier it will for the voter to trust that they will behave correctly. The other possibility is that a Trustee will fail to perform their portion of the decryption. This is a valid attack against the process, which would invalidate the *robustness* property. Again the use of a threshold of Trustees for decryption means that we do not need all Trustees to enable successful decryption. So we need to trust that at least the threshold amount of Trustees will successfully decrypt the tallies. So, given an election with $n \geq 2$ Trustees, of which m where $1 < m \leq n$ are required to decrypt, then we need to trust that:

1. x Trustees, where $x \leq n-m$, will be honest enough not to collude and decrypt sub-tallies or votes.
2. y Trustees, where $y \geq m$, will be honest enough to decrypt the final tally.

The first item here is minimized by minimizing $n - m$. As we also encourage a larger n to distribute trust, this means increasing m towards n. The second item is minimized by *reducing* m. While this may seem contradictory at first, in fact this second item is actually less relevant than the first as the definition of "*honest enough*" is different in both cases. In the second case, breaking the rules will lead to a publicly visible violation which may well have other –perhaps legal— implications for the entity involved. This public violation may be enough to dissuade the Trustee from misbehaving given that they *will* be caught. The first item however

is a violation which may never see the light of day. The colluding Trustees can act together and decrypt the data without being caught, even to anonymously publish the decrypted data —without the proofs— without incriminating themselves. Therefore, from a trust point of view, the first item is vastly more important. Our protocol, by allowing the choice of the number of Trustees and the threshold, has not directly minimized the trust requirements but allowed them to be minimized to the extent required by any particular election.

The Voter as an entity ought to be considered, although in this case it is the *other* Voters that a given Voter must think about. Given that the eligibility of a Voter is the responsibility of the Registrar —whose trust we have already explored— and the Voter can only add their vote to the chain, there is little scope for a Voter to influence the election further than the legitimate casting of their own vote. The only route to this is the possibility of a DoS attack by using the facility for repeated voting. They could repeatedly cast their vote, adding blocks to the chain and preventing other voters from adding their own blocks. The protocol as stands does not address this attack, however it could be mitigated with a rule disallowing repeat votes under a defined combination of conditions such as a set timeframe or without an intermediate vote. So, we must trust the other Voters not to do this, which indeed is not minimal.

Finally, the Auditor is completely trustless. The Auditor role does not really exist as a specific role, indeed any entity whether involved in the election or not can act as Auditor. Any Voter can run the Auditor node to verify the election both in real-time during the event and afterwards provided some peers are still available to provide the chain data.

7.2. Comparison to Existing Protocols

Of the systems covered in section 3.2, the techniques used for achieving the core security properties of voting system where similar in nature. The use of either homomorphic encryption or mix-nets for secrecy, combined with ZKPs for audit without revealing data were common to all of them. Astris is as well very similar in many ways, and this can be attributed to convergent evolution — they are good solutions to the problem at hand and innovative new solutions have not emerged to prove themselves better. However, they all suffer from the same drawbacks as well: they become increasing more fragile and complex as further security guarantees are met. For example, we can reduce trust requirements in the holders of a private key using secret-sharing methods and multi-party-computing to prevent a single entity being able to subvert the whole system. But with every extra multi-party step, we add complexity to the process which gives the opportunity for new attack vectors to creep in. As we make the protocols more complex, the need for an anonymous private communication channel becomes more needed as more side-channel data is produced which could potentially link the voter back to the vote without the requirement for the encryption or mixing algorithms to be compromised. Some proposed voting systems, such as [3], assume the existence of such a channel in order to ensure the security properties they claim. Astris specifically aims to not require such a channel, the properties of a private channel being sufficient.

Considering DLT and its use in existing systems, there are three main classes of use:

- Voting systems that use the cryptocurrencies and their transaction mechanism.
- Those that use cryptocurrencies with complex smart contracts to perform the logic of the voting process.
- Those that use the technology as the Bulletin Board to make the process auditable.

Of the systems this report has touched on that also use DLT in their operation, they all fall into the first two categories. Some aspects of the voting process does end up on the blockchain in all of them, but the primary focus is not on public audit. There are commercial voting systems —such as TiVi [65] and FollowMyVote (`https://followmyvote.com`)— that do use DLT to provide the bulletin board functionality. This report did not consider those systems in depth as they are more difficult to inspect, having little published documentation as to their inner workings. However, Astris does use DLT for exactly this purpose. Blockchains, while a fascinating technology, are not suitable for all situations. The author does not believe that cryptocurrency chains are a wise choice of medium for electronic voting. They are expensive and slow, but their immutability does provide a significant layer of trust to data stored on them. In the author's mind, they are a perfect fit for the bulletin board, as the data are public and verifiable by all. Hence, Astris deviates from the majority of systems using blockchains, as it only uses the blockchain as its audit trail.

7.3. Implementation Success and Analysis

The software implementation of Astris did implement the entire specification as described in this report. It was able to run through a complete election with simulated voters. The Audit node was able to verify the election and produce the final results when available. The peer-to-peer network allowed nodes to share and verify the chain data and new blocks into the system could be distributed, new peers into the network could "catch up" on the state of the chain from just the election identifier and a single seed peer. The software was able to play the role of each of the key entities in the system. As such the implementation did achieve the goal set out for it, which was to prove that such a system as specified could be implemented, with the only data-sharing via the blocks exchanged. Time pressure meant having to compromise on the web-based user-interface and only providing command line access to the functionality.

