Pomona College
Department of Computer Science

Decentralized Group Management in Dissent

Eleanor Cawthon

October 23, 2015, 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

Abstract

Among both humans and computers, decentralized approaches to group decision-making exhibit trade-offs between decentralization and scalability.

We provide two contributions: First, we outline a general specification for election protocols providing instigator anonymity. Next, we sketch how this can be applied to provide bootstrapping and group management for the Dissent anonymity protocol.

We go on to present a protocol for egalitarian groups to determine their leadership in a fashion that is anonymous, verifiable, and fully decentralized. By combining the Dissent protocol for anonymous communication with decentralized trust[CGF10], with a simple voting protocol utilizing linkable ring signatures[LWW04], we show how a group might attain verifiable and anonymous elections with Byzantine trust assumptions, secure against a global passive adversary. As a specific example, we show how this might be used as a server selection and group management mechanism in scalable Dissent [WCGFJ12], so that the scalable protocol is used most of the time, but where the peer-to-peer consensus can always rescind the power it has delegated.

Contents

Chapter 1

Introduction

In Italy in 1922, Benito Mussolini legally became Prime Minister. Over the next several years, the Italian Parliament passed a series of electoral reforms that ultimately allowed for the Fascists control of the Italian Parliament and granted Benito Mussolini complete, authoritarian control of Italy. Benito Mussolini established martial law, and it took years of war and millions of casualties to reverse this. Of course, Benito Mussolini did have a great deal of popular support, at least initially. He promised security and order in a time of lawlessness. As the saying goes, Mussolini made the trains run on time. But whether or not a majority of the people of Italy ever supported the Fascists, by the time the Italian Parliament had given them control, the people could not revoke his power without years of war and millions of casualties.

This is one example of a more general problem in group decision-making: In order to maintain the advantages of democracy and decentralized trust, while also allowing efficient governance at scale, it must be possible to delegate power to individuals or small groups, and it must be possible to revoke that power.

Various computational approaches exist to providing secure and trustworthy democratic elections, where each individual's vote is anonymous and where the result is verifiable. As the case of Benito Mussolini shows, however, securing the election process itself is not enough when a central power determines whether and what elections should be held . 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 for a vote and drafting the ballot must also be decentralized and secure

against coercion. We shall henceforth refer to this process of initiating a vote as a *petition*. If the Italian Parliament had a mechanism for anonymously petitioning for a vote of no confidence, for example, it might have been possible to determine support for removing Benito Mussolini without endangering the instigator of that vote and without resorting to years of war and millions of casualties.

This paper makes three contributions. First, we provide what we believe to be a novel examination of anonymity in the context of electronic voting. Next, we propose a specification and implementation sketch for a verifiable, anonymous, and decentralized petition protocol. Finally, we show how this can be applied to provide group management in the Dissent in Numbers[WCGFJ12] anonymity protocol, with applications for scalable anonymous web browsing.

We begin with an overview of existing tools dealing with various aspects of this problem (Chapter 2). We then outline the properties provided by our protocol (Chapter 3), and specify the threat model against which it is secure (Chapter **??**). 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. Three existing bodies of work are particularly useful here: For verifiability, we turn to offline electronic voting protocols (which centralize computation but not trust). For decentralization, we look to work on fault tolerance and distributed consensus.Finally, we review existing work on anonymous group communication.

2.1 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 [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 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.

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.

2.3 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 transmit 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]

2.3.1 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"[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.

2.3.2 Strong Anonymity with Dissent

Dissent is an alternative to Tor that provides provable anonymity for k honest users even if every other node in the network is dishonest [CGF10]. Provable anonymity is achieved through a modified version of the Dining Cryptographers problem[Cha88]: each node i shares a secret K_{ij} with each other node j. Communication proceeds in rounds, within which each node has a designated b-bit slot. Before any messages are sent, a secure shuffle[Nef01] assigns each node 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 node s's slot, every node generates b 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. Node s also combines (via xor) a b-bit message with its noise to create its ciphertext. The combination (via xor) of all nodes' ciphertext includes the noise stream associated with each shared secret twice, and so all noise cancels out and node s'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].

2.4 New 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.

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 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.

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 3

Security Properties

We identify three crucial properties for a group decisionmaking protocol that keeps power decentralized.

First, the protocol's computation must be distributed. If the adversary has control of the network infrastructure, it can easily block access to any central component of a protocol (such as a Tor relay or voting registrar). If blocking one or a small subset of nodes in the group is sufficient to break the protocol, this again weakens our decentralization guarantees and renders the protocol less useful. A fault tolerant, distributed protocol, however, can easily recover from arbitrary changes in the network topology.

Next, the protocol must be verifiable — any particiapnt or external auditor should be able to examine a transcript of the protocol and determine whether or not it was performed honestly. Without this property, it would be necessary to trust that some portion of the group acted honestly without verification — this constitutes centralized trust, and is therefore inadequate.

