Decentralized Group Management in Dissent

Eleanor Cawthon

October 5, 2015, 2015

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

TODO: In a unionization campaign, the Student/Worker Alliance was established as an equalizing force: workers and students alike could share in governance, with one vote per person. The management, however, could see the transcripts of all meetings of the Student/Worker Alliance. If a worker made a proposal unpopular with management, the worker would be fired; if a student made the same proposal, management could do nothing.

Or,

TODO: 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 dele-

gate 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 Mussolinishows, 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 Parliamenthad a mechanism for anonymously petitioning for a vote of no confidence, for example, it might have been possible to determine support for removing Benito Mussoliniwithout endangering the instigator of that vote and without resorting to years of war and millions of casualties.

This paper makes three contributions. First, we 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 Numbersanonymity 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 4). In Chapter 5, we provide a detailed specification for a protocol providing the properties laid out in Chapter 3. In Chapter 6, we outline one potential implementation of this specification. Finally, we conclude and discuss directions for future work (Chapter 7).

TODO: maybe communist elections are a better example here

TODO: Probably could be better/fix the rest of this section for real outline

TODO: cite

TODO: proofs section?

TODO: fewer "outline"

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 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 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 [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.

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 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]

2.3.1 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 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.

2.3.3   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 [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.

TODO: copy edit Tor vs. Dissent

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 3

Goals

TODO: intro

3.1 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 [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.

3.1.1 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.

TODO: more params; define lrs

TODO: define invalid

3.1.2 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 $M.t(G)$.

3.1.3 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.

3.2 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[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.

3.2.1 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.

3.2.2 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 4

Threat Model

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.

4.1   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 [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.

4.2 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.[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.

4.3   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].

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

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

lulz

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 5

General 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.

5.1 Anonymous Broadcast Protocol

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

5.1.1 Definitions

- An instance of the anonymous communication layer consists of:

    - A fixed peer set $P := p_1, p_2, ..., p_n$.
    - A pseudonym scheme $y : P \longleftrightarrow Y$, where Y is a set of pseudonyms and y is a bijection
    - A monotonically increasing turn number r, associated with a (possibly blank) message m_r in the broadcast channel signed by pseudonym $y(p_i) \in Y$.
    - A $\texttt{Schedule}$ function $s : \mathbb{N} \to Y$ mapping rounds to pseudonyms, establishing whose turn it is.

5.1.2 Interface

Peers have the following functions available to them:

- At any round r, Any peer p_i can pseudononymously broadcast a message $m_{n(p_i)}$ to P at round $r + j$ for some $j \leq n$

- At the end of round r, all $p \in P$ learn the contents of m_r

5.2 Anonymous Petition Protocol

5.2.1 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 $t : E \to (\{\textsc{True}, \textsc{False}, \textsc{Invalid}\}, F)$ 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 $t(e) = \textsc{True}$ for some outcome e, then the proposal corresponding to e 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.

  > TODO: Obsolete. Make generic.

- 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, 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.

> this is kindof out of no where. maybe move?

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. [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.

> TODO: I think this is actually only true if people vote at different times

5.2.2 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).

> TODO: cite

- An *Election* encompasses the operation of the protocol between when a Petition P is first proposed and the round $P.\text{r}$ — 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 `Ballots` that have been cast thus far.

5.2.3 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$_{\text{State}(M,r)}$(`Petition`P), which constructs a `Ballot`$(sig, vote)$ and broadcasts it to the cluster in conjunction with P

- Evaluate$_{\text{State}(M,r)}$(`Petition`P, `Ballot`$(sig, vote)) \to (\{\text{Unfinished}, \text{Valid}, \text{Invalid}\}, proof$: 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$_{\text{State}(M,r)}$(`Petition`P, `Ballot`$(sig, vote)) \to (\{\text{True}, \text{False}\}, proof)$ should return (True, $proof$) if $(sig, vote)$ 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.

Chapter 6

Protocol Description

We now sketch an implementation of the specification from Chapter 5: 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. Our protocol uses this 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.

6.1 Building Blocks

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

6.1.1 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 round i, 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.*

TODO: explain

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

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 thread models in Section **??** *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].

