Pomona College
Department of Computer Science

Decentralized Group Management in Dissent

Eleanor Cawthon

April 25, 2015

Submitted as part of the senior exercise for the degree of
Bachelor of Arts in Computer Science

Professors Bryan Ford and Tzu-Yi Chen, advisors

Copyright © 2015 Eleanor Cawthon

The author grants Pomona College the nonexclusive right to make this work available for noncommercial, educational purposes, provided that this copyright statement appears on the reproduced materials and notice is given that the copying is by permission of the author. To disseminate otherwise or to republish requires written permission from the author.

Abstract

Decentralized approaches to private communication exhibit tradeoffs between decentralization and scalability. We present a protocol for achieving the best of both worlds.

Contents

Chapter 1

Introduction

A classic problem in human group interaction is how to make decisions in a way so that everyone is represented, but progress is still made. In very small groups, action can proceed by consensus - all members have the opportunity to be heard, and only actions that have the support of the entire group proceed. In any moderately sized group, however, this peer-to-peer approach to consensus becomes unweildly. Most governance structures implement some sort of delegation of power , whether by way of an elected legislature or a military dictator.

Although this has traditionally been characterized as a problem of communication at scale, we can also conceptualize it as a problem of trust. Participants in a democratic group place some amount of trust in the will of the consensus, but wish to avoid trusting any individual or small group with enough power for them to usurp the democratic process.

How, then, might such a group minimize the risk of assigning enough power to a small group for that group to misbehave, while maximizing the economies of scale arising from delegating power?

The electoral process itself is a particularly interesting example of this phenomenon. Elections frequently utilize secret ballots in order to prevent voters from being coerced into voting a particular way, but these schemes traditionally involve trusting a centralized entity to honestly count the votes. In contrast, election mechanisms that allow voters to verify their votes have been counted correctly, such as a vote by role call or raise of hand, typically sacrifice ballot secrecy.

Various computational approaches exist to addressing these limitations of traditional elections. The trust considerations decentralized groups face, however, extend beyond the voting process itself. Noam Chomsky writes

that "[t]he smart way to keep people passive and obedient is to strictly limit the spectrum of acceptable opinion, but allow very lively debate within that spectrum" [**?**]. To have truly free elections, the means for calling an election and for drafting the ballot must also be decentralized. For example, in many systems, there is a mechanism for calling a vote of no confidenceto potentially remove elected leaders in the middle of a term. Existing electronic voting protocols do not provide a way of protecting the identity of the voter who decides such a referrendum should take place. Further, voting over the internet poses additional trust problems beyond those inherent to electronic voting in general. A dissident group organizing in defiance of a powerful entity with control over the network must protect its members' anonymity not only from other group members, but from a global passive adversary who can analyze all message transmissions and traffic patterns among all nodes in the system.

We present a survey and analysis of secure, decentralized, consensus-based decision making in untrusted networks. We provide what we believe to be a novel examination of anonymity in the context of electronic voting, and propose a general specification for verifiable, anonymous, and decentralized group management.

We begin with an overview of existing tools dealing with various aspects of this problem (Chapter 2). We then outline important properties for voting protocols operating in this trust model, and introduce several important threat models to consider (Chapter 3). In Chapter 4, we provide a detailed specification for a protocol providing the properties laid out in Chapter 3. In Chapter 5, we outline one potential implementation of this specification.. Finally, we conclude and discuss directions for future work (Chapter 6).

Chapter 2

Related Work

This project brings together several disparate but related areas of computer science research. We examine existing work on decentralized decisionmaking, first in offline electronic voting protocols, then in fault tolerance, and finally in the anonymity protocols Tor and Dissent.

2.1 Existing Systems

2.1.1 Electronic Voting

Secure electronic voting systems have arisen largely out of a desire to retain secret ballots (no one should learn how a particular voter voted) while also guaranteeing accurate and fair counting of votes. Unlike distributed consensus protocols, secure electronic voting systems generally achieve their security properties through decentralization of trust rather than computation. They normally depend on a single executor of the vote aggregation protocol, using verifiability to ensure that each voter can be confident their vote was tallied fairly.

