Pomona College
Department of Computer Science

Decentralized Group Management in Dissent

Eleanor Cawthon

April 25, 2015

Abstract

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

added to make it compile, this abstract is not a draft

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   TODO: cite
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   TODO: cite
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.   TODO: cite

Various computational approaches exist to addressing these limitations   TODO: transition
of traditional elections. The trust considerations decentralized groups face,   sentence?
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" [CBN98]. 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 claling 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.

TODO: cite

TODO: capitalize or hyphenate?

We present a protocol for egalitarian groups to determine their leadership in a fashion that is anonymous, verifiable, and reversible. By combining the Dissent in Numbers protocol for anonymous communication with decentralized trust, with a simple voting protocol utilizing linkable ring signatures, we show how a group might attain accountable scalability: where the scalable protocol is used most of the time, but where the peer-to-peer consensus can always rescind the power it has delegated.

TODO: rewrite this to be in terms of the above desired properties

Chapter 2 provides an overview of existing work on decentralized decisionmaking, with particular attention to anonymity and electronic voting systems, before situating our contribution with discussion of these systems' limitations. Chapter 3 outlines the functionality and security properties our protocol sets out to provide. Chapter 4 describes our protocol specification. Chapter 5 formalizes the properties set out in Chapter 3 and includes proofs that our protocol satisfies them. Chapter 6 concludes.

TODO: conclude

Chapter 2

Background

Anonymous communication significantly constrains the ability of oppressive regimes and vigilante groups alike to suppress dissent. Newly available information about vulnerabilities and global-scale surveillance in today's centralized internet infrastructure has rendered a swath of anonymity tools obsolete, and poses a significant threat to those that remain.

A trustworthy anonymity tool in the post-Snowden era must be resilient to both surveillance and censorship: It should guarantee its users' anonymity even in the face of a global passive adversary, and it should be unrealistic for such an adversary to simply prevent users from accessing it. A useful anonymity tool must also perform with reasonably low latency - a property which often trades off with security and availability.

This review will first examine the existing anonymity tools The Onion Router (Tor) and Dissent, considering their ability to provide strong anonymity guarantees. Higher performance versions of both Tor and Dissent, however, rely on well-known relay servers which present challenges for availability. Section ?? will examine decentralized, peer-to-peer techniques that might be used to improve Dissent's availability.

> TODO: Rework this intro

2.1 Distributed Decision-Making

2.1.1 Goals/Threat Models

> TODO: COPY EDIT, this doesn't make sense here

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. Dissent [CGWF13] and the Neff shuffle [Nef01] both use zero-knowledge proofs to achieve verifiability.

Accountability: In the context of Dissent, accountability refers to the ability of a protocol to detect and exclude participants who disrupt the protocol [SCGW$^+$14], while proving that the disruptor did indeed disrupt the protocol. Such a mechanism is necessary in peer-to-peer protocols like Dining Cryptographers in which a single disruptor can make the result of a round unusable. In Dissent, accountability checks occur without revealing the link between any message and its sender — moreover, it is not possible to deliberately exclude a participant on the basis of valid messages the participant sends. That is, the Dissent accountability mechanism does not break anonymity.

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 [CGWF13]. Protocols that make use of quorums rather than being fully peer-to-peer are able to provide stronger guarantees of forward progress [Lam98].

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 client who completed it [CGF10]. Dissent makes use of pseudonyms to provide this, separating the protocol correctness and accountability layer from the layer in which messages are revealed. Group membership voting could conceivably take place at either.

Fault Tolerance

TODO: I'm not really using any of the fault tolerance stuff; may want to remove this section or just talk briefly about Byzantine threat models

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 [Lam98], 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.

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 [CL99]. Additional optimizations to regular and Byzantine Paxos have also been developed [Lam06]. 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 [KLST11] 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.

2.1.2 Electronic Voting Existing Work

TODO: beef up this section.

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. Voting protocols can be evaluated in their provision of three different kinds of verifiability [KRS10]: *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.

One solution is presented in [Nef01], 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 [Nef01] is based on each voter's retention of their secret key. *Coercion-resistant* electronic voting protocols remain robust even if secret keys are compromised: In [JCJ05], 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.

2.2 Anonymity

TODO: Set this up as problems with electronic voting schemes

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[For14]

2.2.1 Onion Routing with Tor

TODO: this section is probably not needed

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"[DMS04]. 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.

2.2.2 New Threat Models

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.[FF13] highlight five 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.

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.

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.

TODO: conclusion

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

TODO: copy edit Tor vs. Dissent

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 [HBS13], 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.

