AnonVote: Secure Blockchain E-voting through Signature Dilution

This document is an anonymized version of a master dissertation that was published publicly under another name in June 2023.

The research reported in this dissertation served as the basis for the paper with the same name that is submitted to Usenix Security ’24 (fall deadline) for blind review.

This document is referenced anonymously as [4] in that paper such that the blind submission principle is not violated.

Permission is given by the authors to Usenix Security reviewers to consult this document for their reviewing.

In particular, this document backs up the following claims in the submitted paper:

- A full specification was made of the AnonVote protocol (Chapter 5) that is briefly mentioned in Section 4 of the submission.
- A lengthy security analysis was performed (Chapters 6-12) that is summarized in Section 5 of the submission.
- A theoretical performance analysis and simulated evaluations were performed (Chapter 13) that form the basis of the evaluation in Section 6 of the submission.

Please note that additional experiments and analysis have been performed in between the original publication of the dissertation in June and the submission to Usenix Security ’24 in October to collect stronger evidence for some of the claims in Section 6 of the submission.

The results of those additional experiments and analysis are reflected in Section 6 of the submission, but not in this anonymized document. Indeed, apart from the anonymization and the omission of boilerplate information (abstract, preface, permission of use, …), this document is a copy of the original dissertation. If the paper is accepted, reference [4] to this anonymous document will be replaced by a reference to the original dissertation.

The authors,

October 2023

Contents

Part I

E-Voting Introduction

Chapter 1

Introduction

1.1 Problem statement

Societies need leadership but since those who would lead are human like the rest of us, history has shown that time and time and again the authority to make political decisions devolves into tyranny. In the sixth century BC Athenian statesman Solon laid the groundwork for the earliest form of democracy, a system in which political decisions are ultimately made by society as a whole by means of elections. This solution however moves the problem of tyranny over to the election process itself: if humans cannot simply be trusted to make political decisions, then they also cannot simply be trusted to manage an election that determines those same political decisions. Having faith in an authority organizing an election and tallying votes without oversight is fundamentally no different from having faith in a dictator claiming to be benevolent.

Many practical procedures to mitigate this problem were developed over the centuries: voting booths and envelopes to keep votes anonymous, granting partisan inspectors the ability to witness the count, keeping ballots around so they can be recounted etc. These measures however still do not guarantee a fair election so with the advent of computers and modern cryptography, solutions are sought in e-voting. In recent years blockchain technology has developed, enabling decentralized storage of data that could help prevent fraud on the part of any election authority. This however begs the questions: how should an e-voting blockchain system be designed such that it can be considered secure and fair?

1.2 Objectives

Our primary aim is to develop an e-voting protocol that fulfils all the requirements imposed on electronic electoral systems. We thus begin with a literature study on what these requirements are, basing ourselves on earlier e-voting work but also on international legal documents from the United Nations, the Inter-Parliamentary Union, the European Union and the United States of America.

Our objective is to build our protocol around the central idea of Signature Dilution,

a means of generating anonymity which will be explained in Part 2 of this thesis. Since Signature Dilution is uniquely suited to blockchain technology, we aim to design our e-voting system as a blockchain. Our objective is to fully flesh out the Signature Dilution process in such a way that the different participants can coordinate it together while protecting their privacy. We also aim to maximize the opportunities provided to the electorate to take part in this Signature Dilution process.

We aim to implement the protocol as a client application and to implement extra functionality to simulate elections involving multiple clients. We use these simulations to measure the performance of the system under normal circumstances, as well as its robustness against attackers.

1.3 Structure

This thesis is divided into four parts, with this introduction being Chapter 1 of the first part. Chapter 2 describes related work. This includes e-voting systems such as mixnets and blockchains, but it also includes work describing the requirements an e-voting system should meet. These requirements can be found in earlier e-voting work but we also compare international legal documents to ensure e-voting requirements correspond to actual electoral standards.

Part 2 describes our system, AnonVote. It begins with a broad overview of the system and a brief description of each of its phases, along with an explanation of the central concept of Signature Dilution. Next we provide a detailed explanation of the protocol, and lastly an explanation of our implementation.

Part 3 is an evaluation of the system, which mostly takes the form of a theoretical analysis but also includes experiments. We begin with preliminaries describing the roles of the different participants in the system and a communication channel called social discourse, which we assume to exist. Next is a chapter analyzing the system in terms of the requirements we laid out in Section 2.1. We then have four chapters about issues of robustness, respectively Denial of Service attacks, Eclipse attacks, blockchain forks and remaining robustness issues. Next is a chapter about the three other recommendations laid out in Section 2.1 and this part ends with a chapter about performance evaluation, including experiments done with simulations of the system and a section about cost.

Part 4 contains the overall conclusion of the thesis, as well as a note on future work.

Chapter 2

Related work

Much has already been written on the subject of e-voting and we will discuss a selection of e-voting schemes listed in Table 2.1. Several trends are visible. A common building block of e-voting schemes is the *bulletin board*: a communication channel that anyone can read, no one can remove data from, and all active participants can add data to their own designated section [10]. There are also a number of schemes that can be categorized as *mixnets*, while others can be categorized as *blockchain-based* schemes, and yet others fall in neither category. Table 2.1 classifies the schemes we will discuss according to these two categories as well as an "other" category. We will first look at the necessary requirements an e-voting scheme must have and then cover individual schemes according to this categorization and evaluate them based on these requirements: unless otherwise mentioned, this evaluation represents our own view and our own conclusions.

Mixnets	Chaum [7], Prêt à Voter [31], Helios [2], VoteAgain[24]
Blockchains	Abhirama [1], zkHawk [4], Bulut [6], Cooley [8], U.S. [12], Govinda [16], Hanifatunnisa [18], Hjálmarsson [21], Koç [23], Mani [25], Patidar [29], Sadia [32], Shetty [34], Soud [35], Srivastava [36], Wang [37], PriScore [38], d-BAME [39]
Other	Pfitzmann [30], Cramer [10], Estonia [13], Switzerland [17]

Table 2.1: A categorization of the systems cited.

2.1 Requirements

Article 25 of the International Covenant on Civil and Political Rights (a U.N. treaty binding on 173 states) states [22]:

> "Every citizen shall have the right and the opportunity [...] [to] vote and to be elected at genuine periodic elections which shall be by universal and equal suffrage and shall be held by secret ballot, guaranteeing the free expression of the will of the electors".

The Inter-Parliamentary Union distilled seven Voting and Election rights from this principle [9]:

1. Every adult citizen has the right to vote in elections, on a non-discriminatory basis.
2. Every adult citizen has the right to access to an effective, impartial and non-discriminatory procedure for the registration of voters.
3. No eligible citizen shall be denied the right to vote or disqualified from registration as a voter, otherwise than in accordance with objectively verifiable criteria prescribed by law, and provided that such measures are consistent with the State's obligations under international law.
4. Every individual who is denied the right to vote or to be registered as a voter shall be entitled to appeal to a jurisdiction competent to review such decisions and to correct errors promptly and effectively.
5. Every voter has the right to equal and effective access to a polling station in order to exercise his or her right to vote.
6. Every voter is entitled to exercise his or her right equally with others and to have his or her vote accorded equivalent weight to that of others.
7. The right to vote in secret is absolute and shall not be restricted in any manner whatsoever.

It also gives the state the responsibility to "[ensure] the integrity of the ballot through appropriate measures to prevent multiple voting or voting by those not entitled thereto" and "the integrity of the process for counting votes".

The European Commission for Democracy through Law (the Venice Commission) has published guidelines stating [14]:

> "Electronic voting should be used only if it is safe and reliable; in particular, voters should be able to obtain a confirmation of their votes and to correct them, if necessary, respecting secret suffrage; the system must be transparent."

In other words: the system must not simply have integrity, but this integrity must also be verifiable by the voter. It elaborates on the term *secret suffrage*, which corresponds to the seventh Voting Right of the IPU:

> "For the voter, secrecy of voting is not only a right but also a duty, non-compliance with which must be punishable by disqualification of any ballot paper whose content is disclosed."

It appears to also cover voter coercion:

> "Voting must be individual. Family voting and any other form of control by one voter over the vote of another must be prohibited."

The Venice Commission also provides guidelines for the observation of elections:

> "Both national and international observers should be given the widest possible opportunity to participate in an election observation exercise. [...] Observation must not be confined to the election day itself, but must include the registration period of candidates and, if necessary, of electors, as well as the electoral campaign. It must make it possible to determine whether irregularities occurred before, during or after elections. It must always be possible during vote counting."

The Organization for Security and Cooperation in Europe's Election Observation Handbook lists the practical implications of international standards for democratic elections [33]:

1. Periodic elections
2. Genuine elections
3. Free elections
4. Fair elections
5. Universal suffrage
6. Equal suffrage
7. Voting by secret ballot
8. Honest counting and reporting of results

Most of these requirements correspond verbatim to requirements mentioned in the ICCPR, while the last requirement can be termed integrity cf. the IPU.

Similarly in the United States of America the Help America Vote Act requires that a voting system "permit the voter to verify (in a private and independent manner) the

votes selected by the voter on the ballot before the ballot is cast and counted" [20]. It also decrees voters should be able to "change the ballot or correct any error before the ballot is cast and counted". One noteworthy requirement is also that there be a "single, uniform, official, *centralized*, interactive computerized statewide voter registration list" (emphasis ours). While it may seem secure to centralize the list given that this would guarantee that the list is complete, this particular requirement may actually make the system less secure since it requires a trusted third party.

The literature on e-voting does not give us a uniform set of requirements an electronic election system must adhere to: some authors posit requirements that others do not, and some authors use the same requirement under a different name. Patidar et al. for example require that voters cannot vote for invalid candidates [29], but the Swiss system explicitly allows for write-ins in some cantons [17] and the system tested in Emery et al. is also assumed to allow for this [12]. Srivastava et al. require it to be public knowledge who did and did not vote in the election [36], but this would violate the Venice Commission guideline "The list of persons actually voting should not be published" [14]. In most cases *individual verifiability* is used to refer to the ability of individual voters to verify that their votes were counted, but Adida refers to this requirement as *ballot casting assurance* [2] while Hjalmarsson et al. refer to it as *transparency* [21]. *Privacy* generally refers to the inability of a third party to determine how a voter voted, but this is termed *anonymity* in Zaghloul et al. and in Sadia et al. [39][32], while Yang et al., Soud et al. and the Estonian government refer to it as *secrecy* [38][35][13]. The term also corresponds to *secret ballot/suffrage* in the ICCPR and Venice Commission guidelines, and the seventh voting right of the IPU [22][14][9].

Nonetheless there some are requirements that are listed by most authors that list requirements. All five of them correspond to requirements set out by international law and political science as explained above, and we will consider these to be the necessary requirements an e-voting scheme must adhere to:

1. **Verifiability**: This requirement is subdivided in three parts:
   (a) **Individual verifiability**: Voters must be able to verify that their votes are captured.
   (b) **Universal verifiability**: Anyone must be able to verify that all captured votes were counted.
   (c) **End-to-end verifiability**: Anyone must be able to verify the correct execution of any step in the process.
2. **Privacy**: A third party must not be able to determine how a voter voted.
3. **Eligibility**: A voter must be someone who is eligible to vote.
4. **Uniqueness**: Only one vote per voter may be tallied.

5. **Integrity**: Votes must be tallied according to their content, as chosen by the voter.

Both *verifiability* and *integrity* correspond to the ICCPR's call for "genuine" elections "guaranteeing the free expression of the will of the electors" [22]. Verifiability is also strongly related to the Venice Commission's guidelines on election observation [14] and the Help America Vote Act explicitly mentions voters should be able to "verify" their votes [20]. *Eligibility* corresponds to the first five voting rights of the IPU. *Uniqueness* corresponds to the term "equal suffrage" in the ICCPR and the sixth voting right of the IPU [9][22].

Several less common requirements are also of note, and we will consider them to be recommendations:

1. **Coercion resistance**: It should not be possible for a third party to coerce a voter into voting a certain way.

2. **Robustness**: The system must continue working as intended even when someone interacts with it in a way that does not conform to the intended usage of the system, or when someone neglects to carry out a duty imposed by the intended usage of the system.

3. **Fairness**: The running tally should not be available while voters are still able to vote.

4. **Distribution of trust**: The above requirements should not rely on a single trusted third party.

The requirement of *coercion resistance* is interesting since it primarily relies on *privacy*: it is not possible to coerce people to vote a certain way if you cannot know that they voted this way. Some authors however take this requirement much further: it should also not be possible for voters to prove to the coercer how they voted [24]. Adida clearly believes privacy is sufficient and it is voters' personal responsibility to not reveal who they voted for, since he adds a "Coerce Me!" button to his system in order to "make it clear that online voting is inherently coercible" [2]. The Venice Commission appears to agree on the matter of the voter's personal responsibility since it stipulates it should be punishable for voters to reveal their vote [14]. It is indeed true that, for voters to be incapable of proving to a third party how they vote, their physical environment must be controlled[1] (e.g. a voting booth where smartphones and cameras are confiscated upon entry) because otherwise this environment could include a coercer looking over their shoulder and making them vote right before the deadline. In the end coercion resistance is a matter of degree: privacy provides a decent amount of protection against coercion, but as we shall see, some e-voting schemes contain extra mechanisms to make coercion more difficult.

Another requirement of note is *robustness*: it is worth considering what the practical faults are an e-voting scheme should be robust against. A malicious party may carry out

[1] This also implies there must be a trusted third party who controls this environment.

a denial-of-service attack to interfere with the election. It is also possible that potential voters simply abandon their role at some point in the process (and thus do not vote), meaning the system should be robust against imperfect turnout. 100% turnout is unheard of in political elections, the Belgian federal election of 2019 for example had 88,70% turnout while Belgium is one of only 21 countries where voting is compulsory [3].[2]

2.2 Mixnets

Chaum introduced the concept of a mixnet for various purposes, including electronic elections [7]. A mix is a server that receives encrypted messages, decrypts them and outputs them in random order. Eavesdroppers can see who sent the encrypted message but the random order of the output messages ensures (if each message is sent the same number of times) that they cannot link the encrypted message to the decrypted message, ensuring privacy from the point of view of anyone but the mix. A mixnet is a network of mixes which allows senders to send messages through a cascade of mixes: the message is not just encrypted once for a single mix, but instead encrypted many times for many different mixes. It is then sent to these mixes in reverse order: the first mix will be able to decrypt the message to an intermediate message that can be sent to the second mix, which will be able to decrypt the intermediate message to a second intermediate message and so on. This means that, unless all mixes in the cascade conspire, the anonymity of the sender is protected.

For electronic elections a mixnet would be used to create a roster of digital pseudonyms, public keys held anonymously by voters. Voters send an application message (containing their public key) through the mixnet to an authority which decides whether the application should be accepted and then publishes the roster of accepted pseudonyms. Voters can only apply for one pseudonym since it would be publicly visible if they sent multiple applications through the mixnet.

A major drawback of this system is its lack of verifiability making it vulnerable to malicious mix servers: if a cascade of multiple mixes is used the privacy of the messages is not compromised, but there is no protection against mix servers that actually tamper with the messages themselves. A malicious mix server could, when included in a cascade, replace the encrypted pseudonym applications with new pseudonym applications that it has encrypted himself and thereby steal votes. This problem is generally addressed by zero-knowledge proofs, such as the zero-knowledge argument proposed by Bayer and Groth [5]. The system is not coercion-resistant beyond the level of privacy, this is not a design consideration for the author.

[2] A turnout of around 90% is typical of countries with compulsory voting, with the sole exception of North Korea where sanctions are severe and even in that case turnout in the 2019 parliamentary election was 99,99% instead of 100% [41]. Due to its lack of anonymity and single option on ballots, we will not consider the North Korean system as a model for an election protocol.

In AnonVote we will also create digital pseudonyms, but this will be done in a different manner.

2.2.1 Prêt à Voter

Ryan et al. designed the Prêt à Voter system that offers voters in a paper ballot election the opportunity to audit the system to see if their votes are correctly processed and tallied [31]. Voters receive a ballot with the candidates in a randomized order, associated spaces to mark their vote and an identifier of the ballot. This identifier is in fact the permutation of the candidates in encrypted form. Voters then choose to either audit the ballot itself or use it to vote. If they audit the ballot they will be given a new one afterwards. In the auditing process the ballot's identifier is processed by the election's mixnet to produce a list of candidates in a specific order that should correspond to the order on the ballot, if this is not the case the ballot is proven to be fraudulent. If they use the ballot to vote, they mark the candidate they are voting for in the designated space and then destroy the part of the ballot that explicitly states which candidate belongs to which space, thus making the vote encrypted. They submit the vote by scanning it in, but keep the physical ballot as a receipt. The vote is processed (along with all other votes) by the election's mixnet based on the ballot's identifier and the space marked to decrypt the permutation and this mixnet finally publishes a list of all decrypted votes. The voters (or anyone else) can publicly audit every mix in the cascade based on their ballot identifier, revealing which cyphertext was decrypted to which plaintext in that step in the process, but they cannot audit all steps in the cascade for a single vote since that would compromise its privacy.

The ability to audit paper ballots is an elegant way to protect privacy and even coercion resistance, but there are several drawbacks to the system. It is not clear for example what conclusion should be drawn when a ballot is audited and is found to be fraudulent: intuitively this would mean that the entire election is invalid, since this is the point of auditing, but that makes the election vulnerable to a type of denial-of-service attack where a fraudulent ballot is inserted and audited. It is tempting to say that a certain margin of error in terms of fraudulent ballot should be accepted, but this means honest voters must audit ballots more often and the process devolves into an auditing arms race between honest voters and attackers inserting fraudulent ballots. Another type of denial-of-service attack can be carried out by a fraudulent mix to hide the fact that it changed a ballot: it can audit the ballot in every other mix in the mixnet, making it impossible for anyone to audit the ballot in the mix where it was altered.

2.2.2 Helios Voting

Adida developed the popular Helios Voting system based on a mixnet and zero-knowledge proofs [2]. Voters communicate their vote to the Ballot Preparation System (BPS) which generates a random number and uses it to encrypt the vote, sending a hash of the vote back

to the voter. The voters can then choose to either audit the vote or seal it. If they audit it, the BPS sends them the random number and the cyphertext, so they can verify that this cyphertext and the hash are correct. They then have to start the process again and receive a new hash they can either audit or seal. If they choose to seal, the BPS discards the random number and after authentication by the voters, the vote cyphertext will be written to the system's bulletin board. The bulletin board is maintained by a trusted third party but it can be audited. When all the votes have been written to the bulletin board, they are processed in a mixnet based on the ElGamal cryptosystem that generates a shuffle of the votes and a number of shadow mixes [11]. Auditors are then given the opportunity to ask for the shadow mixes to be decrypted, which would give evidence that the shuffle itself were correct. If they do so, a new shuffle with shadow mixes must be generated. Only when auditors are convinced of the integrity of the mixnet, the actual shuffle itself is decrypted and its contents are tallied. The shadow mixes may no longer be audited at this point, since they would reveal the permutation of the shuffle and thus compromise the privacy of the voters.

The system is based on many trusted third parties but allows auditors to verify the behavior of all of them. This provides statistical evidence of the integrity of the election, but no irrefutable proof. No attempt is made to create coercion resistance since the author considers e-voting to be inherently vulnerable to coercion.

2.2.3 VoteAgain

VoteAgain was designed to mitigate the issue of coercion [24]. In the pre-election phase a Polling Authority randomly generates a voter identifier and an initial ballot index for each voter, but keeps these numbers a secret. In the election phase, voters cast a vote by first requesting an ephemeral voting token from the PA including both the voter identifier and the ballot index in encrypted form. The voters append their encrypted vote to this voting token, sign this message and send it to the bulletin board. A voter can do this multiple times, each time the Polling Authority increments the ballot index before encrypting it. This means that, after being coerced, voters can overwrite their coerced vote with a new vote. The Tally Server reads all the encrypted votes from the bulletin board and performs a cryptographic shuffle (with a zero-knowledge proof of its validity) of the ballots after which it decrypts all their voter identifiers and ballot indices, allowing the ballots to be sorted by voter identifier and all but the ballot with the highest index to be filtered out. After the shuffle coercers should be incapable of seeing which voter id belongs to the voter they have tried to coerce, but they could still try to force a voter to vote a specific number of times and use that number to unmask the voter after the shuffle. It is for this reason that the TS inserts a random number of dummy ballots, of a lower index, for each voter id before the shuffle. The encrypted, filtered votes are then sent to a mixnet of Trustees that decrypts the votes so they can be tallied.

Hindering coercers' ability to identify a voter by adding dummy votes is a very interesting idea, but it is questionable how effective this is. If the votes are not salted before encryption (which the authors do not mention), coercers may still create and identifiable

pattern of vote cyphertexts by coercing a voter into voting for different candidates in a specific order. Unsalted vote cyphertexts however are an even greater risk since they allow for brute force decryption by iterating over all candidates, so it should perhaps be assumed that salting is implied. But this allows coercers to identify their coerced votes based on cyphertext and therefore see if they have been overwritten by a later vote. In addition, if the Polling Authority and the Trustees all collude together, they can break the privacy of the voters.

2.3 Blockchain

A decentralized blockchain was introduced by Nakamoto for the purpose of financial transactions [27]. Data is aggregated in a block along with the hash of the previous block, resulting in a chain of blocks that makes it infeasible to alter data. This opens up the possibility of using a blockchain as a Bulletin Board for electronic voting schemes.

Many e-voting blockchains are designed as a simple tallying smart contract or dedicated blockchain with little to no consideration for either privacy or uniqueness of the vote [35][8][21][23][25][34][16][36][1][32][29][6]. Hanifatunnisa et al. only use e-voting for the tallying between districts, thereby circumventing the need for privacy and reducing the problem to simple sums [18]. Wang et al. make interesting claims about properties their system supposedly has, but it is not clear how their system makes good on these promises [37].

> We also use a blockchain to implement a Bulletin Board and other functionality in AnonVote.

2.3.1 U.S. Government

Emery et al. carried out a penetration test of an e-voting blockchain prototype developed by a U.S. government organization playing an important role in U.S. national elections [12]. The prototype was not published and the organization remains anonymous.[3] In the system an Election Officer maintains a database of all registered voters and creates an election by sending out e-mails to these voters. Voters authenticate with the Election Officer and send back a signed plaintext vote. The signature in question is not a cryptographic digital signature but a handwritten signature recorded on a touchscreen device such as a smartphone. The Election Officer manually compares the signature to an image of the voter's signature and if they match the Election Officer logs the vote (without the signature or any identifying information about the voter) in the blockchain.

[3]The prototype was to be used in a large-scale mock election in order to learn practical lessons about E-Voting. It was not used and was never intended to be used in a real U.S. election.

There is no privacy in this system from the point of view of the Election Officer. It is also unclear what the purpose of the handwritten signatures is: signatures are used for authentication but the voters have already authenticated when they send the vote and digital images are easy to copy. The system is not coercion-resistant.

2.3.2 PriScore

PriScore is a decentralized score voting system based on a blockchain and zero-knowledge proofs [38]. At registration the voters each generate a key pair, in addition to key pairs for each candidate (in other words, there is a key pair for each voter-candidate combination). The voting phase is split into two steps: Commit and Vote. The tallying phase is also split into two steps: Self-Tally and Tally. If all goes well (i.e. if every voter cooperates throughout the entire process) only the Vote and Self-Tally steps are actually needed. The vote consists of a ballot and a zero-knowledge proof of that ballot. The ballot consists a.o. of g^p, where g is a common base and p is the actual score the voter is voting for. In Self-Tally all the ballots are multiplied with each other, resulting in g^P where P is the sum of all p's and therefore the final tally, recovered by calculating the logarithm. If the ballots were to solely consist of g^p however this would compromise their privacy, so a mathematical trick is applied to mask the vote: the voter multiplies the ballot by the public keys of all voters with a lower index, to the power of the voter's own private key, and divides it by the public keys of all the voters with a higher index, again to the power of the voter's own private key. When the ballots are multiplied with each other these public keys all cancel each other out and only g^P is left. This system fails when one voter registers but declines to vote, so the Vote step is preceded by a Commit step. In this step the voter encrypt g^p by multiplying it with all the public keys of the other voters, to the power of a random number, and publishes this commitment in the blockchain. If the voter neglects to vote afterwards, the Tally step is executed: the other voters can cooperate to decrypt the vote from the commitment since they collectively have all the private keys corresponding to the public keys used to encrypt the commitment, and then they can add this vote to the tally.

The Commit and Tally steps make it possible to tally if one voter abandons the process but not if more than one voter does, so the system might not require a perfect turnout but instead a near-perfect turnout and is therefore not robust. The presence of a Commit step also opens up the possibility of voters abandoning the process before this step, which also requires the process to start over and thus does not solve anything. In the Vote step itself each voter does a linear amount of work in terms of the number of voters, which can be large in an election. It is also important to consider the size of the digital numbers used to represent the ballots: public and private keys must be significantly larger than regular integers and floating point numbers to provide security, and the ballots must be a product of a number of public keys that depends linearly on the number of voters but also on the value of the private key. Any loss in precision results in an inaccurate tally. Since voters are capable of divulging their private keys, the system is not coercion-resistant.

2.3.3 zkHawk

Banerjee designed an e-voting blockchain based on the zk-SNARK implementation of zCash [4]. Voting is done by having a voter transfer one coin to the candidate, which can be done anonymously in zCash. The final tally is done by an offchain smart contract named zkHawk that guarantees that the sum is correct.

Leveraging zk-SNARKs and zCash in order to provide privacy in e-voting is an intriguing idea, but a fundamental issue in the usage of zk-SNARKs is their reliance on a Structured Reference String which requires trusted setup, and therefore a trusted third party. As the author suggests, this can be mitigated using an updatable zk-SNARK: voters can "update" the SRS and thereby make themselves one of the trusted third parties who would have to be compromised in order to compromise the election. If all voters do this, the election is secure, but it is questionable how scalable this solution is.

2.3.4 d-BAME

Zaghloul et al. designed a blockchain-based system where electronic ballots are provided to voters by two third parties: the registrar and the moderator [39]. They cooperate by means of a protocol that guarantees that voter privacy is not compromised to each party by itself: only when the two parties collude will they be able to discover which individual corresponds to which vote. The registrar and the moderator must therefore be chosen in such a way that they would never collude, for example by choosing two opposing candidates in the election. Voters generate a key pair and register by providing their credentials and their public key to the registrar, which signs the public key and provides this signature as proof to the voters that they are registered. The registrar also secretly generates a set of ballots (as many as there are voters) that contain random nonces and some extra data. When the registration period is over, the registrar publishes all registered voter public keys in the blockchain, without any information that would reveal the identity of the voters corresponding to each public key. Voters who are disenfranchised in this step can prove this by publishing the signature of their public key that they received from the registrar. In the next step, voters pseudonymously request a ballot from the moderator by authenticating with their private key. The moderator in turn requests the ballot from the registrar by providing the public key, encrypted with a blinding factor b. The registrar chooses another blinding factor q and uses it to encrypt the public key even further as k, then it sends the ballot encrypted by k to the moderator, along with an encrypted version of q that can only be decrypted by the private key of the voter. The moderator therefore cannot decrypt the ballot, but can send the encrypted ballot to the voter along with b and the encrypted version of q, which allows the voter to decrypt the ballot by first decrypting q. The voter then appends the vote to the ballot and encrypts this string with the public keys of both the registrar and the moderator and publishes it in the blockchain. When all votes have been cast (as cyphertext) the registrar publishes all the ballots and then both the registrar and the moderator publish their private keys so the votes can be decrypted.

The system seems completely secure under the assumption that the registrar and the

moderator do not collude. One can wonder however to what extent this can be guaranteed: it is possible for one party to infiltrate the other and thus reveal voters' identities via espionage. It is also possible that two adversarial parties would collude in opposition to a third adversarial party, especially if there is a political culture of two entrenched parties (a "two-party system") that is being challenged by a newcomer. At the end of the day the fact that two parties are electoral enemies does not mean they do not have a shared interest in uncovering voters' identities. Voters can be coerced when they have received their ballots.

2.3.5 CoinJoin

Maxwell designed the CoinJoin transaction style to improve privacy within the context of BitCoin financial transactions [26]. In BitCoin a transaction consists of two parts: a set of inputs, each with their own value, and a set of outputs, each with their own value as well. The sum of all the values of the inputs must equal the sum of all the values of the outputs. This means that if we take multiple transactions and add the inputs together, this number will equal the sum of the outputs of all the transactions and therefore these transactions can be combined into a single transaction. At this point the only possible way of determining which inputs originally corresponded to which outputs is by comparing their monetary value, but this information is completely lost if the outputs all have the same value, so there is a degree of anonymity. Specifically an *anonymity set* is created, which is the set of all the entities that took part in this combined transaction and therefore the set of all possible candidate payers for any payment made.

This does not mean that CoinJoin transactions are completely secure: Goldfeder et al. have demonstrated a *cluster intersection attack* which is capable of uncovering the identity of the owner of a coin [15]. This attack relies on the fact that multiple coins can be owned by the same entity and therefore spent together. When this happens and it becomes public knowledge that these coins were owned by the same entity, their anonymity sets are reduced to the intersection of the original anonymity sets, which is likely to be very small and might even single out the owner.

> While CoinJoin is specifically a financial transaction style, we will employ a similar idea in Signature Dilution.

2.4 Other Systems

Several other systems have been proposed that do not fall neatly into categories such as mixnets or blockchains. Some of these systems have even been used in actual political elections.

2.4.1 Pfitzmann and Waidner

Pfitzmann and Waidner designed an election protocol based on pseudosignatures and a Byzantine agreement protocol [30]. In the first phase, all participants $\{P_1, ..., P_n\}$ make a reservation using the Dining Cryptographers protocol, where individual voters P_i broadcast a null message for almost all authentication IDs except for one, the authentication ID chosen by this voter, in which case they broadcasts their secret key M_i. The space of possible authentication IDs must be sparse enough to make collisions unlikely. When broadcasting however they obfuscate their key (or null message) by adding/subtracting secret keys $K_{i,j}$ that they have agreed upon with each other voter P_j in pairwise fashion: they add any secret key agreed upon with a voter of lower index, and subtract any secret key agreed upon with a voter of higher index. The result is a local sum O_i which they broadcast and from which the actual key cannot be reconstructed.

$$O_i = M_i + (K_{1,i} + ... + K_{i-1,i}) - (K_{i+1,i} + .. + K_{n,i})$$

However when all local sums are added together, each pairwise key $K_{i,j}$ is added exactly once and subtracted exactly once and therefore only the sum of the secret keys remains, which is assumed to be only one secret key M_i due to the aforementioned sparseness (or simply zero in case all voters broadcast a null message for this authentication ID).

$$S = O_1 + ... + O_n = M_1 + ... + M_n$$

The order of the authentication IDs defines an order for the voters since they each chose a different one, so after the first phase all voters knows their own position in the order (since they know which authentication ID they chose) but not the position of any other voter.

In phase 2 the same procedure is repeated for every voter in order, but the voters P_i whose turn it is withhold their local sum O_i and instead broadcast random data. The result is that the secret keys cannot be reconstructed from the publicly available information (because S is not public data), but in each round the voter P_i whose turn it is can privately reconstruct all the secret keys from the other voters (since this voter knows the missing local sum O_i and therefore S), without knowing which key belongs to which voter. The voters can then pseudosign their vote with all the secret keys they have each received and then they can broadcast their vote using an untraceable broadcast protocol that again relies on the DC protocol and pseudosignatures.

If there are n voters, then for every authentication ID $\frac{n \cdot (n-1)}{2}$ pairwise keys must be created, and this must be done for all m possible authentication IDs (where $m >> n$), and in the pseudosignature phase this is done for all n voters, resulting in a complexity of $O(n^3 \cdot m)$. This complexity is exacerbated by the fact that the protocol relies on completely honest voters, so in order to take attacks into account there is an ingenious system of traps and investigations that add even more complexity. Unsurprisingly, the authors consider their work to be "constructive proofs of existence rather than practically applicable protocols".

Since voters have the secret keys they use to pseudosign their vote, the system is not coercion-resistant.

2.4.2 Cramer et al.

Cramer et al. designed a multi-authority election protocol based on the ElGamal cryptosystem [10]. Voters publish their encrypted votes nonanonymously on the bulletin board and there is a set of authorities that cooperate to tally, which they will only do at a given time when the election is considered to be over. They cooperate by means of secret sharing: each vote is encrypted with a public key and can only be fully decrypted by a secret key composed of a subset of the individual secret keys of each authority. The fact that only a subset is needed and not the entire set of authorities makes the scheme more robust since it protects against malicious authorities that block the count (assuming there are not too many malicious authorities). However, the authorities do not actually fully decrypt each vote, but instead they make use of the homomorphic properties of the ElGamal cryptosystem to tally the vote within the space of the encryption, and then afterwards they cooperate to fully decrypt this tally [11]. The encrypted votes and decrypted tally are further guaranteed to be valid by means of zero-knowledge proof, usually requiring interaction with independent users ("verifiers"). Since such interactions are a scalability issue, they are instead replaced by a random oracle generating challenge messages. One issue with this protocol is however that a malicious authority, or rather a set of malicious authorities, might be inclined to do more than just stall the tally: voter privacy is protected by the goodwill of authorities that will only decrypt the tally and not the votes, but a large enough subset of authorities could in principle decrypt the votes and thus compromise voter privacy. The authorities can even do this in secret, since they can communicate and conspire with each other in secret and since decryption is possible with only a subset of the partial keys, affording the conspiring authorities plausible deniability. The system is also not coercion-resistant since voters can divulge the plaintext of their votes.

2.4.3 Estonia

Estonia introduced nationwide electronic voting, called i-voting, in 2005 [13]. In the system votes are processed by three government parties: the Collector, the Processor and the Tallier. Voters cooperate with the Collector to first encrypt their vote with the public key of the Tallier and then sign the encrypted vote with their own private key. Voters can do this multiple times during the week before election day, each time overwriting their previous vote, in an attempt to counter coercion. The Collector provides voters with QR codes they can use to request their vote during this stage, verifying that the vote was actually recorded. When the i-voting deadline passes the Collector forwards all these signed encrypted votes to the Processor, which checks all the digital signatures and nullifies any double votes. When the Processor has assembled its final list of signed encrypted votes it removes the signatures and any information identifying the voters and then sends these encrypted votes to the Tallier, which can decrypt and tally the votes.

The Estonian i-voting system, while towering above election proceedings in countries such as neighboring Russia, is not without its faults. The privacy of the voters relies on the assumption that the Collector, Processor and Tallier do not collude. The QR code used to verify that a vote was recorded is of limited use since it only audits the first step in the process: votes can still be removed by the Processor or the Tallier. To mitigate this risk the system introduces Auditors that can repeat the work of the Processor or the Tallier in order to verify that this was done correctly, but access to this role cannot be universal since that would compromise the privacy of the voters. Restricting it to only designated Auditors however reintroduces the problem of collusion. The idea that overwriting votes prevents coercion also does not hold water since the coercion can happen immediately before the deadline.

2.4.4 Switzerland

The Swiss state of Geneva introduced e-voting in 2003 but due to initial flaws completely overhauled it in 2017 [17]. Central to this new version is the role of a number of election authorities that generate the contents of voting cards and process the votes in a distributed manner using the ElGamal cryptosystem: the election authorities cannot decrypt votes by themselves and compromise privacy or integrity. Each election authority first generates a secret key share they then use to collectively generate the public key. They also collectively generate codes and credentials for each unique voting card that will be printed by the printing authority and handed to a voter. In the election phase the voters encrypt their vote with the election authorities' collective public key and generate a non-interactive zero-knowledge proof that this cyphertext contains a valid vote based on the voter's voting code. The election authorities respond by each sending shares of verification codes that the voter can combine to create the actual verification codes that must match the ones written on the voting card. The voter then generates a confirmation based on the confirmation code written on the card and sends this to the election authorities, proving that the vote corresponds to a valid voting card. When all the votes have been cast the election authorities use the homomorphic properties of the ElGamal cryptosystem to first calculate an encrypted tally and then cooperate to decrypt it.

While offering a high level of security, the system assumes no collusion between the election authorities: they can cooperate to decrypt individual ballots. Since voters can divulge the plaintext of their votes, the system is not coercion-resistant.