The performance of the system when verifying large chains was less than hoped, but no time was spent looking to optimize this, so there may well be room for improvement. The system running on an Intel i7-8550U processor could fully validate an election chain with one million voters in approximately 10 hours using only a single core for the bulk of the work. Much of the processing involved in the validation is well suited to parallelization, and so with some effort the author believes this number could be reduced significantly.

Therefore, despite the fact that the user interface for the software was clunky enough to not be suitable for non-technical users and the amount of manual steps involved in running

an election, the author considers the software implementation to be a success.

The design of the software to run locally, means that all cryptographic processes occur on the user's own machine. Provided they can trust the source of the software, they can trust that it will not misbehave. Other remote voting systems (such as Helios) provide a client-server architecture where the user uses a web-browser to connect to the remote voting service. Although in the case of Helios the web UI runs cryptographic code in the user's browser, as a remote system that *sends the client to us*, it is onerous to check that the code is correct, and that the server is running the code we expect. In contrast, with the Astris program, the executable can be verified via out-of-band checksums, or even compiled directly by the user. Although the program does use a web interface for some tasks, the web-server is run locally by the program, and by default picks a random ephemeral port to run on. This mitigates a range of attacks against a traditional remote web-based product.

That being said, there is plenty of room for improvement that came to light during the implementation phase. Too late in the process to be able to integrate directly, but future work could benefit from the experience. The main pain point was that the gRPC protocol for implementing the API was not a great choice. It appeared so during the research phase, as it provided a number of desirable features:

- An API contract definition that has many client and server code generators to enable services in other languages to be built more easily.
- A well-established and mature RPC system.
- Works over HTTPS (over HTTP/2) so TLS would provide a secure channel for communications.
- Has support for authentication use TLS client certificates (and mutual certificate validation)

Unfortunately it became apparent as the peer-to-peer code was being written that this last point did not support the author as intended. We are working peer-to-peer and using a Certificate Authority somewhat defeats the purpose. Instead, the peer-to-peer connections would work best with self-signed certificates and with the certificate itself used as the identity of the peer. The implementations of gRPC did not support this use case and so the API became such that public Audit nodes would use "real" TLS certificates, and the clients looking to perform one of the *short actions* —such as adding a single block— could do so without having to obtain or generate a certificate at all. If the system were re-designed, a networking library more suited to the needs of Astris should be considered.

8. Conclusion

This report has considered voting, e-voting, blockchains, trust and practicality. We have seen the security properties that are desirable in voting systems and how a number of practical and theoretic systems approach them. We have looked at those same systems with respect to the amount of trust they require from their users and which entities within the system must be trusted before the security properties begin to fail. We have looked at a newly proposed system designed to minimize these trust requirements while maximizing the coverage of the security properties.

After considering all this, we now look at the question posed in the introduction: *Can we reach a minimum level of trust while maintaining the desirable security properties?*.

We could interpret this question in two ways:

- Once trust requirements have reached a minimum, have we maintained the desired security properties?
- Is there a minimal trust requirement where the desired security properties are achieved?

The author believes that the answer to the first question is *no*, and the answer to the second is *yes*.

To justify the first answer, we consider the Astris protocol which was designed to be trust-minimal. One of the building blocks for reducing trust requirements is to distribute that trust amongst a number of entities, rather than one. The proportion of the number of entities that must be trustworthy to the number that can be dishonest without consequence for the protocol is key here and the smaller this proportion the lower the trust requirements will be. We discovered that by using such a threshold we also introduce the risk of fragility, reducing the *robustness* of the protocol. Hence, Astris is a system where the trust requirements are configurable and must be traded-off against robustness. This fact opposes the first statement of the question: to minimize trust requirements we will necessarily increase fragility.

For the second answer, the author was sorely tempted to write *it depends*: which is at best a non-answer and at worst facetious. Unfortunately, the answer to so many questions —especially in software engineering— **is** *it depends*, when there is so much context left unspecified. The reason that more context would be useful in this question is that knowing the threat model of the election and therefore a way to assess the trade-offs that must inevitably be made. However, irrespective of the context, once we have decided on the security properties and trade-offs we wish to make then the trust requirements can indeed be reduced to a minimum. Note that we have not specified how low this minimum is — it is certainly non-zero.

In summary, it is indeed possible to minimize trust requirements for an e-voting scheme, but that minimum will be dependent on the threat model of the election being held and how the trade-offs in the configuration are made. It leaves open the question of whether this reduction in trust is *sufficient* for the body of voters. The theoretical arguments aside, the

continued use of and increasing adoption of the state-backed remote voting system in Estonia for state-level elections despite reports outlining its flaws would indicate that the citizens are increasingly willing to trust the system in practice. This report's coverage of the Estonian i-Voting platform (subsection 3.2.3) demonstrates that there are higher trust requirements than the Astris proposal, but they are evidently low enough. This does not mean –and the author is not suggesting— that we should not strive to improve voting systems and reduce the trust requirements for their use, but the opposite: it appears there is a great opportunity for remote voting and the author firmly believes that reduced transparency and increased auditability would be a major improvement to state-level elections.

References

[1] S. Panja and B. K. Roy, "A secure end-to-end verifiable e-voting system using zero knowledge based blockchain," Tech. Rep. 466, 2018.

[2] P. McCorry, S. F. Shahandashti, and F. Hao, "A Smart Contract for Boardroom Voting with Maximum Voter Privacy," Tech. Rep. 110, 2017.

[3] Y. Liu and Q. Wang, "An E-voting Protocol Based on Blockchain," Tech. Rep. 1043, 2017.

[4] X. Yang, X. Yi, S. Nepal, A. Kelarev, and F. Han, "Blockchain voting: Publicly verifiable online voting protocol without trusted tallying authorities," *Future Generation Computer Systems*, vol. 112, pp. 859–874, Nov. 2020.