One solution is presented in [**?**], and the initial assignment of pseudonyms in Dissent already uses a variation on this protocol. In the Neff shuffle, each voter encrypts their vote in such a way that the aggregator must permute the vote ciphertexts before being able to decrypt them. The result is a permutation which no one knows — a voter can verify that their ciphertext is present in the permutation, but gains no information about the correspondence between other ciphertexts and other voters. It is both individually and universally verifiable.

The individual verifiability of [**?**] is based on each voter's retention of their secret key. *Coercion-resistant* electronic voting protocols remain ro-

bust even if secret keys are compromised: In [**?**], the voter uses their secret key only to establish eligibility, at which point they are assigned a random element of a well known set to use in their actual ballot. The ballots are unlinkable to the secret keys, and there is no way for an outsider to confirm whether a particular random element corresponds with a particular voter. This protocol achieves strong resistance to multiple kinds of coercion by deliberately weakening the eligibility verification property to depend on trust in the "registrar" who validates credentials and assigns the voting keys, preventing "forced-abstention" attacks (in which the adversary demands a voter simply not participate) by making it impossible for outsiders to verify the set of voters.

These protocols provide useful templates for how a distributed voting protocol might accomplish similar security properties.

2.1.2 Distributed Consensus

Distributed consensus protocols allow groups of nodes to come to an agreement on canonical values. For example, in a distributed database, if an unreliable power supply causes some portion of servers to be offline for each of several transactions, these protocols allow the servers to reconcile their records so that all servers agree on the transaction history. The problem was popularized by [**?**], which proposed the framing and solution now known as Paxos: A Paxos cluster that consists of $2f + 1$ nodes must have a *quorum* of $f + 1$ participating nodes at any given time. Transactions occur in three phases: First, the single current designated leader (Proposer) proposes the nth change. Next, if no other participants (Acceptors) have received a proposal numbered higher than n, the Acceptors promise to ignore future lower numbered requests. Finally, upon receiving $f + 1$ Promises, the Proposer declares success to all Acceptors. This allows the cluster to maintain a consistent record of transactions as long as a quorum is present.

2.1.3 Anonymity Protocols

Every anonymity tool shares one basic goal: Given a set of assumptions about an adversary's capabilities, an anonymity tool provides a way for a user to broadcast a message without the adversary being able to discover the author of the message. To illustrate the general approach and some common security assumptions, we first consider The Onion Router (Tor), the most widely used anonymity tool today[**?**]

Onion Routing with Tor

Tor aims to provide an anonymity service that, to the end user, behaves like a one-hop proxy: If Alicia wants to send an HTTP request to Badru using Tor, Alicia sends a request through Tor, Tor forwards the request to Badru, Badru replies to Tor, and Tor forwards the response to Alice. Under the surface, when Alicia's traffic enters the Tor network, it is encrypted and transmitted among several "onion routers" before reaching an "exit relay", which decrypts the request and passes it onto Badru. Each intermediate onion router only knows the next hop in the path from Alicia to Badru - no single node knows its traffic originated at Alicia or is en route to Badru - so no one in the network knows the complete path.

Tor provides anonymity from an adversary who "can observe some fraction of the network traffic; who can generate, modify, delete, or delay traffic; who can operate onion routers of [their] own; and who can compromise some fraction of the onion routers"[**?**]. If an adversary can observe much more than a small fraction of traffic, or if the adversary controls many colluding nodes, other attacks become possible, and the anonymity guarantees no longer hold. We now know that the U.S. National Security Agency actively uses such attacks, and so a new protocol is necessary in order to remain anonymous from the N.S.A.