Part II

AnonVote Design & Implementation

Chapter 3

Protocol Overview

AnonVote relies on a Bulletin Board implemented as a blockchain [10][27]. This entails a peer-to-peer network of nodes that connect to each other as in Figure 3.1 to broadcast messages, including the blocks in the blockchain. All nodes in the network implement the same protocol, but during an election there is an election manager, who has special capabilities and responsibilities in the blockchain, and a number of eligible participants. The election manager is defined by a key pair, so it is possible for multiple nodes in the network to act as the same election manager if they share the keys.

Verifiable decentralized voting is implemented in quite a straightforward manner by having voters, who are eligible participants, publish their anonymously signed votes in the blockchain, eliminating the need for a trusted third party. The main novel idea is a Signature Dilution process that generates anonymity and therefore enables privacy of the vote, in a manner reminiscent of CoinJoin [26]. An election consists of five phases:

1. **Setup**: The election manager chooses a pair of asymmetric keys and initializes the blockchain.

2. **Registration**: Voters choose their initial pair of asymmetric keys and register them in the blockchain with the help of the election manager. There is no anonymity in this phase.

3. **Signature Dilution**: Registered voters work together to turn their nonanonymous keys into anonymous keys.

4. **Commitment**: Voters choose who they vote for and publish a hash of their vote.

5. **Voting**: Voters publish the actual readable vote corresponding to their hash from the Commitment phase.

We will discuss each of these phases briefly in order below to sketch the general idea of the protocol. The phases are revisited with all the details of the protocol in Section 4.

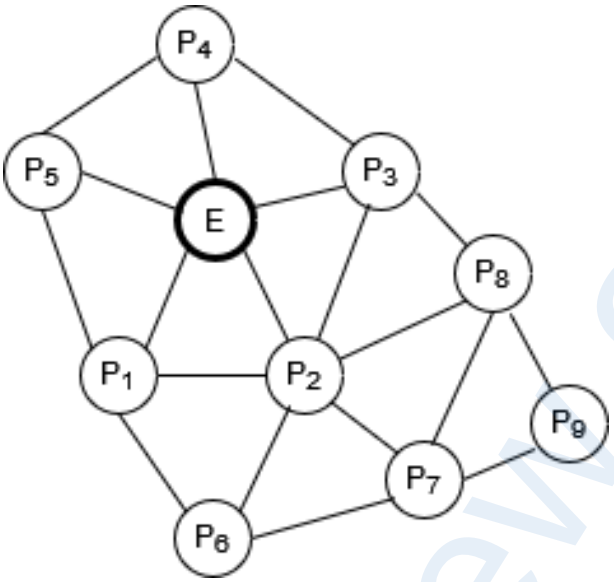

Figure 3.1: An illustration of the network used by AnonVote, consisting of an election manager E and several eligible participants $\{P_1, ..., P_9\}$ that connect to each other.

3.1 Setup

In this phase the *election manager* chooses the asymmetric keypair to be used for this election. The election manager either generates a new public and private key, or reuses one from a previous election. The election manager also generates a new 64-bit election ID, which will be used in conjunction with the public key to uniquely identify this election. Both the public key and the election ID will be included in all messages so they can be distinguished from other elections, so if the public key is being reused it is very important that the election manager choose a new election ID. This data is then included in a special *initialization block* which the election manager signs and publishes, starting the blockchain.

3.2 Registration

During Registration we do not yet worry about privacy, which makes the system quite straightforward. Voters individually generate an asymmetric key pair and then present themselves in person at the office of the election manager where they prove their identity. This can be done in any way the election manager sees fit, such as by means of a photo ID, a handwritten signature or biometric data. Voters hand over their public key and provide a guarantee that they will recognize this as their public key. This guarantee can take any form the election manager sees fit: it could be a simple declaration with a handwritten signature or it could be a video of the voter reading the key out loud, or something else entirely. The election manager then publishes the public key along with the identity of the voter in the election's bulletin board, completing the registration process.

This bulletin board is best implemented by using the blockchain that was initialized in the Setup phase so it can easily interface with the Signature Dilution phase. A *registration block* then simply consists of the public key and identity of one voter and it is validated by the signature of the election manager. Registration closes by means of a special *dilution start block*, also validated by the signature of the election manager. Nevertheless it may be decided that Registration shouldn't happen via a blockchain, for example due to HAVA's insistence on a *centralized* registration list, in which case the election manager can simply publish a single block including the hash of the registration list in this phase. In the remainder of this thesis we will however assume registration is implemented within the blockchain.

3.3 Signature Dilution

All the voters that were registered with a public key in the previous phase can now work together to generate anonymous public keys for themselves. They do this by means of a *signature dilution* process that will be explained here.

Let $V = \{v_1, v_2, ..., v_n\}$ be n eligible voters and let each voter v_i have an initial asymmetric key pair $(pk_{i,0}, sk_{i,0})$, with the public key $pk_{i,0}$ being publicly associated to v_i's identity. Let v_i also have another key pair $(pk_{i,1}, sk_{i,1})$, which we will refer to as the *diluted* key pair, with the public key $pk_{i,1}$ not being publicly known yet and therefore not associated with v_i. If v_i signs a message with the initial private key $sk_{i,0}$ it is proven that this message was signed by a voter but the voter is not anonymous. If v_i signs a message with the diluted private key $sk_{i,1}$ there is complete anonymity but there is also no proof the signer is one of the voters: any attacker could simply have generated a new key pair.

Let a_b be the *assembler* of a new block b, which we will refer to as a *dilution block*, in a process visualized in Figure 3.2. a_b might be one of the voters but this is not necessary. Voters v_i can anonymously send their diluted public keys $pk_{i,1}$ to a_b, who assembles the block as a list $b = \{pk_{\pi(1),1}, pk_{\pi(2),1}, ..., pk_{\pi(n),1}\}$ in a random order. a_b can now broadcast b and all voters v_i can verify that their diluted public key $pk_{i,1}$ is listed in b and if so, they sign it with their initial private key: $s_i = sign(sk_{i,0}, b)$. They then send this signature back to a_b who collects all n signatures and publicly broadcasts $b' = (b, \{s_1, s_2, ..., s_n\})$. The order of the signatures is different from the order of the diluted public keys: there is a random permutation π which is not known to a_b or anyone else. The signatures prove that each initial public key $pk_{i,0}$ is associated with at least one diluted public key $pk_{i,1}$ in the block so if the number of signatures is equal to the number of diluted public keys it is proven that each $pk_{i,1}$ corresponds to exactly one $pk_{i,0}$ and therefore to an eligible voter, but b' does not reveal *which* voter, preserving the voter's privacy. Each $pk_{i,1}$ is therefore proven to belong to exactly one member of the anonymity set V. We say that all voters have *diluted* their public key.

Signature dilution is similar to CoinJoin in the sense that there is a set of input-output relations which are obscured by mixing all the input together and mixing all the output together [26]. In CoinJoin this is done with financial transactions, where the inputs are

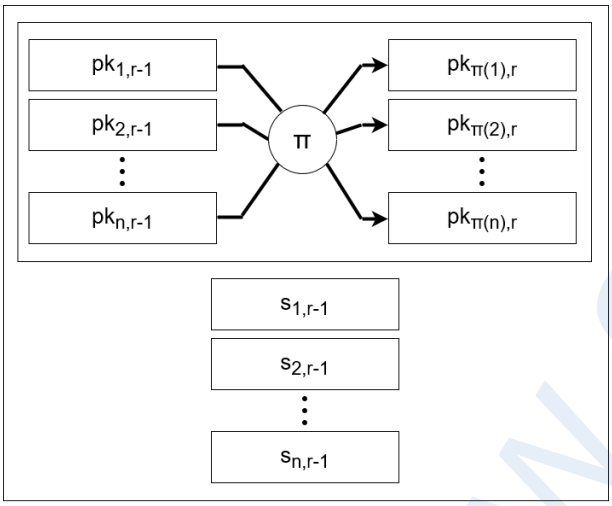

Figure 3.2: An illustration of the signature dilution process. The old keys $(pk_{1,r-1}, ..., pk_{n,r-1})$ are listed alongside the new keys $(pk_{\pi(1),r}, ..., pk_{\pi(n),r})$ and each old key corresponds to one new key by means of a secret permutation π. All this data is signed with the signatures $(s_{1,r-1}, ..., s_{n,r-1}$ of the old keys, proving that the owner of each old key is represented by one of the new keys.

addresses of entities spending money and the outputs are addresses of entities receiving the same money, while in signature dilution it is public keys that replace each other.

Public keys differ from transaction inputs and outputs in that they are indivisible and non-fungible, which means it is not necessary to agree upon a uniform output size. It also means that the number of input public keys must be equal to the number of output public keys (as opposed to the sum of all transaction input values and the sum of all transaction output values in CoinJoin), which guarantees that the number of output identities is equal to the number of input identities. Since each entity only has a single identity to dilute, the cluster intersection attack is not possible [15].

Another difference is the expectation that monetary transactions are part of a broader state change that is not entirely captured within the blockchain: money is paid in exchange for goods, and these goods are handed over in the real world. This is an important difference since it makes a cryptocurrency blockchain far more vulnerable to attacks by means of forking than a signature dilution blockchain: attackers can double-spend because they can undo half of the purchase, i.e. the transfer of money, while the other half, the transfer of goods, is not reversed. This does not happen in signature dilution since orphaning a dilution block reverses the entire state change and the affected parties can simply try again with a new dilution block.

Assembling a dilution block for all voters in an election might lead to scalability issues, so it is more feasible if the voters v_i first dilute their public keys $pk_{i,0}$ within a small group of size k, and therefore a small anonymity set, to validate $pk_{i,1}$. They can then dilute $pk_{i,1}$ once again within another group of k voters to validate a twice diluted public key $pk_{i,2}$, which now has an anonymity set of size k^2. If the voters v_i each take part in r dilution

blocks their keys' anonymity set grows exponentially as k^r.[1]

3.4 Commitment

Since privacy is already guaranteed by Signature Dilution, voting itself can be implemented in a very straightforward way. Most of the systems cited in Section 2 can be appended here, especially the ones that primarily lacked voter privacy. Nevertheless we will develop a Voting phase that combines well with the Signature Dilution phase and the blockchain we have been using so far, while also meeting the requirements sketched in Section 2. The Commitment phase is a preceding phase that serves to provide the recommendation of fairness.

In the simplest form of the Voting phase, voters v_i can sign their votes w_i with their (last) diluted private key $sk_{i,r}$, append the public key $pk_{i,r}$ and publish this message $(w_i, sign(sk_{i,r}, w_i), pk_{i,r})$ in the bulletin board, i.e. the blockchain we have been using so far. This however reveals the content of early votes before later votes have been cast, so we do not have fairness in this system. We therefore first let the voters publish a *commitment block*, which contains a signed hash of the vote: $(hash(w_i, z), sign(sk_{i,r}, hash(w_i, z)), pk_{i,r})$, where z is a random salt chosen by the voter. The block does not reveal the content of the vote, but it does lock the voter into it.

Double votes can be dealt with in multiple ways. It is possible to only count the first vote and therefore the first commitment block, in which case the blockchain may even be configured to reject commitment blocks with a signature that has already passed. Alternatively it is possible to only count the last vote to comply more fully with the guidelines of the Venice Commission and the Help America Vote Act [14][20]. In the remainder of this thesis we assume the latter option.

3.5 Voting

When the Commitment phase is over, voters broadcast their votes along with the salt they used and their public key: (w_i, z, pki, r). At this point it is no longer necessary to append this information to the blockchain: the candidates, or anyone else, can simply listen for votes being broadcast and include them in their tally. It is in the individual candidates' interest to include all the votes for themselves in the tally, so if they all listen and tally we can expect every vote to be counted.

The vote w_i itself can be any kind of data the election manager allows. If there is a fixed set of candidates or choices they can each be associated with a dedicated bit string, but it is also possible to define a format for a structured message for, e.g., score voting, or to allow for write-ins in ASCII format. In the remainder of this thesis we will assume a fixed set of candidates or choices.

[1] If there are eight voters in a block and the voters take part in five blocks, their keys will be diluted among 32768 voters, which is larger than the population of 509 of Belgium's 581 municipalities.

Chapter 4

Protocol Details

This section describes the five phases of the protocol in more detail. The Setup phase is covered in Section 4.1, the Registration phase in Section 4.2, the Dilution phase in Section 4.3, the Commitment phase in Section 4.4 and the Voting phase in Section 4.5. Section 4.6 describes the resolution of forks in the blockchain, which relies on the *chain score* of each block that is detailed in the following sections.

4.1 Setup

The election c is initiated by the *election manager* m_c, who must first generate an asymmetric keypair (pk_{m_c}, sk_{m_c}). This keypair can be reused by m_c between different elections, but for each election a new 64-bit identifier ID_c must be generated. The pair (pk_{m_c}, ID_c) uniquely identifies the election. To start the blockchain, m_c broadcasts an initialization block as shown in Figure 4.1, signing it with the private key sk_{m_c}. This ends the Setup phase and starts the Registration phase.

4.2 Registration

A voter v_i generates the initial keypair (pk_i, sk_i) and has it registered with a string ID_{v_i} of length 128 in ASCII format. m_c creates a registration block and signs it as shown in Figure 4.2.

The chain score of a registration block is one more than the chain score of its predecessor. If its predecessor is an initialization block, its chain score is assumed to be 0 and therefore the registration block will have a chain score of 1.

All voters are registered with registration blocks during this phase and they can even do so multiple times: the last registration overwrites the rest if ID_{v_i} is the same. This phase ends with a dilution start block shown in Figure 4.4.

	Description
Prefix	ASCII-encoded string “Initialization block - ”
pk_{m_c}	Public key of the election manager
ID_c	Index of the election
s_{m_c}	Signature of the election manager

Figure 4.1: Contents of an initialization block

	Description
Prefix	ASCII-encoded string “Registration block - ”
pk_{m_c}	Public key of the election manager
ID_c	Index of the election
ID_b	Index of the block within the blockchain
$hash(b_{ID_b-1})$	Hash of the previous block in the blockchain
C_b	The chain score at this block.
ID_{v_i}	Bytestring uniquely identifying the voter v_i
$pk_{i,0}$	The public key v_i registers
s_{m_c}	Signature of the election manager

Figure 4.2: Contents of a registration block

	Description
Prefix	ASCII-encoded string “2 disenfranchised - ”
Registration block	The registration block containing the disenfranchised voter

Figure 4.3: Contents of a disenfranchisement message

The chain score of a dilution start block is equal to the chain score of its predecessor. In the unlikely case that its predecessor is an initialization block, its chain score is assumed to be 0 and therefore the dilution start block's chain score is also 0.

If a registration block was orphaned due to a fork, it is the election manager's duty to append it later. If this does not happen and a voter is disenfranchised, this voter can use the data of the orphaned block to prove this. This happens by means of a disenfranchisement message, which is simply a prefix followed by the contents of the registration block as shown in Figure 4.3.

Since the election manager is the only one appending blocks in this phase (even though the manager might consist of multiple government employees using the same key), it should be possible to avoid forks with some coordination.

4.3 Signature Dilution

The Signature Dilution phase employs a number of different blocks and a greater number of auxiliary messages, all necessary to coordinate the process. This section explains the process and is divided into several smaller sections: Section 4.3.1 covers the dilution blocks themselves, Section 4.3.2 covers the concept of *maximum depth* and the associated *depth block* and *pre-depth block*, and Section 4.3.3 covers the *dilution pools* that are necessary to create a dilution block, and the way their membership is negotiated. Section 4.3.4 describes the *dilution end block* used to end this phase and list the candidates that can be voted for.

4.3.1 Dilution blocks

Signatures are diluted in a dilution block with a layout as in Figure 4.5. It contains n_b old public keys that must be diluted and therefore also n_b new public keys that are the result of this dilution. If the old public keys are listed in an order $\{1, ..., n_b\}$, the new public keys are listed as a permutation π of that order: $\{\pi(1), ..., \pi(n_b)\}$. π is not fully known to anyone, the voters v_i will each only be able to identify their own place $\pi(i)$ in the permutation. The block is signed with n_b signatures, each signature corresponding to an old public key and these signatures are listed in the same order $\{1, ..., n_b\}$.

	Description
Prefix	ASCII-encoded string "dilution start block - "
pk_{m_c}	Public key of the election manager
ID_c	Index of the election
ID_b	Index of the block within the blockchain
$hash(b_{ID_b-1})$	Hash of the previous block in the blockchain
C_b	The chain score at this block.
s_{m_c}	Signature of the election manager

Figure 4.4: Contents of a dilution start block

The chain score C_{b_j} of a dilution block depends on δ_{b_j}: the number of distinct anonymity sets being diluted in this block b_j. δ_{b_j} is added to the chain score of the previous block and then decremented:

$$C_{b_j} = C_{b_{j-1}} + \delta_{b_j} - 1$$

The chain score thus tends to increase with the length of the blockchain, but if the blocks do not dilute much the chain score only increases slowly. If all its public keys already belonged to the same anonymity set, the chain score does not increase at all.

The depth d_b of a dilution block is defined as such:

$$d_b = 1 + \max_{pk \in b} d_{b'} : \exists sk : (sk = pk^{-1} \wedge \exists s : s = sign(sk, b'))$$

It is one greater than the largest depth of one of the old public keys that are being diluted. The depth of each of these keys corresponds to the depth of either the dilution block they originated in, or zero if they originated in a registration block. This depth is important because it is not allowed to be too high, as will be explained in Section 4.3.2.

The anonymity set A_b is a number and it is also defined recursively:

$$A_b = \min_{pk \in b} A_{b'} : \exists sk : (sk = pk^{-1} \wedge \exists s : s = sign(sk, b'))$$

It is the smallest anonymity set number of each of the blocks that created a signature to be diluted in this block, where the anonymity set of a registration block is simply the public key being registered, interpreted as a number. Keeping track of the anonymity set

	Description	
Prefix	ASCII-encoded string "Dilution block - "	
pk_{m_c}	Public key of the election manager	
ID_c	Index of the election	
ID_b	Index of the block within the blockchain	
$hash(b_{ID_b-1})$	Hash of the previous block in the blockchain	
C_b	The chain score at this block.	
ID_P	The identifier of the pool that is being formed	
pk_{a_b}	Public key of block assembler a_b	
$ID_{b_{pka_b}}$	Index of the block in which a_b's public key originated	
d_b	Depth of the dilution block	
A_b	Anonymity set of the new public keys	
n_b	Number of public keys that are diluted	
$pk_{1,r-1}$	Old public key of voter v_1	Repeated n_b times
$ID_{b_{pk_{1,r-1}}}$	Index of the block in which v_1's old public key originated	
...		
$pk_{n_b,r-1}$	Old public key of voter v_{n_b}	
$ID_{b_{pk_{n_b,r-1}}}$	Index of the block in which v_{n_b}'s old public key originated	
$pk_{\pi(1),r}$	New public key of voter $v_\pi(1)$	Repeated n_b times
...		
$pk_{\pi(n_b),r}$	New public key of voter $v_{\pi(n_b)}$	
$s_{1,r-1}$	Signature of voter v_1 with the old private key $sk_{1,r-1}$	Repeated n_b times
...		
$s_{n_b,r-1}$	Signature of voter v_{n_b} with the old private key $sk_{n_b,r-1}$	

Figure 4.5: Contents of a dilution block

	Description
Prefix	ASCII-encoded string "1 Pre-depth block - "
pk_{m_c}	Public key of the election manager
ID_c	Index of the election
ID_b	Index of the block within the blockchain
$hash(b_{ID_b-1})$	Hash of the previous block in the blockchain
C_b	The chain score at this block.
s_{m_c}	Signature of the election manager

Figure 4.6: Contents of a pre-depth block

in this way allows users to see which blocks, and therefore which signatures, belong to the same anonymity set and so calculate the dilution factor in constant time.

4.3.2 Maximum depth

In order to avoid public keys with a higher depth being diluted alongside public keys with a lower depth (which would negatively impact the ideal exponential growth of anonymity sets), we maintain a maximum depth D_c of a blockchain. The depth of each block is not allowed to exceed this maximum depth, which is initially one. The election manager increments D_c periodically by means of a pre-depth block and a depth block with shown in Figures 4.6 and 4.7 respectively. The two layouts are identical apart from their prefix. In both blocks the chain score is 16 plus the chain score of its predecessor.

The reason there are two distinct blocks used for incrementing D_c is the presence of leftover voters: if voters are only allowed to dilute their signatures once before the next depth block, it is possible that some voters are unable to find anyone to dilute their signature with in this round. In theory this can only be a single voter, since multiple voters should still be able to dilute their signatures with each other, but in practice there might end up being multiple voters who have not diluted their keys. An intuitive way of solving this problem would be to divide the electorate beforehand into groups in such a way that no one is left out, but this would require everyone to participate and is therefore not robust. The risk of having leftover voters simply cannot be avoided, but fortunately for them there are multiple rounds and it is extremely unlikely that they will be leftover in other rounds as well.

	Description
Prefix	ASCII-encoded string “0 Depth block - ”
pk_{m_c}	Public key of the election manager
ID_c	Index of the election
ID_b	Index of the block within the blockchain
$hash(b_{ID_b-1})$	Hash of the previous block in the blockchain
C_b	The chain score at this block.
s_{m_c}	Signature of the election manager

Figure 4.7: Contents of a depth block

The pre-depth block however gives them an extra edge to allow them to avoid this situation. After the pre-depth block is published, blocks with a higher depth are allowed but only if there is at least one old public key with a lower depth in the block, giving preferential treatment to voters who were leftover in the previous round of Signature Dilution. When the depth block is published, this precondition is dropped and all blocks with a higher depth are allowed. The pre-depth blocks and depth blocks thus alternate in the signature dilution phase, with many dilution blocks in between. Other voters are incentivized to work together with these leftover voters because if they can dilute early, they know they won't be leftover themselves in this round. Leftover voters therefore have ample opportunity to avoid being leftover again, they need only take this opportunity.

4.3.3 Dilution Pools

It is possible to start the process of assembling a dilution block by having the voters simply broadcast their new public keys anonymously and wait until an assembler includes them. This can lead to many overlapping blocks however and when voters are included in multiple competing blocks, it is in their own best interest to only sign one of them since they would otherwise spoil the privacy the blocks are supposed to give them.[1] If all voters choose randomly though they are unlikely to choose blocks in a way that is completely compatible

[1]It does not matter that only one of the competing blocks will end up in the blockchain, since an attacker can still listen for signatures that are being broadcast even if those signatures will be discarded by the protocol.

	Description
Prefix	ASCII-encoded string "Application - "
pk_{m_c}	Public key of the election manager
ID_c	Index of the election
$pk_{i,r-1}$	Old public key of voter v_i
$ID_{b_{pk_{i,r-1}}}$	Index of the block in which v_i's old public key originated
$s_{i,r-1}$	Signature of voter v_i with the old private key $sk_{i,r-1}$

Figure 4.8: Contents of a dilution application

and network traffic will descend into chaos, particularly if attackers decide to exploit this weakness and use it in a denial-of-service attack.

It is therefore advisable for voters and block assemblers to first negotiate their membership in a *dilution pool* that will be used to assemble the block. There are eight types of messages used before the full dilution block itself can be broadcast:

1. **Dilution application message**: voters broadcast this message to signal their availability.
2. **Invite**: a_b sends this to each voter to start forming the dilution pool.
3. **Pool response**: voters send this message to a_b to signal that they chose this pool to participate in.
4. **Pool message**: a_b sends this to each voter to inform them of all the other voters and allow them to communicate anonymously within the pool.
5. **Pool acknowledgement**: voters acknowledge that the pool message was correct
6. **New key message**: voters send this message containing their new public key to a_b.
7. **Unvalidated dilution block message**: a_b assembles a dilution block without signatures and sends this to each voter to have it signed.
8. **Signature message**: voters send their signature for the dilution block to a_b.

	Description
Prefix	ASCII-encoded string “invite - ”
pk_{m_c}	Public key of the election manager
ID_c	Index of the election
$pk_{i,r-1}$	Old public key of voter v_i
$ID_{b_{pk_{i,r-1}}}$	Index of the block in which v_i's old public key originated
ID_P	The identifier of the pool that is being formed
$s_{DilutionApplication}$	Signature of the dilution application, listed as $s_{i,r-1}$ in its layout.
pk_{a_b}	Public key of block assembler a_b
$ID_{b_{pk_{a_b}}}$	Index of the block in which a_b's public key originated
s_{a_b}	Signature of assembler a_b with the private key sk_{a_b}

Figure 4.9: Contents of an invite

For their r-th dilution block, voters v_i first broadcast a *dilution application message* signed with their private key $sk_{i,r-1}$ as shown in Figure 4.8.

The block assembler a_b listens for these messages and chooses k voters to invite, a_b will prefer to choose them in such a way that the dilution factor will be high to make sure the whole process isn't in vain. a_b generates a *pool identifier* ID_P which, in combination with a_b's public key, uniquely identifies this pool. The assembler then sends a signed *invite* to v_i, the layout of which is shown in Figure 4.9.

If the voters v_i choose this dilution pool they each send a signed *pool response* back with a layout as shown in Figure 4.10.

If the assembler a_b has received enough responses to form a block, a_b chooses an asymmetric session keypair (pk_P, sk_P) and forms a *pool message* with a layout as in Figure 4.11. This message contains the session public key pk_P and a list of all the members of the pool. For each member v_i the session private key sk_P is also included, encrypted with v_i's old public key $pk_{i,r-1}$. The signature in the message is calculated over everything that comes before it.

The pool members each check if the encrypted session private key actually corresponds to the session public key. If so, each pool member v_i sends a *pool acknowledgement* (Figure

	Description
Prefix	ASCII-encoded string “response - ”
pk_{m_c}	Public key of the election manager
ID_c	Index of the election
pk_{a_b}	Public key of block assembler a_b
$ID_{b_{pk_{a_b}}}$	Index of the block in which a_b's public key originated
ID_P	The identifier of the pool that is being formed
$pk_{i,P}$	Pool public key of voter v_i
$s_{i,P}$	Signature of voter $v_{i,r-1}$ with the private key $sk_{i,r-1}$ for the pool public key $pk_{i,P}$
$s_{DilutionApplication}$	Signature of the dilution application, listed as $s_{i,r-1}$ in its layout.
s_{Invite}	Signature of the invite, listed as s_{a_b} in its layout.
$pk_{i,r-1}$	Old public key of voter v_i
$ID_{b_{pk_{i,r-1}}}$	Index of the block in which v_i's old public key originated
$s_{i,r-1}$	Signature of voter $v_{i,r-1}$ with the private key $sk_{i,r-1}$

Figure 4.10: Contents of a pool response

4.13) acknowledging the pool message. If not, v_i publishes the pool private key $sk_{i,P}$ in a *blame message* (Figure 4.14) with the associated pool message, proving a_b has acted in bad faith.

If all pool members have broadcast a pool acknowledgement however, they subsequently broadcast their new public key $pk_{i,r}$ signed with the session key in a *new key message* with a layout as in Figure 4.15.

After receiving all the new keys, a_b can start assembling the block b itself. Specifically a_b can create everything except for the signatures used to validate the block. a_b can then

	Description	
Prefix	ASCII-encoded string “Pool message - ”	
pk_{m_c}	Public key of the election manager	
ID_c	Index of the election	
pk_{a_b}	Public key of block assembler a_b	
$ID_{b_{pk_{a_b}}}$	Index of the block in which a_b's public key originated	
ID_P	The identifier of the pool that is being formed	
pk_P	Session public key of this pool	
n_b	Number of members in the pool, and therefore the number public keys that are diluted	
$Pool\ Member_1$	Pool member item of voter v_1, detailed in Figure 4.12	Repeated n_b times
...		
$Pool\ Member_{n_b}$	Pool member item of voter v_{n_b}, detailed in Figure 4.12	
s_{a_b}	Signature of the block assembler a_b with the private key sk_{a_b}	

Figure 4.11: Contents of a pool message

send an *unvalidated dilution block message* shown in Figure 4.16. The signature at the end is calculated over everything up to and including the unvalidated block itself.[2]

When v_i receives a valid unvalidated dilution block message with both v_i's old key and new key in it, v_i will sign the dilution block inside and broadcast it as a *signature message* with a layout as in Figure 4.17. The signature is calculated over the unvalidated dilution

[2]The signature is not strictly necessary from the perspective of the integrity of the signature dilution process, but without it attackers could mount a denial-of-service attack where they send a false unvalidated dilution block message (with some incorrect new keys) to a pool member to have it signed. The pool member will quickly discover the mistake when the real unvalidated dilution block message arrives, but at that point the new key must be discarded because signing two different blocks that both contain this new key would compromise its anonymity. As a result the entire pool fails and the process must start anew.

	Description
$pk_{1,r-1}$	Old public key of voter v_i
$ID_{b_{pk_{i,r-1}}}$	Index of the block in which v_i's old public key originated
$pk_{i,P}$	Pool public key of voter v_i
$s_{i,P}$	Signature of voter $v_{i,r-1}$ with the private key $sk_{i,r-1}$ for the pool public key $pk_{i,P}$
$s_{DilutionApplication,i}$	Signature of the dilution application of voter $v_{i,r-1}$, listed as $s_{i,r-1}$ in its layout.
$s_{Invite,i}$	Signature of the invite of voter $v_{i,r-1}$, listed as s_{a_b} in its layout.
$s_{PoolResponse,i}$	Signature of the pool response of voter $v_{i,r-1}$, listed as $s_{i,r-1}$ in its layout.
$E_{pk_{i,P}}(sk_P)$	Session private key, encrypted with v_i's pool public key

Figure 4.12: Contents of a pool member item in a pool message

	Description
Prefix	ASCII-encoded string "4 pool acknowledgement - "
i	Index of the pool member who acknowledges the pool
Pool message	Full contents of the pool message being acknowledged
$s_{i,r-1}$	Signature of the voter v_i with the private key $sk_{i,r-1}$

Figure 4.13: Contents of a pool acknowledgement

	Description
Prefix	ASCII-encoded string “Blame - ”
i	Index of the pool member who received a fake private key
$sk_{i,P}$	The pool private key of v_i
Pool message	Full contents of the fraudulent pool message

Figure 4.14: Contents of a blame message

	Description
Prefix	ASCII-encoded string “New key message - ”
$pk_{i,r}$	New public key of voter v_i
Pool Message	Full contents of the pool message of this pool
s_P	Signature with the session private key sk_P

Figure 4.15: Contents of a new key message

	Description
Prefix	ASCII-encoded string “Unvalidated - ”
Dilution block	Dilution block without signatures
s_{a_b}	Signature with the assembler’s private key sk_{a_b}

Figure 4.16: Contents of an unvalidated dilution block message

	Description
Prefix	ASCII-encoded string "Signature _ "
ID_P	The identifier of the pool that is being formed
pk_{a_b}	Public key of block assembler a_b
$ID_{b_{pk_{a_b}}}$	Index of the block in which a_b's public key originated
$pk_{i,r-1}$	Old public key of voter v_i
$ID_{b_{pk_{i,r-1}}}$	Index of the block in which v_i's old public key originated
$s_{i,r-1}$	Signature with the voter's private key $sk_{i,r-1}$
Dilution block	Dilution block without signatures
s_{Unv}	Signature of the unvalidated dilution block message with the assembler's private key sk_{a_b}, listed as s_{a_b} in it

Figure 4.17: Contents of a signature message

block inside the message.

When a_b has received signature messages from all members, a_b can publish the fully validated dilution block as given in Figure 4.5.

4.3.4 Dilution end block

The Signature Dilution phase is ended by a *dilution end block* as shown in Figure 4.18. It contains a *candidates* formatted message that specifies exactly what votes are allowed in the following two phases. The chain score is 32 more than the chain score of the preceding block: it must be significantly higher than the chain score of a dilution block in order to allow the election manager to be authoritative in ending the Signature Dilution phase, but at the same time it should not be so high that it could enable the election manager to undo a significant amount of signature dilution.

	Description
Prefix	ASCII-encoded string “end block _ ”
pk_{m_c}	Public key of the election manager
ID_c	Index of the election
ID_b	Index of the block within the blockchain
$hash(b_{ID_b-1})$	Hash of the previous block in the blockchain
C_b	The chain score at this block.
Candidates	A formatted message detailing the possible voting options for this election.
s_{m_c}	Signature of the election manager

Figure 4.18: Contents of a dilution end block

4.4 Commitment

In this phase voters v_i commit their votes with a *commitment block*, the layout of which is given in Figure 4.20. It contains a salted hash of the vote: due to the size of the salt z it is not feasible to brute-force the vote itself. The vote content of this commitment block is locked in due to the hash and it is linked to the public key of the voter contained in the block as well, ensuring that only one voter per voter is tallied. Its chain score depends on whether there was already a commitment block for the same public key: if so, the chain score is equal to the chain score of the predecessor block; if not, the chain score is incremented.

The Commitment phase ends with a *commitment end block* as shown in Figure 4.20.

4.5 Voting

This phase does not add any new blocks to the blockchain. Voters simply broadcast a vote message (Figure 4.21) referencing the commitment block that was used for this voter’s final vote. The vote message also contains the voter’s public key and the content of the vote w and the salt z, allowing anyone to verify that this vote corresponds to the hash in the

	Description
Prefix	ASCII-encoded string "Commitment block - "
pk_{m_c}	Public key of the election manager
ID_c	Index of the election
ID_b	Index of the block within the blockchain
$hash(b_{ID_b-1})$	Hash of the previous block in the blockchain
C_b	The chain score at this block.
$hash(w, z)$	Salted hash of the vote w of the voter v
pk_v	Public key of the voter v
$ID_{b_{pk_v}}$	Index of the block in which v's public key originated
s_{pk_v}	Signature of the voter v

Figure 4.19: Contents of a commitment block

commitment block.

	Description
Prefix	ASCII-encoded string “3 Commitment end block - ”
pk_{m_c}	Public key of the election manager
ID_c	Index of the election
ID_b	Index of the block within the blockchain
$hash(b_{ID_b-1})$	Hash of the previous block in the blockchain
C_b	The chain score at this block.
s_{m_c}	Signature of the election manager

Figure 4.20: Contents of a commitment end block

	Description
Prefix	ASCII-encoded string “vote - ”
pk_{m_c}	Public key of the election manager
ID_c	Index of the election
w	Content of the vote
z	Salt
pk_v	Public key of the voter v
$ID_{b_{hash_{w,z}}}$	Index of the commitment block

Figure 4.21: Contents of a vote message

4.6 Forks

When a fork occurs in the blockchain it is resolved on the basis of three criteria. The criteria are applied in the order listed: the second criterion only applies if the first criterion is inconclusive and the third criterion only applies if the first two criteria are inconclusive. The criteria are:

1. The types of the two blocks that start each branch of the fork: in some cases this is enough to determine that one branch wins.

2. If the above was not enough, the chain scores of the two blocks that end each branch of the fork: the highest chain score wins.

3. If the chain scores are equal, there is a final criterion which again depends on the types of the two blocks that start each branch of the fork.

We will now discuss the first and third criteria in more detail, followed by a note on forks and commitment end blocks.

4.6.1 First criterion

Depending on the types of the two blocks that start each branch of the fork the protocol might immediately decide which branch wins, or it might be inconclusive and the second and possibly the third criterion have to be taken into account. Figure 4.22 shows which block loses to which other block according to the first criterion: the block at the base of each arrow loses and the block at the tip of each arrow wins. If no arrow exists between two blocks, the first criterion is inconclusive. The first criterion is always inconclusive if the blocks have the same type.

4.6.2 Third criterion

If the first criterion was inconclusive and the chain scores of both branches are equal, the fork is resolved according to a third criterion. If both blocks were commitment blocks, the branch with the highest length wins. If the chain lengths are equal however, or if the blocks were of the same type but not commitment blocks, the fork is resolved by calculating the *block priority* of each block b_j:

$$p_{b_j} = value_{b_j} - value_{b_{j-1}}$$

A value $value_{b_{j-1}}$ from the previous block, interpreted as a number, is subtracted from the corresponding value $value_{b_j}$ of the current block, again interpreted as a number, within a finite group. The block with the lowest priority wins. Performing the calculation within the finite group ensures that blocks with higher values are not stuck in livelock.