[5] C. Spadafora, R. Longo, and M. Sala, "Coercion-Resistant Blockchain-Based E-Voting Protocol," Tech. Rep. 674, 2020.

[6] T. Dimtiriou, "Efficient, Coercion-free and Universally Verifiable Blockchain-based Voting," Tech. Rep. 1406, 2019.

[7] G. Tsoukalas, K. Papadimitriou, and P. Louridas, "From Helios to Zeus," {*USENIX*} *Journal of Election Technology and Systems ({JETS})*, vol. 1, no. 1, pp. 1–17, 2013.

[8] Z. Xia, C. Culnane, J. Heather, H. Jonker, P. Y. A. Ryan, S. Schneider, and S. Srinivasan, "Versatile Prêt à Voter: Handling multiple election methods with a unified interface," 2010.

[9] P. Ryan, D. Bismark, J. Heather, S. Schneider, and Z. Xia, "PrÊt À Voter: A Voter-Verifiable Voting System," *Information Forensics and Security, IEEE Transactions on*, vol. 4, pp. 662–673, Jan. 2010.

[10] B. Yu, J. Liu, A. Sakzad, S. Nepal, P. Rimba, R. Steinfeld, and M. H. Au, "Platform-independent Secure Blockchain-Based Voting System," Tech. Rep. 657, 2018.

[11] M. Seifelnasr, H. S. Galal, and A. M. Youssef, "Scalable Open-Vote Network on Ethereum," Tech. Rep. 033, 2020.

[12] S. Gajek and M. Lewandowsky, "Trustless, Censorship-Resilient and Scalable Votings in the Permission-based Blockchain Model," Tech. Rep. 617, 2019.

[13] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachene, "An homomorphic LWE based E-voting Scheme," p. 21.

[14] J. Epstein, "Electronic Voting," *Computer*, vol. 40, no. 8, pp. 92–95, Aug. 2007.

[15] S. Delaune and S. Kremer, *Formalising Security Properties in Electronic Voting Protocols*, Apr. 2010.

[16] H. Li, A. R. Kankanala, and X. Zou, "A taxonomy and comparison of remote voting schemes," in *2014 23rd International Conference on Computer Communication and Net-*

works (ICCCN), Aug. 2014, pp. 1–8.

[17] N. Hastings, R. Peralta, S. Popoveniuc, and A. Regenscheid, "Security considerations for remote electronic UOCAVA voting," National Institute of Standards and Technology, Gaithersburg, MD, Tech. Rep. NIST IR 7770, 2011.

[18] D. M. Romano, "The nature of trust: Conceptual and operational clarification," Ph.D. dissertation, Louisiana State University and Agricultural & Mechanical College, United States – Louisiana, 2003.

[19] A. Juels, D. Catalano, and M. Jakobsson, "Coercion-Resistant Electronic Elections," Tech. Rep. 165, 2002.

[20] S. G. Weber, R. Araujo, and J. Buchmann, "On Coercion-Resistant Electronic Elections with Linear Work," in *The Second International Conference on Availability, Reliability and Security (ARES'07)*, Apr. 2007, pp. 908–916.

[21] R. L. Rivest, "On the notion of 'software independence' in voting systems," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 366, no. 1881, pp. 3759–3767, Oct. 2008.

[22] B. Adida, "Helios: Web-Based Open-Audit Voting," in *Proceedings of the 17th Conference on Security Symposium*, ser. SS'08. USA: USENIX Association, Jul. 2008, pp. 335–348.

[23] "Helios Voting - FAQ," https://vote.heliosvoting.org/faq.

[24] "Helios - Helios v3 Verification Specs," https://documentation.heliosvoting.org/verification-specs/helios-v3-verification-specs.

[25] V. Cortier, P. Gaudry, and S. Glondu, "Belenios: A Simple Private and Verifiable Electronic Voting System," in *Foundations of Security, Protocols, and Equational Reasoning*, J. D. Guttman, C. E. Landwehr, J. Meseguer, and D. Pavlovic, Eds. Cham: Springer International Publishing, 2019, vol. 11565, pp. 214–238.

[26] "OSCE/ODIHR Election Expert Team Final Report 2019," Office for Democratic Institutions and Human Rights, Tech. Rep., Jun. 2019.

[27] "Security Analysis of the Estonian Internet Voting System — Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security," https://dl.acm.org/doi/10.1145/2660267.2660315.

[28] Z. Zhao and T.-H. H. Chan, "How to Vote Privately Using Bitcoin," in *Information and Communications Security*, ser. Lecture Notes in Computer Science, S. Qing, E. Okamoto, K. Kim, and D. Liu, Eds. Cham: Springer International Publishing, 2016, pp. 82–96.

[29] P. Tarasov and H. Tewari, "Internet Voting Using Zcash," Tech. Rep. 585, 2017.

[30] K. N. Khaqqi, J. J. Sikorski, K. Hadinoto, and M. Kraft, "Incorporating seller/buyer reputation-based system in blockchain-enabled emission trading application," *Applied Energy*, vol. 209, pp. 8–19, Jan. 2018.

[31] S. B. Patel, P. Bhattacharya, S. Tanwar, and N. Kumar, "KiRTi: A Blockchain-based

Credit Recommender System for Financial Institutions," *IEEE Transactions on Network Science and Engineering*, pp. 1–1, 2020.

[32] J. Gao, K. O. Asamoah, E. B. Sifah, A. Smahi, Q. Xia, H. Xia, X. Zhang, and G. Dong, "GridMonitoring: Secured Sovereign Blockchain Based Monitoring on Smart Grid," *IEEE Access*, vol. 6, pp. 9917–9925, 2018.

