Dissent: Accountable Anonymous Group Messaging

Henry Corrigan-Gibbs and Bryan Ford
Department of Computer Science
Yale University
New Haven, CT, USA
henry.corrigan-gibbs@aya.yale.edu, bryan.ford@yale.edu

ABSTRACT

Users often wish to participate in online groups anonymously, but misbehaving users may abuse this anonymity to disrupt the group's communication. Existing messaging protocols such as DC-nets leave groups vulnerable to denial-of-service and Sybil attacks, Mix-nets are difficult to protect against traffic analysis, and accountable voting protocols are unsuited to general anonymous messaging.

We present the first general messaging protocol that offers provable anonymity with accountability for moderate-size groups, and efficiently handles unbalanced loads where few members wish to transmit in a given round. The N group members first cooperatively shuffle an $N \times N$ matrix of pseudorandom seeds, then use these seeds in N "pre-planned" DC-nets protocol runs. Each DC-nets run transmits the variable-length bulk data comprising one member's message, using the minimum number of bits required for anonymity under our attack model. The protocol preserves message integrity and one-to-one correspondence between members and messages, makes denial-of-service attacks by members traceable to the culprit, and efficiently handles large, unbalanced message loads. A working prototype demonstrates the protocol's practicality for anonymous messaging in groups of $40+$ members.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*Security and Protection*; C.2.2 [**Computer-Communication Networks**]: Network Protocols—*Applications*

General Terms

Algorithms, Security

Keywords

Anonymity, Accountability, Denial of Service, Group Communication, Peer-to-Peer Networks, Verifiable Anonymous Shuffle

1. INTRODUCTION

Anonymous participation is often considered a basic right in free societies [43]. The limited form of anonymity the Internet provides is a widely cherished feature [37, 41], enabling people and groups with controversial or unpopular views to communicate and organize without fear of personal reprisal [34]. Yet anonymity makes it difficult to trace or exclude misbehaving participants [13]. Online protocols providing stronger anonymity, such as mix-networks [9, 21] and DC-nets [10, 22, 32, 40], further weaken accountability and yield forums in which no content may be considered trustworthy and no defense is available against anonymous misbehavior.

This paper focuses on providing anonymous messaging within small, private online groups. We assume a group's membership is closed and known to its members; creating groups with secret membership is a related but orthogonal goal [38]. Members may wish to send messages to each other, to the whole group, or to a non-member, such that the receiver knows that *some* member sent the message but no one knows *which* member. Members may also wish to cast secret ballots in votes held by the group, or to create pseudonyms under which to collaborate with other members.

We also wish to hold members *accountable*, however, not by compromising their anonymity and allowing some authority or majority quorum to unmask a member whose messages prove unpopular, but rather by ensuring that no malicious member can abuse his (strong) anonymity to disrupt the group's operation. For example, a malicious member should be unable to corrupt or block other members' messages, overrun the group with spam, stuff ballots, or create unlimited anonymous Sybil identities [17] or sock puppets [36] with which to bias or subvert the group's deliberations.

As a motivating example, suppose an international group of journalists wishes to form a "whistleblowing" publication analogous to WikiLeaks [42]. To protect journalists and their sources, member journalists wish to submit leaked documents and related information to the group anonymously. Member journalists need assurance that powerful organizations or governments cannot trace the leak to an individual journalist or her source. The journalists wish to prove to their readers that leaked documents come via a trustworthy channel, namely one of the group's known and reputable members, and not from an outsider. The group must be able to analyze and vet each document thoroughly before collectively approving it for publication. The group must protect its internal operation and its members' anonymity even from adversaries who have planted colluding spies within the group. And this security must come at acceptable time and resource costs.

We present an accountable anonymous messaging protocol called Dissent (Dining-cryptographers Shuffled-Send Network), the first we know of with the properties needed in scenarios like the one outlined above. Dissent offers integrity, anonymity, and accountability in the face of strong traffic analysis and compromised members. An experimental prototype shows Dissent to be efficient enough for latency-tolerant messaging in small distributed groups.

In contrast with mix-networks [9,21] and DC-nets [10,22,32,40], Dissent implements a *shuffled send* primitive, whereby each group member sends *exactly* one message per round, making it usable for

CCS'10, October 4–8, 2010, Chicago, Illinois, USA.

voting or assigning pseudonyms with a 1-to-1 correspondence to real group members. Unlike verifiable cryptographic shuffles [20, 26], Dissent uses only readily-available cryptographic primitives, and handles arbitrarily large messages and unbalanced loads efficiently, such as when one journalist has a multi-gigabyte document to leak while the others have nothing to send. While group and ring signatures [4,11,30] can anonymously authenticate messages transmitted via some anonymous transmission channel, signatures offer no protection against anonymous denial-of-service (DoS) or Sybil attacks against the transmission channel itself, as Dissent does.

Dissent operates in two stages, *shuffle* and *bulk transfer*. The shuffle protocol builds on a data mining protocol by Brickell and Shmatikov [7] to permute a set of fixed-length messages, one from each group member, and broadcast the set of messages to all members with cryptographically strong anonymity. Like many anonymous messaging protocols, the original data mining protocol was vulnerable to untraceable DoS attacks by malicious group members. Our refinements remove this vulnerability by adding *go/no-go* and *blame* phases, which can trace and hold accountable any group member maliciously disrupting the protocol.

Dissent's bulk protocol builds on DC-nets [10,22,32,40] to transmit variable-length messages anonymously. In place of the DoS-prone slot reservation systems in prior DC-nets schemes, however, Dissent leverages its shuffle protocol to prearrange the DC-nets transmission schedule, guaranteeing each member exactly one message slot per round. In each round, all group members broadcast bit streams based on pseudorandom seeds distributed via the shuffle protocol, so that XORing all members' bit streams together yields a permuted concatenation of all members' variable-length messages. Cryptographic hashes distributed in the shuffle phase enable members to verify the correctness of each others' bulk transmissions, ensuring message integrity and DoS protection throughout.

Dissent has limitations, of course. It is not intended for large-scale, "open-access" anonymous messaging or file sharing [12,21], although it might serve as a building block in designs like Herbivore [32]. Dissent's accountability properties assume closed groups, and are ineffective if a malicious member can leave and rejoin the group under a new (public) identity after expulsion. Dissent is also not a general-purpose voting system: for example, it provides only a limited form of coercion resistance. Finally, the serialized shuffle protocol imposes a per-round startup delay that makes Dissent impractical for latency-sensitive applications.

We built a working prototype of Dissent and tested it under Emulab [18] on groups of up to 44 nodes connected via simulated wide-area links. Anonymously distributing messages up to 16MB in size among 16 nodes with 100ms inter-node delays, Dissent's shuffle protocol and other startup costs incur a 1.4-minute latency. Dissent handles large message loads, both balanced and unbalanced, in about $3.5\times$ the time required for non-anonymized group messaging via TCP. Varying group size, Dissent can send a 1MB message anonymously in less than 1 minute in a 4-node group, 4 minutes in a 20-node group, and 14 minutes in a 40-node group. While not suitable for interactive workloads, Dissent should be usable for "WikiLeaks"-type scenarios requiring strong security guarantees in small but decentralized groups.

This paper makes four main technical contributions. First, we enhance Brickell/Shmatikov's shuffle protocol [7] to make DoS attackers traceable without compromising anonymity. Second, we use this shuffle protocol to create a DoS-resistant DC-nets variant for bulk transfer, which guarantees each member exactly one transmission slot per round. Third, we introduce the first shuffle protocol that supports arbitrary-size and unbalanced message loads efficiently, e.g., when only one member has data to send. Fourth, we demonstrate through a working prototype the practicality of the protocol, at least for delay-tolerant applications.

Section 2 outlines Dissent's communication model, security goals, and operation. Section 3 describes the shuffle protocol, and Section 4 details the bulk transfer protocol. Section 5 informally covers practical implementation and usage considerations such as protocol initiation, coercion resistance, and liveness. Section 6 describes our prototype implementation and experimental results. Section 7 summarizes related work, and Section 8 concludes.

2. PROTOCOL OVERVIEW

This section first introduces the group communication model our protocol implements, outlines a few applications of this model, and defines the protocol's precise security goals, leaving protocol details to subsequent sections.

Dissent consists of two sub-protocols: a *shuffle* protocol and a *bulk* protocol. The shuffle protocol has two practical limitations: all messages must be of equal length L, incurring $O(NL)$ extra communication if only one member wishes to send; and its decrypt-and-shuffle phase is inherently serial, incurring a long delay if N or L is large. We currently have no solution if the number of participating nodes is large, but our *bulk* protocol addresses the problem of sending large, variable-length messages efficiently. Our shuffle protocol ensures integrity and anonymity exactly as in its precursor [7], but our new *go/no-go* and *blame* phases enable all group members to trace the culprit of any protocol malfunction.