Strong Anonymity with Dissent

Dissent is an alternative to Tor that provides provable anonymity even if only one server in the network is honest[**?**]. In its present form, a Dissent cluster consists of m servers and n connected clients[**?**]. Provable anonymity is achieved through a modified version of the Dining Cryptographers problem[**?**]: each client i shares a secret K_{ij} with each server j. Communication proceeds in rounds, within which each client has a designated k-bit slot. Before any messages are sent, a secure shuffle[**?**] assigns each client to a slot so that the owner of a slot is the only node in the system which knows who owns that slot. In any client s's slot, every client and every server generates k bits of random noise seeded with each of its shared secrets K_{ij}, and combines these with an exclusive or (xor) operation to produce that node's ciphertext. Client s also combines (via xor) a k-bit message with its noise to create its ciphertext. The combination (via xor) of all clients' and servers' ciphertext includes the noise stream associated with each shared secret twice, and so all noise cancels out and client s' message is revealed. However, since deciphering this requires the participation of all

nodes in the system, it is impossible to tell which client transmitted a message in a given slot. Dissent also incorporates an accountability mechanism, allowing any node that disrupts the protocol to be detected and removed from the cluster [**?**].

The original Dissent was fully peer-to-peer [**?**]. The shift to a client-server model allows for significantly improved performance, but it introduces several new concerns, particularly relating to misbehaving servers, a new class of DoS attacks, and group formation.

Chapter 3

Properties and Threat Models

3.1 Desired Properties

3.1.1 Verifiability

A protocol is verifiable if its output can be inspected to confirm that the protocol was carried out correctly. A simple example of this is signing a message with the private key associated with a well-known public key: Anyone who knows the public key can verify the validity of the signature.

Voting protocols can be evaluated in their provision of three different kinds of verifiability [**?**]: *individual* verifiability ensures that a voter can verify their vote was included correctly. *Universal* verifiability requires that anybody can verify the election result correctly represents the collection of ballots cast. Finally, *Eligibility* verifiability allows anybody to verify that only eligible voters voted, and that each voter voted only once.

3.1.2 Anonymity

Members of any group often face reprocussions if they participate in group governance in ways that run contrary to the interests of other members of the group. For this reason, election protocols often incorporate some notion of anonymity. A protocol guarantees *anonymity* in some operation a client can complete if the output of that operation is unlinkable (or, more precisely, cryptographically very difficult to link) to the participant who completed it[**?**].

We are interested in two types of anonymity: First, within an election, each voter's confidentiality should be preserved. Second, the instigator of an election, who may also be the author of the proposed petition, should be anonymous.

3.2 Threat Models

Newly available information about vulnerabilities and global-scale surveillance in today's centralized internet infrastructure has brought new security considerations and threat models to the foreground of networked system research. It has rendered a swath of anonymity and voting tools obsolete, and poses a significant threat to those that remain. Group management protocols aimed at dissidents must take this into account.

3.2.1 Byzantine Fault Tolerance

In shifting our focus from offline voting algorithms to networked protocols, we must consider several types of disruptions not taken into account by those algorithms. In particular, any member tasked with "broadcasting" a message may equivocate, sending different values to different participants. Existing work on distributed consensus provides various approaches to these problems.

Paxos and many similar protocols makes the simplifying assumption that all nodes in the system are honest — the only faults considered are those triggered by nodes suddenly going offline. If nodes may be malicious, they may "fail" not just by disappearing, but by forging messages in an effort to influence the consensus value. *Byzantine* consensus protocols allow the honest nodes in a system to arrive at a canonical value, so long as some minimum portion of nodes are honest. The original Paxos can accommodate Byzantine failures if an additional verification stage, in which all Acceptors communicate with all other Acceptors in order to detect equivocation, is added before the final step [**?**]. Additional optimizations to regular and Byzantine Paxos have also been developed [**?**]. If the adversary can not only send arbitrary messages but also monitor messages exchanged among other nodes, additional attacks are possible. One approach to this divides the nodes into small quorums in an effort to contain malicious nodes [**?**] while also providing better scalability than solutions that require all-to-all communication to thwart equivocators.

In both its standard and Byzantine formulations, the distributed consensus problem assumes discrepancies in the record will only be due to faults —

that is, each assumes all honest participants either agree on what the value should be or agree to accept the value reported by the nodes who do know the value. In the election of rotating leaders or servers, the correct value is not knowable a priori. If our leader election algorithm is modeled on other election protocols, it must be assumed that honest nodes may disagree on which servers they wish to elect.