[33] Y. Wang, Z. Su, N. Zhang, J. Chen, X. Sun, Z. Ye, and Z. Zhou, "SPDS: A Secure and Auditable Private Data Sharing Scheme for Smart Grid Based on Blockchain and Smart Contract," *IEEE Transactions on Industrial Informatics*, pp. 1–1, 2020.

[34] "Blockchains for Business Process Management - Challenges and Opportunities — ACM Transactions on Management Information Systems," https://dl.acm.org/doi/10.1145/3183367.

[35] "Namecoin," https://www.namecoin.org/resources/whitepaper/.

[36] "ENS - Ethereum Name Service," https://ens.domains/about.

[37] "Unstoppable Domains - Documentation," https://docs.unstoppabledomains.com/.

[38] A. Yakubov, W. M. Shbair, A. Wallbom, D. Sanda, and R. State, "A blockchain-based PKI management framework," in *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*. Taipei, Taiwan: IEEE, Apr. 2018, pp. 1–6.

[39] M. Y. Kubilay, M. S. Kiraz, and H. A. Mantar, "CertLedger: A new PKI model with Certificate Transparency based on blockchain," *Computers & Security*, vol. 85, pp. 333–352, Aug. 2019.

[40] S. Matsumoto and R. M. Reischuk, "IKP: Turning a PKI Around with Blockchains," Tech. Rep. 1018, 2016.

[41] K. Martin, "Everyday Cryptography," in *Everyday Cryptography*, 2nd ed. Oxford Unioversity Press, 2017, pp. 214–215.

[42] M. Stevens, E. Bursztein, P. Karpman, A. Albertini, and Y. Markov, "The First Collision for Full SHA-1," in *Advances in Cryptology – CRYPTO 2017*, J. Katz and H. Shacham, Eds. Cham: Springer International Publishing, 2017, vol. 10401, pp. 570–596.

[43] L. Lamport, R. Shostak, and M. Pease, "The Byzantine Generals Problem," *ACM Transactions on Programming Languages and Systems*, vol. 4, no. 3, pp. 382–401, Jul. 1982.

[44] A. Back, "Hashcash - A Denial of Service Counter-Measure," *Tech Report*, p. 10, Aug. 2002.

[45] P. Gaži, A. Kiayias, and A. Russell, "Stake-Bleeding Attacks on Proof-of-Stake Blockchains," in *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*, Jun. 2018, pp. 85–92.

[46] A. Mackenzie, "Memcoin2: A Hybrid Proof-of-Work, Proof-of-Stake crypto-currency," p. 19.

[47] V. Buterin and V. Griffith, "Casper the Friendly Finality Gadget," *arXiv:1710.09437 [cs]*, Jan. 2019.

[48] Y. Wen, F. Lu, Y. Liu, and X. Huang, "Attacks and countermeasures on blockchains: A survey from layering perspective," *Computer Networks*, p. 107978, Mar. 2021.

[49] I. Eyal and E. G. Sirer, "Majority Is Not Enough: Bitcoin Mining Is Vulnerable," in *Financial Cryptography and Data Security*, ser. Lecture Notes in Computer Science, N. Christin and R. Safavi-Naini, Eds. Berlin, Heidelberg: Springer, 2014, pp. 436–454.

[50] A. Sapirshtein, Y. Sompolinsky, and A. Zohar, "Optimal Selfish Mining Strategies in Bitcoin," in *Financial Cryptography and Data Security*, ser. Lecture Notes in Computer Science, J. Grossklags and B. Preneel, Eds. Berlin, Heidelberg: Springer, 2017, pp. 515–532.

[51] G. O. Karame, E. Androulaki, and S. Capkun, "Double-spending fast payments in bitcoin," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, ser. CCS '12. New York, NY, USA: Association for Computing Machinery, Oct. 2012, pp. 906–917.

[52] M. Apostolaki, A. Zohar, and L. Vanbever, "Hijacking Bitcoin: Routing Attacks on Cryptocurrencies," in *2017 IEEE Symposium on Security and Privacy (SP)*, May 2017, pp. 375–392.

[53] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse Attacks on Bitcoin's Peer-to-Peer Network," Tech. Rep. 263, 2015.

[54] J. R. Douceur, "The Sybil Attack," in *Peer-to-Peer Systems*, ser. Lecture Notes in Computer Science, P. Druschel, F. Kaashoek, and A. Rowstron, Eds. Berlin, Heidelberg: Springer, 2002, pp. 251–260.

[55] X. Yang, Y. Chen, and X. Chen, "Effective Scheme against 51% Attack on Proof-of-Work Blockchain with History Weighted Information," in *2019 IEEE International Conference on Blockchain (Blockchain)*, Jul. 2019, pp. 261–265.

[56] R. Anderson and R. Needham, "Programming Satan's computer," in *Computer Science Today*, G. Goos, J. Hartmanis, J. van Leeuwen, and J. van Leeuwen, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, vol. 1000, pp. 426–440.

[57] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," *Journal of the ACM*, vol. 56, no. 6, pp. 34:1–34:40, Sep. 2009.

[58] C. Gentry, "A fully homomorphic encryption scheme," Ph.D., Stanford University, United States – California, 2009.

[59] J. Hoffstein, J. Pipher, and J. H. Silverman, "NTRU: A ring-based public key cryptosystem," in *Algorithmic Number Theory*, ser. Lecture Notes in Computer Science, J. P. Buhler, Ed. Berlin, Heidelberg: Springer, 1998, pp. 267–288.