2.1 The Shuffled Send Primitive

Dissent's purpose is to provide a *shuffled send* communication primitive, providing sender anonymity among a well-defined group of nodes. We assume that the set of members comprising the group and each member's public key (or certificate) is agreed upon and known to all group members. The group may initiate a run of the shuffled send protocol in any way that preserves anonymity. For example, a designated leader, or *every* group member, might initiate runs periodically on a fixed or random schedule. Alternatively, a "client" node *not* requiring anonymity, within or outside the group, might initiate a run to request a service provided by the group collectively. For protection against traffic analysis, however, a member's desire to send anonymously must not be the initiation event.

Each Dissent protocol run is independent and permits each group member to send exactly one variable-length message to some target designated for that run. Ongoing interaction requires multiple protocol runs. A run's designated target may be a particular group member, all members (for anonymous group multicast), or another node such as a non-member "client" that initiated the run. Group members might agree upon the target of a run using a higher-level "wrapper" protocol, for example, as described in Section 5.

Each protocol run operates as shown in Figure 1. Every group member i secretly creates a message m_i and submits it to the protocol. The protocol collects all N secret messages, shuffles their order according to some random permutation π that *no one* knows, concatenates the messages in this shuffled order so that m_i appears at position π_i, and sends the concatenated sequence of messages to the target. Each input message m_i can have a different length L_i, and the protocol's output has length $\sum_i L_i$.

2.2 Applications of Shuffled Send

The shuffled send model combines and generalizes the functionality of several classes of anonymity protocols. Although every participant must submit a message in a given protocol run, members with nothing to send can submit a message of length zero, providing efficient single-sender as well as multiple-sender service.

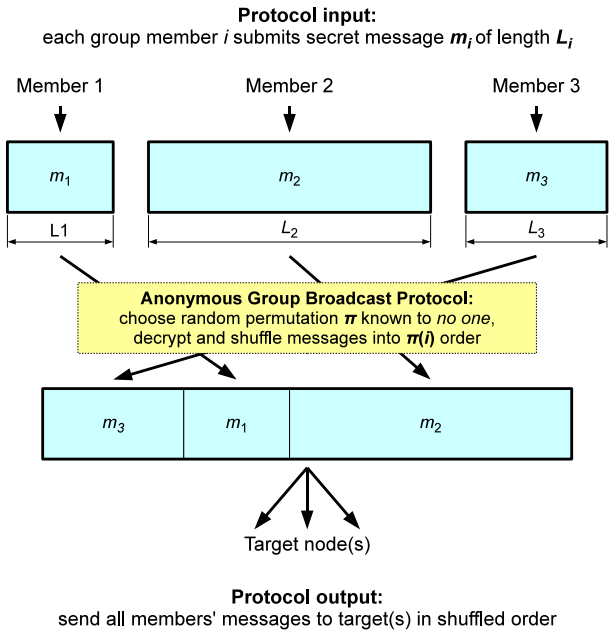

Figure 1: Shuffled send communication model

The protocol still requires each member to send a similar number of bits on the underlying network for traffic analysis protection, but none of these bits are wasted for purposes of padding messages of unbalanced lengths. Members wishing receiver anonymity can first anonymously send a public encryption key to establish a pseudonym, then look for messages encrypted with that key in subsequent shuffled sends targeted at the whole group.

Since each member submits *exactly* one message per shuffled send, one run's messages can serve as ballots in an anonymous vote. Unlike anonymous voting protocols designed for specific types of ballots and tallying methods, Dissent supports ballots of arbitrary type, format, and size. Group members can count and independently verify the ballots in any agreed-upon fashion. Ballots need not be one-shot messages either. A group can use one protocol run to establish a set of pseudonymous signing keys, one per member, then use these pseudonyms in subsequent protocol runs for pseudonymous deliberation, without permitting members to create unlimited pseudonyms for Sybil attacks [17] or sock puppetry [36].

Applications to which shuffled send may be suited include whistleblowing [42], surveys [7], file sharing [32], accountable Wiki-style editing [36], and "cocaine auctions" [33]. The current version of Dissent has some notable limitations: e.g., it may not scale to large groups, it provides only a limited form of coercion resistance (described in Section 5.3), and the latency incurred by its shuffle protocol may make it unsuitable for interactive or real-time messaging. Future work may be able to address these limitations.

2.3 Security Goals

We now define Dissent's attack model and security goals. We assume the attacker is polynomial-time limited, but can monitor all network traffic and compromise any subset of group members. A member is *honest* if she follows the protocol exactly and is not under the attacker's control, and is *faulty* otherwise. Faulty nodes are byzantine: they may collude and send arbitrary messages. For simplicity, our core protocol descriptions in Sections 3 and 4 assume that nodes never just go silent; we address liveness using principles from PeerReview [23] as outlined in Section 5.

The formal security properties we wish the protocol to satisfy are *integrity*, *anonymity*, and *accountability*, as we define below.

- **Integrity:** The protocol maintains *integrity* if, at the end of a protocol run involving N group members, every honest member either: (a) obtains exactly N messages, including each message submitted by an honest group member, or (b) knows that the protocol did not complete successfully.
- **Anonymity:** Following Brickell and Shmatikov [7], the protocol maintains *anonymity* if a group of $k \leq N-2$ colluding members cannot match an honest participant's message to its author with a probability significantly better than random guessing. (If all but one member colludes, no anonymity is possible.)
- **Accountability:** As in PeerReview [23], a member i *exposes* a member j if i holds third-party verifiable proof of j's misbehavior. The protocol maintains *accountability* if no member ever exposes an honest member, and after a run, either: (a) each honest member successfully obtains every honest member's message, or (b) all honest members expose at least one faulty member.

2.4 Simplifying Assumptions

Our core protocol descriptions in Sections 3 and 4 make several simplifying assumptions, which we will relax and address more realistically later in Section 5. We assume for now that: (a) all members know when to initiate a protocol run and how to distinguish one run from another; (b) all members of a group participate in every protocol run; (c) all members have public encryption keys and nonrepudiable signing keys known to all other members; and (d) all members remain connected throughout a protocol run and never stop sending correctly-signed messages until the protocol run has completed from the perspective of all group members. Assumption (d) implies that we address only safety properties for now, deferring liveness issues to Section 5—including the important corner case of a node withholding the last message it is supposed to send, while collecting all other members' final messages, thereby learning a protocol run's results while denying others those results.

3. SHUFFLE PROTOCOL

This section details the shuffle protocol, first covering its cryptographic building blocks, then formally describing the protocol, proving its correctness, and analyzing its complexity.

3.1 Cryptographic Primitives

Dissent relies on a conventional, possibly randomized *signature scheme*, which consists of: (a) a key generation algorithm producing a private/public key pair (u, v); (b) a signing algorithm taking private key u and message m to produce signature $\sigma = \text{SIG}_u\{m\}$; and (c) a deterministic verification algorithm taking public key v, message m, and candidate signature σ, and returning true iff σ is a correct signature of m using v's associated private key u. The notation $\{m\}\text{SIG}_u$ indicates the concatenation of message m with the signature $\text{SIG}_u\{m\}$.

We also require a *public-key cryptosystem*, which must be IND-CCA2 secure [2]. The cryptosystem must also provide access to the random bits it uses in key generation and encryption; Dissent's accountability mechanisms use this capability for commitment and verification of behavior, as described below. A software implementation of RSA-OAEP [19] using a pseudorandom number generator meets these requirements, for example. The cryptosystem specifically consists of: (a) a key generation algorithm producing a private/public key pair (x, y); (b) an encryption algorithm taking pub-