The specific value and finite group used depends on the block type:

- **Registration block:** The value is the voter ID ID_{v_b} in the finite group 2^{1024}. If the voter IDs are equal, the public key is used in the finite group 2^{512}.

- **Dilution block:** The value is the first public key to be diluted $pk_{1,b}$ in the finite group 2^{512}. If the two public keys compared are equal, the block with the lowest key origin index for the first public key $ID_{b_{pk_{1,b}}}$ wins. If those indices are equal too, the second public key of each block is used, and then its key origin block index, and then

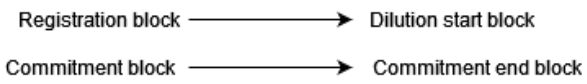

Figure 4.22: The blocks that lose and win a fork according to the first criterion.

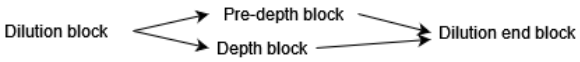

Figure 4.23: The blocks that lose and win a fork according to the third criterion.

the third etc. If all public keys of the block with the fewest are equal to the first public keys of the other block as well as their indices, the block with the most public keys wins. If all public keys to be diluted and their indices are equal and there is an equal number of public keys, the first new public key originating in the blocks is used, and then the second etc. If all new public keys are the same, as well as all keys to be diluted and their origin block indices, then the public key of the block assembler is used. Lastly, if the block assembler is the same as well as all the new keys and the old keys along with their origin indices, the pool identifier is used.

- **Dilution end block:** The value is the hash of the entire block $hash(b)$ in the finite group 2^{512}. If the hashes are equal, the smallest block wins. If the blocks are the same size, the value used for the block priority is the entire content of the block.
- **Commitment block:** The value is the public key pk_v of the voter v in the finite group 2^{512}. If the public keys are equal, the block with the lowest key origin index $ID_{b_{pk_v}}$ wins. If those indices are equal too, the value is the hash of the vote $hash(w, z)$.

For blocks with differing types but equal chain scores, forks are decided in the way shown in Figure 4.23. The block at the base of each arrow loses and the block at the tip of each arrow wins.

4.6.3 Forks & commitment end blocks

The election manager is not allowed to create two commitment end blocks for the same election (which would mean two branches of a fork both end in a commitment end block). It is also not allowed to create a commitment end block before the deadline of election day. If either of these things can be shown to have occurred, the election is invalid and the election manager is blamed for fraud.

Chapter 5

Protocol Implementation

This chapter details our implementation of the protocol and the client that uses it. Section 5.1 describes the client we have implemented, Section 5.2 lists cryptographic standards used, Section 5.3 describes several data structures and Section 5.4 details all the messages used by the protocol and their sizes. The code of our implementation can be found at source code url.

5.1 Client

All nodes in the network run the same client application, whether they function as election managers or eligible participants. We have implemented the client application in the Java language, using Bouncy Castle for all cryptography. Figure 5.1 shows an overview of an AnonVote client at a high level of abstraction. Similar to the Enterprise Ethereum client architecture, the AnonVote client consists of four layers: the Application Layer, the Privacy Layer (combining functionality of the Tooling and 3 P's layers of the Enterprise Ethereum client), the Blockchain Layer and the Network Layer [28]. Table 5.1 shows the number of source lines of Java code for each of the implemented layer classes. Figure 5.2 shows an example network with the layered structure of each client application. There is one election manager and there are three eligible participants, but they all run the same client application with the same layers.

Layer	SLoC count
Application Layer	299
Privacy Layer	502
Blockchain Layer	3773
Network Layer	230

Table 5.1: Java Source-Lines-of-Code count of the implemented layers.

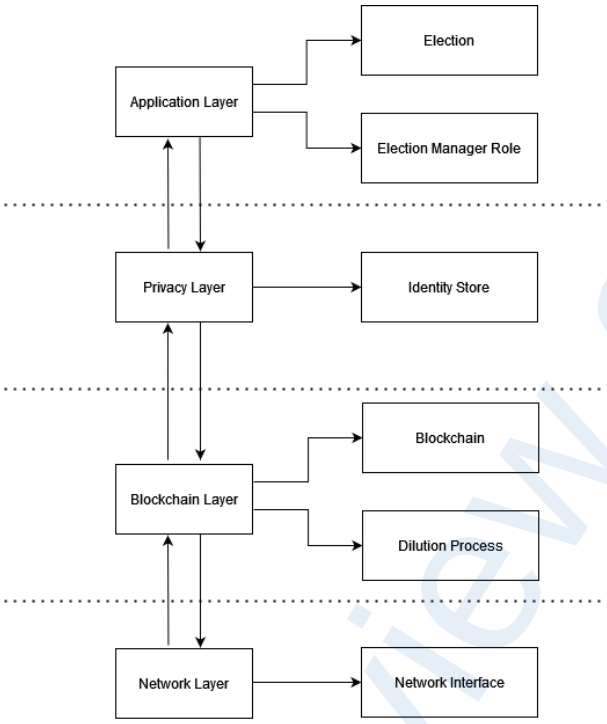

Figure 5.1: A high level overview of the layered structure of the implemented client.

5.1.1 Application Layer

Users interface with the Application Layer, which keeps track of the most abstract data of the elections they interface with: a set of Election objects for the elections they participate in, and a set of ElectionManagerRole objects for every key pair they can use in the capacity of an election manager. The key pairs used by voters do not exist in this layer, instead they are each simply represented by an identity index, an integer indexing the collection of key pairs this client has at its disposal.

5.1.2 Privacy Layer

The Application Layer interfaces with the Privacy Layer, which uses an IdentityStore object to keep track of the Identity objects for every voter key pair this client controls. Besides the public and private key, Identity object keep track of the block indices of the blocks where their public keys were introduced and the blocks where their public keys were invalidated.

This layer's primary purpose it to translate the indices of these Identity objects into their associated public keys and signatures. This layer also translates the Election objects from the Application Layer into the public keys and identifier bytestrings used by the protocol, and it captures vote messages from the Blockchain Layer and tallies the results of an Election object.

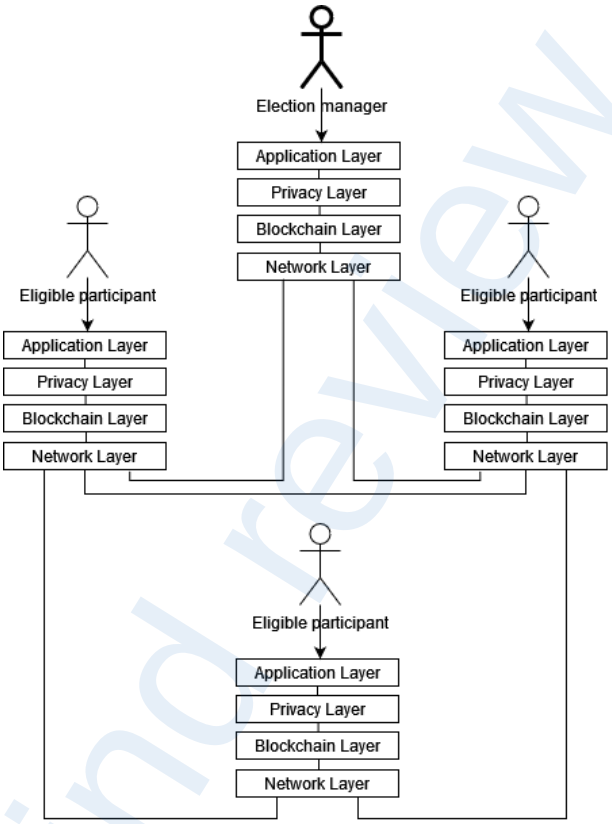

Figure 5.2: An example network consisting of an election manager and three eligible participants, shown with the four layers of the client. Each user interfaces with the Application Layer and the clients connect to each other via the Network Layer.

5.1.3 Blockchain Layer

The Privacy Layer interfaces with the Blockchain Layer, which primarily models the blockchain and the dilution processes. It uses Blockchain objects to keep track of the various election blockchains this client is currently following, each one defined by the public key of its election manager and its identifier bytestring.

For each blockchain, it also keeps track of the dilution processes that this client is a part of, the processes it is managing and a limited set of processes that it is simply observing in order to monitor other clients' behavior. If the layer has broadcast a dilution application, it times out after a while and then it sends another one. If the layer is attempting to start a dilution pool and has invited potential members, these also time out after a while and then the layer invites other potential members.

It does neither of these things indefinitely however: it randomly switches between broadcasting dilution applications and attempting to start dilution pools. Switching from broadcasting dilution applications to attempting to start dilution pools happens based on the *maximum switch counter from applications* s_a: if the layer has sent fewer than s_a dilution applications, there is a $\frac{1}{s_a}$ chance that it switches to attempting to start a dilution pool any time a dilution application times out. If the layer has sent s_a dilution applications, it automatically switches to attempting to start a dilution pool. A similar strategy is employed for switching from attempting to start a dilution pool to sending dilution applications, but this is based on the *maximum switch counter from pools* s_p.

In this layer all messages sent through the network are created, validated and processed. When blockchain forks arise, they are resolved in this layer using the associated Blockchain object.

5.1.4 Network Layer

The Blockchain Layer interfaces with the Network Layer, which interfaces with the network used to communicate the blockchain. This layer keeps track of adjacent nodes in the network by means of their NetworkInterface, identified by their IP address and port number.

The layer uses a flooding protocol to forward all the anonymous broadcast messages it receives, after first validating the messages by interfacing with the Blockchain Layer. Note that this means that, in our current implementation, the anonymous broadcast messages are not fully anonymous, since nodes can track from which side of the network the messages are coming. This can be mitigated by implementing onion routing, but we have not included this in the implementation since it is no longer new research.

The Network Layer also uses unicast messages to make and respond to requests of its adjacent nodes. This can be a request for the last block of a blockchain or the predecessor of a block. These messages are not flooded, the receiving node simply answers by sending the requested block.

5.2 Cryptography

We have implemented all the cryptographic functionality we use by means of the Bouncy Castle library. All asymmetric encryption is implemented using Elliptic Curve Cryptography with public keys of 64 bytes and private keys of 32 bytes, using the named curve secp256r1. Digital signatures are implemented using the Elliptic Curve Digital Signature Algorithm with the SHA256 hash function and public key encryption is implemented using the Elliptic Curve Integrated Encryption Suite.

Block hashes are calculated using the SHA512 hash function, but vote hashes are generated using the BCrypt password hash function with a salt of 16 bytes and a cost of 10. This is because the purpose of vote hashes is that they should not be feasible to crack by brute force.

5.3 Data structures

In order to efficiently access data in the Blockchain Layer, we have developed several data structures that enable fast lookup and LIFO buffer functionality.

5.3.1 Byte set

A byte set is a set of bytestrings. Since bytestrings are stored by reference in Java, a normal hash set cannot look up bytestrings by value in sublinear time: two bytestring objects with the same value still have a different reference and thus a different hash.

Byte sets do provide this functionality by storing the data in a tree structure: each node has a hash set with a limited size where bytestrings are initially stored. In this case the node is a leaf node of the tree. If this hash set overflows, the contents are split up on the basis of the byte at the index of the depth of this node in the tree structure. For each distinct byte, a new child node is created that receives all of the associated bytestrings. The original node is now no longer a leaf node, so the initial hash set is replaced by a hash map that matches bytes by value to child nodes.

5.3.2 Byte map

A byte map is a map in which the keys are bytestrings. It implements the same tree structure as byte sets to enable lookup by value in sublinear time, as opposed to linearly iterating over the keys of a hash map. The difference between a byte map and a byte set is that the leaf nodes in a byte map store a hash map with bytestrings as keys, as opposed to a hash set.

5.3.3 Circle

A circle combines the functionality of a LIFO buffer with fast lookup and removal by content. It is implemented as a linked list with a maximum size: whenever the buffer

overflows, the first element is removed. To enable content-based lookup and removal, it also maintains a byte map mapping content to elements in the linked list. The first element can always be popped from the circle. Based on the type of content that is stored inside the circle, several variants exist with slightly different functionality: the hash circle, the blame circle, the blockchain circle, the dilution application circle and the background dilution process circle.

Hash circle

A hash circle keeps track of hashes of larger messages. The flooding protocol in the Network Layer uses this to store hashes of messages that passed recently, so it can avoid resending them ad infinitum through cycles in the network. The elements in the hash circle contain the hash as a bytestring of a fixed size.

Blame circle

A blame circle keeps track of public keys that look like they are sabotaging dilution processes, so nodes can avoid getting into a new dilution process with them because they fear it will be sabotaged as well. The elements in the blame circle contain the public key that is blamed as a bytestring, as well as the index of its origin block and the pool identifier of the sabotaged pool as a bytestring. The blame circle provides functionality to count the number of elements for a given public key, which corresponds to the number of failing dilution pools this key has been a part of and thus the likelihood that this key was the culprit.

Blockchain circle

A blockchain circle is used for keeping track of alternative branches of the blockchain, so they can be used by the fork resolution protocol. Each element contains a reference to a blockchain and the content based lookup and removal works by means of the hash of the predecessor of the first block: when this predecessor block arrives at the Blockchain Layer, it can thus easily find the branch to prepend it to.

Dilution application circle

A dilution application circle keeps track of the dilution applications that were broadcast. Each element contains a reference to the dilution application and the content based lookup and removal works by means of the public key bytestring inside the dilution application. When popping an element from the dilution application circle, a blame circle can be given as a parameter to indicate which public keys to avoid. This means that the first element will not be popped if it is part of a large number of failing dilution pools, but instead a later element that is not part of a large number of failing dilution pools might be popped.

	Description	Size (B)
Prefix	ASCII-encoded string “Initialization block - ”	23
pk_{m_c}	Public key of the election manager	64
ID_c	Index of the election	8
$l_{s_{m_c}}$	Size in bytes of the following signature	1
s_{m_c}	Signature of the election manager	$l_{s_{m_c}}$

Figure 5.3: Layout of an initialization block

Background dilution process circle

A background dilution process circle keeps track of dilution processes that the node itself is not a part of. This way the node can monitor which dilution processes fail and thus which public keys were part of a failing dilution process, which will then be kept in a blame circle that is used to pop dilution applications from the dilution application circle.

Each element in the linked list contains a reference to a dilution process object and the content based lookup and removal is based on the public key of the block assembler of the dilution process and its pool identifier.

5.4 Blocks & messages

This section details the full layout of each block and each message used in the protocol along with the sizes of individual fields within the blocks and messages as well as the sizes of the blocks and messages themselves. Blocks and messages are listed in their own sections according to the phase to which they belong. Section 5.10 describes control messages necessary to enable communication between the nodes in the network.

5.5 Setup

To start the blockchain, m_c broadcasts an initialization block of roughly 167 bytes in the format shown in Figure 5.3, where the signature is calculated over the first 95 bytes.

	Description	Size (B)
Prefix	ASCII-encoded string “Registration block - ”	21
pk_{m_c}	Public key of the election manager	64
ID_c	Index of the election	8
ID_b	Index of the block within the blockchain	8
$hash(b_{ID_b-1})$	Hash of the previous block in the blockchain	64
C_b	The chain score at this block.	8
ID_{v_i}	Bytestring uniquely identifying the voter v_i	128
$pk_{i,0}$	The public key v_i registers	64
$l_{s_{m_c}}$	Size in bytes of the following signature	1
s_{m_c}	Signature of the election manager	$l_{s_{m_c}}$

Figure 5.4: Layout of a registration block

	Description	Size (B)
Prefix	ASCII-encoded string “2 disenfranchised - ”	20
Registration block	The registration block containing the disenfranchised voter	$366 + l_{s_{m_c}}$

Figure 5.5: Layout of a disenfranchisement message

5.6 Registration

Registration blocks used to register voter v_i and the initial keypair (pk_i, sk_i) consist of roughly 437 bytes and have a layout as shown in Figure 5.4. The signature is calculated over the first 365 bytes.

	Description	Size (B)
Prefix	ASCII-encoded string “dilution start block - ”	23
pk_{m_c}	Public key of the election manager	64
ID_c	Index of the election	8
ID_b	Index of the block within the blockchain	8
$hash(b_{ID_b-1})$	Hash of the previous block in the blockchain	64
C_b	The chain score at this block.	8
$l_{s_{m_c}}$	Size in bytes of the following signature	1
s_{m_c}	Signature of the election manager	$l_{s_{m_c}}$

Figure 5.6: Layout of a dilution start block

The Registration phase ends with a dilution start block with a layout as in Figure 5.6. It is 247 bytes in size and its signature is calculated over the first 175 bytes.

A voter whose registration block was orphaned can send a disenfranchisement message (Figure 5.5) of roughly 457 bytes.

5.7 Signature Dilution

This section details the implementation of all the blocks and messages sent during the Dilution phase.

5.7.1 Dilution blocks

Signatures are diluted in a dilution block with a layout as in Figure 5.7. It contains n_b old public keys that must be diluted and therefore also n_b new public keys that are the result of this dilution. If the old public keys are listed in an order $\{1, ..., n_b\}$, the new public keys are listed as a permutation π of that order: $\{\pi(1), ..., \pi(n_b)\}$. The block is signed with n_b signatures, each signature corresponding to an old public key and these signatures are listed in the same order $\{1, ..., n_b\}$. The block is roughly $315 + n_b \cdot 208$ bytes in size and the signatures are calculated over the first $315 + n_b \cdot 136$ bytes.

5.7.2 Maximum depth

The election manager increments D_c periodically by means of a pre-depth block and a depth block with layouts as in Figures 4.6 and 4.7 respectively. The two layouts are identical

	Description	Size (B)	
Prefix	ASCII-encoded string “Dilution block - ”	17	
pk_{m_c}	Public key of the election manager	64	
ID_c	Index of the election	8	
ID_b	Index of the block within the blockchain	8	
$hash(b_{ID_b-1})$	Hash of the previous block in the blockchain	64	
C_b	The chain score at this block.	8	
ID_P	The identifier of the pool that is being formed	8	
pk_{a_b}	Public key of block assembler a_b	64	
$ID_{b_{pka_b}}$	Index of the block in which a_b’s public key originated	8	
d_b	Depth of the dilution block	1	
A_b	Anonymity set of the new public keys	64	
n_b	Number of public keys that are diluted	1	
$pk_{1,r-1}$	Old public key of voter v_1	64	Repeated n_b times
$ID_{b_{pk_{1,r-1}}}$	Index of the block in which v_1’s old public key originated	8	
...			
$pk_{n_b,r-1}$	Old public key of voter v_{n_b}	64	
$ID_{b_{pk_{n_b,r-1}}}$	Index of the block in which v_{n_b}’s old public key originated	8	
$pk_{\pi(1),r}$	New public key of voter $v_\pi(1)$	64	Repeated n_b times
...			
$pk_{\pi(n_b),r}$	New public key of voter $v_{\pi(n_b)}$	64	
$l_{s_{1,r-1}}$	Size in bytes of the following signature	1	Repeated n_b times
$s_{1,r-1}$	Signature of voter v_1 with the old private key $sk_{1,r-1}$	$l_{s_{1,r-1}}$	
...			
$l_{s_{n_b,r-1}}$	Size in bytes of the following signature	1	
$s_{n_b,r-1}$	Signature of voter v_{n_b} with the old private key $sk_{n_b,r-1}$	$l_{s_{n_b,r-1}}$	

Figure 5.7: Layout of a dilution block

	Description	Size (B)
Prefix	ASCII-encoded string “1 Pre-depth block - ”	20
pk_{m_c}	Public key of the election manager	64
ID_c	Index of the election	8
ID_b	Index of the block within the blockchain	8
$hash(b_{ID_b-1})$	Hash of the previous block in the blockchain	64
C_b	The chain score at this block.	8
$l_{s_{m_c}}$	Size in bytes of the following signature	1
s_{m_c}	Signature of the election manager	$l_{s_{m_c}}$

Figure 5.8: Layout of a pre-depth block

	Description	Size (B)
Prefix	ASCII-encoded string “0 Depth block - ”	16
pk_{m_c}	Public key of the election manager	64
ID_c	Index of the election	8
ID_b	Index of the block within the blockchain	8
$hash(b_{ID_b-1})$	Hash of the previous block in the blockchain	64
C_b	The chain score at this block.	8
$l_{s_{m_c}}$	Size in bytes of the following signature	1
s_{m_c}	Signature of the election manager	$l_{s_{m_c}}$

Figure 5.9: Layout of a depth block

apart from their prefix, the pre-depth block is roughly 244 bytes in size and its signature is calculated over the first 174 bytes, while the depth block is roughly 240 bytes in size and its signature is calculated over the first 168 bytes.

5.7.3 Dilution Pools

The layout of a dilution application message is shown in Figure 5.10 and it consists of roughly 230 bytes, with the signature being calculated over the first 158 bytes.

	Description	Size (B)
Prefix	ASCII-encoded string “Application - ”	14
pk_{m_c}	Public key of the election manager	64
ID_c	Index of the election	8
$pk_{i,r-1}$	Old public key of voter v_i	64
$ID_{b_{pk_{i,r-1}}}$	Index of the block in which v_i’s old public key originated	8
$l_{s_{i,r-1}}$	Size in bytes of the following signature	1
$s_{i,r-1}$	Signature of voter v_i with the old private key $sk_{i,r-1}$	$l_{s_{i,r-1}}$

Figure 5.10: Layout of a dilution application

	Description	Size (B)
Prefix	ASCII-encoded string “invite - ”	9
pk_{m_c}	Public key of the election manager	64
ID_c	Index of the election	8
$pk_{i,r-1}$	Old public key of voter v_i	64
$ID_{b_{pk_{i,r-1}}}$	Index of the block in which v_i’s old public key originated	8
ID_P	The identifier of the pool that is being formed	8
$l_{s_{DilutionApplication}}$	Size in bytes of the following signature	1
$s_{DilutionApplication}$	Signature of the dilution application, listed as $s_{i,r-1}$ in its layout.	$l_{s_{DilutionApplication}}$
$l_{s_{a_b}}$	Size in bytes of the following signature	1
s_{a_b}	Signature of assembler a_b with the private key sk_{a_b}	$l_{s_{a_b}}$

Figure 5.11: Layout of an invite

An invite consists of roughly 305 bytes with a signature calculated over the first 161 bytes, its layout is shown in Figure 5.11.

If the voters v_i choose this dilution pool they each send a signed *pool response* back with a layout as shown in Figure 5.12. In order to do this, v_i needs to generate an ephemeral asymmetric keypair $(pk_{i,P}, sk_{i,P})$ and include $pk_{i,P}$ (v_i's pool public key) and a signature calculated over this new key with $pk_{i,r-1}$. This key will be used to encrypt the session key in the pool message. The pool response has a size of roughly 587 bytes and its signature is calculated over the first roughly 515 bytes.

After receiving enough responses, a_b forms a *pool message* with a layout as in Figure 4.11. The message contains roughly $304 + n_b \cdot 496$ bytes and its signature is calculated over everything that comes before that signature, roughly $232 + n_b \cdot 496$ bytes.

The pool members broadcast their new public key $pk_{i,r}$ signed with the session key in a *new key message* with a layout as in Figure 4.15. The message consists of roughly $458 + n_b \cdot 496$ bytes and the signature is calculated over the first $386 + n_b \cdot 496$ bytes.

After receiving all the new keys, a_b can create the dilution block except for the signatures used to validate it, so the first $315 + n_b \cdot 136$ bytes of the layout in Figure 5.7. a_b can then send an *unvalidated dilution block message*, the layout of which is given in Figure 4.16. The signature at the end is calculated over the first $329 + n_b \cdot 136$ bytes (everything up to and including the unvalidated block itself).

v_i signs the dilution block inside the unvalidated dilution block message and broadcasts it as a *signature message* with a layout as in Figure 5.19. It is $623 + n_b \cdot 136$ bytes in size and the signature is calculated over the unvalidated dilution block inside the message, which is the last $235 + n_b \cdot 136$ bytes.

5.7.4 Dilution end block

A *dilution end block* (Figure 5.20) contains a *candidates* formatted message that specifies exactly what votes are allowed in the following two phases. The chain score is 32 more than the chain score of the preceding block: it must be significantly higher than the chain score of a dilution block in order to allow the election manager to be authoritative in ending the Signature Dilution phase, but at the same time it should not be so high that it could enable the election manager to undo a significant amount of signature dilution.

5.8 Commitment

A commitment block (Figure 5.22) most importantly contains a hash of the vote w along with a salt z. It is important that w cannot be cracked by brute force, so we use the password hashing function Bcrypt with cost 10 and we generate a random 128-bit salt z that will not be published during this phase. The block contains roughly 339 bytes and the signature is calculated over the first 267 bytes.

	Description	Size (B)
Prefix	ASCII-encoded string "response - "	11
pk_{m_c}	Public key of the election manager	64
ID_c	Index of the election	8
pk_{a_b}	Public key of block assembler a_b	64
$ID_{b_{pk_{a_b}}}$	Index of the block in which a_b's public key originated	8
ID_P	The identifier of the pool that is being formed	8
$pk_{i,P}$	Pool public key of voter v_i	64
$l_{s_{i,P}}$	Size in bytes of the following signature	1
$s_{i,P}$	Signature of voter $v_{i,r-1}$ with the private key $sk_{i,r-1}$ for the pool public key $pk_{i,P}$	$l_{s_{i,P}}$
$l_{s_{DilutionApplication}}$	Size in bytes of the following signature	1
$s_{DilutionApplication}$	Signature of the dilution application, listed as $s_{i,r-1}$ in its layout.	$l_{s_{DilutionApplication}}$
$l_{s_{Invite}}$	Size in bytes of the following signature	1
s_{Invite}	Signature of the invite, listed as s_{a_b} in its layout.	$l_{s_{Invite}}$
$pk_{i,r-1}$	Old public key of voter v_i	64
$ID_{b_{pk_{i,r-1}}}$	Index of the block in which v_i's old public key originated	8
$l_{s_{i,r-1}}$	Size in bytes of the following signature	1
$s_{i,r-1}$	Signature of voter $v_{i,r-1}$ with the private key $sk_{i,r-1}$	$l_{s_{i,r-1}}$

Figure 5.12: Layout of a pool response

	Description	Size (B)	
Prefix	ASCII-encoded string “Pool message - ”	15	
pk_{m_c}	Public key of the election manager	64	
ID_c	Index of the election	8	
pk_{a_b}	Public key of block assembler a_b	64	
$ID_{b_{pka_b}}$	Index of the block in which a_b’s public key originated	8	
ID_P	The identifier of the pool that is being formed	8	
pk_P	Session public key of this pool	64	
n_b	Number of members in the pool, and therefore the number public keys that are diluted	1	
$Pool\ Member_1$	Pool member item of voter v_1, detailed in Figure 5.14	$l_{Pool\ Member_1}$	Repeated n_b times
...			
$Pool\ Member_{n_b}$	Pool member item of voter v_{n_b}, detailed in Figure 5.14	$l_{Pool\ Member_{n_b}}$	
$l_{s_{a_b}}$	Size in bytes of the following signature	1	
s_{a_b}	Signature of the block assembler a_b with the private key sk_{a_b}	$l_{s_{a_b}}$	

Figure 5.13: Layout of a pool message

5.9 Voting

A vote message (Figure 5.24) most importantly contains the vote content w and the salt z used to hash it in the Commitment phase, as well as the public key of the voter v and the index of the commitment block where the hash was committed. Its length in bytes is $168 + l_w$, the latter being the length of the vote content itself.

	Description	Size (B)
$pk_{i,r-1}$	Old public key of voter v_i	64
$ID_{b_{pk_{i,r-1}}}$	Index of the block in which v_i's old public key originated	8
$pk_{i,P}$	Pool public key of voter v_i	64
$l_{s_{i,P}}$	Size in bytes of the following signature	1
$s_{i,P}$	Signature of voter v_i with the private key $sk_{i,r-1}$ for the pool public key $pk_{i,P}$	$l_{s_{i,P}}$
$l_{s_{DilutionApplication,i}}$	Size in bytes of the following signature	1
$s_{DilutionApplication,i}$	Signature of the dilution application of voter v_i, listed as $s_{i,r-1}$ in its layout.	$l_{s_{DilutionApplication,i}}$
$l_{s_{Invite,i}}$	Size in bytes of the following signature	1
$s_{Invite,i}$	Signature of the invite of voter v_i, listed as s_{a_b} in its layout.	$l_{s_{Invite,i}}$
$l_{s_{PoolResponse,i}}$	Size in bytes of the following signature	1
$s_{PoolResponse,i}$	Signature of the pool response of voter v_i, listed as $s_{i,r-1}$ in its layout.	$l_{s_{PoolResponse,i}}$
$l_{E_{pk_{i,r-1}}(sk_P)}$	Length in bytes of the following encrypted session private key	1
$E_{pk_{i,P}}(sk_P)$	Session private key, encrypted with v_i's pool public key	$l_{E_{pk_{i,P}}(sk_P)}$

Figure 5.14: Layout of a pool member item in a pool message

	Description	Size (B)
Prefix	ASCII-encoded string “Blame - ”	8
i	Index of the pool member who received a fake private key	1
$sk_{i,P}$	The pool private key of v_i	32
Pool Message	Full contents of the fraudulent pool message	$l_{Pool\ Message}$

Figure 5.15: Layout of a blame message

	Description	Size (B)
Prefix	ASCII-encoded string “4 pool acknowledgement - ”	25
i	Index of the pool member who acknowledges the pool	1
Pool message	Full contents of the pool message being acknowledged	$l_{Pool\ Message}$
$s_{i,r-1}$	Signature of the voter v_i with the private key $sk_{i,r-1}$	$l_{s_{i,r-1}}$

Figure 5.16: Layout of a pool acknowledgement

	Description	Size (B)
Prefix	ASCII-encoded string “New key message - ”	18
$pk_{i,r}$	New public key of voter v_i	64
Pool Message	Full contents of the pool message of this pool	$l_{Pool\ Message}$
l_{s_P}	Size in bytes of the following signature	1
s_P	Signature with the session private key sk_P	l_{s_P}

Figure 5.17: Layout of a new key message

	Description	Size (B)
Prefix	ASCII-encoded string “Unvalidated - ”	14
Dilution block	Dilution block without signatures	$315 + n_b \cdot 136$
$l_{s_{a_b}}$	Size in bytes of the following signature	1
s_{a_b}	Signature with the assembler’s private key sk_{a_b}	$l_{s_{a_b}}$

Figure 5.18: Layout of an unvalidated dilution block message

	Description	Size (B)
Prefix	ASCII-encoded string “Signature - ”	12
ID_P	The identifier of the pool that is being formed	8
pk_{a_b}	Public key of block assembler a_b	64
$ID_{b_{pk_{a_b}}}$	Index of the block in which a_b’s public key originated	8
$pk_{i,r-1}$	Old public key of voter v_i	64
$ID_{b_{pk_{i,r-1}}}$	Index of the block in which v_i’s old public key originated	8
$l_{s_{i,r-1}}$	Size in bytes of the following signature	1
$s_{i,r-1}$	Signature with the voter’s private key $sk_{i,r-1}$	$l_{s_{i,r-1}}$
Dilution block	Dilution block without signatures	$315 + n_b \cdot 136$
s_{Unv}	Signature of the unvalidated dilution block message with the assembler’s private key sk_{a_b}, listed as s_{a_b} in it	$l_{s_{Unv}}$

Figure 5.19: Layout of a signature message

	Description	Size (B)
Prefix	ASCII-encoded string "end block _ "	12
pk_{m_c}	Public key of the election manager	64
ID_c	Index of the election	8
ID_b	Index of the block within the blockchain	8
$hash(b_{ID_b-1})$	Hash of the previous block in the blockchain	64
C_b	The chain score at this block.	8
Candidates	A formatted message detailing the possible voting options for this election.	$l_{Candidates}$
$l_{s_{m_c}}$	Length in bytes of the following signature	1
s_{m_c}	Signature of the election manager	$l_{s_{m_c}}$

Figure 5.20: Layout of a dilution end block

5.10 Control messages

Communication between the nodes in the network requires a number of control messages. When a node wants to begin tracking a blockchain, it first needs to receive the entire blockchain as it exists at that moment. The node sends out a *chain request* (Figure 5.25) containing the public key of the election manager and the index of the election. Any node that tracks this blockchain responds by sending the last block in the chain.

When a node receives a block for a blockchain that it is tracking, but it does not have the block's predecessor (either because the index of the block is higher than the length of the chain or because the hash of the predecessor according to the new block does not correspond to the hash of the predecessor the node has) it must request this predecessor from the node it received the current block from. This happens by means of a *block request* (Figure 5.26) that contains information about the blockchain and the block that is requested. The other node responds with a *requested block message* (Figure 5.27), which most importantly contains the contents of the block that was requested but also some details indicating that it was the response to a previous block request.

	Description	Size (B)	
l_t	Size in bytes of the tag of each candidate	1	
t_1	Tag of the first candidate	l_t	
$l_{Description_1}$	Length of the following description	1	
$Description_1$	Descriptive name of the first candidate in ASCII	$l_{Description_1}$	
$Electable_1$	Whether the candidate can be elected or not	1	Repeated m times
p_1	Index of the parent of the first candidate	4	
...			
t_m	Tag of the last candidate	l_t	
$l_{Description_m}$	Length of the following description	1	
$Description_m$	Descriptive name of the last candidate in ASCII	$l_{Description_m}$	
$Electable_m$	Whether the candidate can be elected or not	1	
p_m	Index of the parent of the last candidate	4	

Figure 5.21: Layout of the candidates message, part of a dilution end block

	Description	Size (B)
Prefix	ASCII-encoded string "Commitment block - "	19
pk_{m_c}	Public key of the election manager	64
ID_c	Index of the election	8
ID_b	Index of the block within the blockchain	8
$hash(b_{ID_b-1})$	Hash of the previous block in the blockchain	64
C_b	The chain score at this block.	8
$hash(w, z)$	Salted hash of the vote w of the voter v	24
pk_v	Public key of the voter v	64
$ID_{b_{pk_v}}$	Index of the block in which v's public key originated	8
$l_{s_{pk_v}}$	Size in bytes of the following signature	1
s_{pk_v}	Signature of the voter v	$l_{s_{pk_v}}$

Figure 5.22: Layout of a commitment block

	Description	Size (B)
Prefix	ASCII-encoded string “3 Commitment end block - ”	25
pk_{m_c}	Public key of the election manager	64
ID_c	Index of the election	8
ID_b	Index of the block within the blockchain	8
$hash(b_{ID_b-1})$	Hash of the previous block in the blockchain	64
C_b	The chain score at this block.	8
$l_{s_{m_c}}$	Size in bytes of the following signature	1
s_{m_c}	Signature of the election manager	$l_{s_{m_c}}$

Figure 5.23: Layout of a commitment end block

	Description	Size (B)
Prefix	ASCII-encoded string “vote - ”	7
pk_{m_c}	Public key of the election manager	64
ID_c	Index of the election	8
l_w	length of w	1
w	Content of the vote	l_w
z	Salt	16
pk_v	Public key of the voter v	64
$ID_{b_{hash_{w,z}}}$	Index of the commitment block	8

Figure 5.24: Layout of a vote message

	Description	Size (B)
Prefix	ASCII-encoded string “chain - ”	8
pk_{m_c}	Public key of the election manager	64
ID_c	Index of the election	8

Figure 5.25: Layout of a chain request

	Description	Size (B)
Prefix	ASCII-encoded string “block - ”	8
pk_{m_c}	Public key of the election manager	64
ID_c	Index of the election	8
ID_b	Index of the requested block b	8
$hash(b)$	Hash of the requested block b	64

Figure 5.26: Layout of a block request

	Description	Size (B)
Prefix	ASCII-encoded string “answer - ”	9
$hash(b)$	Hash of the requested block b	64
b	Full contents of the requested block b	l_b

Figure 5.27: Layout of a requested block message

Part III

Evaluation

Chapter 6

Evaluation preliminaries

This part provides for a theoretical evaluation of the protocol of AnonVote. It includes formal proofs as well as calculations of several aspects of the system. The current chapter outlines some preliminaries we will refer back to in the rest of the evaluation. Section 6.1 describes the roles of the different possible participants in the system, both honest and attackers. Section 6.2 describes a secondary communication channel that we assume to exist, called *social discourse*. It is necessary for some of the proofs to work and we consider it to be common sense. Chapter 7 evaluates the system according to the five requirements set out in Section 2.1. Sections 7.1 through 12.3 evaluate the system according to the five requirements and the four recommendations set out in Section 2.1. Section 13.2 calculates the size of the blockchain and the number of messages that need to be sent.