[60] L. D. Feo, D. Jao, and J. Plût, "Towards quantum-resistant cryptosystems from super-

singular elliptic curve isogenies," Tech. Rep. 506, 2011.

[61] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in *Proceedings 35th Annual Symposium on Foundations of Computer Science*, Nov. 1994, pp. 124–134.

[62] "Most Widely Deployed SQL Database Engine," https://sqlite.org/mostdeployed.html.

[63] P. Maymounkov and D. Mazières, "Kademlia: A Peer-to-peer Information System Based on the XOR Metric," Tech. Rep., 2002.

[64] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishna, "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications," Tech. Rep.

[65] Smartmatic, "TIVI Whitepaper," https://www.smartmatic.com/fileadmin/user_upload/Smartmatic_Whi

[66] J. Camenisch and M. Stadler, "Proof Systems for General Statements about Discrete Logarithms," p. 13.

Appendices

A. Data types in the Astris Protocol

This is a list of the data types used for the block payloads in the Astris system. Some nested structures are broken out to minimize repetition.

All strings in this specification are defined as UTF-8 encoded text, including numeric representations.

Note that in all the serialized representations, integers marked as `BigInt` are arbitrary size and there will be serialized to strings as the base64url (RFC4648[i]) encoding with padding characters stripped. The integers should be treated as unsigned and converted to big-endian bytes before the encoding takes place.

We also list the construction of the signature messages where appropriate, but the full construction of a signature is defined with the `Signature` data type.

Payload hints: `kind`

```
    1 = PayloadElectionSetup
    2 = PayloadTrusteeShares
    3 = PayloadTrusteePublic
    4 = PayloadVoterRegistration
    5 = PayloadVoteCast
    6 = PayloadPartialTally
```

PayloadElectionSetup

```
// PayloadElectionSetup: kind=1
{
    protcolVersion:          "1.0"
    name:                    string
    blockDifficulty:         integer
    encryptionSharedParams:  ElGamalParameters
    trusteesRequired:        integer
    candidates:              Array<string>
    maxChoices:              integer
    trustees:                Array<TrusteeSetup> // index in here is trustee index (1-based)
    registrar:               RegistrarSetup
    timing:                  TimingInfo
}
// ElGamalParameters
{
    p: BigInt // The group prime,
    q: BigInt // The sub-group prime,
    g: BigInt // The generator for the group
}
```

[i] https://tools.ietf.org/html/rfc4648

```
// TrusteeSetup
{
    name:               string
    verificationKey:    PublicKey
    encryptionKey:      PublicKey
    encryptionProof:    ProofOfKnowledge
    publicExponents:    Array<BigInt>    // length = trusteesRequired + 1
    signature:          Signature
}
// RegistrarSetup
{
    name:               string
    verificationKey:    PublicKey
    registrationURL:    string
    signature:          Signature
}
// TimingInfo
{
    timeZone:               string      // IANA timezone specifier
    parameterConfirmation:  TimeBounds
    voterRegistration:      TimeBounds
    voteCasting:            TimeBounds
    tallyDecryption:        TimeBounds
}
// TimeBounds
{
    opens:  string // ISO8601 Timestamp with second precision
    closes: string // and no timezone information. e.g. `2020-01-13T18:00:00`
}
```

The signature for a `TrusteeSetup` is calculated with using a message created by concatenation of:

1. the literal string `"trustee:"`
2. the base10 representation of the trustee index (1-based)
3. the literal string `":"`
4. the `name` property
5. the literal string `":"`
6. the base16 (lowercased) representation of the encryption public key `Y` value
7. for each `publicExponent`:
    a) the literal string `":"`
    b) the base16 (lowercased) representation of the exponent

The signature for a `RegistrarSetup` is calculated with using a message created by concatenation of:

1. the literal string `"registrar:"`
2. the `name` property
3. the literal string `":"`

4. the `registrationURL`

PayloadTrusteeShares

```
// PayloadTrusteeShares
{
    trusteeIndex:   integer               // (1-based)
    shares:         Array<EncryptedShare> // length = number of trustees - 1
}
// EncryptedShare
{
    recipient:  integer     // the trusteeIndex this is for (1-based)
    point:      CipherText  // encrypted with recipient public encryption key
    signature:  Signature
}
```

The trustee shares for a trustee with index i out of N trustees should be an array of encrypted shares for each j where $j \neq i$ and $0 <= j <= N$. The signature on an encrypted share is constructed by the concatenation of:

1. the literal string `"share:"`
2. the base10 representation of the trustee index i (1-based)
3. the literal string `":"`
4. the base10 representation of the recipient trustee index j (1-based)
5. the literal string `":"`
6. the base16 (lowercased) representation of the point ciphertext `a` value
7. the literal string `":"`
8. the base16 (lowercased) representation of the point ciphertext `b` value

PayloadTrusteePublic

```
// PayloadTrusteePublic
{
    trusteeIndex:   integer
    shardKey:       PublicKey
    shardProof:     ProofOfKnowledge // of the PrivateKey paired to "shardKey"
    signature:      Signature
}
```

The signature on the trustee public data is constructed by the concatenation of:

1. the literal string `"shard:"`
2. the base10 representation of the trustee index (1-based)
3. the literal string `":"`
4. the base16 (lowercased) representation of the `shardKey` public key `Y` value

PayloadVoterRegistration

```
// PayloadVoterRegistration
{
    voterId             string    // only the registrar knows the mapping between ↩
        voterId and voter
    verificationKey:    PublicKey // voters own verificationPublic key
    registrarSig:       Signature // signed with the registrar's signing key
    voterSig:           Signature // signed with the voters signing key
}
```