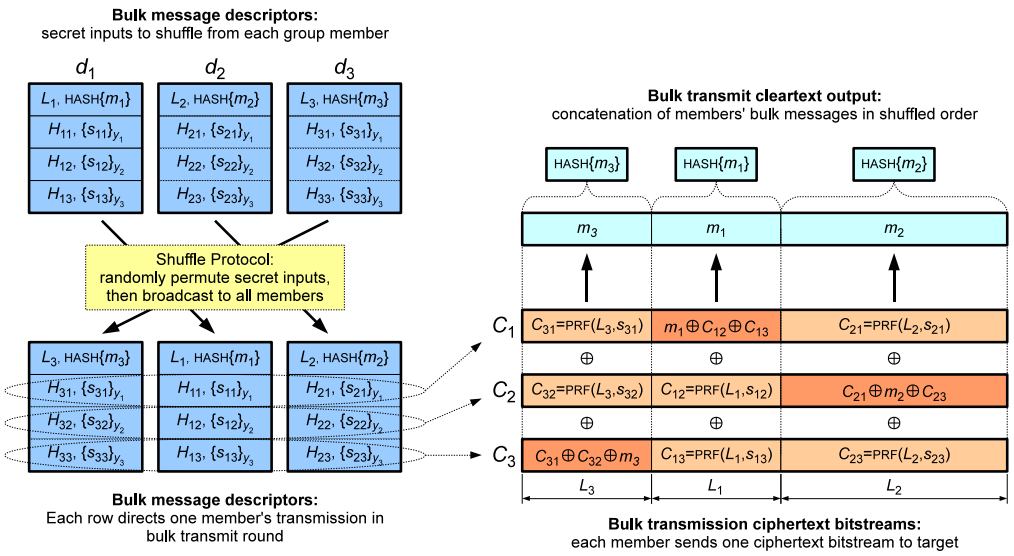

Figure 2: Illustration of bulk protocol operation for 3-member group, shuffled using permutation $\pi = [2, 3, 1]$.

lic key y, plaintext m, and some random bits R, and producing a ciphertext $C = \{m\}_y^R$; (c) a deterministic decryption algorithm taking private key x and ciphertext C, and returning the plaintext m. A node can save the random bits R it uses during encryption, and can encrypt deterministically using a given R, such that given inputs y, m, and R always yield the same ciphertext. We assume that honest nodes can check an arbitrary (x, y) purported to be a key pair, to verify that this (x, y) is indeed a key pair generated according to the specified key generation algorithm. The appendix describes how any public-key cryptosystem can be adapted, if necessary, to satisfy this assumption. The notation $C = \{m\}_{y_1:y_N}^{R_1:R_N}$ indicates iterated encryption via multiple keys: $C = \{\ldots\{m\}_{y_1}^{R_1}\ldots\}_{y_N}^{R_N}$. We omit R when an encryption's random inputs need not be saved.

We use a standard definition [35] of a collision-resistant *unkeyed hash function* and will denote the hash of message m as $\text{HASH}\{m\}$.

We use a standard definition [35] of a *pseudorandom number generator* (PRNG). We will denote the first L bits generated from a PRNG seeded with s as $\text{PRNG}\{L, s\}$.

3.2 Protocol Description

Each group member i (for $i = 1, \ldots, N$) initially has a primary encryption key pair (x_i, y_i), a signing key pair (u_i, v_i), and a secret message m_i of fixed length L to send anonymously.

Before a protocol run, all members agree on a session nonce n_R uniquely identifying this protocol run, the participants' primary public encryption and signing keys, and a common ordering of all members $1, \ldots, N$. Such agreement might be achieved via Paxos [25] or BFT [8], as discussed further in Section 5.

The shuffle protocol operates in *phases*. Each honest member i sends at most one unique message $\mu_{i\phi}$ per phase ϕ. A member i may send the same $\mu_{i\phi}$ to all members, in which case we say i *broadcasts* $\mu_{i\phi}$. An implementation of Dissent may use an underlying broadcast transmission primitive for this purpose, if available, or may simply send the same message N times, once to each group member. A faulty node might *equivocate* during a broadcast by sending different messages to different members.

Each group member maintains a *tamper-evident log* of all messages it sends and receives in a protocol run [23]. Member i signs each $\mu_{i\phi}$ it sends with its private key u_i, and includes in each message the session nonce n_R and a hash $h_{i\phi}$ of i's current log head in phase ϕ. Each $h_{i\phi}$ depends on all messages i received up to phase ϕ, before sending $\mu_{i\phi}$. Members ignore any messages they receive containing a bad signature or session nonce.

- Phase 1: Secondary Key Pair Generation. Each member i chooses an encryption key pair (w_i, z_i), and broadcasts:
$$\mu_{i1} = \{z_i, n_R, h_{i1}\}_{\text{SIG}_{u_i}}$$
- Phase 2: Data submission. Each member i encrypts her datum m_i with all members' secondary public keys:
$$C_i' = \{m_i\}_{z_N:z_1}$$
Member i stores C_i' for later use, then further encrypts C_i' with all members' primary public keys, this time internally saving the random bits used in each encryption:
$$C_i = \{C_i'\}_{y_N:y_1}^{R_{iN}:R_{i1}}$$
If encryption fails at any point, the group moves directly to phase 5b below ("blame"). Member i now sends to member 1:
$$\mu_{i2} = \{C_i, n_R, h_{i2}\}_{\text{SIG}_{u_i}}$$
- Phase 3: Anonymization. Member 1 collects all ciphertexts into a vector $\vec{C}_0 = C_1, \ldots, C_N$, randomly permutes its elements, then strips one layer of encryption from each ciphertext using private key x_1 to form $\vec{C}_1$. Member 1 sends to member 2:
$$\mu_{13} = \{\vec{C}_1, n_R, h_{13}\}_{\text{SIG}_{u_1}}$$
Each member $1 < i < N$ in turn accepts $\vec{C}_{i-1}$, permutes it randomly, strips one encryption layer to form $\vec{C}_i$, then sends $\vec{C}_i$ to member $i + 1$. Member N finally permutes and decrypts $\vec{C}_{N-1}$ to form $\vec{C}_N$, and broadcasts to all members:
$$\mu_{N3} = \{\vec{C}_N, n_R, h_{N3}\}_{\text{SIG}_{u_N}}$$
If any member i detects a duplicate or invalid ciphertext during this phase, member i reports it and the group moves directly to phase 5b below ("blame").
- Phase 4: Verification. All members now hold $\vec{C}_N$, which should be a permutation of $C_1', \ldots, C_N'$. Each member i verifies that

her own C_i' is included in the $\vec{C}_N$ she received, and sets a flag GO_i to TRUE if so and FALSE otherwise.

Each member i creates a vector $\vec{B}$ of all broadcast messages it sent or received in prior phases: all members' public key messages from phase 1, and member N's phase 3 message containing $\vec{C}_N$. Thus, $\vec{B} = \mu_{11}, \ldots, \mu_{N1}, \mu_{N3}$. Member i broadcasts:

$$\mu_{i4} = \{\text{GO}_i, \text{HASH}\{\vec{B}\}, n_R, h_{i4}\}_{\text{SIG}_{u_i}}$$

Each member i then waits to receive such a "go/no-go" message from *every* other member. If *every* member j reports $\text{GO}_j = \text{TRUE}$ for the expected $\text{HASH}\{\vec{B}\}$, then member i enters phase 5a below; otherwise i enters phase 5b ("blame").

- Phase 5a: Decryption. Each member i destroys her copy of C_i' and the random bits she saved in phase 2, then broadcasts her secondary private key w_i to all members:

$$\mu_{i5} = \{w_i, n_R, h_{i5}\}_{\text{SIG}_{u_i}}$$

Upon receiving all keys $w_1, \ldots, w_N$, member i checks that each w_j is the private key corresponding to public key z_j, and if not, exposes j using the signed messages from j containing these invalid keys. Otherwise, i removes the remaining N levels of encryption from $\vec{C}_N$, resulting in a permutation of the submitted data $m_1, \ldots, m_N$, and the protocol completes successfully.

- Phase 5b: Blame. Each member destroys her secondary private key w_i, then reveals to all members the random bits R_{ij} she saved from the primary public key encryptions in phase 2, and all signed messages she received and sent in phases 1–4. Each member i uses this information to check the behavior of each member j in phases 1–4, replaying j's primary key encryptions in phase 2, and verifying that j's anonymized output $\vec{C}_j$ in phase 3 was a decrypted permutation of $\vec{C}_{j-1}$. Member i *exposes* member j as faulty if j signed an invalid z_j in phase 1, an incorrectly encrypted C_j in phase 2, an improperly decrypted or permuted $\vec{C}_j$ in phase 3, a $\text{GO}_j = \text{FALSE}$ or a wrong $\text{HASH}\{\vec{B}\}$ in phase 4 after phases 1–3 succeeded, or if j equivocated by signing more than one message or log head $h_{j\phi}$ in any phase ϕ.

3.3 Protocol Correctness

The shuffle protocol's integrity and anonymity derive almost directly from Brickell/Shmatikov [7], so we only sketch proofs of these properties, focusing instead on the accountability property introduced by our enhancements.