6.1 Participants

We refer to anyone involved in the system as a participant. Participants are further subdivided according to their capabilities: *eligible participants* are individuals with the right to vote, listed on the electoral roll, the *election manager* creates the election blockchain and fulfils the election manager role as described in the protocol, and an *outsider* is anyone who is neither an eligible participant nor an election manager. We also distinguish between *sensible* participants, who will never carry out actions that are obviously harmful to their goals, and *senseless* participants, who might carry out such actions. Finally we also make a distinction between *honest* participants, who intend to participate in a free and fair election, and *attackers*, who have any other goal. When discussing outsiders, we do not make the distinction between sensible and senseless outsiders since it is not meaningful in their case, and we only consider outside attackers: honest outsiders have nothing to do with the election so they will not interact with the system. The different roles are discussed in more detail below, and the specifics of honest sensible behavior are elaborated upon.

6.1.1 Honest sensible participant

Honest participants are people who interact with the blockchain with the intention of participating in an election and without the intention to violate any of the requirements or recommendations set out in Section 2. Honest sensible participants will not carry out actions that are obviously harmful to this goal. This has very specific implications based on the different capabilities of the participants.

Honest sensible election manager: An honest sensible election manager is the election manager of a blockchain who is attempting to hold a free and fair election, and thus the honest sensible election manager is also an honest sensible participant. The honest sensible election manager can consist of a group of people that are all attempting to hold a free and fair election. There is a public key associated with the honest sensible election manager and it is used in each message as part of the identification of the blockchain. All the members of this group possess the private key associated with this public key, but no one else does: sharing the private key with anyone else would obviously be harmful to the election manager's goals, so it is not sensible. When registering an eligible participant in a registration block, the honest sensible election manager obtains proof that the eligible participant chose to register the public key in it.

Honest sensible eligible participant: Honest sensible eligible participants are honest sensible participants who are eligible to vote. They each have a unique voter ID that is publicly linked to their identity. They will never carry out an action that obviously disenfranchises themselves since this would not be sensible. Honest sensible eligible participants will follow a number of steps in order, but they might abandon the process at any point and will in that case not carry out the last steps. Thus it is possible for an honest sensible eligible participant to carry out only the first two steps and then abandon the process, or alternatively to carry out steps 1, 2, 3, and 4 and then abandon the process, but it is not possible to carry out only steps 1 and 4 since this means skipping steps 2 and 3. These are the steps they carry out:

1. They create a key pair. They will keep both the public and the private key in their possession to the best of their abilities and they will not reveal their private key to anyone else unless coerced to do so.

2. They register the public key along with their voter ID with the election manager who creates a registration block containing this information in the blockchain. They keep a copy of the registration block and if they notice at the end of the Registration phase that it has been orphaned, they use it to broadcast a disenfranchisement message proving that they were disenfranchised.

3. During the Dilution phase, they dilute their keys at least twice. They keep the public and private keys resulting from each round of signature dilution in their possession to the best of their abilities and they will reveal neither to anyone else unless coerced

to do so. For each round of signature dilution they participate in, they first attempt to dilute their keys before the pre-depth block of that round is published and only when they fail to do so will they dilute their keys after the pre-depth block has been published as leftover voters. They attempt to have each dilution block they participate in remain a part of the winning branch of the blockchain and they attempt to be part of an anonymity set that is as large as possible given the number of rounds of signature dilution they participate in and the number of members in each dilution pool.

4. During the Commitment phase, they choose the candidate they wish to vote for and create a commitment block that is tied to their vote for this candidate, and append this block to the blockchain.

5. During the Voting phase, they broadcast the vote message associated with the commitment block they appended in the previous phase, so it can be included in the tally.

Honest sensible voter: Honest sensible voters are honest sensible eligible participants who vote, i.e., they carry out the same steps as all other honest sensible eligible participants and they do not abandon the process at any point.

6.1.2 Sensible attacker

Sensible attackers interact with the blockchain with the intention of violating some of the requirements and recommendations set out in Section 2, but they do their best to pretend to be honest participants and so will never carry out an action that reveals they are not.

Sensible attacking election manager: A sensible attacking election manager is the election manager of a blockchain and is masquerading as an honest sensible election manager but is secretly attempting to violate one or more of the requirements and recommendations. Sensible attacking election managers will never carry out an action revealing that they are not honest sensible election managers. They will therefore also obtain proof that an eligible participant chose to register a public key when creating the registration block containing that public key and the voter ID of that eligible participant.

Sensible attacking eligible participant: Sensible attacking eligible participants are people who are eligible to vote, like honest sensible eligible participants, and as such they also have a unique voter ID that is publicly linked to their identity. They do not necessarily want to follow the same steps as honest sensible eligible participants and instead attempt to violate one or more of the requirements and recommendations set out in Section 2. They will however never carry out an attack that enables the victim to publicly prove that they are guilty of this attack, where the proof actually reveals the identities of the attackers.

6.1.3 Senseless participants

Senseless participants could be honest and intend to participate in a free and fair election, or they could be attackers trying to violate the system's requirements, but their intentions do not really matter since they might carry out actions that are obviously harmful to their goals. It is easy to see that honest senseless participants are not protected by the system because they might for example use a public key that they do not have the private key to and therefore be unable to do anything else: participants are not protected from their own stupidity. We can categorize senseless participants according to their capabilities:

Senseless election manager: A senseless election manager is the election manager of a blockchain but might carry out actions that a sensible election manager would not. Since election managers that try their best to manage a free and fair election are honest sensible election managers, and election managers that do not do that behave like attackers, the main difference between a senseless election manager and a sensible attacking election manager is the latter's attempt to masquerade as an honest sensible election manager: senseless election manager can carry out an action that obviously sabotages the election.

Senseless eligible participant: Senseless eligible participants are on the electoral roll and can thus register a public key and subsequently dilute it, but they might register a public key without possessing the corresponding private key, or they might broadcast such a public key as a new key in the dilution process, or they might sign a dilution block that does not include any public key they have broadcast. They might also abuse the messages sent during the dilution process, causing it to fail.

6.1.4 Outside attacker:

Outside attackers are neither election managers nor eligible participants within the given election but they still attempt to violate the requirements and recommendations. Nevertheless they might try to masquerade as any participant in the process.

6.2 Social discourse

We assume that there exists a secondary communication channel that we refer to as *social discourse*. The global state of the election is relayed via this channel and all participants have the ability to both send and listen for messages on this channel. The global state includes the final tally when it becomes available. It also includes the phase the blockchain is currently in and this information is updated within this channel so often that all participants know the phase soon enough to be able to make full use of it. Social discourse can in practical reality take the form of media such as television or the world wide web, or word of mouth between people using the system.

Chapter 7

Requirement Validation

7.1 Verifiability

7.1.1 Individual verifiability

This requirement is defined as: Voters must be able to verify that their votes are captured. AnonVote completely satisfies this requirement.
Proof: A voter v_i casts a vote in the Commitment phase and the Voting phase. During the Commitment phase, v_i appends a commitment block b_i to the blockchain and this can be verified by simply looking at the blockchain. During the Voting phase, v_i broadcasts the vote w_i corresponding to b_i. If the vote is captured it is included in the tally. All tallies are published with all the vote messages tallied in them, therefore v_i can trivially verify that w_i is in this list and thus that it was captured.

7.1.2 Universal verifiability

This requirement is defined as: Anyone must be able to verify that all captured votes were counted. AnonVote completely satisfies this requirement.
Proof: With each tally, all the captured votes w counted are published. It is trivial to recalculate the tally and therefore verify that all captured votes have been counted.

7.1.3 End-to-end verifiability

This requirement is defined as: Anyone must be able to verify the correct execution of any step in the process. AnonVote satisfies this requirement when it comes to sensible participants. Senseless eligible participants can however disenfranchise themselves and transfer their vote to another person, and senseless election managers can openly sabotage the election, publicly showing that they are senseless.
Proof: If in each phase the correct execution of all steps can be verified by anyone, then the correct execution of any steps in the entire process can be verified by anyone. Therefore

we shall prove that the correct execution of all steps with respect to sensible participants in each phase can be verified by anyone.

Setup phase: The only thing that happens during this phase is the generation of an initialization block, which is correct if it was created by the election manager owning the key pair containing the public key listed in the block. The block is signed with a digital signature corresponding to this key pair and therefore the correctness of this phase can be verified by anyone by verifying the signature, assuming a sensible election manager who by definition knows the private key corresponding to the public key.

Registration phase: During this phase eligible participants are registered by means of *registration blocks*. This is correct if each registration block contains a voter ID belonging to a person eligible to vote and if its public key was chosen by this person for the purpose of including it in a registration block. The first requirement can be verified by looking up this voter ID in the electoral roll and the second requirement is verifiable if the election manager has obtained proof that the person eligible to vote chose to register the public key, which a sensible election manager will always do.

Senseless election managers might create a registration block containing the voter ID of an eligible participant who did not agree to this, but this participant can easily see this by looking at the blockchain and then challenge the election manager, who will be unable to prove that the eligible participant chose to register the key in this block. The election manager is then exposed as being senseless.

Senseless eligible participants might also register a public key without possessing the corresponding private key, with the result being that they disenfranchise themselves.
This phase ends with a *dilution start block*, which is correct if none of the participants registered have their registrations annulled. Registrations can only be annulled if the corresponding registration block is orphaned.

If this does happen, the sensible disenfranchised participants publish a *disenfranchisement message* containing the registration block. The registration block itself can be verified since it is signed by the election manager, and the only other requirement for the disenfranchisement to be correct is that the eligible participant was not registered with a different public key during the registration phase. It is trivial to verify this by looking at the blocks in the blockchain.
Besides the creation of registration blocks, dilution start blocks and disenfranchisement messages, nothing else happens in this phase. Therefore any step in this phase can be verified by anyone.

Dilution phase: During this phase eligible participants dilute their public keys in *dilution blocks*, meaning they repeatedly replace their old key pair with a new key pair. This is correct if for every dilution block and for each participant in it, there is exactly one old public key being invalidated and exactly one new public key introduced, and if the participant owns the key pairs of both the old and new public key. The dilution block is

signed with the signatures corresponding to the old public keys, so if the participant did not own the old public key it would not be computationally feasible for this participant to create this signature. If the participant did not own the key pair belonging to at least one of the new keys, signing the block would obviously disenfranchise this participant, which honest sensible eligible participants will never do by definition.

Senseless eligible participants can however disenfranchise themselves by signing a dilution block that does not contain a new public key corresponding to a private key they possess, allowing other people to insert their new keys into the block. Therefore each sensible participant taking part in this dilution process owns at least one of the new keys in the block, while senseless eligible participants can disenfranchise themselves and transfer their ability to vote to another person. Since the number of participants taking part in this dilution process equals the number of old keys which equals the number of new keys, this means each of the participants taking part in this dilution process owns exactly one of the new keys, unless there are senseless eligible participants in the dilution pool. Dilution blocks are therefore correct, with the exception that senseless eligible participants can disenfranchise themselves and transfer their ability to vote to another person, when the signatures are correct, and this is trivial to verify.
Depth blocks and *pre-depth blocks* do not change which keys belong to which participants, so these blocks are always correct as well.
The phase ends with a *dilution end block*, which does not in itself change which keys belong to which participants. It is still possible for a dilution end block to be appended in such a way that dilution blocks are orphaned, but since we have already proven that there is a one-on-one correspondence between the keys being invalidated and the new keys being introduced when all participants behave sensibly, orphaning a dilution block does not make the process incorrect.
Besides the dilution process creating dilution blocks, the creation of pre-depth blocks, depth blocks and the dilution end block, nothing else happens in this phase. Since we have shown that the correct execution of all four of these steps can be verified, the correct execution of the entire Dilution phase can be verified for sensible eligible participants, while senseless eligible participants can disenfranchise themselves and transfer their ability to vote to another person.

Commitment phase: During this phase eligible participants add commitment blocks to the blockchain in order to commit their votes. This is correct if each commitment block was created by the participant listed in it, and if the hash listed in it was also generated by this participant. Since the participant signs the block, it is trivial to verify that commitment blocks are correct. Nothing else happens during this phase so the correct execution of the Commitment phase can be verified.

Voting phase: During this phase voters send vote messages corresponding to the last commitment block they published in the blockchain during the Commitment phase. Such a vote message is correct if the vote and salt in it correspond to the hash in the commitment

block, which is trivial to verify, and if the participant of the commitment block did not create a new commitment block later, which is again trivial to verify. The vote messages are tallied and each tally is published alongside the set of votes it took into consideration. It is thus trivial to verify that the tally is correct by recalculating it. When two different tallies are published and they were both calculated correctly, this means the votes taken into consideration were different. This can happen when two nodes each receive a different subset of the total set of vote messages. This contradiction can be resolved trivially by forming the union of the two sets of voters and recalculating the tally on the basis of this union.
Nothing else happens in this phase and therefore the correct execution of the Voting phase can be verified.

7.2 Privacy

This requirement is defined as: A third party must not be able to determine how a voter voted. We will not consider cases where voters willingly cooperate to reveal their votes since these are covered in Section 12.1 as coercion. Privacy thus refers to determining how voters voted without their cooperation. We cannot prove that this requirement is satisfied because there is a way of violating it, but we can evaluate the system to determine the conditions that must be met for the requirement to be violated.

When we consider a third party trying to determine how a voter voted, we make a distinction between the different kinds of attackers: the attacking election manager, the attacking eligible participant and the outside attacker. It turns out that the attacking election manager has the same capabilities as the outside attacker, so we will cover them first together, followed by the attacking eligible participant.

7.2.1 Attacking election manager/outside attacker

We can prove that attacking election managers and outside attackers are unable to determine how an honest sensible voter voted.
Proof: The content of the vote is contained entirely in the vote message and its associated commitment block, there are no other messages in the protocol that contain this information in any form. Therefore if the attacker is unable to determine which voter corresponds to a commitment block and its associated vote message, then the attacker is unable to determine how the voter voted. The only information in the commitment block and vote message that is related in any way to the voter's identity is the public key and the index of the block in which it originated. This block can be either a registration block or a dilution block, but voters who vote with a key originating in a registration block do not dilute their keys. Since this contradicts the definition of an honest sensible voter we will not consider such voters and therefore we know that the key originated in a dilution block. Therefore if the attacker is unable to determine which new key originating in a dilution block corresponds to a voter, the attacker is unable to determine how that voter voted if

it is a sensible voter.

The only information in a *dilution block* that is related in any way to the voters' identities is their old keys and the associated signatures. Therefore if the attacker is unable to determine which old key and signature corresponds to which new key, the attacker is unable to determine how a voter voted. In the dilution block itself, the order of the new keys is completely independent of the order of the old keys because the block assembler is unable to figure out from each new key message what the old key is to which it corresponds. There is also no extra data within the block that could serve to link the two. Therefore the attacker cannot determine which old key corresponds to which new key by looking at the dilution block alone.

There are however nine types of other messages associated with each dilution block: the dilution application, the invite, the pool response, the pool message, the pool acknowledgement, the blame message, the new key message, the unvalidated dilution block message and the signature message.

In order for the attacker to determine which old key and signature correspond to which new key, the attacker must make use of a message containing a new key.

The first such message in the dilution process is the *new key message*. This message also contains:

- A prefix
- The contents of the pool message of this dilution process
- A signature using the session key of this dilution pool

The prefix and the contents of the pool message are common to all members of the dilution pool, so they cannot be used to discover which old key corresponds to the new key. The signature is generated using a session private key that was listed in encrypted form in the pool message. Each old key has its own session private key cyphertext in the pool message, but in order to link the new key message to this session private key cyphertext, the session private key itself must also be different. There is only one session public key in the pool message and all pool members check that their session private key corresponds to this session public key, and send a pool acknowledgement if this is the case. The new key messages are only sent after all pool acknowledgements were sent, which means the session private keys of all pool members correspond to the session public key and therefore they are all the same. It is thus not possible for an attacking election manager or an outside attacker to determine which old key and signature corresponds to the new key message.

The next message in the dilution process that contains a new key is the *unvalidated dilution block message*. This message is the same as a dilution block, except it does not contain the signatures of the old keys, but it does contain an extra prefix and a signature of the block assembler. Neither the prefix nor the signature serve to link any of the old keys to any of the new keys, therefore since the attacker is unable to determine which new key corresponds to which old key in a dilution block, it is also unable to do so in an unvalidated dilution block message.

The last message in the dilution process that contains a new key before the dilution block is appended to the blockchain is the *signature message*. This message contains:

- A dilution block with signatures
- A prefix
- Information identifying the dilution pool and its block assembler
- One of the old public keys
- The signature associated to this old public key

The prefix and the information identifying the dilution pool and the block assembler cannot be used to link any of the old keys to any of the new keys. The signature and the associated old public key are intended to identify a single voter, but it is not possible to link this voter to any of the new keys listed in the message since there is nothing that would single out one of the new keys. An attacking election manager or an outside attacker is therefore also unable to determine which new key corresponds to which old key in a signature message.

7.2.2 Attacking eligible participant

Eligible participants have an extra capability that allows them to compromise the privacy of another participant. In a single dilution block with N members, $N-1$ conspiring members can compromise the privacy of the remaining member: if they all create a joint alternative signature dilution message with their old keys and new keys and sign this message with both their old keys and their new keys, only one old key and one new key will be left and these must thus also correspond to each other. Fewer than $N-1$ conspiring participants cannot do this since there would be more than one old key and more than one new key left unidentified.

Since participants dilute their keys in multiple rounds and multiple dilution blocks, attackers must repeat this process for every round. At the end of each round the members of the dilution pool all belong to the same anonymity set, so since honest eligible participants attempt to be part of an anonymity set that is as large as possible given the number of rounds of signature dilution they participate in and the number of members in each dilution pool, each round must involve $N-1$ new and different attackers with $N-1$ different keys. If voters dilute their keys in r rounds with N participants per pool, this means $r(N-1)$ participants must conspire in order to compromise the privacy of a single participant, and they must coordinate communication across the network perfectly in order to funnel the targeted participant into each dilution pool every round.

The conspiring participants may however target multiple participants in parallel: in the first round $N-1$ conspiring members are necessary per targeted participant, and it is possible that there is only one targeted participant, as shown above. In every following

round, each pool will contain the targeted participant plus conspiring members from $N-1$ anonymity sets that do not correspond to the targeted participant's anonymity set. Each of them may come from an anonymity set in which they are already compromising the privacy of another participant, but this means the minimum number of targeted participants is N times the minimum number of targeted participants in the previous round. Therefore, in order to keep the ratio of $N-1$ conspiring participants per targeted participant during r rounds of signature dilution, it is necessary for N^{r-1} participants to be targeted by $(N-1)N^{r-1}$ conspiring participants.

It should be noted that such an attack in the first round means the attackers create proof that they are guilty of perpetrating this attack, which contradicts the definition of an attacking eligible participant. This means that if the attackers want to target a victim in the first round, they need the victim to be a leftover participant that they target with their keys of depth 1. There would then have to be N^{r-1} leftover participants and since $r \geq 2$ according to the definition of the honest eligible participant this means at least $N^{2-1} = N$ leftover participants. N leftover participants could however have formed a dilution pool before the pre-depth block and avoided being leftover, so according to the definition of the honest eligible participant there must be fewer than N leftover participants. The ratio of $N-1$ conspiring participants per targeted participant can therefore not be maintained.

The only possible scenario is therefore this: n_l leftover participants are targeted after the first round by $N-1$ attackers each, which results in $n_l(N-1)$ attackers in total. In the next round, all these attackers reshuffle so that each leftover participant is in a dilution pool with $N-1$ attackers that were in different dilution pools during the first round. If $n_l < N$ they cannot form dilution pools of size N in the second round with the $n_l(N-1)$ attackers in the first round since they would have to include attackers from the same anonymity set, which means the pool will not create an anonymity set that is as large as possible and this is something honest eligible voters avoid by definition. Each pool in the second round must therefore be supplemented with $N-n_l$ attackers that have so far not participated in the attack. Therefore the number of attackers necessary for the first two rounds is:

$$n_l(N-1) + n_l(N-n_l) = n_l(2N-1-n_l)$$

After the second round these attackers have all become part of the same anonymity set, which means that none of them can attack the victims anymore in any of the following rounds. For every following round and for every victim there must therefore be $N-1$ attackers that have so far not participated in the attack, which amounts to $n_l(N-1)(r-2)$ new attackers and the total number of attackers is:

$$n_l(2N-1-n_l) + n_l(N-1)(r-2) = n_l(r(N-1)+1-n_l)$$

The ratio of attackers per victim is:

$$\frac{n_l(r(N-1)+1-n_l)}{n_l} = r(N-1)+1-n_l$$

This is minimized when n_l is the maximum value of $N-1$:

$$r(N-1)+1-(N-1) = (r-1)(N-1)+1$$

It should be noted that, if the $n_l(N-1)$ attackers in the last round do not dilute their keys any further just like the victim, it will be clear which commitment blocks and vote message correspond to these attackers and they might be punished by disqualifying their votes. If they do dilute their keys further, they must find $n_l(N-1)(N-1)$ eligible participants who will dilute their keys at least one round further than the victim, and this number is multiplied by $N-1$ for every additional round. Since these participants have nothing to do with the attack, they might decide to mount a counterattack where they use the same strategy as the attackers to expose which new key corresponds to the attacker's old key. There is no guarantee the participants will do this, but besides the fact that honest participants are incentivized to see the election be conducted properly, they are also incentivized to see the attackers being punished by disqualification if they vote differently. The honest voters who do not vote differently from the attackers have no such incentive, but they also do not provide any effective privacy since they reveal what the attacker has voted for.

Therefore the only way of violating voter privacy is for a group of $n_l(r(N-1)+1-n_l)$ attackers to attack $n_l < N$ victims, where they have to carefully coordinate communication across the network so the victims all become leftover participants in the first round and so they funnel the victims into the dilution pools of the attackers in each round, where the victims participate in relatively few dilution rounds and where the attackers potentially expose themselves to disqualification of their votes.

7.2.3 Colluding roles

The attack that can be carried out by eligible participants would not be helped further by collusion with the election manager or outside attackers since they are not involved in the dilution process. There are no extra dangers to privacy when an election manager or an outside attacker conspires with an attacking eligible participant.

7.3 Eligibility

This requirement is defined as: A voter must be someone who is eligible to vote. AnonVote satisfies this requirement for honest sensible voters, while senseless eligible participants are able to transfer their ability to vote to another person.
Proof: A voter is someone who creates a commitment block and sends an associated vote message. The voter must control a keypair that is considered valid within the blockchain at the moment the commitment block is added because the voter would not be able to sign the block otherwise. There are only two reasons why a key would be considered valid for voting: it was either registered in a registration block or it was one of the new keys of a dilution block.

If the key was registered in a *registration block*, it belongs to an eligible participant since by definition the creation of a registration block takes place during a registration process that provides the election manager with proof that the participant chose to register this

key.

If the key originates in a dilution block, it replaces another key that is made invalid by this dilution block and that belongs to the same participant if the participant is a sensible honest eligible participant because sensible honest eligible participants do not disenfranchise themselves. It is however possible that a senseless eligible participant owns the key that is replaced and chooses to sign the dilution block anyway: the participant is thereby disenfranchised and the dilution block can contain one additional new key that may belong to anyone.

Therefore if the old key belongs to a sensible honest eligible participant, the new key belongs to a sensible honest eligible participant as well (because it is the same participant). In order for the dilution block to be valid, this old key that is replaced must be valid at the time the dilution block is added. By virtue of induction this means any key originating in a dilution block belongs to an honest eligible participant with the exception that senseless eligible participants can disenfranchise themselves and transfer their ability to vote to another person.

7.4 Uniqueness

This requirement is defined as: Only one vote per voter may be tallied. AnonVote satisfies this requirement with the exception that senseless eligible participants can disenfranchise themselves and transfer their ability to vote to another person.
Proof: Tallying is done completely in the open on the basis of the public keys used in commitment blocks and their associated vote messages. Since the protocol specifies that only the last commitment block of a public key can be tallied, and since each commitment block only represents a single vote, only one vote per public key is tallied.

This means that, in order for a single voter to have multiple votes that are tallied, this voter must own multiple public keys that are still valid in the blockchain during the Commitment phase. There are two places where the public keys can originate: in a registration block or in a dilution block.

If a public key is registered for a participant in a *registration block*, the protocol dictates that all public keys that were registered for this participant in earlier registration blocks are no longer valid. It is therefore only possible for a participant to have at most one key originating in a registration block that is still valid.

In a *dilution block* each sensible eligible participant that is a member of the dilution pool receives one new valid key, but the participant must also make one old key invalid. Thus, in the absence of senseless eligible participants, a sensible eligible participant always gains one valid key in a dilution block but loses another and therefore the number of valid keys a sensible eligible participant owns remains constant with each dilution block.

Senseless eligible participants are only different in the sense that they can sign a dilution block with their old keys even when they do not possess the private key corresponding to any of the new public keys and thereby disenfranchise themselves and transfer their ability to vote to another person. It is thus not possible for a participant to own multiple keys that

are still valid, and therefore only at most one vote per voter is tallied, with the exception that senseless eligible participants can disenfranchise themselves and transfer their ability to vote to another person.

7.5 Integrity

This requirement is defined as: Votes must be tallied according to their content, as chosen by the voter. AnonVote completely satisfies this requirement.
Proof: The protocol specifies that votes are tallied according to their content w as listed in the vote message. The vote message also contains a salt z, and w and z are hashed together to form $hash(w, z)$, which is included in the commitment block. Votes are therefore tallied according to their content as committed in the commitment block. The commitment block is signed with the valid key of the voter, making it infeasible for anyone other than the voter to create the commitment block or alter its contents. Therefore the commitment block's hash $hash(w, z)$ represents the vote content as chosen by the voter and the vote is tallied according to this content.

Chapter 8

Denial of Service

One danger the system should be robust against is the danger of denial of service attacks: attackers can attempt to sabotage the system by sending large numbers of messages to honest participants and forcing them to process them, wasting time and resources. The messages could be either valid or invalid.

Zargar et al. [40] provide a classification of DDoS attacks with two main categories: network/transport level DDoS flooding attacks and application level DDoS flooding attacks. Even though AnonVote is an application layer protocol, it is actually the network/transport level DDoS flooding attacks that are relevant to it, because AnonVote is a peer-to-peer networking protocol as opposed to a client-server protocol. There are four types of network/transport level DDoS flooding attacks:

1. **(Plain) flooding attacks:** attackers simply send a large number of messages to the victim. The messages are not necessarily valid.

2. **Protocol exploitation flooding attacks:** attackers send a large number of messages to the victim, where each message is actually valid and causes the victim to allocate resources to initiate a process, but the attackers subsequently break the process off and the victim has wasted the allocated resources.

3. **Reflection-based flooding attacks:** attackers do not send messages directly to the victim, but instead to a third party: the messages are valid and cause the third party to send messages to the victim.

4. **Amplification-based flooding attacks:** attackers exploit services to send a large number of messages to the victim. The attackers send messages to these services that are significantly smaller in size or number than the messages that the services send to the victim.

Zargar et al. categorize defense mechanisms according to four broad categories: *source-based mechanisms*, *destination-based mechanisms*, *network-based mechanisms* and *hybrid mechanisms*. This classification is however tailored to networks consisting of core and edge

routers, used by client-server applications. For a peer-to-peer network such as AnonVote many of the defense mechanisms cannot be implemented as is, but some can be modified slightly to fit and sometimes the distinction between two mechanisms is blurred.

Ingress/egress filtering can be applied to any network. *IP Traceback mechanisms*, *Hop-count filtering*, *Path Identifiers*, *Route-based Packet filtering* and *History-based IP filtering* could all be implemented by modifying the protocol, but these mechanisms are detrimental to privacy. The mechanism of *detecting and filtering malicious routers* can however be easily modified to apply to nodes in the AnonVote network instead of routers: when node A discovers that node B is sending malicious traffic, node A can block messages coming from node B. The *Management Information Base* is founded in the idea that certain statistical assumptions about the proper flow of messages can be made, and that traffic deviating from these assumptions can thus be assumed to be malicious. We can apply this same idea to AnonVote, reasoning about the frequency of certain messages.

The degree to which it is possible to send valid messages is heavily dependent on the capabilities and therefore the role of the attacker. We will therefore structure this section according to message validity and the different attacker roles: first we will cover invalid messages coming from any sender in Section 8.1, then valid messages from outside attackers in Section 8.2, followed by attacking eligible participants in Section 8.3 and finally attacking election managers in Section 8.4.

8.1 Invalid messages

Messages can be validated upon arrival and subsequently ingress filtering can be applied: invalid messages will not be processed afterwards. This limits the impact of plain flooding attacks by sending large numbers of invalid messages, but validation itself will still take time and resources so it is important to evaluate its impact.

Blocks created by election manager

The *initialization block* contains the public key of the election manager and is signed with the signature corresponding to that public key. Validation thus consists of verifying the signature over the message.

The *registration block*, *dilution start block*, *pre-depth block*, *depth block*, *dilution end block* and *commitment end block* contain the public key of the election manager, the block index, the hash of the previous block and the chain score, and they are signed with the signature corresponding to the public key of the election manager. Validation thus consists of verifying the signature over the message, verifying that the hash of the previous block corresponds to the hash of the block with an index that directly precedes the block index of this block, and verifying that the chain score is increased correctly compared to the chain score of the preceding block. All of this can be done in constant time with respect to the number of blocks.

Blocks created by eligible participant

The *dilution block* contains a number of old public keys to be diluted and a number of new public keys to replace them. The block is invalid if the number of old keys does not equal the number of new keys, which is trivial to verify. It is also signed with signatures corresponding to all the old public keys, so these signatures must be checked when validating and this can be done in constant time with respect to the number of blocks. These keys must also still be valid at this point in the blockchain however. This can trivially be verified in linear time by, for every key $pk_{i,r}$, iterating over all the blocks in the chain to see whether there is a block where $pk_{i,r}$ originates and whether there is no subsequent block where $pk_{i,r}$ is used and thus made invalid. However, with a little extra memory it can also be verified in constant time: each key is listed alongside the index of the block in which it originates, so this can be verified by simply checking the block at that index. Each node can also store, for every key originating in a block, the index of the block in which it is used and made invalid, thus allowing verification that a key was not yet used in constant time. The node must of course update these indices whenever a new block arrives that makes use of a key in an older block, but this can also be done in constant time.

The dilution block also contains an anonymity set which must be the lowest from among the anonymity sets of all the blocks where the old keys originated, and each of these anonymity sets can be found by looking at their blocks, which is possible in constant time. The block also contains a depth, which must be one higher than the highest depth from among the depths of all the blocks where the old keys originated. These again can be found by looking at their blocks in constant time. Finally the block also contains a block index, hash of the previous block and chain score, which can be found in constant time with respect to the number of blocks.

The *commitment block* contains the block index, hash of the previous block and chain score, which can all be easily verified by comparing the previous block known to the node. It contains the public key of voter v_i and the salted hash of the vote w_i, along with a signature corresponding to v_i's public key. If the signature is valid, the block is valid. Note that it is possible for the salted hash to be gibberish and not correspond to any vote w_i, but we still consider the commitment block to be valid in this case.

Messages sent by eligible participant

The *disenfranchisement message* contains a registration block, which must be valid in order for the disenfranchisement message to be valid. If it is valid, the absence of the block in the blockchain must be confirmed, which can be done in constant time by reading the block index and comparing the block at that index in the chain. If it is indeed absent, the absence of the voter ID in any registration block in the blockchain must be verified. This can trivially be done in linear time by looking at all the blocks from the start of the chain to the dilution start block, but it can also be done in near constant time with respect to the number of blocks if a hashmap is kept mapping voter IDs to the indices of their registration blocks. The size of each entry in this hashmap is smaller than the size

of the corresponding registration block, since it contains only the voter ID and the block index, both of which are also contained in the registration block. The hashmap is therefore smaller than the total number of registration blocks in the chain and thus smaller than the blockchain itself.

The *dilution application* is signed by the public key listed in it, which can be easily verified. The public key must however also still be valid and its depth must be consulted, all of which can be done in constant time with respect to the number of blocks, as we have shown above. When the depth has been consulted, it must be compared to the maximum depth currently allowed in the chain, which can be done trivially in linear time by iterating over all the blocks and counting the (pre-)depth blocks. A node may however also keep track of the maximum depth, allowing it to make the comparison in constant time.

The *invite* is signed by the public key of the block assembler listed in it, which can thus be easily verified. The public key of the block assembler must still be valid however and its depth must be allowed, but we have shown in the above that it is possible to check this in constant time. Finally the invite must correspond to a dilution application, but all the data within the application is also contained within the invite, as well as its signature, so the dilution application can be reconstructed and validated in constant time.

The *pool response* is signed by the public key of the invitee listed in it, which can thus be easily verified. The pool key within it is also separately signed by the same public key, which can therefore also be easily verified. The pool response must correspond to a valid invite, but all of its information, including it signature, is contained within the pool response. It is therefore trivial to reconstruct the invite and validate it.

The *pool message* is signed by the public key of the block assembler pk_{a_b} listed in it, which can thus be easily verified. In addition to this key it also contains the public keys of all the invited members, which must all still be valid and at an acceptable depth, but we have shown above that this can be verified in constant time. Each key also has a pool key which is signed by the public key itself and thus trivial to verify. The pool message must correspond to a number of pool responses in order to be valid, but it contains all the data necessary to reconstruct those pool responses along with their signatures, so this can easily be verified. Lastly the pool message contains a session public key pk_P as well as cyphertexts of the corresponding sk_P, each encrypted by the pool public key of one of the members v_i as $E_{pk_{i,P}}(sk_P)$. Each of these cyphertexts must therefore, when decrypted, contain the right private key, but this cannot be verified using only the cyphertext. It is thus possible for $E_{pk_{i,P}}(sk_P)$ to contain an invalid sk_P and only v_i would know since no one else can decrypt the cyphertext. v_i is however expected to respond with either a *pool acknowledgement* indicating that $E_{pk_{i,P}}(sk_P)$ is valid, or a *blame message* proving that it is invalid. It must thus be assumed that, if the validation checks mentioned earlier in this paragraph succeed, the pool message is valid until a blame message proves otherwise, at which point nodes can hold the block assembler a_b accountable by giving priority to messages not coming from a_b, in a manner similar to the mechanism of detecting and filtering malicious routers. A pool message can therefore be validated in constant time with respect to the number of blocks in the chain, for all purposes not covered by the pool acknowledgement and blame message.

The *pool acknowledgement* contains a pool message as well as the index of the pool member v_i acknowledging that the message is valid, along with a signature corresponding to v_i's public key $pk_{i,r-1}$. The pool acknowledgement is valid if the pool message is valid, if the index is within the range of the members of the pool, and if the signature is valid, which can all be verified in constant time with respect to the number of blocks in the chain.

The *blame message* also contains a pool message as well as the index of the pool member v_i claiming that $E_{pk_{i,P}}(sk_P)$ encrypts an invalid session private key sk_P, alongside the pool private key $sk_{i,P}$. This private key can be used to decrypt $E_{pk_{i,P}}(sk_P)$ and thus verify that the pool message was indeed invalid, which means the blame message is valid.

The *new key message* contains a new public key and a pool message, which can be validated as described in the above, and it is signed by the session private key sk_P communicated via cyphertext in that pool message. The signature can thus be easily verified, proving that the message is valid.

The *unvalidated dilution block message* mostly contains the same data as the dilution block, but it is signed with a signature corresponding to the key of the block assembler pk_{a_b} instead of the keys of all of the pool members. The signature is easy to verify, proving that the message is valid.

The *signature message* contains the contents of the dilution block with a signature of one of the pool members. In order to verify that the signature message is valid, the signature must be verified. The signature message must also correspond to an earlier unvalidated dilution block message, but it contains the signature of that message and all the data necessary to reconstruct it, so this can be verified by validating the signature.