3.2.2 Global Passive Adversary

To provide anonymity from an adversary like the N.S.A., a modern anonymity protocol must protect against several forms of attacks. Feigenbaum et. al.[**?**] highlight several specific attacks to which onion routing is vulnerable:

Global traffic analysis: If the adversary can monitor most of the traffic on the internet globally, the adversary can with high probability see the link from Alicia to the Tor network and from the Tor exit relay to Badru. This means the adversary can observe that the messages Alicia sends correspond to messages Badru receives some short amount of time later. Even if the messages themselves are encrypted, the adversary can analyze the lengths and other metadata about the messages to correlate this traffic.

Intersection attacks: In general, it is possible to tell when users are using an anonymity service — the anonymity comes from the difficulty of linking any particular user to particular messages produced by the service. Over time, however, the client set of an anonymity service is unlikely to remain fixed. A passive adversary monitoring the outputs of an anonymity service as well as the set of users connected can narrow the set of users who potentially, for example, updated a particular blog with a static pseudonym, by excluding users not online during all updates to the blog.

3.2.3 The Trouble with Relays

One potential approach to making Dissent widely available would be to have well-known, globally dispersed Dissent servers available for clients to connect to, similar to the current state of Tor. Any such well-known server list, however, is susceptible to blocking by internet service providers. It would therefore be preferable to have servers be short-lived, or at least not well known. Since Dissent takes place over regular TCP connections, detecting that the protocol is being executed without knowledge of the addresses of servers would be difficult to accomplish without a great number of false

positives [**?**], so this may be enough to realistically preclude most attempts to block access to the protocol entirely. Additionally, while the current version of Dissent guarantees that malicious servers cannot deanonymize a client without the cooperation of all servers, and guarantees that disrupting servers can be exposed, it provides no way to remove a disrupting or malicious server form the system.

One way to resolve these problems would be to have clusters of Dissent clients elect temporary servers among themselves, allowing servers to either step down (e.g., by going offline) or be impeached by some portion of the clients. Doing so in a truly decentralized and fair fashion is a non-trivial problem. We consider several other areas of research relevant to solving it.

Chapter 4

General Specification

In this chapter, we outline a general specification for verifiable, anonymous, and fully decentralized election protocols. For simplicity, we assume all votes have only two options (ratify or not), but not that it can be proven that any other multiple choice ballots can be constructed from these components and thus our analysis is fully generalizable.

4.1 Terminology and Data Structures

- This protocol operates at the level of a `Cluster` of `Node`s. A `Peer` is a `Node` that is a member of a particular `Cluster`. It may be voted out of a cluster. New nodes become `Peer`s if their joining is approved by the existing cluster.
- A `Manifest` defines the group configuration and consists of
  - A `Roster`, representing the set of eligible voters
  - A map `Committees` mapping names of elected statuses to sets of elements of the `Roster`. For example, `Committees` might consist of the key "`servers`", with the value $\{A, B, C\}$, where A, B, and C are the participants listed in R which have been elected to act as `servers` in an instance of Dissent in Numbers.
  - A function $t : E \to (\{\textsc{True}, \textsc{False}, \textsc{Invalid}\}, P)$ where E is the set of all *Election State*s, and P is the set of all possible proofs of correctness of the result. That is, if $t(g) = \textsc{True}$ for some outcome g, then the proposal corresponding to $g$passes. A plausible example is the function which specifies what proportion

of `Peer`s must agree to a change in the composition of the `Roster` in order for the change to take effect.

- A `Petition` is a proposal to change the `Manifest`. It consists of
  - An *Instigator*, the `Peer` who proposed the `Petition`. This information is not publicly associated with the `Petition`.
  - A unique identifier L, which is public
  - A proposed new `Manifest`,
  - An expiration condition, defining when the vote on the petition should end.
- A `Ballot` encodes information about the eligibility of the voter, and information about the voter's preference. To determine the results of the elction while providing the verifiability properties discussed in Section 3.1.1, there must be a public record of some aggregate information about each: An auditor must be able to tell that every voter was eligible, and also what the outcome of the election was. To provide voter confidentiality, we must provide a way for each voter to provide both bits of information without exposing the correlation between the two. In other words, if Badru wants to vote for Alicia to be president, Badru must convey that Badru (or someone with Badru's credentials) voted, and that a vote has been cast for Alicia, without revealing that Badru cast a vote for Alicia. We can represent the information Badru provides as a *Vote* tuple $(sig, vote)$, where *sig* encodes Badru's credentials and *vote* encodes his candidate choice.