3.3.1 Integrity

To preserve integrity, after a protocol run every honest member must either: (a) hold the datum m_i of every honest member i, or (b) know that the protocol did not complete successfully. Suppose that a protocol run appears to complete successfully via phase 5a (decryption), but that some honest member i does not hold the plaintext m_j of some other honest member j. Since j is honest, j's intermediate ciphertext C_j' must be a correct encryption of m_j, and C_j' must have appeared in $\vec{C}_N$. Otherwise, j would have sent $\text{GO}_j = \text{FALSE}$ in phase 4. Since honest member i would not enter phase 5a without receiving $\text{GO}_j = \text{TRUE}$ for the same $\vec{B}$ from *all* members, and $\vec{B}$ includes message μ_{N3} containing $\vec{C}_N$, i must hold C_j'. If all members released correct secondary private keys $w_1, \ldots, w_N$ during phase 5a, C_j' must then decrypt to m_j. If some member k released a secondary private key w_k such that (w_k, z_k) is an invalid key pair, all honest members expose member k.

3.3.2 Anonymity

The protocol preserves anonymity if no group of $k \leq N - 2$ colluding members can win an *anonymity game*, determining with non-negligible probability which of two honest members submitted which of two plaintexts, as detailed in prior work [7]. The attacker might gain advantage either by manipulating protocol messages, or by using only the information revealed by a correct protocol run. In the first case, the attacker can identify the intermediate ciphertext C_i' of some honest member i by duplicating or eliminating other honest members' ciphertexts in phase 3, but any honest member will detect duplication in stage 3 and elimination in stage 4, aborting the protocol before the attacker can decrypt C_i'. In the second case, an attacker who can win the anonymity game with non-negligible probability, using only information revealed by correct protocol runs, can use this ability to win the *distinguishing game* that defines an IND-CCA2 secure cryptosystem [2, 7].

3.3.3 Accountability

A member i *exposes* another member j if i obtains proof of j's misbehavior verifiable by a third party. To maintain accountability, no member may expose an honest member, and at the end of a protocol run, either: (a) the protocol completes successfully, or (b) all honest members expose at least one faulty member.

We first show that no member i can expose an honest member j. A proof of misbehavior by j consists of some "incriminating" message $\mu_{j\phi}$ signed by j in phase ϕ, together with all of the messages in j's log up through phase ϕ, and the random bits each node saved during phase 2 and released in phase 5b. Member i could "truthfully" expose j only if j signs an incorrect message in phases 1–5a, or signs more than one message per phase, contradicting the assumption that j is honest. Member i could also falsely accuse j by exhibiting one of j's messages $\mu_{j\phi}$, together with a false "prior" message $m'_{k\phi'}$ (for $\phi' < \phi$) signed by some colluding node k, different from the message $\mu_{k\phi'}$ that j actually used to compute her message $\mu_{j\phi}$. In this case, the "proof" will contain both $\mu_{k\phi'}$ (from j's log) and the false $m'_{k\phi'}$, exposing the equivocating member k instead of honest member j.

Suppose a protocol run fails, but some honest member i does not expose any faulty member. Member i observes a run to fail if it reaches phase 5a (decryption) but detects a bad secondary private key, or if it reaches phase 5b (blame). A failure in phase 5a exposes the sender of the bad secondary key. Member i enters phase 5b only if it: (a) detects a faulty encryption key in phase 1, (b) detects a duplicate or faulty ciphertext in phase 3, (c) sees a $\text{GO}_j = \text{FALSE}$ in phase 4, or (d) sees an incorrect $\text{HASH}\{\vec{B}\}$ in phase 4. Case (a) immediately exposes the relevant message's sender as faulty.

In case (b), member i can encounter a duplicate ciphertext in phase 3 only if some member $1 \leq j < i$ injected it earlier in the anonymization phase, or if two members j_1 and j_2 colluded to inject it in phase 2. (Two independently encrypted ciphertexts are cryptographically unique due to the random bits used in encryption.) If some member $1 \leq j < i$ duplicated a ciphertext, then using the message logs of members 1 through i and the random bits from phase 2, member i can replay the decryptions and permutations of each member before i in phase 3 to expose j as faulty. If no member duplicated a ciphertext in phase 3, then in replaying phase 3, i identifies the ciphertexts C_{j_1} and C_{j_2} (which decrypt to identical ciphertexts in $\vec{C}_{i-1}$), and exposes their senders j_1 and j_2. If i cannot decrypt a ciphertext in phase 3, it similarly traces the bad ciphertext to the member responsible.

In case (c), the sender j of the $\text{GO}_j = \text{FALSE}$ either truthfully reported its ciphertext missing in phase 4, or sent $\text{GO}_j = \text{FALSE}$

although its ciphertext C'_j appeared in $\vec{C}_N$. In the former case, i replays phase 3 to expose the member who replaced j's ciphertext. In the latter case, the occurrence of C'_j in $\vec{C}_N$ exposes j itself.

In case (d), member i's $\vec{B}$ does not match the $\text{HASH}\{\vec{B}'\}$ in another member j's go/no-go ($\vec{B} \neq \vec{B}'$). Members i and j compare their respective message logs. If i's log of prior broadcast messages does not match its computed $\vec{B}$, this fact exposes i, and similarly for j with its $\vec{B}'$. Otherwise, for some member k and phase ϕ, there must be corresponding signed messages that differ between B and B', i.e., some $\mu_{k\phi} \in B$ and $\mu'_{k\phi} \in B'$ such that $\mu_{k\phi} \neq \mu'_{k\phi}$. These messages expose k as having equivocated during a broadcast.

3.4 Asymptotic Complexity

Since iterated public-key encryptions as performed in phase 2 typically involve plaintext expansion, let $\tilde{L} = L + O(N)$ be the size of an L-bit input message after these $2N$ encryptions.

If the underlying network provides efficient broadcast, then each node transmits $O(N\tilde{L})$ bits during a run, for a total messaging cost of $O(N^2\tilde{L})$. Without efficient broadcast, the "normal-case" phases 1 through 5a still require each node to transmit only $O(N\tilde{L})$ bits, for $O(N^2\tilde{L})$ overall cost, because all broadcasts in these phases are either single messages of length $O(N\tilde{L})$ or N messages of length $O(\tilde{L})$. The blame phase in an unsuccessful run may require $O(N^3\tilde{L})$ total communication for all honest members to expose some faulty member, but an attacker can trigger at most $O(N)$ such runs before the group exposes and removes all faulty members.

Latency is dominated by the N serial communication rounds in phase 3, in which each node must send $O(N\tilde{L})$ bits, for a total latency of $O(N^2\tilde{L})$ transmission bit-times. Other phases require a constant number of unicast messages or parallelizable broadcasts.

Excluding the blame phase, each member's computational cost is dominated by the $2N$ public-key encryptions and decryptions it performs. Each of these operations is on a plaintext of length $O(\tilde{L})$, for a processing cost of $O(N\tilde{L})$ per node or $O(N^2\tilde{L})$ total. The blame phase introduces an additional $O(N)$ factor if all members must replay all other members' encryptions.

4. BULK PROTOCOL

We now describe Dissent's bulk protocol in detail, then analyze its correctness, security properties, and complexity.

4.1 Protocol Description

Members $1, \ldots, N$ initially hold messages $m_1, \ldots, m_N$, now of varying lengths $L_1, \ldots, L_N$. We reuse the cryptographic primitives described in Section 3.1. As before, each member i has a signing key pair (u_i, v_i) and a primary encryption key pair (x_i, y_i). All members know each others' public keys, and have agreed upon session nonce n_R and an ordering of members.

- Phase 1: Message Descriptor Generation. Each member i chooses a random seed s_{ij} for each member j, then for each $j \neq i$, generates L_i pseudorandom bits from s_{ij} to obtain ciphertext C_{ij}:

  $$C_{ij} = \text{PRNG}\{L_i, s_{ij}\} \quad (j \neq i)$$

  Member i now XORs her message m_i with each C_{ij} for $j \neq i$ to obtain ciphertext C_{ii}:

  $$C_{ii} = C_{i1} \oplus \ldots \oplus C_{i(i-1)} \oplus m_i \oplus C_{i(i+1)} \oplus \ldots \oplus C_{iN}$$

  Member i computes hashes $H_{ij} = \text{HASH}\{C_{ij}\}$, encrypts each seed s_{ij} with j's public key to form $S_{ij} = \{s_{ij}\}^{R_{ij}}_{y_j}$, and collects the H_{ij} and S_{ij} for each j into vectors $\vec{H}_i$ and $\vec{S}_i$:

  $$\vec{H}_i = H_{i1}, \ldots, H_{iN}$$
  $$\vec{S}_i = S_{i1}, \ldots, S_{iN}$$

  Finally, member i forms a *message descriptor*, d_i:

  $$d_i = \{L_i, \text{HASH}\{m_i\}, \vec{H}_i, \vec{S}_i\}$$

- Phase 2: Message Descriptor Shuffle. The group runs the shuffle protocol in Section 3, each member i submitting its fixed-length descriptor d_i as the secret message to be shuffled. The shuffle protocol broadcasts all descriptors in some random permutation π to all members, so d_i appears at position $\pi(i)$ in the shuffle.