The *vote message* contains the index of the commitment block and the corresponding vote w_i and salt z_i which make it trivial to recompute the hash in the commitment block. If the recomputed hash is equal to the actual hash in the block, the vote message is valid.

Control messages

The *chain request* does not contain any signature or anything that can be validated, but it is possible that a node receives a chain request for a blockchain that it does not track itself. It is not really useful to mix nodes tracking different blockchains in the same network, so when this occurs the node can choose to consider such a message to be invalid. This already severely limits the number of chain requests that can be sent to other nodes.

The *block request* also does not contain any signature, but it does contain the index of a block within the blockchain and the hash of the block. They can thus be validated by looking up that block in the blockchain in constant time, and then comparing the hash. This decision does however impact the concept of validation in the case of forks in the network: a node that tracks one branch of the fork will consider a block request invalid if it refers to a block in the other branch. This should not be a problem though if nodes send out block requests wisely: they only have a reason to send out a block request if they receive a block that they do not have the predecessor of from another node, so it makes sense to send the block request to that node since it is obviously tracking the right branch if it is honest.

The *requested block message* simply contains a block with its hash and an additional prefix. The block can be validated as described in Sections 8.1 and 8.1. Validation of the message can however be taken one step further: this message is only supposed to be sent in response to a chain request or a block request, so there is no reason for nodes to consider random requested block messages to valid. Instead they limit the requested block messages they validate to the ones that they sent a chain request or block request for.

Conclusion

All the messages that are sent can therefore be validated in constant time with respect to the number of blocks in the blockchain, often by verifying digital signatures. This makes the impact of invalid messages low, but if the numbers are great enough, they can still cause a problem. It is possible to prevent such denial-of-service attacks from a single node by muting it after it sends a number of invalid messages within a short period of time. This does not prevent distributed denial of service attacks, but they can be prevented by limiting the number of new connections a node makes and prioritizing older, trusted connections, applying the same principle as history-based IP filtering.

8.2 Outside attackers

If outside attackers can create valid messages, this allows those messages to move beyond validation and be processed. This would open up the possibility of protocol exploitation flooding attacks, reflection-based flooding attacks and amplification-based flooding attacks. It is therefore important to consider each type of message to see if outside attackers are capable of creating valid versions of them.

The *initialization block*, the *registration block*, the *dilution start block*, the *pre-depth block*, the *depth block*, the *dilution end block* and the *commitment end block* are all signed with the signature of the election manager. Since outside attackers do not know the private key of the election manager, it is not feasible for them to create any of these blocks validly. The *dilution block*, *dilution application*, *invite*, *pool response*, *pool message*, *pool acknowledgement*, *unvalidated dilution block message*, *signature message* and *commitment block* are all signed with a signature corresponding to an eligible participant's key that originated somewhere in the blockchain. Outside attackers do not know the private keys of these eligible participants, therefore it is also not feasible for them to create any of these blocks or messages validly.

A *blame message* contains a pool message, which must have been created by the block assembler since the signature must be valid, and it contains the index of the pool member who supposedly received a fake private key and the pool private key of this participant. The pool private key can be used immediately to decrypt the fraudulent private key and thus verify that the blame message is valid. A valid blame message proves that the block assembler is an attacker, so even if there is a large number of valid blame messages, the blame messages themselves cannot be considered to be an attack: they are exposing an

attack.

A *vote message* contains the public key of the election manager, index of the election, public key of the voter and the index of the commitment block, which all serve to identify the commitment block that this vote message corresponds to. The vote message also contains the content of the vote w and the salt z, which can be verified by hashing them and comparing the result to $hash(w, z)$ in the commitment block. Therefore it is easy to verify that a vote message is valid if the commitment block it corresponds to is valid, and we have already shown that it is easy to verify this as well. In order for outside attackers to be able to create vote messages, they must discover the vote w and salt z that was hashed in a commitment block, and then create a vote message that contains these values. Even if it were feasible for attackers to discover these values, it cannot be considered an attack since they only end up broadcasting vote messages that should be broadcast and tallied anyway.

A *chain request* simply contains a prefix, the public key of the election manager and the index of the election. It is perfectly possible for an outside attacker to send such a message. However, the prefix is invariant and the other two valuables are only considered valid for blockchains the recipient tracks itself (see Section 8.1), so the number of such messages that can be sent is actually severely limited.

A *block request* contains a prefix, the information identifying the election and the index and hash of the block that is requested. For it to be considered valid, the election must be one that the recipient is tracking, and the hash must corresponding to the block at the given index in the blockchain. This limits the number of possible valid block requests one node can send to another, to the size of the blockchain. Sending block requests for all the blocks in the blockchain can however hardly be considered to be a denial-of-service attack, since it is in fact expected behavior for any new node joining the network. Nonetheless this is an avenue for a distributed denial-of-service attack, since a large number of attacker nodes can connect to a victim node, request the entire blockchain, and then do nothing. This can be mitigated if the victim node simply does not accept an unlimited number of connections from other nodes: it is not a server serving requests, it is a node in a peer-to-peer network.

A *requested block message* must contain a valid block and its hash and be sent in response to a chain request or a block request. The only feasible way to respond with a valid message to such requests is by sending the exact message that the recipient is requesting, which cannot be considered an attack.

In conclusion, it is not possible for an outside attacker to carry out a denial-of-service attack using valid messages.

8.3 Attacking eligible participants

Attacking eligible participants are capable of creating many more valid messages than outside attackers and it is thus important to see if each of them can be used in a protocol exploitation flooding attack, reflection-based flooding attack or amplification-based

flooding attack.

8.3.1 Disenfranchisement message

Eligible participants can create disenfranchisement messages if their registration block has been orphaned. They can however only create one disenfranchisement message per orphaned registration block, making it impossible for them to create large numbers of disenfranchisement messages unless the election manager orphans large numbers of their registration blocks. This is however only possible if those registration blocks exist at all, which is not realistic since the number must be large enough to cause problems for a node to process, but small enough so the election manager can register these participants off-chain. It also presupposes an attack by the election manager, who orphans registration blocks. It is thus not feasible to carry out a denial-of-service attack by means of disenfranchisement messages.

8.3.2 Dilution block

During the Dilution phase, eligible participants create dilution blocks, but they do not do this individually: they cooperate with each other to create the block in a dilution process. A single participant can thus not create a large number of dilution blocks since a single participant cannot create a single dilution block. Multiple conspiring participants can however create a large number of dilution blocks and append them one after the other. As long as the participants are actually unifying anonymity sets though, the blocks are useful and cannot reasonably be considered to be an attack: there is actual signature dilution taking place. Conspiring participants might however create a large string of dilution blocks one after the other by diluting the keys of the same participants over and over again. They only unify anonymity sets in the first block, afterwards all the keys to be diluted come from the same anonymity set and therefore the chain score does not increase. This means all these dilution blocks could easily be orphaned in a fork, so nodes can safely just ignore dilution blocks if they notice a large number of them that do not increase the chain score, and thereby protect themselves from a denial-of-service attack.

If the participants within a dilution pool are not all conspiring, some of the members might carry out a denial-of-service attack against the others by means of the messages sent in the dilution process. We must therefore look at each of these messages to determine the actual risk:

8.3.3 Dilution application

Dilution applications contain the key that is to be diluted, and they are signed with that key. Eligible participants who only possess a single valid key pair can only broadcast one dilution application. In order to send large numbers of dilution applications, they would therefore have to possess large numbers of valid key pairs and we have shown in Section

7.4 that this is not possible. On the other hand it is possible that an eligible participant broadcasts a dilution application but does not respond to all invites that follow: this is expected behavior so we do not consider this to be a denial-of-service attack.

8.3.4 Invite

Invites contain:

- The public key of the participant that is invited
- An identifier of the pool ID_P
- The key of the block assembler
- The signature of the dilution application

They are signed with a signature that must correspond to the key of the block assembler. Due to the signature it would not be feasible for an eligible participant to send a large number of invites with different block assemblers in each one of them, but it is definitely possible for an eligible participant to send a large number of invites with different invitees and different pool identifiers. To exclude invites for fake dilution applications, the dilution application of the invitee can be reconstructed (using the public key of the invitee and the fixed prefix) and the signature can be checked to see if there actually was a real dilution application. This limits the number of possible invites, but it is still possible that there are a lot of invitees if there are a lot of dilution applications that have not yet been included in a dilution block. It can hardly be considered an attack to invite them all though, since this situation means that it is urgently needed for a block assembler to do so.

It is however suspicious to see the same block assembler send out invites with multiple different pool identifiers: honest block assemblers are most likely trying to dilute their own keys in a block and are thus not likely to be forming multiple dilution pools at the same time. It is thus very suspicious to see the same block assembler send invites with different pool identifiers shortly after each other. It is not suspicious to see invites with different invitees, since a block will contain multiple keys anyway and not all invitees respond. Nevertheless, inviting a large number of invitees in a very short time span is still suspicious, since one can only send an invite in good faith if one waits to see what the response is.

8.3.5 Pool response

Pool responses contain:

- The key of the block assembler
- The pool identifier

- The pool public key
- The old public key of the participant
- A signature of the pool public key
- The signature of the dilution application
- The signature of the invite
- The signature of this pool response

Due to the signatures, a participant can only send a pool response when an earlier invite exists and thus it is not possible to send large numbers of pool responses all with different old public keys, block assemblers and pool identifiers. It is however possible to send large numbers of pool responses with different pool public keys, since this is new information introduced by the pool response. It is however trivial for nodes to spot such an attack, since an invitee has no good reason to send multiple pool responses with different pool public keys for the same pool identifier one after the other.

8.3.6 Pool message

Pool messages contain:

- The key of the block assembler
- The pool identifier
- The session public key
- The old public key of every member
- The pool public key of every member, with signature
- The signature of the dilution application of every member
- The signature of the invite of every member
- The signature of the pool response of every member
- The session private key, encrypted by each member's pool public key
- A signature of this message by the block assembler

Due to the signatures, this message can only be created by the block assembler that sent the invites, and the dilution applications, invites and pool responses can all be reconstructed and validated. It is thus not possible to create a pool message for members that did not apply and send pool responses. It is however possible that the session private key encrypted for one of the members does not correspond to the session public key listed in the message. Only the member can tell because no one else can decrypt the session private key, so it is possible for a block assembler to send a fake pool message containing the wrong session private keys. The member can then however send a blame message in order to prove this malicious behavior the part of the block assembler, who can then be punished by the network.

There is also a possibility of a protocol exploitation attack here when the block assembler refuses to create a pool message after receiving all the necessary pool responses. The pool will then fail and its members will have wasted time and resources. The network can however punish the block assembler: nodes can randomly decide to monitor the messages of other dilution pools and thus see that all the pool responses of a pool were sent, but the pool message was not. They can then decide to deprioritize this block assembler as a partner in signature dilution.

8.3.7 Pool acknowledgement

Pool acknowledgements contain:

- The index of the member who acknowledges the pool message
- The pool message being acknowledged
- The signature of the member who acknowledges the pool message

The signature ensures that only the acknowledging member can create the pool acknowledgement, and it is ensured to be a response to a valid pool message since it is contained inside the acknowledgement as well.

Similar to the pool message, there is a possibility of a protocol exploitation attack here when the member refuses to send a pool acknowledgement even if the pool message was valid. However if other nodes see neither a pool response nor a blame message from a given pool member, they can conclude that that pool member is carrying out such an attack and again deprioritize this member as a partner in signature dilution.

8.3.8 Blame message

Blame messages contain:

- The index i of the member v_i who received a fake session private key

- The pool private key of v_i
- The pool message that was fraudulent

Using the index and the pool private key, anyone can decrypt the session private key within the pool message and thus verify that it is indeed fraudulent. Similar to the disenfranchisement message, it is not possible to mount a denial of service attack with many valid blame messages since this presupposes that there exist so many fraudulent pool messages that they would constitute a denial of service attack in and of themselves.

8.3.9 New key message

New key messages contain:

- The new public key of the pool member
- The pool message
- The signature corresponding to the session public key of this pool

The signature guarantees that an actual member of the pool sent this message, and there must be an earlier pool message since it is contained within this message. It is nevertheless possible for one of the members in the pool to send a large number of new key messages and thus mount a denial of service attack. However, it is easy to spot such an attack since the number of new key messages that are to be sent for a single pool is known: it must be equal to the number of old public keys listed in the pool message. If more new key messages are sent, the dilution process is going to fail even if it is not a denial of service attack, so it is perfectly reasonable to simply consider the pool to be invalid when too many new key messages are sent. This in turn however opens up the possibility of protocol exploitation attack: the attacker does not need to send a really large number of messages, but can simply sabotage a dilution pool by sending too many new key messages or none at all. It is not possible for observers to spot which pool member is sabotaging the process here, since the protocol is set up in such a way that new key messages are sent anonymously. Other nodes can however conclude that someone in the pool is an attacker and thus collectively punish them all by deprioritizing them as partners in signature dilution: if this happens once, the deprioritized participant will not suffer much, but if the same participant is deprioritized by multiple failed pools, this will accumulate and the participant will be very unlikely to be part of another dilution pool soon. It is likely that the participant is an attacker in this case, since the participant was the common element in all of these failing dilution pools.

8.3.10 Unvalidated dilution block message

Unvalidated dilution block messages contain the data of a dilution block without its signatures, but they also contain the signature of the block assembler. There is nothing inside them to prove that the earlier messages in the dilution process were actually sent: it is possible the block assembler simply created an unvalidated dilution block message for pool members who never agreed and who will thus not sign it either. Nodes can however quickly notice this by seeing the same block assembler broadcasting multiple unvalidated dilution block messages one after the other. The block assembler can then be deprioritized as a partner in signature dilution.

8.3.11 Signature message

Signature messages contain:

- The identifier of the pool and the key of its block assembler
- The old public key of v_i, who will sign the block
- The signature of the block, corresponding to the public key of v_i
- The dilution block without signatures

The maximum number of valid signature messages that can be sent is the number of members in the pool: a signature message corresponding to a key not in the pool will simply be considered invalid. It is thus not possible to send a large number of signature messages, but it is possible to neglect to send a signature message and thus sabotage the dilution process in a protocol exploitation attack. Outside observers cannot distinguish between the situation where a malicious pool member refuses to send a signature message, the situation where a malicious block assembler did not include the key of an honest pool member who then refuses to send a signature message, and the situation where an honest block assembler did not include the key of an honest pool member because another pool member, who is malicious, sent two new key messages. They must therefore collectively punish all members of the pool in the same way as they would when an incorrect number of new key messages is broadcast.

8.3.12 Commitment block

Commitment blocks contain:

- The salted hash of the vote w_i of voter v_i
- The public key of voter v_i

- A signature corresponding to the public key of v_i

The signature ensures that the block was created by v_i, but this still allows v_i to create a large number of commitment blocks one after the other. Since only the first commitment block will contribute to the chain score, it is quite reasonable for nodes to ignore a large number of commitment blocks coming from the same public key: they can be orphaned anyway.

8.3.13 Conclusion

There are thus a lot of messages that cannot be used in a denial-of-service attack, but some are. In some of these cases it is obvious that the blame falls on the shoulders of one particular participant, but in others (such as the new key message) it is unclear who is to blame. In those cases the only option is to blame all members in the pool equally: the innocent members will suffer a little bit but if a participant is part of many different failing pools, it is likely that this participant is the one to blame in reality.

8.4 Attacking election managers

Election managers create *initialization blocks*, *dilution start blocks*, *dilution ends blocks* and *commitment end blocks*. All of these blocks can only be appended in their specific phase, and they also end that phase, meaning they can no longer be appended after they have been appended once. Therefore it is not possible to use these blocks in a denial-of-service attack.

Election managers also create *registration blocks*, but they need to be able to back these up with an off-chain attestation of this registration (e.g. a declaration signed by the eligible participant). Election managers cannot add registration blocks of non-existent participants since the registration block would contain a voter ID that is not on the electoral roll, and if it adds a large number of fraudulent registration blocks of eligible participants the last registration blocks of these participants (and therefore the ones considered valid) will not be backed up by an attestation. This means the eligible participants can point out that they have been disenfranchised and the election manager does not have the attestation to prove them wrong. The result is that the attacking election manager can no longer masquerade as an honest election manager, which means sensible attacking election managers will not carry out such an attack. The system however is vulnerable to senseless attacking election managers.

The election manager also creates *pre-depth blocks* and *depth blocks*. It is not possible to create a large number of pre-depth blocks all at once or a large number of depth blocks all at once, since the two have to alternate. It is however possible to create a long sequence of alternating pre-depth blocks and depth blocks and thereby impede the normal functioning of the Dilution phase. It will be very obvious to any observer though that attacking election managers are doing this, meaning they can no longer masquerade as honest election managers, but this is again a possibility for senseless attacking election managers.

Any denial-of-service attack will therefore prevent attacking election managers from masquerading as honest election managers, which means they will only carry them out if they are senseless.

8.5 Conclusion

Denial-of-service attacks are possible with the protocol as described in Chapter 4, but nodes can utilize various strategies to defend against them. Senders of large numbers of invalid messages can be muted and the number of new connections a node receives can be limited. In the Dilution phase, attacking eligible participants can carry out protocol exploitation attacks by engaging in a dilution process but abandoning it before the dilution block is created and signed. This can however be mitigated by blaming all the pool members of the dilution process and seeing how the blame accumulates: eligible participants who participate in a lot of failing dilution processes are likely to be attackers. Attacking election managers have extra capabilities of carrying out a denial-of-service attack, but these reveal their culpability so they can no longer masquerade as honest election managers.

Chapter 9

Eclipse attacks

Another type of attack the system should be robust against is the eclipse attack. "In an eclipse attack, the attacker monopolizes all of the victim's incoming and outgoing connections, thus isolating the victim from the rest of its peers in the network ... [and filters] the victim's view of the blockchain." [19] Any communication network that is not a complete graph is vulnerable to an eclipse attack: if there are two nodes v_1 and v_2 that are not adjacent, then they can be isolated from each other by all the nodes that are adjacent to v_1. The success of an eclipse attack greatly depends on whether the victim realizes that the attack is taking place: victims who know this can attempt to break the eclipse by connecting to new nodes. In order to evaluate how robust the system is against eclipse attacks, we must look at the impact of eclipse attacks in each of the five phases of an election:

9.1 Setup phase

During the Setup phase only the *initialization block* is created. Since this is the first block, any node in the network must have this block in order to be able to have any view of the blockchain at all. The only possible eclipse attack in this phase is therefore an attack where victims know that they are being denied access to the blockchain. The victims can therefore keep searching for new nodes to connect to until they receive the initialization block.

9.2 Registration phase

During the Registration phase only the election manager creates blocks: the *registration block* and the *dilution start block*. We shall first consider eclipse attacks targeting these blocks from attacking eligible participants and outside attackers, and then we consider eclipse attacks by attacking election managers.

9.2.1 Attacking eligible participant

Since the election manager creates the initialization block, the registration block and the dilution start block, no network communication is necessary for the election manager to have a full view of the blockchain up to the dilution start block if the election manager consists of a single node. If the election manager consists of multiple nodes, they can be isolated from each other if they are not adjacent, but it is trivial for the election manager to prevent this: the election manager knows which nodes it consists of, and can just make sure that these nodes are all directly connected to each other. The election manager will thus have a complete view of the blockchain when it creates the dilution start block and therefore this block will be appended to the full blockchain.

This means that any other participant who receives and validates this block must also have received all the preceding blocks. An eclipse attack during this phase targeting these blocks and participants who are not the election manager will therefore result in these participants being stuck in the Registration phase. We have however stipulated the existence of a communication channel called social discourse, that will make it obvious to the victims that they are stuck in the Registration phase when it should already have ended, which allows the victims to start searching for other nodes until they see the eclipse has ended.

9.2.2 Attacking election manager

An attacking election manager however has more options: the election manager can create a fork in the Registration phase and direct the view of each branch to a different subgraph of the network. There are two possibilities from this point onward: either at some point the isolation ends and the fork is resolved, or the isolation and the fork persist forever. In the latter case, the election will proceed to the Dilution phase, the Commitment phase and the Voting phase with multiple blockchain branches each viewed by different subgraphs of the network and each resulting in a different tally. Since we have assumed a communication channel called social discourse exists that communicates the tallies, the discrepancy will be discovered. In such a situation one of the tallies must be chosen as authoritative and therefore its branch must be communicated to everyone, which breaks the isolation and then the fork will be resolved anyway.

If the fork is resolved, at least one branch is orphaned and it is possible that this branch contains registration blocks that do not exist on the other branch. It is also possible that the resolution of the fork happens after the Registration phase has ended, which means it is possible for a participant to be disenfranchised. The participant can however use the orphaned registration block to create a disenfranchisement message proving this and thus discrediting the election manager. If the fork does not disenfranchise the participant it can still keep the participant busy on the wrong branch for so long that the participant can no longer make proper use of the Dilution phase, except the definition of social discourse specifies that it is updated often enough to allow the participant to make full use of the Dilution phase.

9.2.3 Conclusion

Victims of an eclipse attack in the Registration phase can therefore either escape the attack soon enough to make it useless or prove that the election manager has disenfranchised them.

9.3 Dilution phase

The Dilution phase is complex and it is the only phase in which both the election manager and the eligible participants create multiple blocks. We will cover eclipse attacks in this phase based on the role of the victim: first we cover eclipse attacks on eligible participants in Section 9.3.1, followed by eclipse attacks on election managers in Section 9.3.2.

9.3.1 Victimized eligible participants

During this phase a large number of blocks are created, but also an even larger number of other messages is sent. We will consider eclipse attacks filtering blocks first, followed by eclipse attacks filtering other messages.

Filtering blocks

During the Dilution phase the dilution block, pre-depth block, depth block and dilution end block are created. Eligible participants cannot create the pre-depth block, the depth block or the dilution end block. If they are block assemblers, they can create a dilution block. The only effective block-based eclipse attack on eligible participants can therefore be in preventing these blocks going to the participants, or preventing a dilution block coming from a block assembler. There will not be a fork but the participants will have a view of the blockchain in an earlier state.

This can affect the signature dilution process in the following way: a difference in state of the blockchain between a state S_0 and a state S_1, where both S_0 and S_1 are in the Dilution phase, means there have been a number of blocks appended to the blockchain in state S_0. Adding a block changes the length and chain score of the blockchain and also the last block (and thus its hash), but depending on the block there are also other state changes. The only blocks that can be appended in this phase without ending the phase are the dilution block, pre-depth block and depth block. The dilution block causes some keys to become invalid and some other keys to become valid, the pre-depth block increases the maximum depth and adds a condition limiting which keys can validly be in a dilution block, and the depth block removes such a condition. Therefore the possible state changes between S_0 and S_1 are:

1. The length of the blockchain changes.
2. The chain score of the blockchain changes.
3. The last block of the blockchain changes.

4. A key becomes valid.
5. A key becomes invalid.
6. The maximum depth changes.
7. A condition limiting which keys can validly be in a dilution block is added.
8. A condition limiting which keys can validly be in a dilution block is removed.

We will now consider the effects of these state changes on any of the messages sent during the dilution process.

Dilution application: When a participant sends out a dilution application it only contains a prefix, information identifying the blockchain, the key the participant wishes to dilute and a signature corresponding to that key. Only the status of the key (and by extension the signature) can be affected by any of the possible state changes: a key might become usable in a dilution pool or it might become unusable in a dilution pool.

Therefore eclipsed participants, thinking the blockchain is in state S_0 while it is actually in state S_1, might send dilution applications with an unusable key and refrain from sending dilution applications with a usable key and thus never have a key diluted.

Eclipsed block assemblers, when receiving a dilution application, might incorrectly conclude that the key within it is unusable and therefore refrain from including this key in their dilution pools, which limits their prospects and therefore makes it more difficult to create a dilution block. Conversely they might receive a dilution application from an attacker with a key that is no longer usable in state S_1 but still usable in state S_0 and thus be tricked into forming a dilution pool that can never lead to a valid dilution block. This again makes it more difficult to create a dilution block.

Invite: When a participant receives an invite from a block assembler it contains a prefix, information identifying the blockchain, the key the participant wishes to dilute, the signature of its dilution application, the pool identifier, the key of the block assembler and a signature corresponding to the latter key. Here again only the status of the keys (both of the participant and of the block assembler, and by extension the signature) can be affected by any of the possible state changes: one of the keys might become usable or unusable in a dilution pool.

Participants will only react however to invites they receive concerning a dilution application they have earlier sent out, which means the key in the dilution application must be identical to the participant's key in the invite. This means an honest eligible participant and an honest block assembler must both consider the participant's key to be usable in order for the invite to exist at all.

No such limitation exists for the key of the block assembler though: it is perfectly possible that honest block assemblers consider their keys to be usable, knowing the blockchain is in state S_1, while honest (eclipsed) participants consider these keys to be unusable,

thinking the blockchain is in state S_0. The same problem arises when block assemblers are eclipsed: they might honestly consider their keys to be usable, thinking the blockchain is in state S_0, even though the other honest participants consider them to be unusable, knowing the blockchain is in state S_1. This limits the number of invites the participant will react to and therefore makes it more difficult for eligible participants to have a key diluted and for block assemblers to create a dilution block.

Worse still the block assembler could collude with the attackers and send an invite using a key that is no longer usable in state S_1 but still usable in state S_0, tricking the participant into reacting to an invite that cannot lead to a valid dilution block. The effect is again that it makes it more difficult for participants to have a key diluted. Eclipsed block assemblers can also be tricked into sending an invite using a key that is no longer usable to a colluding participant, who then wastes the block assembler's time and makes it more difficult to create a dilution block.

Pool response: When an eligible participant sends out a pool response it contains a prefix, information identifying the blockchain, the key of the block assembler, the pool identifier, the pool public key of the participant, the key the participant wishes to dilute, the signatures of the dilution application and invite and two signatures corresponding to the key to be diluted. Once again only the status of two of the keys (both of the participant and of the block assembler, and by extension the two signatures) can be affected by any of the possible state changes: one of the keys might become usable or unusable in a dilution pool. The pool public key is always usable.

The only risks concerning this message in a block-based eclipse attack are therefore the risks of the participant's key or the block assembler key being incorrectly deemed usable or unusable, but since these two keys correspond to the keys used in the dilution application and the invite, these risks are limited to the risks already discussed in the previous two paragraphs.

Pool message: When a participant receives a pool message it contains a prefix, information identifying the blockchain, the key of the block assembler, the pool identifier, the session public key of the pool, the number of members in the pool, the keys the participants wish to dilute, the signatures of their dilution application, invite and pool response, the pool public keys of the participants (with the signature provided in the pool response), the session private key encrypted with each of the participants' public keys, and a signature corresponding to the key of the block assembler. The session public key is always considered valid and participants only check the session private key encrypted by their own public keys, which correspond to the public keys used in the previous messages and therefore the participants and the block assembler agree that these are usable. After decryption the session private key is checked for validity by comparing it to the session public key, this does not depend on the state of the blockchain.

Only the status of the keys of the participants and the block assembler can be affected by the any of the possible state changes: again one of the keys might become usable or

unusable in a dilution pool. The block assembler and one of the participants' keys must correspond to the keys used in the earlier messages sent between that participant and the block assembler, but this is not the case for the other participants' keys. There is a risk that one of the keys is usable in state S_1 but not in state S_0, which would lead the eclipsed participant to reject a perfectly valid pool message and consequently make it more difficult to have a key diluted. An eclipsed block assembler might also include a key from a participant who colludes with the attackers, resulting in the other participants rejecting the pool message. This will make it more difficult for the block assembler to create a dilution block.

There is also a risk that a dishonest block assembler might include a key that is usable in state S_0 but not in state S_1, which would lead eclipsed participants to react to a pool message that is actually invalid and can never lead to a valid dilution block, thus wasting these participants' time and again making it more difficult for them to have a key diluted.

Pool acknowledgement: A pool acknowledgement contains a prefix, the index of the acknowledging member's key in the pool message, the contents of the pool message, and a signature corresponding to the member's key. None of this data can be affected by any of the possible state changes except for the pool message, which was already covered in the above, and indirectly the signature, because it depends on the pool message. Therefore there are no extra risks from block-based eclipse attacks on this message.

Blame message: A blame message contains a prefix, the index of the participant within the pool message, the fake private key and the contents of the pool message. The contents of the pool message can at this point no longer be affected by state changes since all the plaintext data in it must be considered valid by both the participant and the block assembler before the blame message can be sent. The prefix, index and fake private key are also not affected by any state changes, so there are no extra risks from block-based eclipse attacks on this message.

New key message: When a participant sends a new key message it contains a prefix, a new key, the contents of the pool message and a signature corresponding to the session private key. The contents of the pool message can at this point no longer be affected by state changes since all the plaintext data in it must be considered valid by both the participant and the block assembler before the new key message can be sent. The new key does not depend on the state of the blockchain since it is introduced here, and neither the prefix nor the signature depend on the state. Therefore there are no extra risks from block-based eclipse attacks on this message.

Unvalidated dilution block message: When a participant receives an unvalidated dilution block message it contains two prefixes, information identifying the blockchain, the hash of the previous block, the chain score at this block, the pool identifier, the key of the

block assembler, the depth of this block, the anonymity set and both the old and new keys of all the participants.

The information identifying the blockchain, the pool identifier, the key of the block assembler and the old keys of the participants were all already featured in the pool message and both the participant and the block assembler must have considered it to be valid in order for this message to have been sent. The new keys originate in the block that is about to be made so they are not affected by the state of the blockchain up until now.

The hash of the previous block and the chain score are however affected by the state: they might be accurate in state S_1 but not in state S_0, which means the eclipsed participant will refuse to sign the dilution block even though it is valid, or an eclipsed block assembler might send an unvalidated dilution block that is not valid. Alternatively a dishonest block assembler might send an unvalidated dilution block message with a hash and chain score that are inaccurate in state S_1 but accurate in state S_0, making the participant sign a block that will not be valid and thus waste time. An eclipsed block assembler might also send an unvalidated dilution block with an inaccurate hash and chain score to attacking participants, who will sign the block and again waste time. In all these cases the risk is that it becomes more difficult for the participant to have a key diluted and for the block assembler to create a dilution block.

Signature message: When a participant sends a signature message it contains a prefix, the pool identifier, the key of the block assembler, the key of the participant and a signature corresponding to this key, and the contents of the dilution block. Besides the prefix and the signature, all of this data was already featured in earlier messages in this process and thus considered valid by both the participant and the block assembler. The prefix does not depend on the state of the blockchain and the signature must correspond to the key that is already considered valid by both the participant and the block assembler. Therefore there are no extra risks from block-based eclipse attacks on this message.

Conclusion: There is a risk that an eclipse attack might filter dilution blocks and thus sabotage dilution applications, invites, pool messages and unvalidated dilution blocks. This makes it more difficult for eclipsed eligible participants to have a key diluted, and when they notice this can break through the eclipse by connecting to new nodes.

Filtering other messages

Besides the blocks created in the Dilution phase, the messages sent during the dilution process can also be blocked in an eclipse attack. When considering the risks of an eclipse attack on honest eligible participants we must therefore consider the risk of each of these messages being blocked.

Dilution application: These messages allow block assemblers to begin a dilution process by responding with an invite. If eclipsed participants are unable to send their dilution

applications, no block assembler will invite them to a dilution pool and therefore it is more difficult for them to have a key diluted. If eclipsed block assemblers are unable to receive dilution applications, they have fewer participants to invite and it becomes therefore more difficult for them to create a dilution block.

Invite: These messages allow the invited participants to take part in a dilution process. If eclipsed participants are unable to receive invites they cannot respond with pool responses and proceed to dilute their keys, therefore it is more difficult for them to have a key diluted. If eclipsed block assemblers are unable to send invites, no one will take part in the dilution pools they create and therefore it becomes more difficult for them to create a dilution block.

Pool response: These messages allow the block assembler to know that the invitees are willing to take part in this dilution pool and they communicate the necessary information to create a pool message. If eclipsed participants are unable to send pool responses, block assemblers will not be able to send the pool message and dilute the participants' keys, therefore it becomes more difficult for them to have a key diluted. If eclipsed block assemblers are unable to receive pool responses, they can never create a pool message and dilute any keys, therefore it becomes more difficult for them to create a dilution block.

Pool message: These messages allow the members of the dilution pool to securely send their new key to the block assembler. If eclipsed participants are unable to receive pool messages, they cannot send a new key to a block assembler, therefore it becomes more difficult for them to have a key diluted. If eclipsed block assemblers are unable to send pool messages, no one will take part in the dilution pools they create and therefore it becomes more difficult for them to create a dilution block.

Pool acknowledgement: These messages allow the members of the dilution pool to know that the pool message was correct and that they can send a new key message without their privacy being compromised. If eclipsed participants are unable to send pool acknowledgements, the other members of the pool cannot send their new key messages and therefore the dilution pool cannot dilute any keys. It thus becomes more difficult for eclipsed participants to have a key diluted. If eclipsed participants are unable to receive pool acknowledgements, they cannot respond with new keys messages and therefore the dilution pool cannot dilute any keys. Therefore it again becomes more difficult for eclipsed participants to have a key diluted. Block assemblers do not do anything with pool acknowledgements if they are not also members of the dilution pool, in which case they will treat pool acknowledgements the same way as regular members of the dilution pool. Therefore there are no extra risks for block assemblers regarding pool acknowledgements.

Blame message: These messages allow the members of the dilution pool to prove that the block assembler has acted in bad faith, by sending a pool message that does not deliver

the same session private key to all members. This in turn allows senders of blame messages to shift the blame of the dilution processes failing from themselves to the block assembler. If eclipsed participants are unable to send blame messages, other participants will be more likely to blame them for failing dilution processes and therefore it becomes more difficult for them to have a key diluted. If eclipsed participants are unable to receive blame messages, they will assign the blame of a failing dilution process to the wrong participant and be more likely to take part in a dilution process with a participant acting in bad faith, which will again make it more difficult for them to have a key diluted.

New key message: These messages allow the members of the dilution pool to anonymously send their new keys to the block assembler. If eclipsed participants are unable to send new key messages, they cannot dilute the new keys in them and the dilution pool will fail. Therefore it becomes more difficult for them to have a key diluted. If eclipsed block assemblers are unable to receive new key messages, they cannot create the dilution block of the dilution pool. Therefore it becomes more difficult for them to create a dilution block.

Unvalidated dilution block message: These messages allow the block assembler to broadcast the dilution block of the dilution pool so the members can sign it. If eclipsed participants are unable to receive unvalidated dilution block messages, they cannot sign the dilution block and the dilution pool will fail. Therefore it becomes more difficult for them to have a key diluted. If eclipsed block assemblers are unable to send unvalidated dilution block messages they cannot procure the signatures necessary to validate the dilution block of the dilution pool and the pool will fail. It is therefore more difficult for them to create a dilution block.

Signature message: These messages allow the members of the dilution pool to provide the block assembler with the signatures the dilution block needs. If eclipsed participants are unable to send signature messages, they cannot validate the dilution block and the dilution pool will fail. Therefore it becomes more difficult for them to have a key diluted. If eclipsed block assemblers are unable to receive signature messages, they cannot add the necessary validation to the dilution block of the dilution pool. Therefore it becomes more difficult for them to create a dilution block.

Conclusion: An eclipse attack in the Dilution phase targeting an eligible participant can therefore make it more difficult for the participant to have a key diluted or to create dilution blocks. Since participants can see their success or failure in these two areas reflected in the responses they get to their messages and the state of the blockchain, they might conclude from repeated failure that they are likely to be eclipsed and thus connect to new nodes in order to break the eclipse.

9.3.2 Victimized election managers

If the *election manager* is eclipsed there will be no effect as long as the election manager does not create a new block: election managers do not base their actions on which keys are and are not valid during the Dilution phase. Once the election manager does create a new block, the result might be that it is appended to a block that is actually not the last block in the chain.

This creates a fork which will not be resolved by the first criterion: since this is the Dilution phase the competing block is either a dilution block, a pre-depth block or a depth block and in all those cases the chain score of the branches is used to resolve the fork. The competing branch contains only dilution blocks, since the other three blocks that are created in this phase are created by the election manager and therefore knowledge of them cannot be withheld from the election manager, and the blocks in subsequent phases cannot be appended since it is the election manager who ends the phase and thus must have knowledge that the phase has ended.