Finally, the protocol must be anonymous — it should be computationally hard to attribute any vote or any petition to any specific participant. This is essential to preserve the safety of members suggesting or voting for unpopular proposals.

We now formalize these notions:

3.1 Distributed Computation

Our intention in defining a decentralization property is to approximate the human notion of decentralized power: We would like to ensure that initially, all nodes are situated equally, and that it is impossible for the system to enter a state in which any centralization or delegation of power is irrevokable.

In practice, we take this to mean

- We make no assumptions about any subset of nodes having particular capabilities not available to all nodes,
- k honest nodes can always recover from any attempt to disrupt the protocol by $n - k$ colluding malicious nodes.

3.2 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 [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.

3.3 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[For14].

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.

3.4 Threat Model

We assume the adversary controls an arbitrary subset of the nodes in the network — that is, it sees their internal state, and also controls what messages

they send. We additionally assume the adversary can monitor all messages transmitted among all nodes (an NSA-like global passive adversary).

3.5 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[LWW04].

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]. Since we do not handle traditional fault tolerance or network churn, this is an area for future work.

Chapter 4

Formal Specification

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

The protocol involves two layers: An anonymous broadcast channel whereby participants can pass arbitrary messages, and the petition protocol which utilizes this to construct petitions.

4.1 Anonymous Broadcast Protocol

We assume an anonymous communication layer that provides the following functionality:

4.1.1 Definitions

An instance of the anonymous communication layer consists of:

- A fixed peer set $P := p_1, p_2, ..., p_n$.
- A pseudonym scheme $d : P \longleftrightarrow Y$, where Y is a set of pseudonyms and d is a bijection
- A monotonically increasing turn number r, associated with a (possibly blank) message m_r in the broadcast channel signed by pseudonym $d(p_i) \in Y$.

- A `Schedule` function $s : \mathbb{N} \to Y$ mapping rounds to pseudonyms, establishing whose turn it is. Each client should have their i^{th} term before any client has its $i + 1^{th}$ turn.
- A history oracle, which all clients can query to learn the messages transmitted in previous rounds.

4.1.2 Interface

Every `Peer` p has the following functions available to it in polynomial time:

- RECEIVE$(r) \to (y, m)$ allows p to query the history oracle and returns the r^{th} message along with its owner y (with $s(r) = y$)
- SEND(m) — if called at `Turn` r, the `Peer` p broadcasts its message m at the next turn where $s(r + i) = p$, with $i \leq n$. Equivalently, the history oracle is updated such that, after turn $r + i$, any peer calling RECEIVE$(r + i)$ will retrieve $(d(p), m)$.

4.2 Anonymous Petition Protocol

This protocol operates at the level of a `Cluster` of `Peer`s. A `Member` is a `Peer` that is participating in a particular `Cluster`. It may be voted out of a cluster. New `Peer`s become `Member`s if their joining is approved by the existing cluster.

4.2.1 Data Structures

A `State` is a tuple $(r, \cdot)$ encoding the number of rounds r of messages elapsed since the start of the protocol, and any additional state information about the cluster.

A `Petition` is a proposal for the `Cluster` to vote on. It consists of

- An `Instigator`, the `Peer` who proposed the Petition. This information is not publicly associated with the `Petition`,
- A proposal text m,
- A set of ballot choices C, and
- An expiration condition $x : E \to \{\text{TRUE}, \text{FALSE}\}$ defining when the vote on the petition should end.

A `Ballot` encodes information about the eligibility of the voter (e.g., a signature *sig*), and the voter's preference $c \in$ `Petition`.C. [1]

An *Election* encompasses the operation of the protocol between when a `Petition` P is first proposed and when the expiration condition is met — 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 $((r, \cdot), P, V)$, where $(r, \cdot)$ is the current `State`, P is the Petition being voted on, and V is a collection of `Ballot`s that have been cast thus far.

A `Manifest` defines the group configuration and consists of

- A `Roster`, representing the set of eligible voters
- A function $t : E \to (\{(\textsc{Valid}, c), \textsc{Invalid}\}, F)$ where E is the set of all *Election State*s, and F is the set of all possible proofs of correctness of the result. That is, if for some outcome e, we have $t(e) = ((\textsc{Valid}, c), proof)$, then the ballot choice $c \in e.P.C$ passes. A plausible example of such a t 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.
- Any other group configuration information the group should be able to vote on.

[1] To determine the results of the elction while providing the verifiability properties discussed in Section 3.2, 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 `Ballot` 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. It is impossible to design a performant and Byzantine fault tolerant protocol where both are kept secret. (I have discovered a truly marvelous proof of this, which this margin is too narrow to contain). This leaves two possibilities: Either Badru can encode his credentials in a *sig* that is anonymous (c.f. [LWW04]) 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 (as in Dissent).