  To provide the necessary information while preserving his confidentiality, Badru must encrypt part or all of his ballot. We show in Appendix **??**that it is impossible to design a Byzantine Fault Tolerant protocol where both are kept secret. This leaves two possibilities: Either Badru can encode his credentials in a *sig* that is anonymous (c.f. [**?**]) or he can encrypt *vote* so that Badru's choice of candidates can only be decyphered in aggregate, once the connection to Badru's public signature has been lost.

4.2 States

- A *State* is a tuple (M, r) defining the current cluster, where M is a `Manifest`, and r is a monotonically increasing unique state identifier (i.e., a logical clock).

- An *Election* encompasses the operation of the protocol between when a `Petition` P is first proposed and the round P.`rid` — that is, the portion of the protocol where members are aware that that `P` is being considered, but in which no member knows the outcome of the election.
- The *Election State* can be described by a tuple $((M, r), P, V)$, where (M, r) is the current *State*, P is the `Petition` being voted on, and V is a collection of `Votes` that have been cast thus far.

4.3 Functions

Each `Peer` should implement the following functions: Within finite time and in a way that is fair, every `Peer` should be able to call each of the following functions:

- PROPOSE(`Petition` P): broadcasts P to the cluster and initiates an *Election*
- VOTE$_{\texttt{State}(M,r)}$(`Petition` P), which constructs a ballot (sig, $vote$) and broadcasts it to the cluster in conjunction with P
- EVALUATE$_{\texttt{State}(M,r)}$(`Petition` P, `Votes` V) $\rightarrow$ ($\{$UNFINISHED, VALID, INVALID$\}$, $proof$: Given an election state, every peer should be able to determine whether the election has ended, and if it has, what the result was. If it cannot, or if any part of the election state is invalid, it should be able to provide a $proof$ of misbehavior.
- EVALUATE$_{\texttt{State}(M,r)}$(`Petition` P, `Vote` v) $\rightarrow$ ($\{$TRUE, FALSE$\}$, $proof$) should return (TRUE, $proof$) if v was produced by a valid `Peer` according to (M, r), and is a vote on P. Otherwise, TRUE and a proof of misbehavior should be produced.

4.4 Formal Properties

4.4.1 Verifiability

Individual

A group management protocol provides *individual verifiability* if, in any Election State $((M, r), P, V)$, any member u either knows its own vote is correctly represented in V (that is, either u voted and u's signature for $P.L$is contained in G, or u did not vote and u's signature is absent from G), or can produce a zero-knowledge proof that the Election State is invalid.

Universal

A group management protocol provides *universal verifiability* if, in any finished election state, $((M, r), P, V)$anybody can verify that V is a valid signing of P or else produce a proof that it is not. Consequently, any auditor (member or otherwise) can verify the canonical value of $M.t(G)$.

Eligibility

A group management protocol provides *eligibility verifiability* if, in any finished election state, anybody can verify that all elements of V were cast by `Member`s of the current cluster.

4.4.2 Anonymity

For Instigators

A group management protocol provides *instigator anonymity* if, during and after any election, no member and no outside observer can determine which member proposed the ballot in question.

Of Ballots

A group management protocol provides *secret ballots* if, during and after any election, either no outside observer can reconstruct which member submitted which vote, or no outside observer can reconstruct how any member voted. The same restrictions apply to knowledge gained by other participants, except that each member can trivially reconstruct its own vote.

Chapter 5

Protocol Description

We operate a simple voting protocol based on linkable ring signatures [**?**], on top of a Dissent instance that provides instigator anonymity, along with acccountability to handle byzantine faults.