There are now two possibilities: either the election manager's branch wins the fork, and the dilution blocks on the other branch are orphaned, or the election manager's branch loses the fork, and the manager's block is orphaned. We will cover each of these two cases in their owns sections, followed by a section calculating the maximum amount of signature dilution that can be undone due to such a fork.

Manager losing fork

If the manager's block loses the fork, the effects depend on what kind of block the manager's block is: if it is a *pre-depth or depth block*, eligible participants will simply continue creating dilution blocks on the winning branch. If the election manager receives these blocks, the eclipse is broken. If not, the election manager will notice a conspicuous lack of dilution blocks, which is curious since the pre-depth and depth blocks actually serve to enable participants to dilute their keys further. Election managers can thus guess that they are being eclipsed, allowing them to break the eclipse by connecting to new nodes.

If the manager's losing block is a *dilution end block*, the Dilution phase is not ended yet. The election manager however thinks that the phase has been ended, since that is the case according to the branch of the blockchain that includes the dilution end block. Since we assumed the existence of social discourse though the election manager will be aware that the consensus of all nodes is that the Dilution phase has not yet ended. Eclipsed election managers can thus become aware of the eclipse and break it by connecting to new nodes, and end the Dilution afterwards by appending a dilution end block to the actual last block of the blockchain. The effects of an eclipse attack are thus limited to a minor inconvenience for the election manager.

Manager winning fork

If the branch of the election manager wins the fork, the effects again depend on what kind of block the manager's block is: if it is a pre-depth block or a depth block, the

members of the associated dilution pool can immediately create a new dilution block that is functionally the same and thus negate any effects of the eclipse attack. If it is a dilution end block however, a certain amount of signature dilution is undone, negatively impacting the privacy of some eligible participants.

The Dilution phase ends now, so the effects of an eclipse during this phase will be limited to this single fork. (There may still be effects of an eclipse during the two following phases, but they are covered in their respective paragraphs below.) The dilution end block increases the chain score by 32 and its branch wins the fork according to the third criterion, i.e. in case the chain scores of the two branches are equal. Therefore the branch of the dilution block wins if the dilution blocks on the competing branch together increase the chain score by at most 32, and it loses if they increase the chain score by more than 32. The amount by which these dilution blocks increase the chain score is equal to the sum of their δ, the number of distinct anonymity sets that is diluted in each block, minus one per block:

$$C_{b_l} - C_{b_m} = \sum_{j=m+1}^{l} \delta_{b_j} - 1 = (\sum_{j=m+1}^{l} \delta_{b_j}) - (l - m)$$

Where m is the index of the last block before the fork occurs and l is the length of the competing chain, and thus the index of the last block in it. Therefore $(l - m)$ is the number of dilution blocks on the competing branch. If the chain score increases by at most 32, this means:

$$C_{b_l} - C_{b_m} = (\sum_{j=m+1}^{l} \delta_{b_j}) - (l - m) \leq 32$$

This does not impose any limit on the number of dilution blocks: the δ of a dilution block can be 1 and such a block does not increase the chain score at all. Such a block is however useless in terms of signature dilution, so all useful dilution blocks have a δ of 2 or higher. If we maximize the sum of the δ of all useful dilution blocks on the orphaned branch, we must maximize the number of useful dilution blocks and minimize the δ of each dilution block to 2:

$$(\sum_{j=m+1}^{l} \delta_{b_j}) - (l - m) = \sum_{j=m+1}^{l} 2) - (l - m) = 2(l - m) - (l - m) = l - m = 32$$

In this case there are 32 dilution blocks that each doubled an anonymity set.

We can also maximize the average δ of all useful dilution blocks on the orphaned branch. Since the average is the sum divided by the number of blocks $l - m$, it increases as the sum increases and decreases as the sum decreases, but it also increases as $l - m$ decreases and decreases as $l - m$ increases. We know that the difference of the sum and $l - m$ is a constant number 32 if we are maximizing and thus if a value is added to one term, the

same value must be added to the other term. Since we are attempting to maximize the quotient of these two terms, that means the smallest of the two terms must be minimized. The smallest term is $l - m$ since δ is at least 2 and therefore the sum is at least $2(l - m)$, and the smallest number of blocks on a branch is 1, therefore:

$$(\sum_{j=m+1}^{l} \delta_{b_j}) - (l - m) = \sum_{j=m+1}^{m+1} \delta_{b_j}) - 1 = \delta_{b_{m+1}} - 1 = 32$$

$$\delta_{b_{m+1}} = 33$$

$$\frac{\delta_{b_{m+1}}}{l - m} = \frac{33}{1} = 33$$

Maximum anonymity set increase

The factor 33 is in fact also the maximum factor δ_{branch} by which the average anonymity set that existed at the point of the fork and that was diluted afterwards, can be diluted in a branch where the chain score increases by $C_{branch} = 32$. In general for a branch where the chain score increases by C_{branch} the factor by which the average anonymity set that existed at the point of the fork and that was diluted afterwards, can be diluted, is $\delta_{branch} \leq C_{branch} + 1$, as we shall prove by (strong) induction.
Proof:
If $C_{branch} = 1$ the only possible useful dilution block is a single block b with $\delta_b = 2$, where two anonymity sets are diluted: a block with a single anonymity set would not be useful, and a block with two anonymity sets already increases the chain score by 1 on its own. Since the branch only consists of this block, all anonymity sets that existed at the start of the branch and are diluted in the branch, must be diluted in this block and therefore:

$$\delta_{branch} = \delta_b = 2 = 1 + 1 = C_{branch} + 1$$

Therefore in the base case where $C_{branch} = 1$ it is true that $\delta_{branch} \leq C_{branch} + 1$.
We shall now prove that, under the hypothesis:

$$\forall x \in \{1, ..., L\} : C_{branch} = x => \delta_{branch} \leq C_{branch} + 1$$

... the following holds:

$$C_{branch} = L + 1 => \delta_{branch} \leq C_{branch} + 1$$

The existence of dilution blocks that are not useful since they only use a single anonymity set as input, does not contribute anything to either the chain score C_{branch} or the factor δ_{branch}. For all practical purposes a branch containing such a dilution block is the same as a branch without this block. We shall therefore only consider branches in which those dilution blocks do not exist and extend the proof of such branches to all branches where those dilution blocks do exist.

Among the anonymity sets A_{branch} that existed at the start of the branch and are diluted during it, there are two subsets: the anonymity sets A_{end} that are directly or indirectly diluted by the last block b_{end} in the chain, and the anonymity sets A_{early} that are not diluted either directly or indirectly by b_{end}. "Indirectly diluting" here has the following meaning: a block b indirectly dilutes an anonymity set a if and only if it directly dilutes a dilution set a', which originates in a block b' that itself dilutes a either directly or indirectly. Blocks in A_{branch} must fall in either category, so $A_{end} \cup A_{early} = A_{branch}$. The sets in A_{early} are not diluted by b_{end} but they must be diluted, directly or indirectly, in some blocks B_{early} in the branch. We can form a superset of B_{early} called B_{early+}: this set also includes all blocks in the branch that output an anonymity set that any block in B_{early+} uses as input. B_{early+} is a proper subset of the branch, since it can only contain blocks that are in the branch but does not contain the last block in the branch. The last block in the branch b_{end} dilutes at least 2 anonymity sets and must therefore contribute at least 1 to C_{branch}. The blocks B_{early+} therefore contribute less than $C_{branch} = L + 1$ to the chain score. We shall refer to this contribution as C_{early+}.

If all blocks that are not in B_{early+} are removed from the branch, we still have a valid branch since we included all the blocks in the branch that form an anonymity set that is used as input in any other block in B_{early+}. Their contribution to the chain score is C_{early+} and there are no other blocks to contribute anything to the chain score in this branch, so this is a branch where the chain score increases by $C_{early+} < C_{branch} = L + 1$ and so we know by the induction hypothesis that the factor by which the average anonymity set that is diluted in this branch, is diluted, is $\delta_{early+} \leq C_{early+} + 1 < L + 2$ and thus $\delta_{early+} \leq L + 1 = C_{branch}$.

The blocks diluting A_{end} form a subset B_{end} of the branch since they can only contain blocks in the branch by definition, but this is not necessarily a proper subset of the branch since B_{early} may be empty. Since they include all blocks diluting the anonymity sets in the last block b_{end} both directly and indirectly, they include all blocks in the branch that output an anonymity set that any block in B_{end} uses as input and therefore they form a valid branch as well. We shall name the amount by which this branch increases the chain score C_{end} and since we know that this is the contribution of a subset of the branch, we know that $C_{end} \leq C_{branch} = L + 1$. We can create a subset of B_{end} by removing the last block b_{end} and name this subset B_{before_end}. The blocks in this subset form a valid branch since they come from a set B_{end} that forms a valid branch and only the last block is removed. We name the contribution of this branch to the chain score C_{before_end} and we get:

$$C_{end} = \delta_{b_{end}} - 1 + C_{before\_end} \leq L + 1$$

$$\delta_{b_{end}} + C_{before\_end} \leq (L + 1) + 1 = C_{branch} + 1$$

If B_{before_end} is the empty set, its contribution $C_{before\_end} = 0$ and the above formula becomes:

$$\delta_{b_{end}} + C_{before\_end} = \delta_{b_{end}} + 0 = \delta_{b_{end}} \leq C_{branch} + 1$$

Since B_{before_end} is the empty set, B_{end} only consists of b_{end} and thus:

$$\delta_{B_{end}} = \delta_{b_{end}} \leq C_{branch} + 1$$

We shall now consider the case where B_{before_end} is not the empty set. Since b_{end} is a useful dilution block we know $\delta_{b_end} > 1$ and therefore:

$$\delta_{b_{end}} + C_{before\_end} > 1 + C_{before\_end}$$

Combined with the above this gives us:

$$C_{before\_end} + 1 \leq L + 1$$
$$C_{before\_end} \leq L$$

Combined with the induction hypothesis we now know:

$$\delta_{before\_end} \leq C_{before\_end} + 1$$
$$\delta_{before\_end} - 1 \leq C_{before\_end}$$
$$\delta_{b_{end}} + \delta_{before\_end} - 1 \leq C_{branch} + 1$$

The δ_{before_end} anonymity sets are diluted down to a fewer number of anonymity sets, but this cannot be fewer than 1 if B_{before_end} is not the empty set. These sets must be diluted by b_{end} by the definition of B_{before_end}, which means that at least one of the δb_{end} anonymity sets that b_{end} dilutes must originate in one of the blocks in B_{before_end}. This means:

$$\delta_{B_{end}} < \delta_{b_{end}} + \delta_{before\_end}$$
$$\delta_{B_{end}} \leq \delta_{b_{end}} + \delta_{before\_end} - 1 \leq C_{branch} + 1$$

Therefore no matter whether B_{before_end} is empty or not, we know:

$$\delta_{B_{end}} \leq C_{branch} + 1$$

We can take the following two propositions we have proven:

$$\delta_{early+} \leq C_{branch}$$
$$\delta_{B_{end}} \leq C_{branch} + 1$$

Each δ is an average statistic over a set of blocks. An average statistic over a union of two sets can never be larger than the same average statistic over both of the two sets, so we know:

$$\delta_{early+ \cup B_{end}} \leq C_{branch} + 1$$

We know that $A_{early} \subseteq A_{early+}$ and $A_{early} \cup A_{end} = A_{branch}$, therefore $A_{branch} \subseteq A_{early+} \cup A_{end}$. Both A_{early+} and A_{end} are subsets of A_{branch}, so we know $A_{early+} \cup A_{end} \subseteq A_{branch}$ and thus $A_{early+} \cup A_{end} = A_{branch}$. This means:

$$\delta_{branch} \leq C_{branch} + 1$$

Therefore δ_{branch} is at most $C_{branch} + 1 = 33$ when the branch is orphaned due to the election manager being eclipsed. The effects of an eclipse attack targeting the election manager during the Dilution phase are therefore that a small amount of signature dilution can be undone.

Conclusion

When election managers are eclipsed, this can lead to some minor inconveniences for them but also to an amount of signature dilution being undone. The maximum average factor by which anonymity sets increase during this undone signature dilution is 33.

9.4 Commitment phase

During the Commitment phase, eligible participants create commitment blocks and append them to the blockchain. If eclipsed participants are unable to send a commitment block, they cannot use that commitment block to create a vote message in the Voting phase and therefore they are disenfranchised. If eclipsed participants are unable to receive commitment blocks, they likely cannot append their own commitment blocks to the last block in the chain which again results in them being disenfranchised.

It is worth considering why a participant might be eclipsed in this phase: since the only effect is disenfranchisement, the attackers must be aiming for this. The attackers however do not know who the participant is they are disenfranchising, nor do they know the content of the vote since it is hashed, so they only know they are lowering the turnout without knowing in which direction they are steering the result.

Depending on the implementation of the anonymous broadcast channel the participants use however, the commitment blocks may be sent in encrypted form from the geographical location of the participants, which allows attackers to eclipse encrypted messages originating at certain geographical locations. Attackers could use this information to estimate the content of the vote and therefore use the eclipse attack to steer the election away from candidates they do not support. For this to work however the victim node must be adjacent only to nodes that estimate the victim node will vote for a candidate they do not support. This becomes significantly less likely if the victims makes sure to connect to nodes affiliated with all candidates: their preferred candidate must incorrectly estimate

that the victims will vote for someone else, while the other candidates correctly estimate that the victims do not voter for them, and at the same time all candidates are carrying out an eclipse attack.

No matter how unlikely this scenario may be, it is still possible. In this case however the victims will notice that their commitment blocks are not appended to the chain and they can connect to more nodes to break the eclipse.

9.5 Voting phase

During the Voting phase, voters broadcast the content of their vote using vote messages. The content must correspond to the hash they published in their commitment blocks. If eclipsed voters are unable to send their vote messages, they will not be included in the tally and thus the voters are disenfranchised. Voters will however notice that their own tally differs from the tally communicated in the communication channel stipulated in assumption 1 and thus realize they are being targeted in an eclipse attack. They can then connect to new nodes to break the eclipse and allow their votes to be tallied.

9.6 Conclusion

The effects of eclipse attacks are very limited. Eligible participants can be disenfranchised in the Registration phase, but this requires the election manager to be one of the attackers and the disenfranchised eligible participants can prove this. Signature dilution can be made more difficult by an eclipse attack, but if this problem becomes too big the victims will simply notice they are being attacked and break the eclipse. When the election manager is eclipsed, a small amount of signature dilution can be undone. This negatively impacts privacy of voters but they can compensate in advance by diluting their signatures further than they really need to. Eclipse attacks in the Commitment phase are extremely unlikely and will be noticeable to the victims, who can then break the eclipse. The same applies to the Voting phase.

Chapter 10

Blockchain forks

In a blockchain, forks may arise and the protocol must be able to resolve these to ensure the consistency of the blockchain. This can be considered a form of robustness. When a fork exists in the blockchain, it means there is an initial part of the chain $\{b_1, ..., b_f\}$ that is undisputed and ends in b_f, the first block before the fork, followed by a finite set of n branches $B = \{\beta_1, ..., \beta_n\}$. Each branch β_i consists of the latter part of the chain $\{b_{f+1,\beta_i}, ..., b_{l_{\beta_i},\beta_i}\}$, with l_{β_i} being the length of the blockchain according to this branch which is $l_{\beta_i} > f$. The initial part of the chain is defined as being as long as possible, that is to say that b_f is the last undisputed block in the chain of undisputed blocks. This means that, given two branches β_1 and β_2 the first blocks of each branch b_{f+1,β_1} and b_{f+1,β_2} must always be different. In order for the blockchain to function, there must be a consensus mechanism that resolves forks: for each fork and its associated finite set of branches B, if B is not the empty set there must be one branch β_{win} that is in that set and that is deterministically chosen by the consensus mechanism from each subset of B that includes β_{win}. We can show that this is the case by constructing a strict total order over all the branches that can exist for any given fork: a strict total order over a finite non-empty set S defines exactly one maximum element e_{max} in S and it will be the maximum element of any subset of S that includes e_{max}. The consensus mechanism then chooses e_{max} to resolve the fork.

We will now prove that the fork resolution protocol is such a consensus mechanism by showing that, for every possible b_f, the fork resolution protocol defines a strict total order where the maximum element is chosen to resolve the fork. A strict total order on a set S is defined by a relationship $<$ between every $a, b, c \in S$ where:

1. Not $a < a$. (Irreflexivity)
2. If $a < b$ then not $b < a$. (Asymmetry)
3. If $a < b$ and $b < c$ then $a < c$. (Transitivity)
4. If $a \neq b$ then $a < b$ or $b < a$. (Connectedness)

The relationship $<$ we use to define the strict total orders used in the resolution is the relationship where $a < b$ means b wins the fork against a: this means the maximum element is a branch that wins the fork against every other branch. The last block before the fork b_f can be categorized according to the phase the election is in immediately after b_f and we will construct a strict total order for each of these phases:

10.1 Setup phase

This phase only consists of the initialization block, which ends this phase and is not preceded by any other block. It is thus not possible to find a block b_f for which it is true that the blockchain is in the Setup phase immediately after b_f, so we do not need to prove that a strict total order exists in this phase.

10.2 Registration phase

10.2.1 Irreflexivity

First we shall prove that for every branch β it is true that: Not $\beta < \beta$. If $\beta < \beta$ were true, the protocol should resolve a fork between β and itself using any of the three criteria listed in 4.6. The first criterion can only resolve forks between branches that start with a block b_{f+1} of a different type, but β cannot start with a different block than itself so the first criterion does not resolve a fork between β and itself. The second criterion can only resolve forks between branches that have different chain scores, but β has the exact same chain score as itself so the second criterion does not resolve a fork between β and itself.

The third criterion can resolve blocks between blocks of the same type if they are registration blocks, dilution blocks, dilution end blocks or commitment blocks. Of those four types of blocks only registration blocks can be appended in this phase. The third criterion uses the voter IDs of the blocks to resolve the fork only if the two blocks have unequal voter IDs, but β must necessarily have the same voter ID as itself. The third criterion alternatively uses the public keys of the blocks to resolve the fork, but β contains the same public key as itself so this cannot resolve the fork either. Therefore the protocol cannot resolve a fork between β and itself in the Registration phase and thus $\beta < \beta$ is not true.

10.2.2 Asymmetry

We shall now prove that for every two branches β_1, β_2: if $\beta_1 < \beta_2$ then not $\beta_2 < \beta 1$. There are two types of blocks in this phase: the registration block and the dilution start block. Given b_f, all contents of a dilution start block are fixed and thus there is only one possible dilution start block. The dilution start block always wins a fork against a registration block in the first criterion and we have shown in the preceding paragraph that it cannot win or lose a fork against itself, so if $\beta_1 < \beta 2$ we know that b_{f+1,β_1} cannot be a dilution

start block and must therefore be a registration block. This leaves two possibilities: either b_{f+1,β_2} is a dilution start block or it is a registration block as well.

If b_{f+1,β_2} is a *dilution start block* and b_{f+1,β_1} a registration block, the fork is resolved according to the first criterion and β_2 wins, therefore $\beta_2 < \beta_1$ is not true.

If b_{f+1,β_2} and b_{f+1,β_1} are both *registration blocks*, the fork is not resolved according to the first criterion, so it must be resolved by either the second or the third criterion.

If it is resolved according to the second criterion, the chain scores of the branches must be different and $\beta_1 < \beta_2$ means the chain score of β_2 is higher than that of β_1. Since the chain score is an integer and the "higher than" relationship defines a strict total order over the integers, this means the chain score of β_1 cannot be higher than that of β_2 and thus $\beta_2 < \beta_1$ cannot be true.

If the fork is resolved according to the third criterion, either the voter ID of b_{f+1,β_1} and b_{f+1,β_2} is used to resolve the fork or not. If the voter ID is used, it is interpreted as a number and again a strict total order over the numbers is defined. This means that, since $\beta_1 < \beta_2$, the voter ID of b_{f+1,β_1} must be higher than the voter ID of b_{f+1,β_2} and thus the voter ID of b_{f+1,β_1} cannot be lower than the voter ID of b_{f+1,β_2}, so $\beta_2 < \beta_1$ cannot be true.

If the voter ID is not used to resolve the fork, the only other way to resolve the fork is using the public keys of b_{f+1,β_1} and b_{f+1,β_2} and again interpreting them as a number and seeing which is lower. This is once again a strict total order, so we know if the public key of b_{f+1,β_1} is higher than the public key of b_{f+1,β_2}, it cannot be true that the public key of b_{f+1,β_1} is lower than the public key of b_{f+1,β_2} and thus $\beta_2 < \beta_1$ cannot be true.

Therefore there is no case where $\beta - 1 < \beta_2$ is true but $\beta_2 < \beta_1$ is also true.

10.2.3 Transitivity

We shall now prove that for every three branches $\beta_1, \beta_2, \beta_3$: if $\beta_1 < \beta_2$ and $\beta_2 < \beta_3$ then $\beta_1 < \beta_3$. The first block of each of these branches is either a registration block or a dilution start block. We have shown in the above that a branch starting with a dilution start block cannot lose a fork, so since we know $\beta_1 < \beta_2$ and $\beta_2 < \beta_3$, we know that b_{f+1,β_1} and b_{f+1,β_2} are registration blocks.

This leaves two possibilities: either b_{f+1,β_3} is a dilution start block or b_{f+1,β_3} is a registration block. If it is a dilution start block, it wins all forks against registration blocks according to the first criterion, so since we know b_{f+1,β_1} is a registration block we know $\beta_1 < \beta_3$. If b_{f+1,β_3} is a registration block, then the fork between β_2 and β_3 must be resolved either by the second or the third criterion.

If it is resolved by the second criterion, the chain score of β_3 is higher than the chain score of β_2. The fork between β_1 and β_2 is also resolved by either the second or the third criterion. If it is resolved by the second criterion, the chain score of β_2 is higher than the chain score of β_1. Since the chain score is a number and therefore a strict total order applies, this means that the chain score of β_3 is higher than the chain score of β_1 and thus $\beta_1 < \beta_3$. If the fork between β_1 and β_2 is resolved by the third criterion, the chain scores of β_1 and β_2 must be equal. Since we already know that the chain score of β_2 is lower than the chain score of β_3, this means the chain score of β_1 is lower than the chain score of β_3

and therefore $\beta_1 < \beta_3$. Therefore if the fork between β_2 and β_3 is resolved according to the second criterion, we know $\beta_1 < \beta_3$.

If the fork between β_2 and β_3 is resolved according to the third criterion, the chain scores of β_2 and β_3 are equal. The fork between β_1 and β_2 is again resolved either according to the second or the third criterion. If it is resolved according to the second criterion, the chain score of β_1 is lower than the chain score of β_2, which we know to be equal to the chain score of β_3. Therefore we know that the chain score of β_1 is lower than the chains core of β_3 and thus $\beta_1 < \beta_3$. If the fork between β_1 and β_2 is resolved according to the third criterion, then the voter ID of b_{f+1,β_1} is either equal to the voter ID of b_{f+1,β_2} or higher. The voter ID of b_{f+1,β_2} must also be either equal to or higher than the voter ID of b_{f+1,β_3}. If the voter ID of b_{f+1,β_1} is higher than the voter ID of b_{f+1,β_2}, it will also be higher than the voter ID of b_{f+1,β_3} since it is equal to or lower than the voter ID of b_{f+1,β_2}, and thus we know that $\beta_1 < \beta_3$.

If the voter ID of b_{f+1,β_1} is equal to the voter ID of b_{f+1,β_2}, and the voter ID of b_{f+1,β_2} is equal to the voter ID of b_{f+1,β_3}, the voter ID of b_{f+1,β_1} is equal to the voter ID of b_{f+1,β_3}. In this case the public key of b_{f+1,β_1} must be higher than the public key of b_{f+1,β_2}, and the public key of b_{f+1,β_2} must be higher than the public key of b_{f+1,β_3}, which means the public key of b_{f+1,β_1} must be higher than the public key of b_{f+1,β_3} due to the public keys being interpreted as numbers which are strictly totally ordered. Therefore the third criterion resolves this fork as $\beta_1 < \beta_3$.

Therefore for all possible branches in the Registration phase we know that: if $\beta_1 < \beta_2$ and $\beta_2 < \beta_3$ then $\beta_1 < \beta_3$.

10.2.4 Connectedness

We shall now prove that for every two branches β_1, β_2: if $\beta_1 \neq \beta_2$ then $\beta_1 < \beta_2$ or $\beta_2 < \beta_1$. By definition of the fork we know that $b_{f+1,\beta_1} \neq b_{f+1,\beta_2}$, so they cannot both be dilution start blocks since there is only one possible dilution start block if its predecessor is already defined.

It is however possible that one of the two is a *dilution start block* and the other a registration block, in which case the dilution start block wins the fork according to the first criterion. If b_{f+1,β_1} is the dilution start block then $\beta_2 < \beta_1$, and if b_{f+1,β_2} is the dilution start block then $\beta_1 < \beta_2$. Therefore whenever one of the two branches starts with a dilution start block we know $\beta_1 < \beta_2$ or $\beta_2 < \beta_1$.

If both b_{f+1,β_1} and b_{f+1,β_2} are *registration blocks* the fork cannot be resolved according to the first criterion, so we have to see if the second or third criterion can resolve it. If the chain scores of the two branches are different, this will resolve the fork: if the chain score of β_2 is higher than that of β_1 then $\beta_1 < \beta_2$, and if the chain score of β_1 is higher than that of β_2 then $\beta_2 < \beta_1$.

If the chain scores are equal however the second criterion is inconclusive and only the third criterion could possibly resolve the fork. When comparing two registration blocks, the third criterion initially compares the voter IDs of each registration block and resolves the fork if these voter IDs are different: if the voter ID of b_{f+1,β_1}, interpreted as a number,

is lower than the voter ID of b_{f+1,β_2}, interpreted as a number, then $\beta_2 < \beta_1$. If the voter ID of b_{f+1,β_2}, interpreted as a number, is lower than the voter ID of b_{f+1,β_1}, then $\beta_1 < \beta_2$. Therefore if the two voter IDs are not equal, then either $\beta_1 < \beta_2$ or $\beta_2 < \beta_1$.

If b_{f+1,β_1} and b_{f+1,β_2} have the same voter ID, then the protocol cannot use it to resolve the fork and instead uses the public keys of b_{f+1,β_1} and b_{f+1,β_2}: if the public key of b_{f+1,β_1}, interpreted as a number, is lower than the public key of b_{f+1,β_2}, interpreted as a number, then $\beta_2 < \beta_1$. If the public key of b_{f+1,β_2}, interpreted as a number, is lower than the public key of b_{f+1,β_1}, then $\beta_1 < \beta_2$. Therefore if the public keys are not equal, then either $\beta_1 < \beta_2$ or $\beta_2 < \beta_1$.

Therefore the only case that cannot be resolved is the case where the first blocks of both branches are registration blocks with the same voter ID and the same public key. They must also have the same prefix since this does not change and they must have the same information identifying the blockchain, index, predecessor hash and chain score, so all the data the signature is calculated over is the same. The signature must also be the same since it is the signature of the election manager, so all the data in the two blocks is the same and therefore the two blocks are the same, which contradicts the definition of the fork.

Therefore if $\beta_1 \neq \beta_2$ then $\beta_1 < \beta_2$ or $\beta_2 < \beta_1$.

10.2.5 Conclusion

The fork resolution protocol therefore satisfies all four requirements to be a strict total order in the Registration phase.

10.3 Dilution phase

In the Dilution phase, four types of blocks exist: the dilution block, the pre-depth block, the depth block and the dilution end block. Given b_f, all contents of a pre-depth block are fixed and thus there is only one possible pre-depth block. The same logic applies to depth blocks. Furthermore the initial part of the chain $\{b_1, ..., b_f\}$ determines whether it is even possible to append a pre-depth block or a depth block: the two are in fact in complementary distribution. The forking protocol treats the two blocks the same as well: the first criterion does not do anything, the second criterion simply compares the chain scores, and according to the third criterion both these blocks win against a dilution block but lose against a dilution end block. We can thus group these two types of blocks together as a (pre-)depth block and reason on the basis of this abstract type.

10.3.1 Irreflexivity

First we shall prove that for every branch β it is true that: Not $\beta < \beta$. If $\beta < \beta$ were true, the protocol should resolve a fork between β and itself using any of the three criteria listed in 4.6. The first criterion cannot resolve forks in the Dilution phase. The second criterion

can only resolve forks between branches that have different chain scores, but β has the exact same chain score as itself so the second criterion does not resolve a fork between β and itself.

The third criterion can resolve blocks between blocks of the same type if they are registration blocks, dilution blocks, dilution end blocks or commitment blocks. Of those four types of blocks only dilution blocks and dilution end blocks can be appended in this phase.

If β's first block is a *dilution block* the third criterion uses the first old public key of each block to resolve the fork only if the two blocks have unequal first old public keys, but $b_{f+1,\beta}$ must necessarily have the same first old public key as itself.

The third criterion then looks at the indices of the origin blocks of these keys, but $b_{f+1,\beta}$ must again have the same index as itself. The third criterion then attempts the same with the second old public key and its origin block index and continues with all the other old public keys until one is found that is different (or its origin block index is different), which is not the case since $b_{f+1,\beta}$ is compared to itself. The number of public keys in the blocks is then compared, but $b_{f+1,\beta}$ has the same number of public keys as itself so this does not resolve the fork either.

The third criterion subsequently proceeds to compare the new public keys in the same way as the old public keys were compared, so this cannot resolve the fork for the same reason: the list of new public keys in $b_{f+1,\beta}$ is the same as itself. Then the third criterion compares the block assembler of the blocks to be compared, but $b_{f+1,\beta}$ must have the same block assembler as itself.

Finally the third criterion compares the pool identifiers of the blocks to be compared, but $b_{f+1,\beta}$ must have the same pool identifier as itself. The protocol can thus not resolve a fork between β and itself if $b_{f+1,\beta}$ is a dilution block.

If $b_{f+1,\beta}$ is a *dilution end block* the third criterion compares the hash of the block to itself, which will be inconclusive. It then compares the size of the block to itself, which will also be inconclusive. Lastly it compares the entire content of the block to itself, which will again be inconclusive and thus the protocol cannot resolve a fork between β and itself if $b_{f+1,\beta}$ is a dilution end block either.

Therefore the protocol cannot resolve a fork between β and itself in the Dilution phase and thus $\beta < \beta$ is not true.

10.3.2 Asymmetry

We shall now prove that for every two branches β_1, β_2: if $\beta_1 < \beta_2$ then not $\beta_2 < \beta 1$. The fork between these two branches cannot be resolved according to the first criterion since this is the Dilution phase, so it is either resolved according to the second criterion or according to the third criterion.

If it is resolved according to the second criterion, the chain scores of the branches must be different and $\beta_1 < \beta_2$ means the chain score of β_2 is higher than that of β_1. Since the chain score is an integer and the "higher than" relationship defines a strict total order over

the integers, this means the chain score of β_1 cannot be higher than that of β_2 and thus $\beta_2 < \beta_1$ cannot be true.

If it is resolved according tot the third criterion, either the two blocks are of the same type or they are of a different type. If they are of a different type, b_{f+1,β_2} cannot be a dilution block since it loses to all other types, and b_{f+1,β_1} cannot be a dilution end block since it wins against all other types. This leaves three possibilities: either b_{f+1,β_1} is a dilution block and b_{f+1,β_2} is a (pre-)depth block, in which case $\beta_2 < \beta 1$ cannot be true, or b_{f+1,β_1} is a dilution block and b_{f+1,β_2} is a dilution end block, in which case $\beta_2 < \beta 1$ also cannot be true, or b_{f+1,β_1} is a (pre-)depth block and b_{f+1,β_2} is a dilution end block, in which case $\beta_2 < \beta 1$ again cannot be true. Therefore when the two blocks have different types, $\beta_2 < \beta 1$ cannot be true.

If the two blocks have the same type, the fork resolution depends on which type it is. It cannot be a (pre-)depth block since there is only one possible (pre-)depth block for a given b_f, so there would not be a fork in this case. The two blocks must therefore either be dilution blocks or dilution end blocks.

If they are *dilution blocks*, either one of the old public keys is used to resolve the fork or not. If one of the old public keys is used, it is interpreted as a number and again a strict total order over the numbers is defined. This means that, since $\beta_1 < \beta_2$, the old public key of b_{f+1,β_1} at a given index i must be higher than the old public key of b_{f+1,β_2} at index i and thus the old public key of b_{f+1,β_1} at index i cannot be lower than the old public key of b_{f+1,β_2} at index i, so $\beta_2 < \beta_1$ cannot be true.

If no old public key is used to resolve the fork, the index of the origin block of one of the old public keys is used if it differs. Since this is a number a strict total order is defined, and since $\beta_1 < \beta_2$ the origin index of β_1 at index i must be higher than the origin index of β_2 at index i, so the origin index of β_2 at index i cannot be higher than the origin index of β_1 at index i and therefore $\beta_2 < \beta_1$ cannot be true.

If the indices of the origin blocks of all old public keys of the two blocks are the same, the number of old public keys is used if it is different: b_{f+1,β_1} must have fewer old keys than b_{f+1,β_2}, so b_{f+1,β_1} cannot have more old keys than b_{f+1,β_2} and therefore $\beta_2 < \beta_1$ cannot be true.

If the number of old keys is the same, and the old keys and their origin indices are also the same in every case, the third criterion attempts to resolve the fork by using the new public keys: unless all the new public keys are the same, the new public key of b_{f+1,β_1} at a given index i must be higher than the new public key of b_{f+1,β_2} at index i because $\beta_1 < \beta_2$, and thus the new public key of b_{f+1,β_1} at index i cannot be lower than the new public key of b_{f+1,β_2} at index i, so $\beta_2 < \beta_1$ cannot be true.

If the new keys are all the same, the block assemblers of b_{f+1,β_1} and b_{f+1,β_2} are compared: if they are not the same, the block assembler of b_{f+1,β_1} must be higher than the block assembler of b_{f+1,β_2}, which means it cannot be lower than the block assembler of b_{f+1,β_2} and thus $\beta_2 < \beta_1$ cannot be true.

Lastly if all else is the same, the pool identifiers of the two blocks are compared: since $\beta_1 < \beta_2$, the pool identifier of b_{f+1,β_1} must be higher than the pool identifier of b_{f+1,β_2}, so $\beta_2 < \beta_1$ cannot be true. It is thus not possible for $\beta_2 < \beta_1$ to be true if $\beta_1 < \beta_2$ is true and

the initial blocks of the branches are dilution blocks.

It the two initial blocks of the branches are *dilution end blocks*, the third criterion first compares the hashes of the two blocks: if they are not equal, the hash of b_{f+1,β_1} must be higher than the hash of b_{f+1,β_2} when interpreted as a number, since $\beta_1 < \beta_2$, so the hash of b_{f+1,β_1} cannot be lower than the hash of b_{f+1,β_2} when interpreted as a number and therefore $\beta_2 < \beta_1$ cannot be true.

If the hashes are the same, the size of the blocks is compared: if the size is different, b_{f+1,β_1} must be larger than b_{f+1,β_2}, so b_{f+1,β_1} cannot be smaller than b_{f+1,β_2} and therefore $\beta_2 < \beta_1$ cannot be true. Lastly if both the hashes and the sizes are equal, the entire contents of each block is compared, interpreted as a number: since $\beta_1 < \beta_2$ we know b_{f+1,β_1} must be higher than b_{f+1,β_2}, so b_{f+1,β_1} cannot be lower than b_{f+1,β_2} and therefore $\beta_2 < \beta_1$ cannot be true.

Therefore there is no case where $\beta_1 < \beta_2$ is true but $\beta_2 < \beta_1$ is also true.

10.3.3 Transitivity

We shall now prove that for every three branches $\beta_1, \beta_2, \beta_3$: if $\beta_1 < \beta_2$ and $\beta_2 < \beta_3$ then $\beta_1 < \beta_3$. Since this is the Dilution phase, all three of these forks must be resolved according to either the second or the third criterion. This means the chain score of β_1 must be lower than or equal to the chain score of β_2, and the chain score of β_2 must be lower than or equal to the chain score of β_3, which in turn means the chain score of β_1 is lower than or equal to the chain score of β_3.I f it is lower, the fork is resolved according to the second criterion and $\beta_1 < \beta_3$. If the chain scores are equal, all three forks must be resolved according to the third criterion.