4.2.2 Interface

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

- p.PROPOSE(M, P): p constructs a `Petition` P and invokes SEND(P) to send it to the cluster for consideration.
- p.VOTE$(M, P, (sig, vote))$: If $P.x(\cdot) =$ FALSE, then a p may VOTE on it by creating a `Ballot` (sig, vote)indicating its preferred outcome, and invoking SEND$(P, (sig, vote))$ to send it to the group.
- EVALUATE$(M, P, V) \rightarrow (\{$UNFINISHED, VALID, INVALID$\}, c, proof)$: Given a vote state, every peer should be able to determine whether the vote 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$(M, P, (sig, vote)) \rightarrow (\{$TRUE, FALSE$\}, proof)$ should return (TRUE, *proof*) if $(sig, vote)$ was produced by a valid `Peer` according to $(r, \cdot)$, and is a vote on P. Otherwise, FALSE and a proof of misbehavior should be produced.

4.3 Security Properties

Verifiability

Individual A group management protocol provides *individual verifiability* if, in any Election State $((r, \cdot), P, V)$, 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 V, or u did not vote and u's signature is absent from V), 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, $((r, \cdot), P, V)$, 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 $M.t(e)$.

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.

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 now sketch an implementation of the specification from Chapter 4: a simple voting protocol based on linkable ring signatures [LWW04], on top of a Hardened Dissent [SCGW+14] instance that provides instigator anonymity, along with acccountability to handle byzantine faults.

5.1 Building Blocks

We combine several existing cryptosystems and distributed systems to construct our protocol.

5.1.1 Hardened Dissent

A security analysis of the original peer-to-peer Dissent resulted in a modified protocol providing the following properties:

Theorem 1. *At the end of turn r, either all participants see the same value of all messages, any peers disrupting the protocol are detected and eliminated, or the round does not terminate.*

Theorem 2. *For any turn r, all participants know the results of round r before any know the results of round $r + 1$.*

Theorem 3 (Anonymity)**.** *When Hardened Dissent is executed and completes in a cluster of n participants, of which k are honest, none of the avenues of attack descibed in our threat models... provide non-negligable advantage over uniform guessing in determining which of the k honest participants sent any particular message.*

All properties are rigorously proven in [SCGW+14].

This satisfies the specification for an anonymous messaging protocol given in Section 4.1.

5.1.2 Linkable Ring Signatures

Our voting protocol is based on the concept of a linkable ring signature (LRS), introduced in [LWW04]. 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.
>
> [...]
>
> [G]iven 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. ... 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.

[For15]

5.2 Algorithms

5.2.1 Initial formation

We assume the member set is well known, 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 [CGF10] using some consensus protocol such as Byzantine Paxos, as discussed in [SCGW+14].

Initially, the `Manifest` of the Petition protocol layer includes all `Node`s in the Broadcast layer as `Member`s of the `Cluster`.

5.2.2 Voting with Linkable Ring Signatures

We now sketch implementations of the functions from Section 4.2.2.

To PROPOSEa `Petition`,

By associating a unique link scope with each petition, we allow `Member`s to vote by producing a signature with the current `Roster`and the proposer's link scope. Each `Member` that wishes to vote for the petition broadcasts the petition along with its signature. Dissent provides acccountable, verifiable broadcasting of anonymous messages, and so given our reliability assumptions, every `Member` will eventually end up with a list of anonymous signatures associated with the petition. At the specified round, each member should verify all signatures received. If the function t applied to the set of unique, valid signatures results in TRUE, then the petition passes.

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 V, 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 `Petition` passes, the new server set should immediately set up the next iteration of the DIN layer.

5.3 Arguments for Correctness

5.3.1 Verifiability

Our protocol provides all three types of verifiability specified in Section 3.2.

Theorem 4. *This protocol is Verifiable*

Proof.

Lemma 5. *The* Election State *of any* Election*at any time is well defined.*

Proof. This follows from properties of peer-to-peer Dissent [SCGW+14]. The *Election State* can only be updated by transmission of messages (votes) over Dissent. Recall that communication proceeds in serialized *rounds*, so by Theorem 5.1.1, it is impossible for two `Peer`s to have conflicting versions of the election state at any given round. □

Given this, we know that every `Peer` has accurate knowledge of the election state if it has any knowledge of the election state. From here, the verifiability properties follow directly from the properties of linkable ring signatures.

□

5.3.2 Anonymity

Although the voting protocols discussed in Section 2.1 provide voter anonymity (secret ballots) within an election, they do not on their own provide instigator anonymity. This protocol provides both.