The signature `registrarSig` on the voter registration object is produced by the Registrar. The message is constructed by concatenation of:

1. the literal string `"voter:r:"`
2. the base16 (lowercased) representation of the SHA256 sum over the `voterId`.
3. the literal string `":"`
4. the base16 (lowercased) representation of the `verificationKey` public key `y` value

The signature `voterSig` on the voter registration object is produced by the Voter, using the private key associated with the `verificationKey` in the object. The message is constructed by concatenation of:

1. the literal string `"voter:v:"`
2. the base16 (lowercased) representation of the SHA256 sum over the `voterId`.
3. the literal string `":"`
4. the base16 (lowercased) representation of the `registrarSig` `r` value

PayloadVoteCast

```
// PayloadVoteCast
{
    voterId:    string
    votes:      Array<CipherText>                  // length = number of candidates
    proofs:     Array<ZeroKnowledgeProofOneOrZero> // length = number of candidates
    proof:      ZeroKnowledgeProofOneOfX
    signature:  Signature
}
```

The `proofs` array contains ZKPs for each vote, to prove that the encrypted vote is either a 0 or 1.

The `proof` glszkp is a proof that the sum of ciphertexts contains a number between 0 and the configured `maxChoices` from the `PayloadElectionSetup`.

The signature is created with the voter's private key and the message is constructed by concatenation of:

1. the literal string `"ballot:"`
2. the base16 (lowercased) representation of the SHA256 sum over the `voterId`.
3. for each element of `votes`
   a) the literal string `":"`
   b) the base16 (lowercased) representation of the ciphertext's `a` value
   c) the literal string `"|"`
   d) the base16 (lowercased) representation of the ciphertext's `b` value

PayloadPartialTally

```
// PayloadPartialTally
{
    trusteeIndex:   integer
    tallies:        Array<CipherText>           // in candidate order
    decrypted:      Array<BigInt>               // in candidate order
    proofs:         Array<ZeroKnowledgeProof>   // in candidate order
    signature:      Signature
}
```

The `proofs` array contains ZKPs for each decryption, proving the correct partial application of the shard key of the trustee.

The signature is created with the trustee's private signing key and the message is constructed by concatenation of:

1. the literal string `"tally:"`
2. the base10 representation of the trustee index (1-based)
3. for each ciphertext of `tallies`
   a) the literal string `":"`
   b) the base16 (lowercased) representation of the `a` value
   c) the literal string `"|"`
   d) the base16 (lowercased) representation of the `b` value
4. for each element of `decrypted`
   a) the literal string `":"`
   b) the base16 (lowercased) representation of the value

Common Structures

```
// PublicKey
{ y: BigInt }
// SecretKey
{ x: BigInt }
// KeyPair
{
    pk: PublicKey
    sk: SecretKey
}
// CipherText
{
    a: BigInt // known as "alpha"
    b: BigInt // known as "beta"
}
// ProofOfKnowledge (of a SecretKey) / Signature (over a defined message)
// Both constructs share the same properties
{
    c: BigInt
    r: BigInt
}
// ZeroKnowledgeProof
{
    a: BigInt,
    b: BigInt,
    c: BigInt,
    r: BigInt,
}
// ZeroKnowledgeProofOneOrZero
// represented as a tuple of 2 proofs
[ZeroKnowledgeProof, ZeroKnowledgeProof]

// ZeroKnowledgeProofOneOfX
// represented as an X-tuple of proofs,
[ZeroKnowledgeProof, ..., ZeroKnowledgeProof] // length = X
```

B. Common Processes in the Astris Protocol

We note here that many cryptographic operations assume a random oracle. That is a function that given a sequence of bytes, produces a deterministic second sequence of bytes where there is no discernable relation between the two sequences. Cryptographic Hash functions fulfil this property with high confidence and therefore when a random oracle is required, we instead use a hash function. For Astris `1.0` the hash function used for all random oracle substitution is the SHA256 digest function.

Signature

Astris uses Schnorr signatures described fully in RFC8235[i]. We start with our system parameters p, q and g, the secret key a (with public key $A = g^a \bmod p$) and a random number $v \in_R \mathbb{Z}_q$. We then calculate $V = g^v \bmod p$. Now we create the message to be signed in a manner specific to the data we wish to sign, each data type with a signature has the details of how their message should be constructed. Assume that message is created as an array of bytes: m.

Now we create the commitment $c = H(V||A||m) \bmod q$ where H is the SHA256 hash function and the concatenation is done as follows:

1. Start with the string (UTF8 encoded): `"sig|"`.
2. Then append the base16 (lowercased) representation of V.
3. Append a pipe character `"|"`.
4. Then append the base16 (lowercased) representation of A.
5. Append a pipe character `"|"`.
6. Then append the raw bytes of the message m.

Now we calculate the response $r = v - ac \bmod q$. The signature is now the pair (c, r).

To verify the signature we recreate $V = g^r \times A^c \bmod p$, and therefore the expected challenge message $C = H(V||A||m) \bmod q$ in the same way as during creation. Then we check:

1. A is a valid public key for this system
2. $C \equiv c$ the challenge we created is the same as the given one.

Proof Of Knowledge of Secret Key

The Proof of Knowledge construct is a ZKP. In fact, it is exactly the same construct as the Signature. Consider that to create a valid signature, one must possess the secret key, so *any*

[i] `https://tools.ietf.org/html/rfc8235`

signature that is valid by definition proves the possession of the private key. So we perform the exact same process as the defined for Signature, with the fixed message of `"pok"`.