The resolution of the third criterion depends on whether the blocks are of the same type or not. If the first blocks of all three branches have different types, then b_{f+1,β_2} cannot be a dilution block, since it needs to win against b_{f+1,β_1} and dilution blocks lose against all other types, but it also cannot be a dilution end block, since it needs to lose against b_{f+1,β_3} and dilution end blocks win against all other types. Therefore b_{f+1,β_2} is a (pre-)depth block. This means b_{f+1,β_1} must be a dilution block, since this is the only type that loses against a (pre-)depth block, and b_{f+1,β_3} must be a dilution end block, since this is the only type that wins against a (pre-)depth block. A dilution end block wins against a dilution block according to the third criterion, so $\beta_1 < \beta_3$ is true.

If b_{f+1,β_1} and b_{f+1,β_2} are of the same type but b_{f+1,β_3} is of a different type, then $\beta_2 < \beta_3$ automatically means $\beta_1 < \beta_3$ since the fork is resolved according to block types. If b_{f+1,β_1} is of a different type but b_{f+1,β_2} and b_{f+1,β_3} are of the same type, then $\beta_1 < \beta_2$ automatically means $\beta_1 < \beta_3$ since, again, the fork is resolved according to block types. It is not possible that b_{f+1,β_1} and b_{f+1,β_3} are of the same type while b_{f+1,β_2} is of a different type, since that would mean the former type both loses and wins against the latter type. Therefore the only other option is that the three blocks are all of the same type.

They cannot be (pre-)depth blocks in this case, since there is only one possible (pre-)depth block for a given b_f, so they have to be either dilution blocks or dilution end blocks. If they are *dilution blocks*, initially the old public keys are compared between the blocks.

The lists of old public keys of the blocks start with $i - 1 \geq 0$ public keys that are identical in all three blocks, where the key origin block index of each block is identical in all three blocks as well. It is possible that this covers the entire list of old public keys in the block with the shortest list, but if not, there is an index i where at least one of the three blocks has a different old public key or key origin block index.

If the old public key in b_{f+1,β_1} is different from the old public key in the other two, it must be higher, when interpreted as a number, than the old public key in b_{f+1,β_2}, when interpreted as a number, since $\beta_1 < \beta_2$. The old public key in b_{f+1,β_2}, when interpreted as a number, must be equal to or higher than the old public key in b_{f+1,β_3}, since $\beta_2 < \beta_3$. Therefore it is also true that $\beta_1 < \beta_3$ due to the strict total order of numbers.

If the old public key in b_{f+1,β_3} is different from the old public key in the other two, it must be lower, when interpreted as a number, than the old public key in b_{f+1,β_2}, when interpreted as a number, since $\beta_2 < \beta_3$. The old public key in b_{f+1,β_2}, when interpreted as a number, must be lower than or equal to the old public key in b_{f+1,β_1}, since $\beta_1 < \beta_2$. Therefore it is also true that $\beta_1 < \beta_3$ due to the strict total order of numbers.

If the old public key of b_{f+1,β_2} is different from the other two, the other two must also be different from each other since otherwise the key in b_{f+1,β_2} would have to be both higher and lower than the key in the other two at the same time. The case where the three old public keys are all different has already been covered twice in the above.

Therefore the only case that is left to be covered is where all three old public keys are the same: in this case the fork must be decided on the basis of the key origin block index. The index in b_{f+1,β_1} must be higher than or equal to the index in b_{f+1,β_2}, since $\beta_1 < \beta_2$. If it is higher, the index in b_{f+1,β_2} must be higher than or equal to the index in b_{f+1,β_3}, since $\beta_2 < \beta_3$, and therefore the index in b_{f+1,β_1} must be higher than the index in b_{f+1,β_3} and thus $\beta_1 < \beta_3$.

If the index in b_{f+1,β_1} is equal to the index in b_{f+1,β_2}, the latter must be higher than the index in b_{f+1,β_3}, since $\beta_2 < \beta_3$, and thus the index in b_{f+1,β_1} is higher than the index in b_{f+1,β_3} and therefore $\beta_1 < \beta_3$. Therefore in all cases where not the entire list of old keys in the block with the shortest list is equal to the beginning of the list in the other two blocks, we know $\beta_1 < \beta_3$.

If the entire list of old keys in the block with the shortest list is equal to the beginning of the list in the other two blocks, the length of the list is compared: b_{f+1,β_1} must have a list that is shorter than or equal in length to the list in b_{f+1,β_2}, which must have a list that is shorter than or equal in length to the list in b_{f+1,β_3}. Since the length is a natural number, which is strictly totally ordered, we know that b_{f+1,β_1} must have a list that is shorter than or equal in length to the list in b_{f+1,β_3}. If it is shorter, we know that $\beta_1 < \beta_3$. If it is equal in length, all three blocks have the same number of old public keys and they are all identical as well. In this case the only way the forks can be resolved anymore is by comparing the new public keys. The lists of new public keys of the blocks start with $j - 1 \geq 0$ public keys that are identical in all three blocks.

If this does not cover the entire list, at least one of the three blocks must have a different new key at index j in the list. If b_{f+1,β_1} has a different new key at index j than the other two, it must be higher than the new key in b_{f+1,β_2} when both are interpreted as a number,

since $\beta_1 < \beta_2$. The new key in b_{f+1,β_2} must be higher than or equal to the new key in b_{f+1,β_3}, since $\beta_2 < \beta_3$. Therefore the new key in b_{f+1,β_1} is higher than the new key in b_{f+1,β_3} due to the strict total order of the numbers, and thus $\beta_1 < \beta_3$.

If the new key in b_{f+1,β_1} is equal to the new key in b_{f+1,β_2}, then b_{f+1,β_3} must have a new key that is lower than the new key in b_{f+1,β_2} when interpreted as a number, since $\beta_2 < \beta_3$. Therefore the new key in b_{f+1,β_2} must also be higher than the new key in b_{f+1,β_3} and thus $\beta_1 < \beta_3$.

If all new keys in the list are the same amongst all three blocks, then the block assemblers are interpreted as numbers and compared: if the block assembler of b_{f+1,β_1} is higher than the block assembler of b_{f+1,β_2}, we know the block assembler of b_{f+1,β_2} must be higher than or equal to the block assembler of b_{f+1,β_3} and therefore the block assembler of b_{f+1,β_1} is higher than the block assembler of b_{f+1,β_3}.

If the block assembler of b_{f+1,β_1} is the same as the block assembler of b_{f+1,β_2}, then the block assembler of b_{f+1,β_2} is either same as the block assembler of b_{f+1,β_3} or higher: if it is higher, then so is the block assembler of b_{f+1,β_1} and thus $\beta_1 < \beta_3$. If it is the same, then the block assemblers of all three blocks are the same.

In this case the pool identifier is used to resolve the fork: the pool identifier of b_{f+1,β_1} must be higher than or equal to the pool identifier of b_{f+1,β_2}, and the pool identifier of b_{f+1,β_2} must be higher than or equal to the pool identifier of b_{f+1,β_3}, then the pool identifier of b_{f+1,β_1} must be higher than or equal to the pool identifier of b_{f+1,β_3} due to the strict total order of numbers. If it is higher, then $\beta_1 < \beta_3$, and if it is equal the pool identifiers of all three blocks are identical.

In that case the three blocks are identical: they have the same old keys and therefore also the same anonymity set, depth and signatures, and they have the same new keys, block assembler and pool identifier, and they also have the same prefix and information identifying the blockchain. Therefore if the first blocks of all three branches are dilution blocks, we know that, if $\beta_1 < \beta_2$ and $\beta_2 < \beta_3$ then $\beta_1 < \beta_3$.

If the first blocks of all three branches are *dilution end blocks*, the third criterion initially compares the hashes of the blocks: since $\beta_1 < \beta_2$, the hash of b_{f+1,β_1} must be higher than or equal to the hash of b_{f+1,β_2}, when interpreted as numbers. Since $\beta_2 < \beta_3$, the hash of b_{f+1,β_2} must be higher than or equal to the hash of b_{f+1,β_3}, when interpreted as numbers. Therefore the hash of b_{f+1,β_1} must be higher than or equal to the hash of b_{f+1,β_3}. If it is higher, then $\beta_1 < \beta_3$. If it is equal, then the hashes of all three blocks are the same and instead the sizes of the blocks are compared. Since $\beta_1 < \beta_2$, b_{f+1,β_1} must be larger than b_{f+1,β_2} or the same size, and since $\beta_2 < \beta_3$, b_{f+1,β_2} must be larger than b_{f+1,β_3} or the same size. Therefore b_{f+1,β_1} is larger than b_{f+1,β_3} or the same size.

If it is larger, then $\beta_1 < \beta_3$. If it is the same size, all three blocks are the same size and the full contents of each block is compared, interpreted as a number. Since $\beta_1 < \beta_2$, b_{f+1,β_1} must be higher than b_{f+1,β_2} or equal, and since $\beta_2 < \beta_3$, b_{f+1,β_2} must be higher than b_{f+1,β_3} or equal. Therefore b_{f+1,β_1} must be higher than b_{f+1,β_3} or equal. If it is higher, then $\beta_1 < \beta_3$. If it is equal, the full contents of all three blocks are equal and thus there is no fork.

Therefore for all possible branches in the Dilution phase: if $\beta_1 < \beta_2$ and $\beta_2 < \beta_3$ then

$\beta_1 < \beta_3$.

10.3.4 Connectedness

We shall now prove that for every two branches β_1, β_2: if $\beta_1 \neq \beta_2$ then $\beta_1 < \beta_2$ or $\beta_2 < \beta_1$. The chain score of β_1 is either lower than the chain score of β_2, higher or equal. If it is lower, then $\beta_1 < \beta_2$ is true, and if it is higher, then $\beta_2 < \beta_1$ is true.

If the chain scores are equal, the third criterion is used to resolve forks. In this case the two initial blocks of each chain, b_{f+1,β_1} and b_{f+1,β_2} are compared to each other. They either have the same type or they have a different type.

If one is a dilution block and the other a (pre-)depth block, the (pre-)depth block wins. If b_{f+1,β_1} is the dilution block, $\beta_1 < \beta_2$ is true. If b_{f+1,β_2} is the dilution block, $\beta_2 < \beta_1$ is true.

If one is a dilution block and the other a dilution end block, the dilution end block wins. If b_{f+1,β_1} is the dilution block, $\beta_1 < \beta_2$ is true. If b_{f+1,β_2} is the dilution block, $\beta_2 < \beta_1$ is true.

If one is a (pre-)depth block and the other a dilution end block, the dilution end block wins. If b_{f+1,β_1} is the (pre-)depth block, $\beta_1 < \beta_2$ is true. If b_{f+1,β_2} is the (pre-)depth block, $\beta_2 < \beta_1$ is true.

If the blocks have the same type, they are either both dilution blocks, (pre-)depth blocks or dilution end blocks. By definition of the fork we know that $b_{f+1,\beta_1} \neq b_{f+1,\beta_2}$, so they cannot both be (pre-)depth blocks since there is only one possible (pre-)depth block if its predecessor is already defined.

If they are both *dilution blocks*, it is either true that all the public keys to be diluted and their associated origin block indices in the block with the smallest number of public keys are equal to those in the other block, or it is not true. If it is not true, there must be an index $i \geq 1$ at which either the old public key in b_{f+1,β_1} is different from the old public key in b_{f+1,β_2}, or the key origin block index in b_{f+1,β_1} is different from the key origin block index in b_{f+1,β_2}, while the old public keys and key origin block indices are the same in both blocks for every index below i. If the old public keys are different, then the old public key at index i in b_{f+1,β_1}, when interpreted as a number, is either higher or lower than the old public key at index i in b_{f+1,β_2}, when interpreted as a number. If it is higher, then $\beta_1 < \beta_2$ is true. If it is lower, then $\beta_2 < \beta_1$ is true. If the two public keys are the same, the key origin block index must be different: if the key origin block index of b_{f+1,β_1} is higher than the key origin block index of b_{f+1,β_2}, then $\beta_1 < \beta_2$ must be true. If the key origin block index of b_{f+1,β_1} is lower than the key origin block index of b_{f+1,β_2}, then $\beta_2 < \beta_1$ must be true.

If it is true that all the public keys to be diluted and their associated origin block indices in the block with the smallest number of public keys are equal to those in the other block, then there are two possibilities: either both blocks have the same number of keys or they do not. If they do not have the same number of keys, then either b_{f+1,β_1} has fewer keys, in which case $\beta_1 < \beta_2$ is true, or b_{f+1,β_2} has fewer keys, in which case $\beta_2 < \beta_1$ is true.

If they do have the same number of keys, all the old keys and their associated block origin indices are the same, in which case the new keys must be compared.

It is either true that all the new keys in the two blocks are the same, or it is not true. If it is not true, there must be an index $i \geq 1$ at which the new public key in b_{f+1,β_1} is different from the new public key in b_{f+1,β_2}, so the new public key at index i in b_{f+1,β_1}, when interpreted as a number, is either higher or lower than the new public key at index i in b_{f+1,β_2}, when interpreted as a number. If it is higher, then $\beta_1 < \beta_2$ is true. If it is lower, then $\beta_2 < \beta_1$ is true.

If all the new keys are the same, the public key of the block assembler in each block is compared: if they are different, then the block assembler in b_{f+1,β_1}, when interpreted as a number, is either higher or lower than the block assembler in b_{f+1,β_2}, when interpreted as a number. If it is higher, then $\beta_1 < \beta_2$ is true, and if it is lower, then $\beta_2 < \beta_1$ is true. If the two block assemblers are the same, only the pool identifiers are left to compare.

If the pool identifier in b_{f+1,β_1}, when interpreted as a number, is higher than the pool identifier in b_{f+1,β_2}, when interpreted as a number, then $\beta_1 < \beta_2$ is true. If it is lower, then $\beta_2 < \beta_1$ is true.

If the pool identifier is the same the blocks are identical: they have the same old keys and therefore also the same anonymity set, depth and signatures, and they have the same new keys, block assembler and pool identifier, and they also have the same prefix and information identifying the blockchain. Therefore a fork is not possible.

Thus we know that, if the branches start with dilution blocks, if $\beta_1 \neq \beta_2$ then $\beta_1 < \beta_2$ or $\beta_2 < \beta_1$.

If the branches start with *dilution end blocks*, the hashes of the blocks are compared: they are either the same, or the hash of b_{f+1,β_1}, when interpreted as a number, is higher or lower than the hash of b_{f+1,β_2}, when interpreted as a number. If it is higher, then $\beta_1 < \beta_2$, and if it is lower, then $\beta_2 < \beta_1$. If it is the same, the size of the blocks is compared: if b_{f+1,β_1} is larger than b_{f+1,β_2}, then $\beta_1 < \beta_2$, if it is larger, then $\beta_2 < \beta_1$. If the blocks are the same size, the full contents of each block are interpreted as a number and compared: if b_{f+1,β_1}, when interpreted as a number, is higher than b_{f+1,β_2}, when interpreted as a number, then $\beta_1 < \beta_2$. If it is lower, then $\beta_2 < \beta_1$. If they are equal, the two blocks are the same, which contradicts the definition of a fork.

Therefore no matter the chain scores of the branches or the blocks they start with, for every two branches β_1, β_2: if $\beta_1 \neq \beta_2$ then $\beta_1 < \beta_2$ or $\beta_2 < \beta_1$.

10.3.5 Conclusion

The fork resolution protocol satisfies all four requirements to be a strict total order in the Dilution phase.

10.4 Commitment phase

10.4.1 Irreflexivity

First we shall prove that for every branch β it is true that: Not $\beta < \beta$. If $\beta < \beta$ were true, the protocol should resolve a fork between β and itself using any of the three criteria listed in 4.6.

The first criterion can only resolve forks between branches that start with a block b_{f+1} of a different type, but β cannot start with a different block than itself so the first criterion does not resolve a fork between β and itself.

The second criterion can only resolve forks between branches that have different chain scores, but β has the exact same chain score as itself so the second criterion does not resolve a fork between β and itself.

The third criterion can resolve blocks between blocks of the same type if they are registration blocks, dilution blocks, dilution end blocks or commitment blocks. Of those four types of blocks only commitment blocks can be appended in this phase. The third criterion first looks at the length of the chains being compared in case both branches start with a commitment block, but the length of β is equal to itself so the fork cannot be resolved this way. The third criterion subsequently uses the voter public keys of the blocks to resolve the fork only if the two blocks have unequal voter public keys, but $b_{f+1,\beta}$ must necessarily have the same voter public key as itself. The third criterion then looks at the indices of the origin blocks of these keys, but $b_{f+1,\beta}$ must again have the same index as itself. The third criterion alternatively uses the vote hashes of the blocks to resolve the fork, but $b_{f+1,\beta}$ contains the same vote hash as itself so this cannot resolve the fork either.

Therefore the protocol cannot resolve a fork between β and itself in the Commitment phase and thus $\beta < \beta$ is not true.

10.4.2 Asymmetry

We shall now prove that for every two branches β_1, β_2: if $\beta_1 < \beta_2$ then not $\beta_2 < \beta 1$. There are two types of blocks in this phase: the commitment block and the commitment end block.

Given b_f, all contents of a commitment end block are fixed and thus there is only one possible commitment end block. The commitment end block always wins a fork against a commitment block in the first criterion and we have shown in the preceding paragraph that it cannot win or lose a fork against itself, so if $\beta_1 < \beta 2$ we know that b_{f+1,β_1} cannot be a commitment end block and must therefore be a commitment block. This leaves two possibilities: either b_{f+1,β_2} is a commitment end block or it is a commitment block as well.

If b_{f+1,β_2} is a *commitment end block* and b_{f+1,β_1} a commitment block, the fork is resolved according to the first criterion and β_2 wins, therefore $\beta_2 < \beta_1$ is not true.

If b_{f+1,β_2} and b_{f+1,β_1} are both *commitment blocks*, the fork is not resolved according to the first criterion, so it must be resolved by either the second or the third criterion. If it is resolved according to the second criterion, the chain scores of the branches must be

different and $\beta_1 < \beta_2$ means the chain score of β_2 is higher than that of β_1. Since the chain score is an integer and the "higher than" relationship defines a strict total order over the integers, this means the chain score of β_1 cannot be higher than that of β_2 and thus $\beta_2 < \beta_1$ cannot be true.

If the fork is resolved according to the third criterion, either the length of the two branches is the same or it is not the same. If the length is not the same, the longest branch wins and thus β_2 must be longer than β_1. Since the length is a natural number and thus strictly totally ordered, this means β_1 cannot be longer than β_2 and thus $\beta_2 < \beta_1$ cannot be true.

If the length is the same, either the voter public key of b_{f+1,β_1} and b_{f+1,β_2} is used to resolve the fork or not. If the voter public key is used, it is interpreted as a number and again a strict total order over the numbers is defined. This means that, since $\beta_1 < \beta_2$, the voter public key of b_{f+1,β_1} must be higher than the voter public key of b_{f+1,β_2} and thus the voter public key of b_{f+1,β_1} cannot be lower than the voter public key of b_{f+1,β_2}, so $\beta_2 < \beta_1$ cannot be true.

If the voter public key is not used to resolve the fork, the index of the origin block of the voter public key is used if it differs. Since this is a number a strict total order is defined, and since $\beta_1 < \beta_2$ the origin index of β_1 must be higher than the origin index of β_2, so the origin index of β_2 cannot be higher than the origin index of β_1 and therefore $\beta_2 < \beta_1$ cannot be true.

If the indices of the origin blocks of the public keys of the two blocks are the same, the only other way to resolve the fork is using the vote hashes of b_{f+1,β_1} and b_{f+1,β_2} and again interpreting them as a number and seeing which is higher. This is once again a strict total order, so we know if the vote hash of b_{f+1,β_1} is higher than the vote hash of b_{f+1,β_2}, it cannot be true that the vote hash of b_{f+1,β_1} is lower than the vote hash of b_{f+1,β_2} and thus $\beta_2 < \beta_1$ cannot be true.

Therefore there is no case where $\beta - 1 < \beta_2$ is true but $\beta_2 < \beta_1$ is also true.

10.4.3 Transitivity

We shall now prove that for every three branches $\beta_1, \beta_2, \beta_3$: if $\beta_1 < \beta_2$ and $\beta_2 < \beta_3$ then $\beta_1 < \beta_3$. The first block of each of these branches is either a commitment block or a commitment end block. We have shown in the above paragraph that a branch starting with a commitment end block cannot lose a fork, so since we know $\beta_1 < \beta_2$ and $\beta_2 < \beta_3$, we know that b_{f+1,β_1} and b_{f+1,β_2} are commitment blocks. This leaves two possibilities: either b_{f+1,β_3} is a commitment end block or b_{f+1,β_3} is a commitment block.

If it is a *commitment end block*, it wins all forks against commitment blocks according to the first criterion, so since we know b_{f+1,β_1} is a commitment block we know $\beta_1 < \beta_3$.

If b_{f+1,β_3} is a *commitment block*, then the fork between β_2 and β_3 must be resolved either by the second or the third criterion. If it is resolved by the second criterion, the chain score of β_3 is higher than the chain score of β_2. The fork between β_1 and β_2 is also resolved by either the second or the third criterion. If it is resolved by the second criterion, the chain score of β_2 is higher than the chain score of β_1. Since the chain score is a number

and therefore a strict total order applies, this means that the chain score of β_3 is higher than the chain score of β_1 and thus $\beta_1 < \beta_3$. If the fork between β_1 and β_2 is resolved by the third criterion, the chain scores of β_1 and β_2 must be equal. Since we already know that the chain score of β_2 is lower than the chain score of β_3, this means the chain score of β_1 is lower than the chain score of β_3 and therefore $\beta_1 < \beta_3$. Therefore if the fork between β_2 and β_3 is resolved according to the second criterion, we know $\beta_1 < \beta_3$.

If the fork between β_2 and β_3 is resolved according to the third criterion, the chain scores of β_2 and β_3 are equal. The fork between β_1 and β_2 is again resolved either according to the second or the third criterion. If it is resolved according to the second criterion, the chain score of β_1 is lower than the chain score of β_2, which we know to be equal to the chain score of β_3. Therefore we know that the chain score of β_1 is lower than the chains core of β_3 and thus $\beta_1 < \beta_3$. If the fork between β_1 and β_2 is resolved according to the third criterion, the length of the chain of β_1 is either equal to the length of the chain of β_2 or not.

If the lengths are not equal, the length of β_1 must be lower than the length of β_2 since the longest chain wins. Since we know the fork between β_2 and β_3 is also resolved according to the third criterion, we know that the chain of β_3 cannot be shorter than the chain of β_2 since it wins the fork. Therefore we know that β_2 is shorter than β_2 which is at least as short as β_3, so we know β_1 is shorter than β_3 because length is a natural number and thus strictly totally ordered. Therefore we have $\beta_1 < \beta_3$. If β_1 is of equal length to β_2, then either β_3 is longer or also of equal length. If β_3 is longer than β_2, which is of equal length to β_1, we know that β_3 is also longer than β_1 and thus we know $\beta_1 < \beta 3$.

If the lengths of all three branches are the same, then the voter public key of b_{f+1,β_1} is either equal to the voter public key of b_{f+1,β_2} or higher. The voter public key of b_{f+1,β_2} must also be either equal to or higher than the voter public key of b_{f+1,β_3}. If the voter public key of b_{f+1,β_1} is higher than the voter public key of b_{f+1,β_2}, it will also be higher than the voter public key of b_{f+1,β_3} since it is equal to or higher than the voter public key of b_{f+1,β_2}, and thus we know that $\beta_1 < \beta_3$. If the voter public key of b_{f+1,β_1} is equal to the voter public key of b_{f+1,β_2}, and the voter public key of b_{f+1,β_2} is equal to the voter public key of b_{f+1,β_3}, the voter public key of b_{f+1,β_1} is equal to the voter public key of b_{f+1,β_3}.

In this case the key origin block indices of the blocks is used to resolve the fork: since $\beta_1 < \beta_2$ the key origin block index of b_{f+1,β_1} must be higher than or equal to the key origin block index of b_{f+1,β_2}, and since $\beta_2 < \beta_3$ the key origin block index of b_{f+1,β_2} must be higher than or equal to the key origin block index of b_{f+1,β_3}. Therefore the key origin block index of b_{f+1,β_1} must be higher than or equal to the key origin block index of b_{f+1,β_3}. If it is higher, we know that $\beta_1 < \beta_3$. If it is equal, the vote hash of b_{f+1,β_1} must be higher than the vote hash of b_{f+1,β_2}, and the vote hash of b_{f+1,β_2} must be higher than the vote hash of b_{f+1,β_3}, which means the vote hash of b_{f+1,β_1} must be higher than the vote hash of b_{f+1,β_3} due to the vote hashes being interpreted as numbers which are strictly totally ordered. Therefore the third criterion resolves this fork as $\beta_1 < \beta_3$.

Therefore for all possible branches in the *Commitment* phase we know that: if $\beta_1 < \beta_2$ and $\beta_2 < \beta_3$ then $\beta_1 < \beta_3$.

10.4.4 Connectedness

We shall now prove that for every two branches β_1, β_2: if $\beta_1 \neq \beta_2$ then $\beta_1 < \beta_2$ or $\beta_2 < \beta_1$. By definition of the fork we know that $b_{f+1,\beta_1} \neq b_{f+1,\beta_2}$, so they cannot both be commitment end blocks since there is only one possible commitment end block if its predecessor is already defined.

It is however possible that one of the two is a *commitment end block* and the other a commitment block, in which case the commitment end block wins the fork according to the first criterion. If b_{f+1,β_1} is the commitment end block then $\beta_2 < \beta_1$, and if b_{f+1,β_2} is the commitment end block then $\beta_1 < \beta_2$. Therefore whenever one of the two branches starts with a commitment end block we know $\beta_1 < \beta_2$ or $\beta_2 < \beta_1$.

If both b_{f+1,β_1} and b_{f+1,β_2} are *commitment blocks* the fork cannot be resolved according to the first criterion, so we have to see if the second or third criterion can resolve it. If the chain scores of the two branches are different, this will resolve the fork: if the chain score of β_2 is higher than that of β_1 then $\beta_1 < \beta_2$, and if the chain score of β_1 is higher than that of β_2 then $\beta_2 < \beta_1$.

If the chain scores are equal however the second criterion is inconclusive and only the third criterion could possibly resolve the fork. When comparing two commitment blocks, the third criterion initially compares the lengths of the chains of the two branches: the branch with the longest chain wins. If β_1 has the longest chain, then $\beta_2 < \beta_1$ is true. If β_2 has the longest chain, then $\beta_1 < \beta_2$ is true. Thus we know that either $\beta_1 < \beta_2$ or $\beta_2 < \beta_1$ is true if the lengths are different.

If the lengths are the same, the third criterion compares the voter public keys of each commitment block and resolves the fork if these voter public keys are different: if the voter public key of b_{f+1,β_1}, interpreted as a number, is lower than the voter public key of b_{f+1,β_2}, interpreted as a number, then $\beta_2 < \beta_1$. If the voter public key of b_{f+1,β_2}, interpreted as a number, is lower than the voter public key of b_{f+1,β_1}, then $\beta_1 < \beta_2$. Therefore if the two voter public keys are not equal, then either $\beta_1 < \beta_2$ or $\beta_2 < \beta_1$.

If b_{f+1,β_1} and b_{f+1,β_2} have the same voter public key, then the protocol cannot use them to resolve the fork and instead uses the key origin block indices of the two blocks: if the key origin block index of b_{f+1,β_1} is lower than the key origin block index of b_{f+1,β_2}, then $\beta_2 < \beta_1$. If the key origin block index of b_{f+1,β_2} is lower than the key origin block index of b_{f+1,β_1}, then $\beta_1 < \beta_2$.

If the key origin block indices of the two blocks are equal, the protocol uses the vote hashes of b_{f+1,β_1} and b_{f+1,β_2}: if the vote hash of b_{f+1,β_1}, interpreted as a number, is lower than the vote hash of b_{f+1,β_2}, interpreted as a number, then $\beta_2 < \beta_1$. If the vote hash of b_{f+1,β_2}, interpreted as a number, is lower than the vote hash of b_{f+1,β_1}, then $\beta_1 < \beta_2$. Therefore if the vote hashes are not equal, then either $\beta_1 < \beta_2$ or $\beta_2 < \beta_1$.

Therefore the only case that cannot be resolved is the case where the first blocks of both branches are commitment blocks with the same voter public key and the same vote hash. They must also have the same prefix since this does not change and they must have the same information identifying the blockchain, index, predecessor hash and chain score, so all the data the signature is calculated over is the same. The signature must also be the

same since it is the signature corresponding to the voter public key, so all the data in the two blocks is the same and therefore the two blocks are the same, which contradicts the definition of the fork.

Therefore if $\beta_1 \neq \beta_2$ then $\beta_1 < \beta_2$ or $\beta_2 < \beta_1$.

10.4.5 Conclusion

The fork resolution protocol therefore satisfies all four requirements to be a strict total order in the Commitment phase.

10.5 Voting phase

There are no blocks in this phase so no branches can exist: any hypothetical branch β could not satisfy the requirement $l_\beta > f$. Therefore the set of branches is the empty set and thus all propositions are true for every branch in the set.

10.6 Conclusion

The fork resolution protocol therefore resolves forks in all phases, and thus it is a valid consensus mechanism.

Chapter 11

Remaining robustness issues

Robustness is a broad topic since there are many possible ways someone might interact with a system in a way that was not intended by the designer of the system. Denial-of-service attacks, eclipse attacks and blockchain forks were important dangers the system should be robust against and have been analyzed in the previous chapters. This chapter will look at a number of remaining issues the system should be robust against. Section 11.1 covers voter accidents and Section 11.2 covers network failure.

11.1 Voter accidents

There are two kinds of accidents we can consider: accidents where eligible participants lose access to the data they need to vote, and accidents where eligible participants lose the right to vote. [1]

If eligible participants completely lose access to the data they need to vote (their key pairs) during the Registration phase, they can simply register a new key pair. If they lose access to their key pairs when they have already diluted them, they cannot register a new key pair anymore. There is no one to help them recover their keys because that information does not exist elsewhere. They can also not ask for their public key to be disabled in order to get a new key pair since no one knows which public key corresponds to them due to voter privacy.

The same logic applies to eligible participants who lose the right to vote: if this happens during the Registration phase, or before the first round of signature dilution they participate in, it is known which public keys correspond to them and these could be invalidated. However when signature dilution has happened, this results in an anonymity set and thus it is no longer possible to invalidate someone's key since it is not clear which key that is.

The system is therefore robust against some voter accidents but definitely not all of

[1] The law might disenfranchise eligible participants when for example convicted for a felony, but participants might also lose the right to vote by dying: the dead participants themselves are obviously no longer able to vote, but if they divulge their keys to someone else before they die, the other person can vote on behalf of a dead person.

them. It should be noted that this is in fact a necessary risk: giving people control means giving people responsibility. The requirement of eligibility demands that certain actions, such as voting, must be restricted to certain people. An e-voting system consists entirely of the communication of data and as such the only way to restrict access for people is by requiring that they possess a piece of data. Therefore if authorized people lose this data, they lose the ability to vote. This could be mitigated by giving them another piece of data that also authorizes them to vote, but the requirement of uniqueness entails that each person can at most possess one piece of data authorizing the vote. This in turn means that, when a new piece of data is given, either the eligible person must prove that the old data is lost, or the system must invalidate this old data. Eligible participants will not be able to prove that they do not possess data since this requires proving a negative, and for the system to invalidate the data it must know which data within the system determines the validity of this lost piece of data uniquely. This means that a certain piece of data in the system is linked to the identity of the eligible participant. This piece of data must be used when determining the validity of the participant's vote, which means the vote can be linked to the identity of the eligible participant. This in turn violates the requirement of privacy.

11.2 Network failure

Without the internet no e-voting system can work and if part of the electorate has no access to the internet, that part cannot make use of the system. Nevertheless, some systems might allow for participants to be temporarily disconnected from the network and still participate to the full extent of the system's possibilities, while others might require that participants are connected at a very specific time or even for the entire duration of the election. The former can be said to be more robust against network failure than the latter.

Since AnonVote is a blockchain-based system, we can first look at the robustness of blockchains in general against network failure. The core functionality of a blockchain can be summarized as follows: one of the nodes in the network creates a new block, appending it to the chain, and broadcasts it to the other nodes in the network, who each update their own view of the chain.

Network failure can prevent nodes from broadcasting blocks, in which case the state of the blockchain will not be updated. If the network failure is resolved after a while, the nodes can broadcast their blocks after all. In the meantime the blockchain has either grown or it has not: if it has not there is no problem at all, but if it has, a fork arises which will be resolved through the fork resolution protocol. If the network failure persists indefinitely, the blocks will never be broadcast.

Network failure can also prevent nodes from receiving blocks, in which case they cannot see the current state of the blockchain. If the network failure is resolved after a while, the nodes can request the last block of the chain and subsequently request all the other blocks they have missed, and thus they will eventually have the entire blockchain. Thus the dangers of temporary network failure for a blockchain can be summarized as: participants

may be temporarily unable to see the state of the blockchain and change the state of the blockchain, and they may be outcompeted when changing the state due to forks.

We can look at the effects network failure has on the different roles of participants in the system, specifically the honest participants. We will not consider network failure affecting attackers as a problem, since this is arguably even a desirable outcome. We will look at network failure during all five phases of the election.

11.2.1 Setup

The only thing that happens in this phase, is the creation of the initialization block by the election manager. When election managers cannot connect to the internet, they must simply wait until they can and then broadcast the initialization block. Eligible participants do not need to be connected at this moment, since they do not create any blocks and they can retrieve the chain afterwards, when they are connected.

11.2.2 Registration

During this phase, election managers create registration blocks and eventually a dilution start block. No one else creates blocks in this phase. If the election manager is disconnected from the internet, this phase is stalled until the connection is reestablished. Eligible participants can only create a disenfranchisement message during this phase, but this can actually also be done in a later phase. Thus if they are disconnected and their registration blocks are orphaned in a fork, they can still prove this after the phase has ended.

This will however not be possible if the network failure caused them to be unable to receive the registration block in the first place: then they cannot create the disenfranchisement message at all. If eligible participants have their registration blocks orphaned, they cannot vote using this system. It is however also trivial to prove this, which makes it possible for a fallback paper ballot system to allow them to vote anyway.

11.2.3 Dilution

During this phase, eligible participants create *dilution blocks*. If they are temporarily not connected, they are temporarily unable to dilute their keys and they may be outcompeted when they try. In order to create dilution blocks however, eligible participants also send a number of messages to each other. If they are temporarily not connected, they cannot send and receive these messages in that time, which makes it more difficult for them to dilute their keys. If this persists for an entire round of signature dilution, this will however make them leftover and they will be given preferential treatment after the pre-depth block.

It is possible though that network failure persists for so long that the eligible participants have no opportunity to dilute their keys and therefore they have no privacy when voting afterwards. Since it is trivial to prove this, it is again possible for a fallback paper ballot system to allow them to vote in an alternative way. Besides these messages, they

also cannot send a blame message when not connected, meaning an attacker might temporarily avoid being exposed. Election managers do not do anything with these messages, so if they are temporarily disconnected and cannot see them, this has no effect. If election managers do not see dilution blocks due to a temporary network failure, this will only serve to temporarily make it more difficult for them to append blocks.

Election managers create *pre-depth blocks*, *depth blocks* and eventually a *dilution end block* in this phase. These blocks all serve to change the phase, or part of the phase, of the election. When election managers cannot connect to the internet, they must simply wait until they can and then change the (part of the) phase. If eligible participants do not receive a pre-depth or depth block due to network failure, this makes it more difficult for them to take part in a dilution process. This however requires an internet connection anyway, so this risk is covered entirely in the preceding paragraph. If eligible participants do not receive a dilution end block due to network failure, they will be unaware that the blockchain is in the Commitment phase until they eventually do receive the block. In this case they will not be able to do anything in the Dilution phase anymore, since that requires an internet connection, so the only real effect is they are temporarily unable to take part in the Commitment phase. This is equivalent to network failure during the Commitment phase, which we will cover in the paragraph below.