- Phase 3: Data transmission. Each member j now recognizes its own descriptor d_j in the shuffle, and sets $C'_{jj} = C_{jj}$. From all *other* descriptors d_i ($i \neq j$), j decrypts S_{ij} with private key x_j to reveal seed s_{ij}, computes ciphertext $C_{ij} = \text{PRNG}\{L_i, s_{ij}\}$, and checks $\text{HASH}\{C_{ij}\}$ against H_{ij}. If decryption succeeds and the hashes match, member j sets $C'_{ij} = C_{ij}$. If decryption of S_{ij} fails or $\text{HASH}\{C_{ij}\} \neq H_{ij}$, then j sets C'_{ij} to an empty ciphertext, $C'_{ij} = \{\}$.

  Member j now signs and sends each C'_{ij} to the designated target for the protocol run, in π-shuffled order:

  $$\{C'_{\pi^{-1}(1)j}, \ldots, C'_{\pi^{-1}(N)j}, n_R, h_{j3}\}_{\text{SIG}_{u_j}}.$$

- Phase 4: Message Recovery. The designated target (or each member if the target is the group) checks each C'_{ij} it receives from member j against the corresponding H_{ij} from descriptor d_i. If C'_{ij} is empty or $\text{HASH}\{C'_{ij}\} \neq H_{ij}$, then message slot $\pi(i)$ was corrupted and the target ignores it. For each uncorrupted slot $\pi(i)$, the target recovers i's message by computing:

  $$m_i = C'_{i1} \oplus \ldots \oplus C'_{iN}$$

- Phase 5: Blame. If any messages were corrupted in phase 4, all members run the shuffle protocol again, during which each member i whose message was corrupted anonymously broadcasts an *accusation* naming the culprit member j:

  $$A_i = \{j, S_{ij}, s_{ij}, R_{ij}\}$$

  Each accusation contains the seed s_{ij} that i assigned j and the random bits i used to encrypt the seed. Each member k verifies the revealed seed by replaying its encryption $S_{ij} = \{s_{ij}\}^{R_{ij}}_{y_j}$, and checks that $H_{ij} = \text{HASH}\{\text{PRNG}\{L_i, s_{ij}\}\}$. If the accusation is valid, then k exposes j as faulty. If the shuffle reveals no valid accusation for a corrupted message slot $\pi(i)$, then k does nothing: either the anonymous sender i has corrupted its own message or has chosen not to accuse the member who did, which is equivalent to i sending a valid but useless message.

4.2 Protocol Correctness

We now sketch proofs of the bulk protocol's correctness.

4.2.1 Integrity

The shuffle protocol ensures that the message descriptor d_i of each honest member i is correctly included in the shuffled output. The target can use either the individual ciphertext hashes H_{ij} or the cleartext hash $\text{HASH}\{m_i\}$ from d_i to verify the integrity of i's message in the bulk output. The cleartext hash $\text{HASH}\{m_i\}$ is technically redundant, but enables all members to verify the output if only one node collects and combines the ciphertexts for efficiency.

4.2.2 Anonymity

Suppose an attacker controls all but two honest members i and j, and wishes to win the anonymity game [7] by determining with

non-negligible advantage over random guessing which honest member sent one of their plaintexts, for example, m_i. The attacker knows which two message slots $\pi(i)$ and $\pi(j)$ belong to the honest members, and must find the exact permutation π. Since the shuffle protocol preserves anonymity (Section 3.3.2) and the shuffled message descriptors depend only on random bits and the messages themselves, the attacker learns nothing about π from the message descriptors. The only other information the attacker obtains about m_i are the ciphertexts C'_{ik} produced by all members k. But since each bit of C'_{ii} and C'_{ij} is encrypted with a pseudorandom one-time pad generated from a seed s_{ij} that only i and j know, the attacker learns nothing from these ciphertext bits.

4.2.3 Accountability

We first show that no dishonest member i can expose an honest member j. Since the shuffle protocol maintains accountability, we need only show that the bulk protocol never exposes an honest member in its blame phase. To expose j, i must anonymously submit a valid accusation naming j as faulty. This accusation must include a seed s'_{ij} such that $\text{PRNG}\{L_i, s'_{ij}\} \neq \text{PRNG}\{L_i, s_{ij}\}$ and $H_{ij} = \text{HASH}\{\text{PRNG}\{L_i, s'_{ij}\}\}$, thus violating our assumption that the hash function is collision resistant.

Now suppose the bulk protocol violates accountability, such that at the end of a protocol run, some honest member j does not hold the plaintext of another honest member i and does not expose any dishonest member. Since the shuffle protocol maintains accountability, member j must have received i's message descriptor d_i, or have exposed some group member k. Since i is honest, d_i contains correctly computed hashes H_{ik} and correctly encrypted seeds S_{ik} for ciphertexts C'_{ik} that, XORed together, would reveal i's message m_i to j. Some member k must therefore have sent an incorrect ciphertext in the bulk phase. But since i is honest, i would have sent a correct accusation of k in the blame phase, exposing k as faulty.

4.3 Asymptotic Complexity

With efficient broadcast, in the normal case each member transmits $O(N^2)$ bits to shuffle N message descriptors of length $O(N)$, then sends $L_{tot} + O(1)$ bits of bulk ciphertext, where $L_{tot} = \sum_i L_i$. Normal-case communication complexity is thus $O(N^2) + L_{tot}$ bits per node. An unsuccessful run may transmit $O(N^3) + L_{tot}$ bits per node due to the shuffle protocol's blame phase.

If N is small so that L_{tot} dominates, only one member wishes to transmit ($L_i = L_{tot}$ and $L_j = 0$ for $j \neq i$), and the transmitted data is incompressible, then Dissent's communication efficiency is asymptotically optimal for our attack model: trivial traffic analysis reveals that any member sending fewer than L_{tot} bits cannot be the sender. An interesting question for future work is whether better communication efficiency is feasible, while preserving strong traffic analysis resistance, when several members transmit at once.

The shuffle protocol incurs an $O(N^3)$ startup latency, as the N nodes serially shuffle N descriptors of length $O(N)$, but the data transmission phase is fully parallelizable, for a total latency of $O(N^3 + L_{tot})$ transmission bit-times overall.

Each member i performs N cryptographic operations on $O(N)$ bits each during the shuffle, N operations on L_i bits to compute C_{ii}, and one operation on L_j bits to compute C_{ij} for each $j \neq i$. Computational complexity is thus $O(N^2 + NL_{tot})$ per node.

5. USAGE CONSIDERATIONS

In describing Dissent's shuffle and bulk protocols, we made a number of simplifying assumptions, which we now address by placing these core protocols in the context of a more realistic, high-level "wrapper" protocol. We merely sketch this wrapper protocol without formal definition or analysis, since it is intended only to illustrate one way to deploy Dissent in a realistic environment, and not to define the "right" way to do so. The wrapper protocol addresses five practical issues: protocol initiation, member selection, deniable keying, liveness assurance, and end-to-end reliability.

5.1 Protocol Initiation

Our shuffle and bulk protocols assume that all group members "just know" when to commence a protocol run, but in practice some node must initiate each run. Members must *not* initiate a protocol run out of a desire to send anonymously, however, since doing so would make the sender's identity obvious to traffic analysis.

In our wrapper protocol, therefore, each protocol run is unilaterally initiated by some node, whom we call the *leader*. To enable members to send "spontaneously" without compromising their anonymity, *every* group member periodically initiates a protocol run independently of its own desire to send, on either a fixed or randomized time schedule. Anonymity would be equally well served if the leader was the same for all protocol runs, but requiring every member to act as leader occasionally makes it easier to address the liveness issues discussed below. If group policy permits, a non-anonymous outsider may also lead a protocol run, effectively invoking the collective services of the group as in anonymous data-mining applications [7].

5.2 Selecting Available Participants

The core protocols above assume that every group member participates in a given protocol run, but in practice at least a few members of a long-lived group are likely to be unavailable at any given time, making it pragmatically important for the group to be able to make progress in the absence of some members. The wrapper protocol therefore distinguishes a group's *long-term membership* M from the set of members M_R participating in a particular run R, where $M_R \subseteq M$. In the wrapper protocol, the leader of run R is responsible for detecting which members are presently available, and for bringing those members available to a consensus on the precise set of participants M_R for protocol run R.