2.2.3 Strong Anonymity with Dissent

> TODO: Especially if I do take out the related work, I should add a short intro to why Dissent is a good thing to use

Dissent is an alternative to Tor that provides provable anonymity even if only one server in the network is honest[CGF10]. In its present form, a Dissent cluster consists of m servers and n connected clients[WCGFJ12]. Provable anonymity is achieved through a modified version of the Dining Cryptographers problem[Cha88]: 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[Nef01] 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 [CGWF13].

The original Dissent was fully peer-to-peer [CGF10]. 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.

Dissent in Numbers

TODO: Explain it and its properties

2.2.4   Linkable Ring Signatures

TODO: Explain

Chapter 3

Goals and Non-Goals

In transforming Dissent in Numbers[WCGFJ12] to a fully decentralized context, some differential terminology is necessary. We define several components of our protocol before moving on to discuss desired security properties.

3.1 Terminology

The protocol consists of two logical layers: A Dissent in Numbers layer, where most communication takes place, and an underlying peer-to-peer layer where management of the Dissent configuration occurs. We refer to the entire protocol as Dissent, and refer to the Dissent in Numbers layer as *DIN*.

We refer to any single running instance of our protocol on a single machine as a `Peer`. When a `Peer` is part of a Dissent cluster, it is a `Member` of that cluster.

Within the DIN layer, each `Member` takes on the role of a `Client` as described in [WCGFJ12]. In addition, a collectively-chosen subset of these `Member`s also runs a `Server` process (as described in [WCGFJ12]).

> TODO: Probably not relevant to mention Low Latency Dissent, but may be worth reworking this paragraph to refer to the possibility of generalizing

Based on membership operations, `Member`s may take on additional roles. In Dissent in Numbers a subset of `Member`s would also be `Server`s. In Low-Latency Dissent one `Member`s would also be (a) `Relay`(s). These roles are subject to change with future membership proposals.

3.2 Assumptions/Non-Goals

TODO: turn bullet points into paragraphs

TODO: formalize in distributed systems verification terms (initially, next, etc.)

We present the assumptions and goals of our model in informal terms here, before formalizing the definitions as part of proving the properties are satisfied in Chapter 5.

3.2.1 Assumptions

- There is an initial `Manifest` describing a Dissent-in-Numbers setup, known to all `Members` it describes.

- While the `Member` set may change according to votes, the `Peer` set is fixed.

3.2.2 Limitations

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

- 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 `Members` into revealing their private keys after the fact may prove that a particular member voted a particular way.

TODO: Need some kind of analysis of why this is still useful despite that

- Further, intermediate vote counts are necessarily public. We prove generally in Appendix A that we cannot do better.

3.3 Goals

3.3.1 Desired Properties

Progress:

- If a valid `Member` wants to propose a `Ballot`, it can do so within a finite number of rounds

- If a `Ballot` is proposed, every `Member` should have the opportunity to vote on it

- If t of n clients vote for a `Ballot`, that `Ballot` should take effect within a finite number of rounds.

Decentralization:

- No changes to the manifest occur without t of the n clients' approval of the new manifest.

- From any possible state, a cadre of t of n clients can cause the manifest (and corresponding configuration) to change to any other valid manifest.

TODO: Define state better

Security:

- `Member`s who disrupt the protocol can be detected and removed, without deanonymizing participants

TODO: Cite [CGWF13]; this is solved

- Votes should be

  - Anonymous — No member can learn which member voted which way.
  - Verifiable — Once a vote is complete, any member can, given a canonical `Ballot` containing s signatures, verify how many of the s correspond to distinct, valid members' votes, and also that their own vote is included.

- While there are at least t honest `Member`s, no changes to the topology should occur without t of the n clients' approval

- If there are fewer than t honest `Member`s, those `Member`s should still retain strong anonymity among the honest clients, even if they no longer control the topology

TODO: This is fuzzy, explain better

3.3.2 Adversary Model

We assume the adversary can

- Monitor all network traffic

- Control some fraction of the `Peers`, which may send arbitrary messages to each other through secret channels and to the group through ordinary channels

- Inject traffic

TODO: Do we? Specify this

13

Chapter 4

Protocol Description

4.1 Peer-to-Peer Layer

Every `Member` maintains a TCP connection to every other `Member`.

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 ratio t specifying 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,

TODO: Describe how to start a Dissent instance. This is a solved problem, I just need to summarize it.

4.2 Dissent-in-Numbers Layer

The communication involved in establishing the `Manifest` takes place over an instance of Dissent in Numbers [WCGFJ12]. 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*.

> TODO: Move the security properties to the goals section and only describe the functionality here

The DIN configuration file specifies the size and ordering of clients' message slots.