Lemma 6. *Votes are k-anonymous among the k honest users of the system*

Proof. This follows directly from Theorem 5.1.1. □

Using this, we can show:

Theorem 7. *This protocol provides Secret Ballots*

Proof. Anyone monitoring traffic can trivially link a vote to its Dissent pseudonym, but since this changes every round [SCGW+14], this provides no information beyond the scope of a single petition. □

Theorem 8. *This protocol provides Secret Instigators*

Proof. Similarly, since the Hardened Dissent layer runs at all times, every participant is essentially actively submitting an anonymous petition or non-petition every round, and so neither intersection attacks nor traffic analysis can determine the identity of the instigator of any petition. □

Chapter 6

Conclusion

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.

Future work will involve a more detailed specification and analysis of the protocol sketched in Chapter 5. Work has already begun on implementing it and integrating the election of servers into future versions of Dissent, utilizing the implementation of linkable ring signatures available in Go [For15]. Once it is known to be secure, the changes should be fully implemented and performance should be analyzed in the face of various kinds of disruption, and also client churn.

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 believe this will be useful in constructing systems for use by activists and governments alike as these infrastructures develop.

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.

[For15] Bryan Ford. Go crypto library: github.com/dedis/crypto/anon. 2015.

[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.

[LWW04] Joseph K. Liu, Victor K. Wei, and Duncan S. Wong. Linkable Spontaneous Anonymous Group Signature for Ad Hoc Groups. In Huaxiong Wang, Josef Pieprzyk, and Vijay Varadharajan, editors, *Information Security and Privacy*, number 3108 in Lecture Notes in Computer Science, pages 325–335. Springer Berlin Heidelberg, 2004.

[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

Use Case: Resilient Dissent in Numbers

Anonymous communication significantly constrains the ability of oppressive regimes and vigilante groups alike to suppress dissent. Newly available information about global-scale surveillance in today's centralized internet infrastructure has rendered a swath of anonymity tools vulnerable.

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.

In its current form, Dissent in Numbers lacks the ability to handle server faults, and it depends on a pre-existing well-known set of servers. However, it also provides dramatically superior performance as compared with Hardened Dissent. Our protocol can be used to elect servers for an instance of Dissent in Numbers [WCGFJ12], which can then be used for latency-sensitive applications without sacrificing the superior security properties of Hardened Dissent.

A.1 Background: Dissent in Numbers

Peer-to-peer Dissent provides stronger anonymity guarantees than Tor, but at the cost of significantly degraded performance. A new version of the protocol called Dissent in Numbers [WCGFJ12] aims to provide similarly strong anonymity properties at scale. For a single sender to transmit a 128

kB message to a 16-member group under [CGF10] requires more than a minute; under [WCGFJ12] this takes less than 5 seconds.

Dissent in Numbers achieves this performance by using a n clients/m server model. Rather than requiring n^2 shared secrets (one for each pair of clients), Dissent in Numbers uses only one shared secret per server/client pair, for a total of mn. Instead of communicating directly with every other node, each client comunicates only with every server in each round.

Dissent in Numbers provides anonymity among the honest clients so long as at least one of the servers is honest. This means that in a system with k clients out of n, peer to peer Dissent can be shown to always preserve k-anonymity among the honest clients, whereas in Dissent in Numbers, only m need be compromised, regardless of the number of honest clients. Further, peer-to-peer Dissent is able to respond to changes in the topology of the group, including the case where one or more members is removed due to misbehavior. In Dissent in Numbers, the servers are assumed to always be reliable, and while clients can detect server misbehavior, there is no built-in mechanism to respond to this while continuing to use the protocol.

Even where a sufficient number of servers are honest and non-disruptive, Dissent in Numbers introduces additional vulnerability to censorship. 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 [HBS13], so this may be enough to realistically preclude most attempts to block access to the protocol entirely.

We now show how the protocol presented in this paper can be used to bootstrap Dissent in Numbers, so that in the usual case clients can achieve its superior performance, but in error cases the cluster can recover from these sorts of faults. Since our protocol is suitable for execution at the level of a local cluster, it further extends the advantages of Dissent in Numbers to environments where server identities must be ephermeral to avoid censorship.

A.2 Anonymous Petitions for Group Management in Dissent

Our protocol enables clusters of Dissent clients to 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.

We assume a cluster of n peers connected by TCP connections. To launch an instance of Dissent in Numbers, they first launch an instance of our protocol. The `Manifest`should contain a `Server`s field, containing the subset of `Roster`members currently acting as servers. This field may initially be blank, or it may be specified offline.

Once a server set is spefified, all participants have all information they need to launch a Dissent in Numbers instance [WCGFJ12]. Initially, and whenever a vote to change the `Manifest` passes at the Anonymous Petition layer, 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.

By using this protocol to handle group membership, server election, and fault tolerance, we bring resiliance, fault tolerance, and some forms of censorship resistance to Dissent in Numbers, providing a framework for a more versatile system.