5.1 Initial formation

We assume the member set is well known, and that every member has a secure channel through which it can communicate with every other member, and that members remain connected. Initially, the members organize themselves into a Peer-to-Peer Dissent cluster [**?**] using some consensus protocol such as Byzantine Paxos, as discussed in [**?**].

The Dissent configuration file specifies the size and ordering of clients' message slots. which will consist of, for each of the n clients: space for a `Ballot`, and space for up to n signatures so the client can sign whatever other proposals are in play. We only allow any given client to have one proposal at any given time.

5.2 Peer-to-Peer Layer

Every `Member` maintains a TCP connection to every other `Member`, and uses this to execute the peer to peer version of Dissent [**?**]. The primary limitation of this version is performance — there is a detailed security analysis demonstrating the anonymity preserving properties of this layer, and so we use it as a fallback layer for reconfiguring if servers misbehave by dropping out, and for initial setup of the Dissent in Numbers (DIN) layer.

For any instance of the DIN layer, there is a canonical `Manifest` maintained at the P2P layer describing its parameters. The `Manifest` consists of

- A `Roster` R, mapping public keys to IP addresses for all `Client`s,
- A `Servers` list S, which is a subset of R,
- A function $t : G \to \{\textsc{True}, \textsc{False}\}$ mapping a set of signatures to an election result. So, if $t(g) = \textsc{True}$ for some outcome g, then the proposal corresponding to g should be adopted at the specified expiration round. A plausible example is the function which specifies what proportion of `Member`s must agree to a change in the composition of R or S in order for the change to take effect

When a vote to change the `Manifest` passes, the `Member`s newly designated as servers begin running server instances of Dissent in Numbers, and all `Member`s (including the ones now running servers) begin running client instances.

5.3 Dissent-in-Numbers Layer

The communication involved in establishing the `Manifest` takes place over an instance of Dissent in Numbers [**?**]. We sketch a black box model of Dissent in Numbers as it relates to our protocol.

An instance of DIN consists of n clients and m servers. Communication takes place in *rounds*, wherein each client has an opportunity to broadcast a message to the entire client set. Assuming k of the n clients are honest, each client is guaranteed that, at the protocol level, its message will be anonymous among the k honest clients unless all servers collude with each other.Since all clients receive each client's messages in a deterministic order, there is a well-defined sequence of rounds which we can associate with a monotonically increasing *round ID*.

5.4 Voting with Linkable Ring Signatures

Our voting protocol is based on the concept of a linkable ring signature (LRS), introduced in [**?**]. The documentation of the CryptoBook implementation in Go explains:

"The caller supplies one or more public keys representing an anonymity set, and the private key corresponding to one of those public keys. The resulting signature proves to a verifier that the owner of one of these public keys signed the message, without revealing which key-holder signed the message, offering anonymity among the members of this explicit anonymity set. The other users whose keys are listed in the anonymity set need not consent or even be aware that they have been included in an anonymity set: anyone having a suitable public key may be "conscripted" into a set.

If the provided anonymity set contains only one public key (the signer's), then this function produces a traditional non-anonymous signature, equivalent in both size and performance to a standard ElGamal signature.

The caller may request either unlinkable or linkable anonymous signatures. If linkScope is nil, this function generates an unlinkable signature, which contains no information about which member signed the message. The anonymity provided by unlinkable signatures is forward-secure, in that a signature reveals nothing about which member generated it, even if all members' private keys are later released. For cryptographic background on unlinkable anonymity-set signatures - also known as ring signatures or ad-hoc group signatures - see [**?**].

If the caller passes a non-nil linkScope, the resulting anonymous signature will be linkable. This means that given two signatures produced using the same linkScope, a verifier will be able to tell whether the same or different anonymity set members produced those signatures. In particular, verifying a linkable signature yields a linkage tag. This linkage tag has a 1-to-1 correspondence with the signer's public key within a given linkScope, but is cryptographically unlinkable to either the signer's public key or to linkage tags in other scopes. The provided linkScope may be an arbitrary byte-string; the only significance these scopes have is whether they are equal or unequal. For details on the linkable signature algorithm this function implements, see [**?**].