Zero Knowledge Proofs

We have two ZKPs that both rely on the same core process. We will describe it once and then describe the two distinct processes. The general method is specified in [66].

Firstly, we have the system parameters (g, p, q) and choose a random $w \in \mathbb{Z}_q$. Let us consider the following parameters we choose: $h \in \mathbb{Z}_p$ is public knowledge and $s \in \mathbb{Z}_q$ is the secret knowledge we wish to prove we know, but not reveal. Let $G = g^s$ and $H = h^s$ be public. Finally, we have a function F which produces a challenge $C \in \mathbb{Z}_q$ in a random oracle model. The exact specifics depend on the type of proof.

The prover calculates:

1. Commitment A: $t_1 = g^w$
2. Commitment B: $t_2 = h^w$
3. A challenge with: from $C = F()$, F may use any public parameters known so far.
4. A Response $R = (w - sC) \bmod q$

The proof is the tuple (t_1, t_2, C, R, G, H).

The verifier can now take the public information (g, p, q, h) and the proof $(t)_1, t_2, C, R, G, H)$ and check:

1. $g^R modp = (t_1 G^C) \bmod p$
2. $h^R modp = (t_2 H^C) \bmod p$
3. That the challenge C was created correctly from the data, which will be specific to the type of proof.

Zero Knowledge Proof of Correct Decryption

As we are using ElGamal over a finite field for encryption, we can prove that a plaintext comes from a ciphertext without revealing our secret key. Let us remember the encryption and decryption functions over the parameters (p, q, g).

To encrypt a message m with a public key y we need randomness $r \in_R \mathbb{Z}_q$. Then we calculate $a = g^r \bmod p$ and $b = mh^r \bmod p$. The ciphertext is (a, b)

To decrypt the ciphertext (a, b) with private key x we calculate $m = ba^{-x} \bmod p$.

The decryption function can be rearranged as $b/m = a^x$ and the public key is $y = g^x$. So we can create a ZKP with a simple challenge function over (t_1, t_2) that we have defined using our random oracle function over the message created by concatenation of:

1. the literal string `"zkp:dec:"`
2. the base16 (lowercased) representation of t_1
3. the literal string `"|"`
4. the base16 (lowercased) representation of t_2

The public parameters are $s = x$, $g = g$, $G = g^s = g^x = y$, $h = a$, $H = a^s = a^x = b/m$ (we have the ciphertext and plaintext as we have decrypted the value).

Zero Knowledge Proof of Encryption of One of a Set of Plaintexts

This one is more complex. We wish to prove that the ciphertext given is a valid encryption of one of a set of possible plaintexts. The possible texts are public.

In this situation we use the method for create a proof that we know one piece of knowledge from a group described in [66] to create a set of proofs over the possible plaintexts, where all but the *real* values are faked. We use the fake commitments from the fake proofs, along with the real commitment, to create an overall challenge value. We then subtract the fake challenge values from the overall challenge value to create the challenge for the real proof.

In this situation, the verifier cannot tell which proof is the real one, but can check that they all verify. They can also sum the challenge values and check that they match the calculated challenge value for a commitment over all the proofs. If so, then the plaintext must indeed be one of the given values.

The prover has the list of plaintexts $(m_1, m_2, ..., m_k)$, the ciphertext (a, b) and the randomness r used in the encryption. There is also a publicly known piece of metadata d (in our case a hash of the voter's ID in the system) to prevent re-use of the ZKP in a separate context.

For each plaintext in the list where i is the index:

If the plaintext m_i *isn't* the one encrypted, we create a fake proof. The fake proof calculates $b_i = b/m_i$ and two random values for the challenge and response $C \in_R \mathbb{Z}_q$ and $R \in_R \mathbb{Z}_q$. They then calculates the values for (t_1, t_2) (which is done the other way around in a real proof). $t_1 = g^R a^{-C}$ and $t_2 = y^R b_i^{-C}$.

If the plaintext m_i *is* the actual one encrypted, we create a real proof with ZKP parameters $s = r$, $h = y$, $g = g$. This implies that $G = g^r = a$ and $H = h^r = y^r = (g^x)^r = (g^r)^x = a^x = b/m_i$. As all of these values are known we can construct the ZKP commitments (t_1, t_2). To create the challenge for the real proof we combine *all* the commitments for all fake and real proofs in order by using the random oracle over a message constructed by concatenation of:

1. the literal string `"zkp:enc:"`
2. for each proof in order:
    a) the base16 (lowercased) representation of t_1
    b) the literal string `"|"`

c) the base16 (lowercased) representation of t_2

d) the literal string `":"`

3. the extra metadata passed in d as raw bytes

The final challenge value is the random oracle's response, minus each of the other proof's challenge values. This ensures the sum of the challenges is the overall challenge value.

The proof is then the list of tuples $(t_1, t_2, C, R)_1, (t_1, t_2, C, R)_2, ..., (t_1, t_2, C, R)_k$.

The verifier must verify the list of proofs, against the list of plaintexts, without doing the challenge check, but instead creating a sum of the challenges. Then the verifier should re-create the challenge using all the commitments in order and the known metadata d. The sum of the individual challenges should match the overall re-created challenge value.

Canonical JSON Encoding

In order to use JSON serialization as a precursor to hashing to generate the block hashes, we need to ensure that all clients will encode the data in exactly the same way. As such we define a canonical JSON encoding which is used to serialize objects into for hashing and storage on the chain. JSON is not a unique representation of any object, but we can make it so by tightening some loose constraints.

Whilst not directly related to canonical JSON, the *big* integers in this spec —for example those describing numbers in ciphertexts, signatures and keys— are not serialized as raw integers despite the JSON spec allowing it. As most JSON decoding implementations would likely not accept these number, we prefer a string representation. This is described in the preamble of Appendix A.