> TODO: Need to define client and slot somewhere

Illustrate the round message format, 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.

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

> TODO: Jamming by proposing ballots??? Infinite timeouts???

- A proposed *Manifest*, as described above,

- A *Link Scope*

> TODO: Explain

- A collection of `Signatures`

> TODO: Explain

- A *Round ID* when the ballot will expire.

Once a *Ballot* has been proposed, the other `Members` 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.

> TODO: I think this is wrong

By the designated expiration round, all `Members` have enough information to determine whether or not the `Ballot` *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 prepare for the next iteration of the DIN layer.

> TODO: What if there are conflicting versions?

> TODO: finish describing new setup

Chapter 5

Security Properties and Correctness

5.1 Progress

Given a `Collective`a `Peer` can be in any one of the following states:

- *Vanguard*: The `Member` has a new `Manifest` it wishes to advance, either by proposing it or by adding its signature to an already-proposed version.

> TODO: Add transitions (as in words)

> TODO: define

> TODO: This may not actually make sense as a proof structure. Within a configuration, communication is serial.

5.2 Security

5.2.1 Privacy

> What the adversary should not learn

5.2.2 Integrity

> What the adversary should not break

Accountability: t of n nodes can reverse any decision Verifiability: individual and universal, not coercion resistant. Anonymity: No one knows how anyone else voted

Chapter 6

Conclusion

Todo list

Bibliography

[CBN98]   N. Chomsky, D. Barsamian, and A. Naiman. *The common good*. Real story series. Odonian Press, 1998.

[CGF10]   Henry Corrigan-Gibbs and Bryan Ford. Dissent: Accountable anonymous group messaging. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, pages 340–350, New York, NY, USA, 2010. ACM.

[CGWF13]   Henry Corrigan-Gibbs, David Isaac Wolinsky, and Bryan Ford. Proactively accountable anonymous messaging in verdict. In *Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13)*, pages 147–162, Washington, D.C., 2013. USENIX.

[Cha88]   David Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1(1):65–75, January 1988.

[CL99]   Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, New Orleans, USA, February 1999.

[DMS04]   Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, August 2004.

[FF13]   Joan Feigenbaum and Bryan Ford. Seeking anonymity in an internet panopticon. *arXiv preprint arXiv:1312.5307*, 2013.

[For14]   Bryan Ford. Hiding in a panopticon: Grand challenges in internet anonymity, February 2014.

[HBS13]   A. Houmansadr, C. Brubaker, and V. Shmatikov. The parrot is dead: Observing unobservable network communications. In *2013 IEEE Symposium on Security and Privacy (SP)*, pages 65–79, May 2013.

[JCJ05]   Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society*, WPES '05, pages 61–70, New York, NY, USA, 2005. ACM.

[KLST11]   Valerie King, Steven Lonargan, Jared Saia, and Amitabh Trehan. Load balanced scalable byzantine agreement through quorum building, with full information. In Marcos K. Aguilera, Haifeng Yu, Nitin H. Vaidya, Vikram Srinivasan, and Romit Roy Choudhury, editors, *Distributed Computing and Networking*, number 6522 in Lecture Notes in Computer Science, pages 203–214. Springer Berlin Heidelberg, January 2011.

[KRS10]   Steve Kremer, Mark Ryan, and Ben Smyth. Election verifiability in electronic voting protocols. In Dimitris Gritzalis, Bart Preneel, and Marianthi Theoharidou, editors, *Computer Security ESORICS 2010*, number 6345 in Lecture Notes in Computer Science, pages 389–404. Springer Berlin Heidelberg, January 2010.

[Lam98]   Leslie Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, May 1998.

[Lam06]   Leslie Lamport. Fast paxos. *Distributed Computing*, 19(2):79–103, October 2006.

[Nef01]   C. Andrew Neff. A verifiable secret shuffle and its application to e-voting. In *Proceedings of the 8th ACM Conference on Computer and Communications Security*, CCS '01, pages 116–125, New York, NY, USA, 2001. ACM.

[SCGW+14] Ewa Syta, Henry Corrigan-Gibbs, Shu-Chun Weng, David Wolinsky, Bryan Ford, and Aaron Johnson. Security analysis of accountable anonymity in dissent. *ACM Transactions on Information and System Security*, 17(1):1–35, August 2014.

[WCGFJ12] David Isaac Wolinsky, Henry Corrigan-Gibbs, Bryan Ford, and Aaron Johnson. Dissent in numbers: Making strong anonymity

scale. In *Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*, pages 179–182, Hollywood, CA, 2012. USENIX.

Appendix A

Why Intermediate Vote Counts Can't Be Secret

I have discovered a truly marvellous proof of this, which this margin is too narrow to contain.