Linkage tags may be used to protect against sock-puppetry or Sybil attacks in situations where a verifier needs to know how many distinct members of an anonymity set are present or signed

> messages in a given context. It is cryptographically hard for one anonymity set member to produce signatures with different linkage tags in the same scope. An important and fundamental downside, however, is that linkable signatures do NOT offer forward-secure anonymity. If an anonymity set member's private key is later released, it is trivial to check whether or not that member produced a given signature. Also, anonymity set members who did NOT sign a message could (voluntarily or under coercion) prove that they did not sign it, e.g., simply by signing some other message in that linkage context and noting that the resulting linkage tag comes out different. Thus, linkable anonymous signatures are not appropriate to use in situations where there may be significant risk that members' private keys may later be compromised, or that members may be persuaded or coerced into revealing whether or not they produced a signature of interest."

[**?**]

Within a round, each `Member` may transmit a `Petition`. A `Petition` consists of:

- A proposed *Manifest*, as described above,
- A *Link Scope*
- A *Round ID* when the ballot will expire.

Once a *Petition* has been proposed, the other `Member`s have the opportunity to *vote*. A `Member` votes by transmitting the most recent version of the `Ballot`, but with the `Signatures` field modified to include the proposed `Manifest` signed with the voting `Member`'s private key for this link scope.

By the designated expiration round, all `Member`s have enough information to determine whether or not the `Petition` *passes*: Each `Member` should verify all signatures on the most recent versionand compare the total number of valid signatures to the threshold t. If the `Ballot` passes, the new server set should immediately set up the next iteration of the DIN layer.

5.5 Limitations and Non-Goals

Intersection Attacks: The `Peer` and `Member` sets are known. In Dissent in Numbers, clients need not know the IP addresses of any other clients. We

believe it is useful to have a protocol for a group with static membership. If there is membership churn, our protocol remains vulnerable to intersection attacks.

Coercion-Resistance: In order to use anonymous ring signatures for voting, it is necessary for each signature within a scope to correspond to a specific member. An adversary with the power to coerce `Member`s into revealing their private keys after the fact may prove that a particular member voted a particular way[**?**].

Forward Progress: A protocol that guarantees forward progress given certain conditions will eventually make progress as long as those conditions are met. More strict bounds on what "eventually" means are possible. In the original Dissent, for example, forward progress can be guaranteed if all clients follow the protocol and remain online, but not otherwise: The accountability mechanism was so arduous that f disrupting clients could prevent any messages from being transmitted for f hours [**?**]. Protocols that make use of quorums rather than being fully peer-to-peer are able to provide stronger guarantees of forward progress [**?**]. Since we do not handle traditional fault tolerance or network churn, this is an area for future work

Active attacks: Global traffic analysis attacks only require the adversary to be able to monitor global traffic. If the adversary can also modify or generate traffic, several other attacks are possible. The adversary can launch *man-in-the-middle* (MITM) attacks in which it impersonates Alicia, Badru, or one or more Tor nodes. The adversary can launch *Sybil* attacks, in which many different Tor clients controlled by the same adversary join the network as individual clients. Either of these can be used to create *Denial-of-Service* (DoS) attacks, which might either prevent users from connecting to Tor at all, or force Tor traffic to go through particular, potentially adversary-controlled, Tor nodes — de-anonymizing the users.

Chapter 6

Conclusion

As networked communication becomes increasingly integral to the communications of humans in collectives, questions of security too become essential to analysis of voting systems. We have argued that verifiability is not enough for a post-Snowden electronic voting protocol. We have specified and analyzed the properties a voting protocol would need to provide in order to conform to a revised trust model, wherein peers are not required to trust one another, and wherein the adversary is assumed to be able to monitor all network traffic. We have further contributed the first voting protocol we know of that provides the verifiability guarantees of the electronic voting literature, the strong anonymity of Dissent, and the fully decentralized trust model of Byzantine peer-to-peer systems. We believe this will be useful in constructing systems for use by activists and governments alike as these infrastructures develop.