6.1.2   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.
>
> 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 [RST01].
>
> 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 [LWW04].

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."

[For15]

> TODO: TODO: explain in own words instead of having a page and a half long quote

6.1.3   Dissent in Numbers

An instance of Dissent in Numbers 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*.

6.2 Algorithms

6.2.1 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 [CGF10] using some consensus protocol such as Byzantine Paxos, as discussed in [SCGW⁺14].

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.

6.2.2 Peer-to-Peer Layer

Every `Member` maintains a TCP connection to every other `Member`, and uses this to execute Hardened 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 `Clients`,

- A `Servers` list S, which is a subset of R,

- A function $t : G \to \{\text{TRUE}, \text{FALSE}\}$ mapping a set of signatures to an election result. So, if $t(g) = \text{TRUE}$ 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
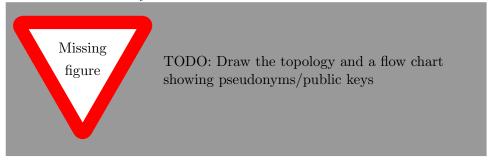
When a vote to change the `Manifest` passes, the `Member`s newly designated as servers begin running server instances of Dissent in Numbers, and all `Member`s (including the ones now running servers) begin running client instances.

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

6.2.3   Dissent-in-Numbers Layer

TODO: Describe uses of DIN

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 adds group management, fault tolerance, and some forms of censorship resistance to Dissent in Numbers, providing a framework for a more versatile system.

Missing figure

TODO: Draw the topology and a flow chart showing pseudonyms/public keys

6.2.4   Voting with Linkable Ring Signatures

By associating a unique link scope with each petition, we allow `Member`s to vote by producing a signature with the current member set 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, veriiable 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.

Missing figure

TODO: 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 `Petition`. A `Petition` consists of:

- A proposed *Manifest*, as described above,

- A *Link Scope*

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

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

Once a *Petition* 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: Explain

TODO: I think this is wrong

By the designated expiration round, all `Members` have enough information to determine whether or not the `Petition` *passes*: Each `Member` should verify all signatures on the most recent version and 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.

TODO: What if there are conflicting versions?

TODO: finish describing new setup

6.3 Arguments for Correctness

6.3.1 Verifiability

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

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 6.1.1, 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.

> TODO: Here's more proof detail:
>
> **Lemma 6.** *Given accurate knowledge of the current election state* $((M, r), P, V)$, *a* `Peer` *can verify that its vote has been included or excluded correctly.*
>
> *Proof.* a `Peer` p can call $\mathtt{Sign}(M.\mathtt{Roster}.keys, P.\mathtt{LinkScope}, m.k, P)$ to produce a signature s, then examine V to see if it contains s. □

□

6.3.2 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 7. *Votes are k-anonymous among the k honest users of the system*

Proof. This follows directly from Theorem 6.1.1. □

Using this, we can show:

Theorem 8. *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 9. *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 7

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 6. 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.

Todo list

27

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 Tre-
           han. Load balanced scalable byzantine agreement through
           quorum building, with full information. In Marcos K. Aguil-
           era, Haifeng Yu, Nitin H. Vaidya, Vikram Srinivasan, and
           Romit Roy Choudhury, editors, *Distributed Computing and
           Networking*, number 6522 in Lecture Notes in Computer Sci-
           ence, pages 203–214. Springer Berlin Heidelberg, January 2011.

[KRS10]   Steve Kremer, Mark Ryan, and Ben Smyth. Election veri-
           fiability 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. Com-
           put. 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. Link-
           able Spontaneous Anonymous Group Signature for Ad Hoc
           Groups. In Huaxiong Wang, Josef Pieprzyk, and Vijay Varad-
           harajan, 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.

[RST01]   Ronald L. Rivest, Adi Shamir, and Yael Tauman. How to Leak a Secret. In Colin Boyd, editor, *Advances in Cryptology ASI-ACRYPT 2001*, number 2248 in Lecture Notes in Computer Science, pages 552–565. Springer Berlin Heidelberg, 2001.

[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.