A key issue in choosing M_R is preventing a malicious leader from packing M_R with colluding members to the exclusion of most honest members, limiting the anonymity of the few honest members remaining. Group policy should therefore define some minimum *quorum* Q, and honest nodes must refuse to participate in a proposed run where $|M_R| < Q$. If there are at most $f \leq Q - 2$ faulty nodes, honest nodes will be guaranteed at least $(Q - f)$-anonymity within a run, regardless of how M_R is chosen.

If members establish and reuse long-lived pseudonyms across multiple runs, however, then a quorum requirement may be insufficient to protect these pseudonyms from intersection attacks [3] by a malicious leader who selectively exludes different nodes in each run. As a further defense, honest members might protect each other against malicious exclusion as follows. If honest member i receives a proposal from would-be leader l_R to initiate run R while excluding some other member j, but i believes j to be reachable, then i demands that l_R add j to M_R—forwarding messages between l_R and j if necessary—as a precondition to i participating in round R.

5.3 Coercion Resistance via Deniable Keying

Dissent's shuffle protocol assumes each group member i has a signing key pair (u_i, v_i) with which it signs all messages, creating the nonrepudiable "accountability trail" that the blame phase (5b) requires to trace a misbehaving member. Unfortunately, this nonrepudiable record also enables members to prove to a third party which message they sent (or didn't send) in a given protocol run.

In anonymous communication scenarios, we often desire not just anonymity but also repudiability [6]: after a protocol run, no one should be able to prove to a third party which message any member sent, or ideally, whether a member participated at all. In anonymous voting applications, we often desire the closely related property of resistance to coercion or "vote-buying."

Our wrapper protocol can provide some repudiability or coercion resistance as follows. We assume each group member i's well-known identity is defined *only* by its primary encryption key pair (x_i, y_i), and members now choose a fresh, temporary signing key pair (u_i, v_i) for each protocol run. To initiate a run, the would-be leader l uses a deniable authenticated key exchange algorithm such as SKEME [28] to form a secure channel with each potential participant i, using l's and i's primary encryption keys for this authentication. Each member i uses this pairwise-authenticated channel to send the leader i's fresh public signing key v_i for the run.

Once l forms a tentative list of $N = |M_R|$ participants, l broadcasts to all participants a *round descriptor* D_R containing a round nonce, all participants' primary public keys $y_1, \ldots, y_N$, and all participants' temporary signing keys $v_1, \ldots, v_N$ for the run. Each member i now forms a *challenge* c_{ij} for each node j, containing a random seed S_{ij} and a hash of D_R keyed on S_{ij}. Member i encrypts c_{ij} with j's public key y_j to yield C_{ij}. Member i sends its encrypted challenges to the leader, who forwards each C_{ij} to member j. Member j decrypts C_{ij}, verifies the keyed hash it contains against the D_R that j received from the leader, and returns c_{ij} to the leader, who forwards it to i. On a decryption failure or challenge mismatch, the leader must decide whether to exclude i or j from a retry attempt; i can prove its innocence by revealing the random bits it used to encrypt its original challenge to j.

Once all members confirm D_R with all other members, the shuffle proceeds using the temporary signing keys in D_R. These signing keys are nonrepudiable only *within the protocol run*, so the leader can trace misbehaving members and exclude them from subsequent runs. No node is left with proof that any member i actually used signing key u_i during a given run, however, since anyone can unilaterally forge all the authenticated key exchanges, challenges, and subsequent messages in the shuffle and bulk protocols.

Of course, this form of repudiability is useful only against an attacker who actually requires third-party verifiable "proof of responsibility" in order to coerce group members. If the attacker can see all network traffic, as our attack model assumes, *and* the attacker's traffic logs alone constitute "proof" of which network packets a given member sent, then we know of no way to achieve deniability or coercion resistance. Similarly, a member might be coerced *before* a protocol run into sending some sufficiently unique, attacker-supplied message or ballot. If the mere appearance of that message/ballot in the run's output satisfies the attacker that the member "stayed bought," then no anonymity mechanism based purely on a random shuffle will address this form of coercion.

5.4 Ensuring Liveness

As we have seen, tracing active disruptors of the shuffle or bulk protocols presents particular technical challenges due to the need to protect the anonymity of honest senders. A member might *passively* disrupt either protocol, however, by simply going offline at any time, either intentionally or due to node or network failure. Fortunately, given the core protocols' resistance to both active disruption and traffic analysis, we can ensure liveness and handle passive disruption via more generic techniques.

Each phase of the shuffle and bulk protocols demand that particular members send properly signed messages to other members. Again borrowing terminology and ideas from PeerReview [23], when the protocol demands that member i send member j a message, and member j has not received such a (properly signed) message for some time, we say that j *suspects* i. Once j suspects i, j informs another node k (the leader, for example) of j's suspicion; k in turn contacts i demanding a (signed) copy of i's message to j. If i fails to offer this message to k, then after some time k suspects j as well and notifies other members in turn, eventually causing all honest, connected members to suspect i. Member i can dispel any honest member's suspicion at any time by offering a copy of the demanded message. If i honestly cannot send to j due to asymmetric connectivity, for example, then i responds to k's demand with the required message, which k forwards back to j, dispelling both j's and k's suspicion and enabling the protocol to proceed.

Since our wrapper protocol makes the leader responsible for initiating protocol runs, we also make it the leader's responsibility to decide when a protocol run has failed due to a suspected node going offline—or deliberately withholding a required message—for too long. At this point, the leader starts a new protocol run, excluding any exposed or persistently suspected nodes from the previous run, and the remaining members attempt to resend their messages. If the leader fails, members can retry their sends in a future run initiated by a different leader.

5.5 End-to-End Reliability

A corner-case liveness challenge for most protocols is *closure*: determining when participants may consider the protocol "successfully concluded." In a byzantine model, a malicious member might intentionally withhold the last message it was supposed to send—e.g., its own secondary private key in phase 5a of the shuffle protocol, or its own ciphertext in the bulk protocol—while collecting the last messages of other members, thereby learning the results of the protocol run while denying those results to other members.

We approach this class of problems in general by treating our shuffle and bulk protocols as a "best-effort" anonymous delivery substrate, atop which some higher-level protocol must provide end-to-end reliable delivery and graceful closure if desired. If a faulty member denies other members a protocol run's results, the honest members will soon suspect the faulty member. The same or a different leader will eventually start a new protocol run without the faulty member, in which the members may retransmit their messages. If a member i wishes to ensure that a message it sends anonymously is reliably seen by a particular member j, for example, then i must resend the message in successive protocol runs until j acknowledges the message. Member j might sign acknowledgments via public or pseudonymous keys, or group or ring signatures [4, 11, 30].

If the messages sent in a protocol run are interrelated, such as the ballots comprising an anonymous vote, the group may wish to ensure that some quorum of members sees the result. The group can follow such a voting run with an acknowledgment run, discarding and repeating unsuccessful voting runs (with successively smaller membership sets as members are exposed or go offline) until the required number of members acknowledges the results. If the group wishes to provide reliable broadcast semantics or maintain some consistent group state across successive protocol runs, the group can implement byzantine consensus [8] atop the shuffled send primitive, ensuring both liveness and strong consistency as long as over two thirds of the group members remain live.

6. PROTOTYPE IMPLEMENTATION

To evaluate Dissent's practicality, we built and tested a simple proof-of-concept prototype implementing the protocol. The prototype is written in Python, using OpenSSL's implementations of 1024-bit RSA-OAEP with AES-256 for public-key encryption and

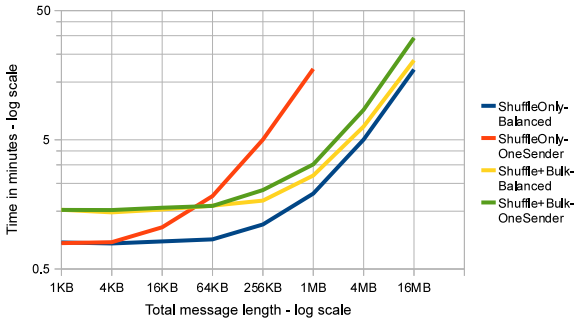

Figure 3: Time required for anonymous broadcast of balanced and unbalanced message loads among 16 nodes, via shuffle alone or full Dissent protocol.

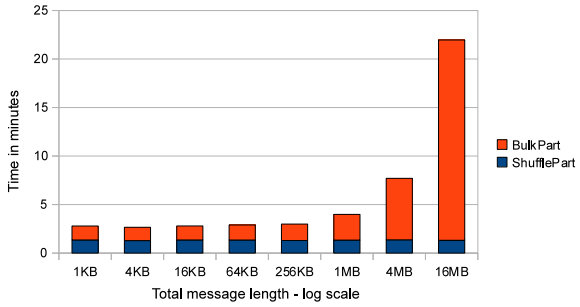

Figure 4: Time required to send varying message sizes, broken into shuffle and bulk transfer protocol portions.