11.2.4 Commitment

During this phase, eligible participants create *commitment blocks* committing their votes. If they are temporarily not connected, they are temporarily unable to commit their votes. They are however not affected by missing other participants' commitments apart from the effects of network failure in a blockchain, as discussed above. This means that eligible participants can simply create their commitment blocks when they reconnect to the internet, assuming this happens during the Commitment phase. If however they experience network failure throughout the entire phase, they are disenfranchised. Election managers do not do anything with the contents of commitment blocks, so if they cannot receive them due to network failure, the only effects are the effects of network failure in a blockchain, as discussed above.

Election managers only create the *commitment end block* in this phase. If they cannot broadcast it due to network failure, the phase will continue until the network failure is resolved.

11.2.5 Voting

During this phase, voters broadcast their vote messages, allowing their votes to be tallied. If network failure prevents voters from broadcasting their vote messages, their votes will not be included in the tally until the network failure is resolved. After that the voters can broadcast their vote messages anyway, and the tally can be updated to include them. If network failure prevents participants from receiving vote messages, they will not include them in the tally as long as the network failure persists. When the network failure is

resolved however, they can request other participants' tallies when they notice their tally results in a different outcome of the election. Since tallies are published with the vote messages that were counted, these will now be known to the participants who experienced network failure as well.

11.2.6 Conclusion

The system is not protected against persistent network failure because no distributed system is, but due its nature as a blockchain it can mostly recover from the effects of temporary network failure. In the worst case scenario, network failure in the first three phases or the last phase can lead to voters being unable to participate in the system. Since this will be publicly visible, they can fall back to a paper ballot. Network failure throughout the entire Commitment phase is however more severe, since the affected eligible participants can be disenfranchised.

Chapter 12

Other recommendations

Besides robustness, three other recommendations were mentioned in Section 2.1. We will cover coercion resistance in Section 12.1, fairness in Section 12.2 and distribution of trust in Section 12.3.

12.1 Coercion resistance

This recommendation is defined as: It should not be possible for a third party to coerce a voter into voting a certain way.

Coercion resistance relies entirely on the knowledge the coercer has over the way the voter voted: if the coercer knows how the voter voted, the coercer can coerce the voter; if the coercer does not know how the voter voted, the coercer cannot coerce the voter. The coercer may try to obtain this knowledge either with or without the cooperation of the voter and these represent two levels of coercion resistance. At the first level the coercer is unable to obtain this knowledge without the cooperation of the voter. This is equivalent to privacy and was already evaluated in Section 7.2.

At the second level the coercer should not be able to obtain this knowledge even with the cooperation of the voter. This level of resistance cannot be achieved with e-voting since it is possible for participants to let coercers look over their shoulders every step of the way. There is a general problem combining this with individual verifiability: on the one hand voters should be able to see proof that their votes are counted according to the content they chose, but on the other hand they should be unable to share this proof with anyone else. In AnonVote specifically this level of coercion is possible because voters can divulge which key they used to vote and they can prove ownership of this key via a challenge-response mechanism or even just divulging the private key.

The system can be amended to achieve this second level of coercion resistance however by turning it from an e-voting system into a hybrid system. The specific changes that would have to be made include voting booths with voting machines where voters are first searched and all their cameras, smartphones and other recording equipment are confiscated. It is not necessary for the voters' identities to be checked and they may enter the voting booth

multiple times. They must carry out all steps in the Dilution, Commitment and Voting phases from within the voting booth on the voting machine and they can be forced to do so by adding the signature of a key stored within the voting machine to all messages in these phases.

In order to prevent the voters from divulging which keys belong to them the link between their private and public keys must be obfuscated. In ECDSA this can be done by replacing the private key d_A by a product of two private keys $d_{A,v} \cdot d_{A,m}$. The voter only knows $d_{A,v}$ because $d_{A,m}$ is stored in secure memory of the voting machine. This makes it impossible for voters to sign anything with their private keys outside of the voting booth since they do not have d_A in full.

It should be noted that, beside the practical limitations of requiring all voters to be in voting booths simultaneously for every step in the last three phases, this solution also introduces a trusted third party that controls access to the voting booth. It is therefore offered merely as a theoretical exercise showing what is necessary to achieve coercion resistance, and not included in the protocol proposed by this thesis.

12.2 Fairness

This recommendation is defined as: The running tally should not be available while voters are still able to vote. This recommendation can be violated if a secondary communication channel exists where all voters willingly broadcast the contents of their votes.

This would have to happen during the Commitment phase since a running tally can only exist after the first commitment block was added and voters are unable to change their votes after the commitment end block was added. Voters could broadcast w and z along with their public keys to prove how they voted. If all voters do this an accurate running tally can be calculated, the fewer voters participate the less accurate the tally becomes. Such an attack can probably be mitigated by making it illegal for voters to disclose their vote during the Commitment phase.

If voters do not broadcast w and z during the Commitment phase, then only $hash(w, z)$ is publicly known during this time. It is not feasible to derive w and z from this hash and therefore a tally cannot feasible be calculated until the Commitment phase ends and voters are no longer able to vote.

12.3 Distribution of trust

This recommendation is defined as: the above requirements should not rely on a single trusted third party. AnonVote satisfies this recommendation because none of the requirements rely on any trusted third party. We will look at the two roles in the system: Section 12.3.1 covers election managers and Section 12.3.2 covers eligible participants.

12.3.1 Election manager

Election managers are a single third party, but they are not trusted: we have covered potential attacks by election managers in the above and they are not able to forge votes or violate voter privacy. We shall look at each of the five phases of the election to see in more detail to what extent trust in the election manager is required.

Setup phase

In this phase, election managers create an initialization block. This includes their public key and the election ID, and it is signed with the private key corresponding to their public key. There is therefore no way to do this the wrong way if the intialization block is valid.

Registration phase

In this phase, election managers create registration blocks for eligible participants. In isolation this would require trust, since it is the election managers that certify that certain keys correspond to certain eligible participants. The need for this trust is however removed by having an off-chain process where election managers obtain proof that eligible participants chose to register their keys. Election managers could also disenfranchise eligible participants by orphaning their registration blocks, but those participants then have recourse to disenfranchisement messages so there is no need for trust in the election manager.

Dilution phase

In this phase, signatures are diluted to create anonymity, but election managers are completely uninvolved with the dilution process. They only increment the maximum depth of dilution blocks by adding pre-depth and depth blocks, but no trust is needed here since it is plainly visible whether they do this or not. They also end the phase by adding a dilution end block which includes the candidate information, but this is again plainly visible and thus requires no trust.

Commitment phase

In this phase, voters commit their vote hashes but election managers do not do anything, so no trust in them is needed.

Voting phase

In this phase, voters broadast their votes in plaintext but again election managers do not do anything, so no trust in them is needed.

12.3.2 Eligible participant

Single eligible participants are a single party, but they are again not trusted because we have covered attacks by eligible participants in the above. They can frustrate the proper functioning of the protocol by disenfranchising themselves in various ways, but this does not negatively impact other eligible participants. We shall look at each of the five phases of the election to see in more detail to what extent trust in an eligible participant is required.

Setup phase

In this phase, eligible participants do not do anything so no trust in them is required.

Registration phase

In this phase, eligible participants do not do anything within the blockchain. They do however participate in the off-chain registration process, registering their keys and creating proof that they did so. It is possible for them to frustrate this process by creating proof that they registered a public key, when in reality they do not possess its corresponding private key, but this only serves to disenfranchise themselves.

They can also create disenfranchisement messages when their registration blocks are orphaned, but this requires no trust in them since the disenfranchisment messages are validated by checking if the registration block they cite actually has the valid signature of the election manager.

Dilution phase

In this phase, eligible participants work together to create dilution blocks. The signatures on the dilution blocks serve to indicate that the dilution is correct and none of the participants can be disenfranchised without their own cooperation. None of the participants can, by themselves, discover the identity of the other participants because the new key message is signed with a pool session key that is shared among all the members of the pool. Therefore no trust is required in any of the eligible participants in this phase.

Commitment phase

In this phase, eligible participants commit their vote hashes in commitment blocks. These commitment blocks are signed by their keys and so it is proven that they were created by the right eligible participant, but the vote hashes are not checked to see if they truly correspond to a vote. It is thus possible for eligible participants to publish a false vote hash, but they only harm themselves this way. Therefore no trust is required in any of the eligible participants.

Voting phase

In this phase, eligible participants publish the plaintext votes corresponding to the vote hashes they have published during the previous phase. Anyone can verify that these plaintext votes truly correspond to the vote hash, so no trust is required.

12.3.3 Conclusion

Therefore neither election managers nor eligible participants are required to be trusted in any of the phases of the election. Since these are the only two roles in the system, there is no trusted third party.

Chapter 13

Performance evaluation

This chapter evaluates the memory consumption and the number of messages sent during an AnonVote election. We measure this experimentally in simulations using the client we introduced in Section 5.1, but we also cover theoretical calculations of the size of the blockchain in bytes and the number of messages that need to be sent. Section 13.1 describes the experimental setup in terms of the algorithm of the simulations that we run and the parameters we set, Section 13.2 describes formulae for the theoretical cost of an election in terms of the length and size of the blockchain and the number of messages broadcast per voter and Section 13.3 details the results of our simulations.

13.1 Methodology

We implement the client described in Section 5.1 and we run simulations of elections using these clients. In each simulation, first the client of the election manager m_c is instantiated. Next a number N of voter clients $\{v_1, ..., v_N\}$ is instantiated, and possibly a number n_a of attackers $\{\alpha_1, ..., \alpha_{n_a}\}$, and they are connected with each other and the election manager in a randomized connected graph G. The graph is constructed using Algorithm 1 to ensure its connectivity.

We base the number of attackers n_a on the number of voters N by the factor $o = \frac{n_a}{N}$. During each iteration, each attacker broadcasts 1000 messages to simulate a denial-of-service attack. Since none of these attackers are registered in the Registration phase, they are all outside attackers who send invalid messages. For each message there is a probability of 0.5 that it consists of random bytes and a probability of 0.5 that it is designed to look like a message that could be sent during this phase, but it fails a thorough validation check.

After the network graph is constructed, we run Algorithm 2, which makes use of Algorithms 3 and 4. Besides the election manager m_c and the voters V, Algorithm 2 uses the following parameters:

- A propagation delay $\pi_{delay} = 5$ that allows each message to propagate through the network before the next message is sent.

Input: an election manager m_c, a set of voters $V = \{v_1, ..., v_N\}$
Output: a connected graph G in which the vertices are m_c and all elements of V

```
G ← {m_c};
for v_i ∈ V = {v_1, ..., v_N} do
    pick a random vertex v_j in G;
    add v_i to G;
    add the edge (v_i, v_j) to G;
end
for α_i ∈ {α_1, ..., α_{n_a}} do
    pick a random vertex v_j in G;
    add α_i to G;
    add the edge (α_i, v_j) to G;
end
return G;
```

Algorithm 1: ConstructNetwork(m_c, $\{v_1, ..., v_N\}$)

- A depth increase interval $I_{depth} = 1000$ that determines after how many iterations in the Dilution phase m_c broadcasts a depth block.
- A pre-depth interval $I_{pre-depth} = 100$ that determines how many iterations in advance the pre-depth block is broadcast. In other words, after the last depth block was broadcast, 900 iterations later the new pre-depth block is broadcast.

During the simulation's Dilution phase, the participants simply try to dilute their signatures in pools of size $k = 2$.

We simulate with different maximum switch counters from both applications and pools (s_a and s_p, described in Section 5.1.3), both varying between 5, 10 and 20. We vary the number of voters N between 4, 8, 16 and 32 and have them participate in $r = 1$ or $r = 2$ rounds of signature dilution. In these simulations we have $o = 0$ and thus there are no attackers. We run 10 simulations for each combination of parameters.

We also run simulations where $o = 0.5$ to simulate the situation where there are half as many attackers as voters. This means that, when $N = 16$, there are 16 voters 8 attackers and 1 election manager, and thus 25 nodes in the network. In these simulations we keep both $s_p = 10$ and $s_a = 30$ constant. We compare the results of these simulations side by side with the results of simulations where $o = 0$, again keeping $s_p = 10$ and $s_a = 30$ constant. Again we run 10 simulations for each combination of parameters.

For each batch of 10 simulations, we measure the average length of the blockchain, the average number of anonymous broadcasts sent per voter, the average number of anonymous broadcasts received per voter and the average number of unvalidated messages passing through each voter node.

Input: an election manager m_c, a set of voters $V = \{v_1, ..., v_N\}$

$\pi = 0$;
$R \leftarrow Copy(V)$;
$C \leftarrow Copy(V)$;
while $phase_c! = Voting$ **do**
  PhaseIteration(m_c, $V = \{v_1, ..., v_N\}$, π, π_{delay}, R, C, I_{depth}, $I_{pre-depth}$);
  let m_c update its network layer and handle all its messages;
  for $v_i \in V$ **do**
    let v_i update its network layer and handle all its messages;
  end
  for $\alpha_i \in \{\alpha_1, ..., \alpha_{n_a}\}$ **do**
    let α_i update its network layer and handle all its messages;
  end
  for $v_i \in V$ **do**
    let v_i update its blockchain layer, possibly rebroadcasting dilution applications or invites;
  end
  for $\alpha_i \in \{\alpha_1, ..., \alpha_{n_a}\}$ **do**
    let α_i broadcast 1000 invalid messages;
  end
end

Algorithm 2: Simulate(m_c, $V = \{v_1, ..., v_N\}$)

Input: an election manager m_c, a set of voters $V = \{v_1, ..., v_N\}$, a propagation counter π, a propagation delay π_{delay}, a set of voters R that still need to be registered, a set of voters C that have not committed their vote yet

if $\pi > 0$ **then**
  decrement π;
end
else if m_c *has not started election* **then**
  let m_c create initialization block;
  $\pi = \pi_{delay}$;
end
else if R *is not empty* **then**
  pick and remove random voter v_i from R;
  let m_c create registration block for v_i;
  $\pi = \pi_{delay}$;
end
else if *election is in Registration phase* **then**
  let m_c create dilution start block;
  $\pi = \pi_{delay}$;
end
else if $\exists v_i \in V$ *: v_i has not yet diluted keys for r rounds or more* **then**
  DilutionPhaseSimulationRound(m_c, $V = \{v_1, ..., v_N\}$, dilution counter, I_{depth}, $I_{pre-depth}$);
  $\pi = \pi_{delay}$;
end
else if *election is in Dilution phase* **then**
  let m_c create dilution end block;
  $\pi = \pi_{delay}$;
end
else if C *is not empty* **then**
  pick and remove random voter v_i from C;
  let v_i create commitment block;
  $\pi = \pi_{delay}$;
end
else if *election is in Commitment phase* **then**
  let m_c create commitment end block;
  $\pi = \pi_{delay}$;
end

Algorithm 3: PhaseIteration(m_c, $V = \{v_1, ..., v_N\}$, π, π_{delay}, R, C, I_{depth}, $I_{pre-depth}$)

Input: an election manager m_c, a set of voters $V = \{v_1, ..., v_N\}$, a dilution counter, a depth increase interval I_{depth}, a pre-depth interval $I_{pre-depth}$

pick a random voter $v_i \in V$ that has not yet diluted keys for D_c rounds or more;
if *v_i is not currently either in a dilution process, broadcasting a dilution application or trying to start a dilution process as a block assembler* **then**
  if *random choice* **then**
    let v_i broadcast a dilution application;
  end
  else
    let v_i try to start dilution process as block assembler;
  end
end
increment dilution counter;
if *dilution counter* $= I_{depth} - I_{pre-depth}$ **then**
  let m_c create pre-depth block;
end
if *dilution counter* $= I_{depth}$ **then**
  let m_c create depth block;
end

Algorithm 4: DilutionPhaseSimulationRound(m_c, $V = \{v_1, ..., v_N\}$, dilution counter, I_{depth}, $I_{pre-depth}$)

13.2 Theoretical cost

We can calculate the size of the blockchain in bytes and the number of messages each voter has to send on the basis of several variables.

If N voters each register their keys, dilute their signatures in r rounds with dilution pools of size k to generate an anonymity set of size k^r and subsequently commit their vote hashes once, with m candidates that each have a tag of length t and a description of length d, the total blockchain will consist of:

1. 1 initialization block (95 bytes).
2. N registration blocks (437 bytes each).
3. 1 dilution start block (247 bytes).
4. $\frac{Nr}{k}$ dilution blocks with k old keys, new keys and signatures ($315 + 208k$ bytes each).
5. $r - 1$ pre-depth blocks (244 bytes each).
6. $r - 1$ depth blocks (240 bytes each).
7. 1 dilution end block ($237 + m(6 + t + d)$ bytes).

8. N commitment blocks (339 bytes each).
9. 1 commitment end block (249 bytes).

The total number of blocks is thus:

$$l_c = 1 + N + 1 + \frac{Nr}{k} + (r-1) + (r-1) + 1 + N + 1 = 2 + 2N + \frac{Nr}{k} + 2r$$

This amounts to a total size in bytes of:

$$95+437N+247+\frac{Nr}{k}(315+208k)+244(r-1)+240(r-1)+237+m(6+t+d)+339N+249$$

$$= 344 + 484r + m(6+t+d) + (776 + r(\frac{315}{k} + 208))N$$

The messages the average voter has to send, are:

1. $\frac{r(k-1)}{k}$ dilution applications.
2. $\frac{r(k-1)}{k}$ invites.
3. $\frac{r(k-1)}{k}$ pool responses.
4. $\frac{r}{k}$ pool messages.
5. r pool acknowledgements.
6. r new key messages.
7. $\frac{r}{k}$ unvalidated dilution blocks.
8. $\frac{r(k-1)}{k}$ signature messages.
9. $\frac{r}{k}$ dilution blocks.
10. 1 commitment block.
11. 1 vote message.

This amounts to a total number of messages per voter:

$$4\frac{r(k-1)}{k} + 3\frac{r}{k} + 2r + 2$$

$$= (\frac{k-1}{k} + 5)r + 2$$

If $N = 500000$, roughly the size of Belgium's most populous municipality Antwerp, the number of rounds is $r = 5$, the size of each pool is $k = 8$, the number of candidates is $m = 10$ and they each have a tag of length $t = 16$ and a description of length $d = 256$, this amounts to a blockchain of size $1.006GB = 0.9373GiB$ and 31.38 messages per voter.

For the parameters that we mentioned we would test in Section 13.1 this means a blockchain length and a number of messages as shown in Table 13.1.

N	r	k	l_c	$n_{broadcast}$	$n_{received}$	$n_{messages}$
4	1	2	14	7.5	22.5	30
	2		18	13	39	52
8	1		24	7.5	52.5	60
	2		30	13	91	104
16	1		44	7.5	112.5	120
	2		54	13	195	208
32	1		84	7.5	232.5	240
	2		102	13	403	416

Table 13.1: Theoretical blockchain length and number of messages broadcast per voter.

13.3 Results

Table 13.2 shows the blockchain length l_c, number of messages each voter broadcasts $n_{broadcast}$, number of messages each voter receives $n_{received}$ and the number of unvalidated messages each voter handles $n_{messages}$, averaged over maximum switch counters from pools of 5, 10 or 20 and maximum switch counters from applications of 5, 10, 20, or 30, and for each combination of parameters we average over 10 runs. In other words, every measured number in the table is the mean of $3 \cdot 4 \cdot 10 = 120$ values.

There appears to be a quadratic increase in $n_{broadcast}$ as N increases. This is reflected as a cubic increase in $n_{received}$ and $n_{messages}$, which is to be expected since each node should receive the messages broadcast by all other nodes. The most significant effect however appears to be an increase in r: $n_{broadcast}$ increases by roughly 90 messages. It is likely that this is due to participants sending dilution applications after they have already diluted their keys in a vain attempt to find leftover participants during the pre-depth phase.

Table 13.3 shows l_c, $n_{broadcast}$, $n_{received}$ and $n_{messages}$ for different maximum switch counters from pools and Table 13.4 shows the same values for different maximum switch counters from applications. Varying the maximum switch counter from pools does not appear to have much influence, but a maximum switch counter from applications of 30

seems to be optimal. Nevertheless, the effect is much smaller than the effects of increasing N or r.

N	r	k	l_c	$n_{broadcast}$	$n_{received}$	$n_{messages}$
4	1	2	14.00	9.100	36.39	45.49
	2		18.00	89.26	356.9	446.2
8	1		24.00	9.295	74.11	83.40
	2		30.09	96.81	755.6	852.4
16	1		44.00	10.11	160.3	170.4
	2		53.99	96.05	1530	1626
32	1		83.99	13.70	420.7	434.4
	2		102.0	107.0	3381	3489

Table 13.2: Measured average blockchain length and number of messages broadcast per voter.

s_p	l_c	$n_{broadcast}$	$n_{received}$	$n_{messages}$
5	46.27	53.71	831.9	885.7
10	46.26	53.68	842.1	895.9
20	46.25	54.36	844.2	898.6

Table 13.3: Measured average blockchain length and number of messages broadcast per voter for different maximum switch counters from pools.

s_a	l_c	$n_{broadcast}$	$n_{received}$	$n_{messages}$
5	46.26	57.43	899.9	957.5
10	46.28	54.52	853.7	908.4
20	46.26	52.14	808.0	860.2
30	46.25	51.58	795.9	847.6

Table 13.4: Measured average blockchain length and number of messages broadcast per voter for different maximum switch counters from applications.

N	r	k	l_c		$n_{broadcast}$		$n_{received}$		$n_{messages}$	
			$o = 0$	$o = 0.5$	$o = 0$	$o = 0.5$	$o = 0$	$o = 0.5$	$o = 0$	$o = 0.5$
4	1	2	14.00	14.00	9.500	10.44	37.96	54.58	47.46	65.02
	2		18.00	18.00	85.64	84.34	342.6	364.7	428.2	449.1
8	1		24.00	24.00	8.944	9.556	71.40	111.9	80.34	121.5
	2		30.00	30.00	88.37	89.69	705.5	761.5	793.9	851.2
16	1		44.00	44.00	10.58	10.12	167.8	223.9	178.4	234.1
32	1		83.90	84.00	14.65	14.21	457.5	560.8	472.2	575.0
	2		102.0	101.9	98.95	101.2	3142	3401	3241	3503

Table 13.5: Measured average blockchain length and number of messages broadcast per voter for networks without outside attackers compared to networks where there are half as many outside attackers as voters.

Table 13.5 shows l_c, $n_{broadcast}$, $n_{received}$ and $n_{messages}$ for different numbers of voters N and different numbers of rounds r, comparing the situation where there are no attackers ($o = 0$) to the situation where there are half as many attackers as voters ($o = 0.5$). In all cases $s_p = 10$ and $s_a = 30$. We see the length of the blockchain l_c is not affected while the number of messages broadcast per voter $n_{broadcast}$ increases on average by 1. Both $n_{received}$ and $n_{messages}$ however increase by a much higher number, and this number is not constant but increases as both N and r increase. It is less than a linear increase however, and the higher $n_{received}$ and $n_{message}$ get, the smaller the percentual increase caused by the attackers: for $n_{messages} = 47.46$ it increases by 37% and for $n_{messages} = 3241$ it increases by 8%.

Part IV

Conclusions & future work

Chapter 14

Conclusion

AnonVote satisfies the requirements of verifiability, privacy, eligibility, uniqueness and integrity for all practical purposes. Verifiability is ensured with the theoretical exception that senseless participants might disenfranchise themselves and thereby enable another person to take their ability to vote. Privacy is guaranteed after signature dilution except in the unlikely event that a group of conspiring voters, far outnumbering the victims they target, carefully direct network traffic to funnel the victims into the right dilution pools and subsequently bust their anonymity within these pools. This will however expose them as attackers to roughly the same extent as they expose the identity of the victim, presumably proving that they broke the law. Eligibility and uniqueness are both guaranteed, again with the theoretical exception that senseless participants might disenfranchise themselves and transfer their ability to vote to another person. The requirement of integrity is completely proven.

Of the four recommendations, coercion resistance, robustness, fairness and distribution of trust, some are satisfied but others are not. The system is not coercion resistant because it is an e-voting system. Fairness is achieved however as long as the electorate does not collectively circumvent it, and distribution of trust is achieved by default since no trust is needed.

The system is robust against denial-of-service attacks in the sense that nodes have various strategies at their disposal to defend against these attacks. The strategy to defend against outside attackers, who are not identified as members of the electorate, was tested experimentally with success: the number of messages handled by each node does not increase dramatically when attackers are present and the impact is lessened even further if the number of voters increases.

The system is as robust to eclipse attacks as a distributed system can be: if nodes are persistently isolated, the system is sabotaged, but if it is only temporary the system can recover. An eclipse attack on the election manager can undo a small amount of signature dilution, but voters can easily compensate for this risk by diluting their signatures one round further than they think strictly necessary.

When forks occur in the blockchain, they will be resolved since the fork resolution protocol has been proven to be valid.

If voters lose their keys while they have already diluted them, but before committing their vote, they are disenfranchised. This is a necessary consequence of using a digital system without a trusted third party. It is thus very important for voters to ensure that they will never lose their keys. It is not advisable to implement the system on a smartphone or another device that can easily be lost, but instead a PC should be used and voters should create backups of their keys that they can use in case they lose access to the data on the PC.

Robustness against network failure is similar to robustness against eclipse attacks: if the failure persists, the system cannot work because no distributed system can work without the internet. If the failure is temporary however, the system can mostly recover. If network failure causes voters to be unable to dilute their keys, they can fall back on a paper ballot. Network failure that persists during the entire Commitment phase however can disenfranchise voters, so it is important that care is taken that this phase lasts long enough and the internet works throughout the electoral precinct. In essence this issue is the digital counterpart of the possibility that some voters in a paper ballot election are unable to reach the polling station on election day.

The size of the blockchain is very manageable if elections are subdivided by municipality or electoral precinct, and there is no reason not to do this since anonymity beyond the electoral precinct is not expected. Voters theoretically only send a small number of messages that does not depend on the total number of voters but does depend linearly on the dilution rounds. In practice however the current implementation sees a quadratic increase in the number of messages broadcast as the number of voters increases, but more importantly there is a very high increase in the number of messages as the number of dilution rounds increases. The size of this increase does not appear to depend on the number of voters, but it is not clear whether the increase depends linearly on the number of dilution rounds, or by any other function. Therefore it is not clear if the increase will be dramatic if signature dilution takes place to a degree that creates decently sized anonymity sets.

Overall we can say that the protocol satisfies all the requirements to the extent that can be reasonably expected from an e-voting system, while it is unclear if the current implementation is ideal.

Chapter 15

Future work

It is important to carry out more simulations at a larger scale to measure the performance of the system in more realistic settings. It is of particular importance to increase the number of rounds of signature dilution during these simulations to see if the effects are an issue. It is also important to vary the number of members in each dilution pool, since all simulations used dilution pools of size 2. The effect of an increase in the number of voters can also be further tested to see if problems arise at the level of an electoral precinct.

It is also possible to see if the implementation of the client application can be improved upon to mitigate the effects of an increase in number of voters or number of dilution rounds. It is possible that the current strategy of randomly simply responding to any dilution application with an invite is not ideal for participants trying to start a dilution pool. Maybe it is more efficient for participants to judge dilution applications by looking at certain bits within their keys and prioritizing those where the value of these bits are equal to the value of the same bits in their own keys. This heuristic can be expected to naturally sort the participants and avoid conflicting network traffic. If the current strategy causes dramatic performance issues as the number of rounds of signature dilution increases, it may be advisable for the client application to be more judicious about sending out dilution applications after a pre-depth block is sent.

Robustness against attackers can also be further studied via simulations: the system was found to be robust against outside attackers sending invalid messages, but Chapter 8 also described attacks from eligible participants and strategies to defend against them. It is important to test the efficacy of these strategies as well.

The protocol can also be amended to account for different types of elections. In the current form, elections with multiple rounds are treated as separate elections that each require a new blockchain and thus a new Registration phase. It is possible to amend the protocol such that the last three phases can be repeated as a cycle, for each round of an election. For example there might be one Setup phase and one Registration phase, followed by a Dilution phase, a Commitment phase and a Voting phase for a round with all candidates, followed by another Dilution phase, Commitment phase and Voting phase for a runoff between the two winners of the previous round.

In Section 7.2.2 we described the possibility of eligible participants counterattacking

against a group of eligible participants that attack someone's privacy. It is possible to further study this idea to develop a protocol for eligible participants to actually carry out such a counterattack.

Bibliography

[1] A.V. Abhirama. "BLOCKCHAIN: ITS USES IN CRYPTOCURRENCY AND ELECTION SYSTEM". In: *International Research Journal of Modernization in Engineering Technology and Science* (2021), pp. 1571–1577.

[2] Ben Adida. "Helios: Web-based Open-Audit Voting". In: *USENIX security symposium*. Vol. 17. 2008, pp. 335–348.

[3] Federal Public Services Home Affairs. *Official Results: 2019 Belgian Federal Election*. 2019. URL: https://elections2019.belgium.be/en/results-figures?el=CK&id=CKR00000.

[4] Aritra Banerjee. "A Fully Anonymous e-Voting Protocol Employing Universal Zk-SNARKS and Smart Contracts". In: *International Congress on Blockchain and Applications*. 2021, pp. 349–354.

[5] Stephanie Bayer and Jens Groth. "Efficient Zero-Knowledge Argument for Correctness of a Shuffle". In: *Advances in Cryptology – EUROCRYPT 2012*. Ed. by David Pointcheval and Thomas Johansson. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 263–280. ISBN: 978-3-642-29011-4.

[6] Rumeysa Bulut et al. "Blockchain-Based Electronic Voting System for Elections in Turkey". In: *2019 4th International Conference on Computer Science and Engineering (UBMK)*. 2019, pp. 183–188.

[7] David L. Chaum. "Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms". In: *Communications of the ACM* 24.2 (1981).

[8] Rafer Cooley, Shaya Wolf, and Mike Borowczak. "Blockchain-Based Election Infrastructures". In: *2018 IEEE International Smart Cities Conference (ISC2)*. 2018, pp. 1–4.

[9] Inter-Parliamentary Council. *Declaration on Criteria for Free and Fair Elections*. 1994. URL: https://www.ipu.org/our-impact/strong-parliaments/setting-standards/declaration-criteria-free-and-fair-elections.

[10] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. "A Secure and Optimally Efficient Multi-Authority Election Scheme". In: *Advances in Cryptology — EUROCRYPT '97*. Ed. by Walter Fumy. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 103–118. ISBN: 978-3-540-69053-5.

[11] Taher ElGamal. "A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms". In: *IEEE Transactions on Information Theory* 4 (31 1985), pp. 469–472.

[12] Shawn M. Emery, C. Edward Chow, and Richard White. "Penetration Testing a US Election Blockchain Prototype". In: *Sixth International Joint Conference on Election Voting – E-Vote-ID 20221*. 2021, pp. 82–97.

[13] State Electoral Office of Estonia. "General Framework of Electronic Voting and Implementation thereof at National Elections in Estonia". In: (2017). URL: `https://www.valimised.ee/sites/default/files/uploads/eng/IVXV-UK-1.0-eng.pdf`.

[14] Council of Europe - European Commission for Democracy through Law (Venice Commission). *Electoral Law*. 2013.

[15] Steven Goldfeder et al. "When the cookie meets the blockchain: Privacy risks of web payments via cryptocurrencies". In: (2017). URL: `https://arxiv.org/pdf/1708.04748.pdf`.

[16] H.S. Govinda et al. "Implementation of Election System Using Blockchain Technology". In: *2021 International Conference on Innovative Computing, Intelligent Communication and Smart Electrical Systems (ICSES)*. 2021, pp. 1–9.

[17] Rolf Haenni et al. "CHVote System Specification Version 3.0". In: *IACR Cryptol. ePrint Arch.* 2017 (2017), p. 325.

[18] Rifa Hanifatunnisa and Budi Rahardjo. "Blockchain Based E-Voting Recording System Design". In: *11th International Conference on Telecommunication Systems Services and Applications (TSSA)*. 2017, pp. 1–6.

[19] Ethan Heilman et al. "Eclipse attacks on bitcoin's peer-to-peer network". In: *24th USENIX Security Symposium (USENIX SECURITY 15)*. 2015, pp. 129–144.

[20] *Help America Vote Act*. 2002.

[21] Fririk . Hjálmarsson et al. "Blockchain-Based E-Voting System". In: *2018 IEEE 11th international conference on cloud computing (CLOUD)*. 2018, pp. 983–986.

[22] *International Covenant on Civil and Political Rights*. 1966.

[23] Ali Kaan Koç et al. "Towards Secure E-Voting Using Ethereum Blockchain". In: *2018 6th International Symposium on Digital Forensic and Security (ISDFS)*. 2018, pp. 1–7.

[24] Wouter Lueks, Iñigo Querejeta-Azurmendi, and Carmela Troncoso. "VoteAgain: A scalable coercion-resistant voting system". In: *29th USENIX Security Symposium (USENIX SECURITY 20)*. 2020, pp. 1553–1570.

[25] Aanchal Mani et al. "College Election System using Blockchain". In: *ITM Web Conferences*. 2022, pp. 1–5.

[26] Gregory Maxwell. *CoinJoin: Bitcoin privacy for the real world*. 2013. URL: `https://bitcointalk.org/?topic=279249`.

[27] Satoshi Nakamoto. "Bitcoin: A Peer-to-Peer Electronic Cash System". In: (2008). URL: `www.bitcoin.org`.

[28] Chaals Nevile et al. *Enterprise Ethereum Alliance Client Specification v6*. 2020. URL: `https://entethalliance.org/wp-content/uploads/2020/11/EEA_Enterprise_Ethereum_Client_Specification_v6.pdf`.

[29] Kriti Patidar and Swapnil Jain. "Decentralized E-Voting Portal Using Blockchain". In: *2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)* (2019), pp. 1–4.

[30] Birgit Pfitzmann and Michael Waidner. "Unconditionally Untraceable and Fault-tolerant Broadcast and Secret Ballot Election". In: *Communications of the ACM* 21.21 (1992), pp. 7–8.

[31] Peter Y.A. Ryan et al. "The Prêt à Voter Verifiable Election System". In: *IEEE Transactions on Information Forensics and Security* 4.4 (2009), pp. 662–673.

[32] Kazi Sadia et al. "Blockchain-Based Secure E-Voting with the Assistance of Smart Contract". In: *IC-BCT 2019. Blockchain Technologies* (2019), pp. 161–176.

[33] Organization for Security and Cooperation in Europe. *Election Observation Handbook*. 2010.

[34] Shehan Shetty, Vishal Thakur Adhij Vartak, and Shraddha Dabhade. "Election Portal Using Blockchain". In: *SJCEM Journal of Engineering Sciences* 3 (1 2021), pp. 22–25.

[35] Majd Soud et al. "TrustVote: on elections we trust with distributed ledgers and smart contracts". In: *2nd Conference on Blockchain Research Applications for Innovative Networks and Services (BRAINS)*. 2000, pp. 176–183.

[36] Ishaan Anand Srivastava et al. "Secure and Transparent Election System for India using Block chain Technology". In: *2018 IEEE Punecon*. 2018, pp. 1–6.

[37] Baocheng Wang et al. "Large-scale Election Based On Blockchain". In: *Procedia Computer Science* (219 2018), pp. 234–237.

[38] Yang Yang et al. "PriScore: Blockchain-Based Self-Tallying Election System Supporting Score Voting". In: *IEEE Transactions on Information Forensics and Security* 16 (2021), pp. 4705–4720.

[39] Ehab Zaghloul, Tongtong Li, and Jian Ren. "d-BAME: Distributed Blockchain-Based Anonymous Mobile Electronic Voting". In: *IEEE Internet of Things Journal* 8.22 (2021), pp. 16585–16597.

[40] Saman Taghavi Zargar, James Joshi, and David Tipper. "A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks." In: *IEEE communications surveys tutorials* 4 (15 2013), pp. 2046–2069.

[41] Colin Zwirko. "Kim Jong Un left off list of officials elected to 14th Supreme People's Assembly". In: *NK News* (2019). URL: https://www.nknews.org/2019/03/kim-jong-un-left-off-list-of-officials-elected-to-14th-supreme-peoples-assembly/.