In order to create a *canonical* encoding of an object the following tweaks to the standard are made:

1. Objects will have their keys sorted lexically.
2. Insignificant whitespace will be removed, i.e. no spacing or indentation
3. No spurious Unicode replacements in strings —i.e. '`<`' ← '`\u003c`'— **except** for the `LINE SEPARATOR U+2028` and `PARAGRAPH SEPARATOR U+2029` which should be encoded as `\u2028` and `\u2029` respectively.
4. After the encoding, we add a single line feed character, `ASCII 0x0A`, as a convenience for viewing the data manually.

For a simple example, the object:

```
{    "k": "<v/>",
     "a": { "b": 1,
              "a": 2 },
     "0": ["c",
           "b",
           "a"
] }
```

Would have the canonical representation (note the trailing newline):

```
{"0":["c","b","a"],"a":{"a":2,"b":1},"k":"<v/>"}

```

For reference, the SHA256 hash of the Canonical JSON encoding of that object —including the trailing new line character— encoded as hex is:

```
"5a5f0c64d0dd15c9ed2e2baa994c36dac4ba55c5a64ffde7e984b35ff004a543"
```

C. Astris gRPC Service Definition

Astris gRPC specification, although part of the source code at `https://github.com/thechriswalker/go-astris`, is useful to include.

```
syntax = "proto3";
option go_package = "github.com/thechriswalker/go-astris/protocol";
package astris;

service AstrisV1 {
  rpc PeerExchange(Peer)      returns (stream Peer) {}
  rpc RecvBlocks(Empty)       returns (stream BlockHeader) {}
  rpc GetBlock(BlockID)       returns (FullBlock) {}
  rpc FromBlock(BlockID)      returns (stream FullBlock) {}
  rpc AtDepth(Depth)          returns (BlockID) {}
  rpc PublishBlock(FullBlock) returns (Acceptance) {}
}

message Empty {}

message Peer {
  string target = 1;
}

message Depth {
  fixed64 depth = 1;
}

message Acceptance {
  bool accepted = 1;
}

message BlockID {
  bytes id = 1;
}

message BlockHeader {
  bytes   prev_id = 1;
  bytes   payload_hash = 2;
  fixed32 timestamp = 3;
  fixed32 proof = 4;
  uint64  depth = 5;
  uint32  kind = 6;
}

message FullBlock {
  BlockHeader header = 1;
  bytes       payload = 2;
}
```

Some points to note in the specification.

1. `FromBlock` allows traversal from a block towards the end of the chain. The blocks themselves only have the hash of the previous block in them so iterating forward is more complex than iterating backwards. This method implies that implementations must be able to iterate over the chain in either direction.

2. `AtDepth` allows a node to check the block ID that another node has at a specific depth in the chain allowing efficient calculation of where the two chains might diverge.

D. The Astris Blockchain

The Astris Blockchain implementation is fairly simple. Each block's payload is an arbitrary set of bytes. These encode various data structures, but to the chain they are just bytes. The block header contains all the other metadata which can be used to validate the block. Let us use the gRPC Protocol Buffer message definition as that is the externally visible definition.

```
message BlockHeader {
    bytes   prev_id = 1;      // SHA256 hash of previous block
    bytes   payload_hash = 2; // SHA256 hash of the payload
    fixed32 timestamp = 3;    // unix timestamp of block creation
    fixed32 proof = 4;        // proof of work
    uint64  depth = 5;        // block height or chain depth
    uint32  kind = 6;         // type hint for the payload
}
```

The chain is maintained via `prev_id` which is the previous block's ID, except for the genesis block which has all 0 bytes here. The `payload_hash` is a convenience, so we don't need to hash the entire payload each time we want to check the block ID. The `timestamp` is a 32bit unsigned integer representing seconds since the Unix epoch (00:00:00 on the 1st January 1970). The `depth` field is 64bit unsigned integer representing the block depth of this block in the chain, again a convenience to assist with validation and ordering. The `proof` field is a 32bit unsigned integer which is the result of our proof-of-work scheme explained later. Finally, we have the `kind` field which holds a hint as to the type of payload in this block. This is purely a convenience for the decoder to know in advance which data structure ought to be contained in the payload.

The block ID is defined as the SHA256 hash of the concatenation of the previous block ID, the payload hash, the timestamp, the proof and the kind hint. The previous block ID and payload hash are bytes already, but the timestamp, proof and kind are 32bit unsigned integers and must be converted to bytes using big-endian encoding. Note that the depth is in the block ID — it is just metadata that could be calculated externally and attempts to place fake data in this field will be noticed swiftly so its integrity as part of the ID is superfluous.

In order to provide the proof of work, we define that the block ID must have n leading 0 bits. That is, when considered as a number it should be less than the *target* number. This is very straightforward to verify, but requires performing (on average) 2^n hashes before a valid value for the proof is found. The number of leading 0-bits is known as the *difficulty* and is defined as 32 for the genesis block —which is less susceptible to post-facto forgery as the block ID is published as the election identifier— and for every subsequent block by the value in the election setup data — i.e. it is under control of the election authority.

Implementations should always consider the deepest valid chain as the correct one, however we should take careful note when a large change takes place as it may indicate adversarial behaviour. As such implementations are encouraged to keep all valid chains in storage, only considering the longest as true. The chains will make up a directed acyclic graph of blocks,

with a single genesis and one longest branch. If more than one branch is equal longest, the implementation should pick the earliest final block as the true chain, but wait for new blocks to confirm which branch should be taken.