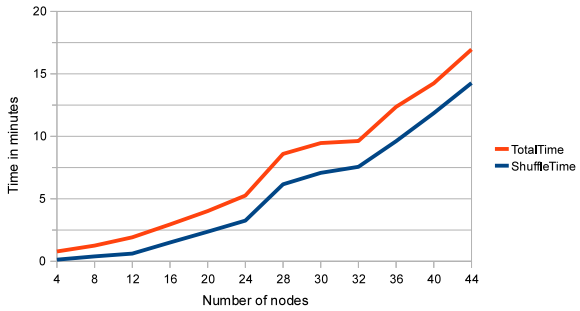

Figure 5: Time required to send 1MB of data (balanced) using shuffle and bulk protocols together, with varying group size.

signing, AES-256 in counter mode as the bulk protocol's pseudo-random number generator, and SHA-1 as the hash algorithm.

We used the Emulab [18] network testbed to test the prototype under controlled network conditions. We ran the prototype on recent x86 PCs machines running Ubuntu 7.04 and Python 2.5, on a simulated star topology in which every node is connected to a central switch via a 5Mbps connection with a latency of 50ms (100ms node-to-node latency). We make no claim that this topology is "representative" of likely deployment scenarios for Dissent, since we know of no available data on the network properties "typical" of online groups that might wish to run Dissent. Our simulated topology is merely intended to reflect *plausible* communication bandwidths and delays for wide-area Internet communication.

We rely on the analysis in previous sections to evaluate Dissent's security properties, and assume that the accountability measures in a full implementation of Dissent will deter or eventually exclude misbehaving members. For experimentation purposes, therefore, we implement and test only the "normal-case" aspects of the protocol in the current prototype. The prototype does not use a secure public key infrastructure, and does not implement the "blame" phases or the full wrapper protocol. Nodes sign and verify all messages, however, ensuring that performance measurements accurately reflect Dissent's normal-case costs.

The prototype uses TCP for communication, maintaining TCP connections throughout a given protocol run to minimize startup overhead, but closing all connections at the end of each run. Where Dissent requires broadcast, nodes implement these broadcasts atop TCP by sending their messages to a leader, who bundles all broadcasts for that phase and sends each node a copy of the bundle.

6.1 Performance Evaluation

Figure 3 shows the total time the prototype requires to broadcast messages of varying sizes anonymously among 16 nodes, using either the shuffle protocol alone or the full Dissent protocol. In each case, we test two message loads: a *Balanced* load in which each node sends $1/16$th of the total message data, and a *OneSender* load in which one node sends all the data and other nodes send nothing.

For a single node to send a 16MB message, Dissent ran in about 31 minutes on the experimental topology, or $3.6\times$ longer than one node required to broadcast the same data to the other 15 nodes with no encryption or anonymization. While significant, a 3–$4\times$ slowdown may be a reasonable price to pay for strong anonymity.

As expected, the full protocol incurs a higher startup delay than the shuffle protocol alone, but handles unbalanced loads more gracefully, maintaining similar performance for a given total message length regardless of balance. We are not aware of any other verifiable shuffles [20, 26] for which working implementations and performance data are available, but given their typical assumption of small, equal-length messages, we expect their performance on unbalanced loads to be at best on par with our shuffle protocol alone.

Figure 4 breaks the runtime of the full Dissent protocol into its shuffle and bulk protocol components, illustrating that the shuffle's cost remains constant with message size and becomes negligible as total message length grows.

The full Dissent protocol still showed some slowdown under highly unbalanced load: although balance does not affect Dissent's communication cost, it *does* affect computation costs. When only one node is sending, that node must compute and XOR together $N-1$ pseudorandom streams of message length L, while other nodes each compute only one L-byte stream. This timing difference could lead to a side-channel attack if not handled carefully in implementation, e.g., by pre-computing all required bit strings before commencing a send. We have made no attempt to analyze the protocol in detail for side-channel attacks, however.

Figure 5 measures the prototype's runtime with varying group sizes. In a successful run, each node sends $O(N^2)$ bits in the shuffle and $L_{tot} + O(1)$ bits in the bulk protocol. As expected, the shuffle's runtime increases much more quickly with N than the bulk protocol, although the superlinear N^2 curve manifests only slightly for the small groups we tested.

7. RELATED WORK

Dissent's shuffle protocol builds directly on an anonymous data collection protocol by Brickell and Shmatikov [7], adding DoS resistance via our new go/no-go and blame phases. Dissent's bulk protocol is similarly inspired by DC-nets [10], which are compu-

tationally efficient and provide unconditional anonymity. DC-nets traditionally require nondeterministic "reservation" schemes to allocate the anonymous channel's communication bandwidth, however, and are difficult to protect against anonymous DoS attacks by malicious group members. Strategies exist to strengthen DC-nets against DoS attacks [22, 40], or to form new groups when an attack is detected [32]. Dissent's use of a shuffle protocol to set up a *deterministic* DC-nets instance, however, cleanly avoids these DoS vulnerabilities while providing the additional guarantee that each member sends *exactly* one message per protocol run, a useful property for holding votes or assigning 1-to-1 pseudonyms.

Mix-networks [9] offer scalable, practical anonymous unicast communication, and can be adapted to group broadcast [27]. Unfortunately, mix-networks are difficult to protect against traffic analysis [31] and DoS attacks [16, 24], and in fact, lose security under DoS attack [5]. Crowds [29] are more computationally efficient than mix-networks, but are vulnerable to statistical traffic analysis when an attacker can monitor many points across the network. k-anonymous transmission protocols [39] provide anonymity only when most members of a group are honest. Dissent, in contrast, is provably secure against traffic analysis, preserving anonymity even when up to $N - 2$ members maliciously collude.

Anonymous voting protocols solve a problem closely related to group broadcast. Each user casts a ballot whose contents should be publicly known but whose author should be unknown to both the election officials and other voters. Many voting protocols allow transmission of only fixed-length messages, e.g., "Yes" or "No" [1].

Cryptographically verifiable shuffles [20, 26] might replace our shuffle protocol, making the shuffle verifiable offline. These algorithms require more exotic and complex cryptography, however, and generally verify only a shuffle's *correctness* (i.e., that it is a permutation), and not its *randomness* (i.e., that it ensures anonymity). All existing techniques of which we are aware to assure a shuffle's randomness and anonymity, in the presence of compromised members, require passing a batch of messages through a series of independent shuffles, as in Dissent or mix-networks [15].

Group signatures [4, 11] and ring signatures [30] provide anonymous *authentication* rather than anonymous transmission. Combining group/ring signatures with classic DC-nets transmission can meet the first two of Dissent's three key security goals, integrity and anonymity (see Section 2.3). Such a combination fails to provide accountability, however: malicious group members can still anonymously disrupt the DC-nets transmission channel, preventing communication from occurring at all. Layering group/ring signatures atop DC-nets also does not provide the 1-to-1 mapping needed for anonymous voting or assigning Sybil attack-resistant pseudonyms.

Tor [14] and Herbivore [32] are two well-known practical systems for providing anonymous communication over the Internet. These systems scale to far larger groups than Dissent does, and also permit interactive communication. These systems do not provide Dissent's strong guarantees of anonymity or accountability, however. As a system based on mix-networks, Tor is vulnerable to traffic analysis attacks. Herbivore provides unconditional anonymity, but only within a small subgroup of the total group of participants. Dissent may be more suitable for non-interactive communication between participants willing to sacrifice protocol execution speed for strong assurances of anonymity and accountability.

8. CONCLUSION

Dissent is a novel protocol for anonymous and accountable group communication. Dissent allows a well-defined group of participants to exchange variable-length messages anonymously without the risks of traffic analysis or anonymous DoS attacks associated with mix-networks and DC-nets. Dissent improves upon previous shuffled-send primitives by adding accountability—the ability to trace misbehaving nodes—and by eliminating the message padding requirements of earlier schemes. We have reviewed practical concerns associated with a real-world deployment of Dissent, and have proposed potential solutions for each. Our implementation demonstrates Dissent to be practical, at least for non-interactive anonymous communication within moderate-size groups.

Acknowledgments

We would like to thank Vitaly Shmatikov, Michael Fischer, Bimal Viswanath, Animesh Nandi, Justin Brickell, Jacob Strauss, Chris Lesniewski-Laas, Pedro Fonseca, Philip Levis, and the anonymous CCS reviewers for valuable feedback and discussion. This work was supported in part by the National Science Foundation under grant CNS-0916413.

