Scalable Shuffling with Round Ids in Dissent
Proposal by Sam Anklesaria for CPSC 490
Advised by Bryan Ford
February 11, 2015

Abstract
Yale’s Dissent project for anonymous communication requires an initial Shuffle phase in order to assign slots of message bandwidth to clients anonymously. Initially, this process required serial communication between each trustee server, and could be prohibitively slow with more than tens of trustees. More scalable strategies have been proposed, but so far, these designs suffer from high computational overhead or are susceptible to DoS attacks. I propose a modification to the Random Shuffler Subgroup strategy which does not suffer these drawbacks. Rather than allowing each node to elect itself into the shuffler subgroup, I believe the protocol can pick the shuffler subgroup using a PRNG seeded with the random round id collectively generated by all nodes during the round bootstrapping phase.

1 Introduction
We live in an age in which anonymous communication online is difficult. Governments monitor internet traffic on a massive scale. Private companies can make billions mining users’ browsing habits and linking their disparate internet identities. Yet truly free speech requires anonymity, or those with unpopular ideas can be threatened with physical harm. Popular anonymous communication solutions such as Tor offer limited anonymity, but cannot protect users from sufficiently sophisticated adversaries. Additionally, Tor’s onion encryption mechanism is useful for point to point communication, but is not easily applied to situations where each client wants to broadcast messages to a large group, as is the case in group messaging or voting. Yale’s Dissent project, on the other hand, provides an alternative approach to anonymous communication which is resistant to traffic analysis and applies well to broadcasting [1].

2 Background
In the following section, I will attempt to give a simplified overview of Dissent’s architecture and the underlying protocols that make it possible. In particular, I will describe the generation of round ids, the need for a shuffle protocol, and how the result of the shuffle is used for anonymous communication. Other unique features of Dissent such as the overlay protocol, tree addressing, trap bits and dynamic channel management will not be addressed. This description summarizes the one provided in Dissent: Architecture, Development Plan, Status [4].

Dissent consists of a number of phases. Initially, the system needs to establish the set of trustees or servers that provide clients with anonymity, a relay node to which all clients can form low latency connections, and the ephemeral public keys of these servers to be used during protocol exchanges. This phase is known as round bootstrapping, and is handled separately by the Collective Consensus protocol [3] (which, not strictly relevant to the generation of round ids, is not discussed here). Once the protocol is complete, each server takes the hash of this list of servers and their ephemeral keys to form the round id, a random value specific to this round of communication that prevents replay attacks. Because each server and its ephemeral key are used in constructing the round id hash, every server contributes to the round id’s randomness.

Next, Dissent must assign a slot to each client such that each client knows only its own slot. This is handled by the Shuffle phase. In its simplest form, based on Neff’s shuffle protocol [2], each client generates a pseudonym and onion encrypts it with the public keys of each of the trustees. All onion encrypted pseudonyms are passed to a known starting trustee. Each trustee successively peels off a layer of encryption from all pseudonyms in the list and shuffles the list before passing it on to the next trustee. The last
trustee then reveals the final list, which will contain the unencrypted pseudonyms shuffled into a random order. Each client checks for its pseudonym’s position in the final list to establish its slot.

Before the Bulk protocol begins (in which actual message transmission occurs), each client shares a secret with each trustee. Trustees use these shared secrets to seed separate random number generators for each client. Each trustee xors the output of all its random number generators together and sends a long string of these random bits to the relay node. Clients use the same shared secrets to seed their own random number generators and xor the results together to create their own random stream of bits.

Finally, the Bulk protocol begins. Say each transmission consists of $M$ bits. Each client will take $M$ bits from its random bit stream and xor its message into the slot it was given by the Shuffle phase. It will send the resulting cyphertext to the relay node. The relay node waits for all clients to send in their cyphertexts and then xors all the cyphertexts together. The random bits generated by the clients and the trustees will cancel out, leaving only the cleartext messages, which are distributed back to all the clients. This process continues until one of the trustees goes offline, in which case the round bootstrapping phase begins again.

### 3 Modified Shuffle Protocol

The simple shuffle protocol discussed previously (Neff Shuffle) does not scale well as the number of trustees increases. Specifically, it requires $O(N)$ latency in serial message-passing steps, where $N$ is the number of trustees. One way around this problem is to select only a subset of all trustees to participate in the shuffle (the Random Shuffler Subgroup approach). If we assume some fraction $r$ of the trustees are dishonest, then randomly picking $k$ trustees as shufflers means that at least one of the chosen shufflers will be honest with probability $1 - r^k$.

The challenge, then is choosing this random subset of trustees to act as shufflers. One approach might be to have each trustee privately pick a random number between 1 and $N$ and elect itself to be a shuffler if its number is not over $k$. In this approach, however, dishonest nodes might elect themselves without picking a random number. This would inflate the expected number of shufflers, slowing the shuffling process considerably (effectively performing a DoS attack).

More generally, the protocol cannot depend on random numbers generated by a single trustee, as a dishonest trustee can always lie. All trustees must contribute to the randomness required in choosing the set of shufflers. If we look back to the round bootstrapping process, however, all trustees have already contributed to the randomness of the round id. Therefore, if the round id is used to seed a PRNG, its stream of random values will be unbiased as long as all nodes maintain their connections. These random values can be used to pick the $k$ trustees in the shuffler subgroup.

In particular, rather than generating a random number for each trustee and including only those whose number lie below a certain value, seeding a single PRNG allows the protocol to pick the shuffler subgroup directly. A value between 1 and $N$ will be generated to pick the first subgroup member, a value between 1 and $N - 1$ to pick the second, and so on until $k$ shufflers have been picked. This approach has the benefit of ensuring that there will always be exactly $k$ trustees in the shuffle, rather than $k$ trustees in expectation.

### 4 Potential Challenges

With the proposed modification to the Random Shuffler Subgroup protocol, nodes still need to agree on a value for $k$, the expected number of nodes to be included in the shuffler subgroup. Ideally, the value chosen for $k$ should be high when a large number of dishonest nodes may be present. But how will nodes know when this is the case? And how will new values for $k$ be proposed? These questions will need to be explored before a prototype implementation of the protocol can be constructed.

Additionally, the membership of the random shuffler subgroup is not enough to determine how the protocol will run. Pseudonyms must be passed between shufflers in a predetermined order. I will also need to explore how to have all nodes agree on this common order before starting the shuffle procedure.

Finally, subtle bias issues can arise in the face of unreliability. It is unclear currently how nodes leaving or entering the group will affect the randomness of the round id. Careful examination of the Collective Consensus protocol will be required before we can make any assumptions about the security properties of using the round id as a seed.
5 Deliverables

Primarily, this project will result in the design of a scalable shuffle protocol for use with random, collectively generated round ids. I will examine the advantages and disadvantages of the design with regard to both latency and security. In particular, I will compare the design with the unmodified Neff shuffle used in the original Dissent architecture as well as Pairwise Encrypted Message Swapping, a more scalable but computationally intensive approach to shuffling proposed for later versions of Dissent.

These comparisons will be backed up by data from extensive simulations, varying both node configurations as well as the number of dishonest trustees. Simulations will run locally, in the form of communicating goroutines, but also over the Internet, through PlanetLab. To conduct these simulations, I will construct a prototype of the modified shuffle protocol. This work will integrate with the existing project of building a new prototype of the full Dissent system. With the right parameters, this new Dissent implementation should be able to use my round id based protocol for its shuffle phase. To facilitate this interaction, my prototype will be implemented in the Go programming language currently being used for the new Dissent prototype, and will make use of Yale’s Crypto library for Go.

References