9. REFERENCES

[1] Ben Adida. *Advances in cryptographic voting systems*. PhD thesis, Cambridge, MA, USA, 2006.

[2] Mihir Bellare, Anand Desai, David Pointcheval, and Phillip Rogaway. Relations among notions of security for public-key encryption schemes. *Advances in Cryptology —CRYPTO '98*, pages 549–570, 1998.

[3] Oliver Berthold, Andreas Pfitzmann, and Ronny Standtke. The disadvantages of free MIX routes and how to overcome them. In *Workshop on Design Issues in Anonymity and Unobservability*, July 2000.

[4] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In *CRYPTO*, August 2004.

[5] Nikita Borisov, George Danezis, Prateek Mittal, and Parisa Tabriz. Denial of service or denial of security? How attacks on reliability can compromise anonymity. In *14th ACM CCS*, October 2007.

[6] Nikita Borisov, Ian Goldberg, and Eric Brewer. Off-the-record communication, or, why not to use PGP. In *WPES*, pages 77–84, October 2004.

[7] Justin Brickell and Vitaly Shmatikov. Efficient anonymity-preserving data collection. In Tina Eliassi-Rad, Lyle H. Ungar, Mark Craven, and Dimitrios Gunopulos, editors, *KDD*, pages 76–85. ACM, 2006.

[8] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *3rd OSDI*, pages 173–186, February 1999.

[9] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2), February 1981.

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

[11] David Chaum and Eugène Van Heyst. Group signatures. In *Eurocrypt*, April 1991.

[12] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Workshop on Design Issues in Anonymity and Unobservability*, July 2000.

[13] David Davenport. Anonymity on the Internet: why the price may be too high. *Communications of the ACM*, 45(4):33–35, April 2002.

[14] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: the second-generation onion router. In *SSYM'04: Proceedings of the 13th conference on USENIX Security*

Symposium, pages 21–21, Berkeley, CA, USA, 2004. USENIX Association.

[15] Roger Dingledine, Vitaly Shmatikov, and Paul Syverson. Synchronous batching: From cascades to free routes. In *WPET*, May 2004.

[16] Roger Dingledine and Paul Syverson. Reliable MIX cascade networks through reputation. In *Financial Cryptography*, March 2002.

[17] John R. Douceur. The Sybil attack. In *1st International Workshop on Peer-to-Peer Systems*, March 2002.

[18] Emulab network emulation testbed. `http://emulab.net/`.

[19] Eiichiro Fujisaki, Tatsuaki Okamoto, David Pointcheval, and Jacques Stern. RSA-OAEP is secure under the RSA assumption. *Journal of Cryptology*, 17(2):81–104, 03 2004.

[20] Jun Furukawa and Kazue Sako. An efficient scheme for proving a shuffle. In *CRYPTO*, August 2001.

[21] David Goldschlag, Michael Reed, and Paul Syverson. Onion routing for anonymous and private internet connections. *Communications of the ACM*, 42(2):39–41, February 1999.

[22] Philippe Golle and Ari Juels. Dining cryptographers revisited. *Eurocrypt*, May 2004.

[23] Andreas Haeberlen, Petr Kouznetsov, and Peter Druschel. PeerReview: Practical accountability for distributed systems. In *21st SOSP*, October 2007.

[24] Jan Iwanik, Marek Klonowski, and Miroslaw Kutylowski. DUO-Onions and Hydra-Onions — failure and adversary resistant onion protocols. In *IFIP CMS*, September 2004.

[25] Leslie Lamport. The part-time parliament. *TOCS*, 16(2):133–169, 1998.

[26] C. Andrew Neff. A verifiable secret shuffle and its application to e-voting. In *8th CCS*, pages 116–125, November 2001.

[27] G. Perng, M.K. Reiter, and Chenxi Wang. M2: Multicasting mixes for efficient and anonymous communication. In *26th ICDCS*, pages 59–59, 2006.

[28] Mario Di Raimondo, Rosario Gennaro, and Hugo Krawczyk. Secure off-the-record messaging. In *WPES*, November 2005.

[29] Michael K. Reiter and Aviel D. Rubin. Anonymous web transactions with crowds. *Communications of the ACM*, 42(2):32–48, 1999.

[30] Ronald Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. In *ASIACRYPT*, December 2001.

[31] Andrei Serjantov, Roger Dingledine, and Paul Syverson. From a trickle to a flood: Active attacks on several mix types. *Information Hiding*, pages 36–52, 2003.

[32] Emin Gün Sirer et al. Eluding carnivores: File sharing with strong anonymity. In *11th SIGOPS European Workshop*, September 2004.

[33] Frank Stajano and Ross Anderson. The cocaine auction protocol: On the power of anonymous broadcast. In *3rd Information Hiding Workshop*, September 1999.

[34] Edward Stein. Queers anonymous: Lesbians, gay men, free speech, and cyberspace. *Harvard Civil Rights-Civil Liberties Law Review*, 38(1), 2003.

[35] Douglas R. Stinson. *Cryptography: Theory and Practice, Third Edition (Discrete Mathematics and Its Applications)*. Chapman & Hall/CRC, November 2005.

[36] Brad Stone and Matt Richtel. The hand that controls the sock puppet could get slapped. *New York Times*, July 2007.

[37] Al Teich, Mark S. Frankel, Rob Kling, and Ya-ching Lee. Anonymous communication policies for the Internet: Results and recommendations of the AAAS conference. *Information Society*, May 1999.

[38] Eugene Vasserman, Rob Jansen, James Tyra, Nicholas Hopper, and Yongdae Kim. Membership-concealing overlay networks. In *16th ACM CCS*, November 2009.

[39] Luis von Ahn, Andrew Bortz, and Nicholas J. Hopper. k-anonymous message transmission. In *10th CCS*, pages 122–130, New York, NY, USA, 2003. ACM.

[40] Michael Waidner and Birgit Pfitzmann. The dining cryptographers in the disco: Unconditional sender and recipient untraceability with computationally secure serviceability. In *Eurocrypt*, page 690, April 1989.

[41] Jonathan D. Wallace. Nameless in cyberspace: Anonymity on the internet, December 1999. Cato Briefing Paper No. 54.

[42] Wikileaks. `http://wikileaks.org/`.

[43] The constitutional right to anonymity: Free speech, disclosure and the devil. *Yale Law Journal*, 70(7):1084–1128, June 1961.

APPENDIX: KEY PAIR VERIFICATION

In the decryption phase of Dissent's shuffle protocol, honest group members receive secondary private keys from other, potentially malicious group members, and must verify both that this private key w_i is valid for the cryptosystem in use, and that it corresponds to the public key z_i distributed during phase 1. Such a check is not a standard function of public-key cryptosystems, but any public-key cryptosystem can be augmented to support such a check. The key point is that disclosure of the private key in phase 5a eliminates all secrecy requirements associated with that key pair, so the member who generated a key pair can "prove" the key's validity simply by including enough information with the private key for the receiving member to replay the key generation process exactly.

Given a public-key cryptosystem [19] with a *probabilistic* key generation algorithm $\mathcal{K}(\rho)$ taking security parameter ρ as input and producing key pair (x, y) as output, we define a *deterministic* construction $\mathcal{K}(\rho, r)$ of the same algorithm, where r contains the random bits supplied to the key generation algorithm. We define the *augmented key pair* of the original private/public key pair (x, y) to be the pair $((x, r), (y, \rho))$. Using this augmented algorithm, members participating in the shuffle protocol broadcast (y, ρ) during phase 1, and reveal (x, r) during phase 5a.

An honest member who receives an augmented key pair need not rely on the correctness of the received private key x and its claimed correspondence to the public key y. Instead, the receiver runs the deterministic key generation algorithm to compute $(x', y') = \mathcal{K}(\rho, r)$, and verify $x = x'$ and $y = y'$. Since ρ has a well-defined validity range and r is an unstructured bit string for which any sufficiently long value is by definition valid, a correct key generation algorithm must produce a working key pair for any valid input combination. Replay thus enables the receiver to verify that the purported key pair is a correct output of the key generation algorithm, before using the released private key for decryption.

A faulty member might choose the "random" bits r non-randomly during initial key generation. A non-random r might compromise the secrecy of ciphertexts encrypted using the public key generated from r, but such behavior harms the security only of the faulty member itself, as if the faulty member incorrectly revealed its private key before phase 5a. Independently of how the random input r was chosen or who knows it, a correct public-key cryptosystem must encrypt and decrypt reliably using the resulting key pair.