Prospectus: Distributed Generation of a Nonce

Motivation
Nonces are random strings used only once that are combined with a standard message, in order to hopefully make it impossible to send messages a second time. Often these nonces are also combined with a timestamp, which makes it even more difficult for a replay attack to occur. [1]

The Dissent project, a scalable method of anonymous communication, uses nonces in all message signed, with a new nonce for each round of Dissent. Each message can only validly be sent once. Currently, however, the nonce is produced from the leader, and starts at 0. Since the nonce just begins at 0 and increments, it is quite easy to replay past messages from other rounds of Dissent. Thus, it is necessary to generate a nonce in a more secure fashion.

Since Dissent has an any-trust model, that is, clients only need to trust one of the servers, no single server could decide the nonce. Rather, the nonce must be generated in a distributed fashion, with each server contributing a portion of the nonce. The protocol would additionally require cryptographic techniques that guarantee that no one participant could unfairly affect the nonce generated or cause honest members to have different nonces for the same round.

In the prospectus that follows, I will first describe current practices in distributed nonces. Next I will describe the Dissent project in more detail, and then some potential attacks on nonces in the context of Dissent. Finally, I will describe in more detail my proposed project, and give deliverables.

Current practice in distributed nonces
The current most common method of producing nonces is using large random numbers, as in UUIDs in UNIX. UUIDs are randomly chosen 128-bit integers that are often used to uniquely identify a machine, though since the generation algorithm described in RFC-4122 can allocate new UUIDs very quickly, they can also be used as nonces. This method of creating a nonce can be distributed across different nodes and does not require central moderation, since each UUID generated is globally unique regardless of the machine that generated it [2]. Dissent, however, cannot use this method because no one machine can be trusted to provide all of the nonce. However, UUIDs could possibly form the basis of the individual group members' contributions to the nonce.

Other methods of producing nonces use timestamps or just start at 0, but these methods are unsuited for use in Dissent.

One thesis, the work of Diogo Monica in 2009, does deal with the problem of the collaborative generation of nonces, suggesting two algorithms for a commonly-known, random, and collaboratively generated nonce with random inputs from n participants specifically for ad hoc wireless networks [3]. In the both method, each participant suggests their random contribution in turn. The first method detects collisions in nonces at the expense of the possibility of attacks by adversaries with infinite resources, and the second method does not detect collisions but is guaranteed to terminate. However, adversarial participants could wait until they heard from all of the honest participants and then skew or impact the nonce in some way, so that method could be improved. Additionally, all these protocols
assume that all participants are aware of wireless collisions. Although this version of Dissent is not done via wireless communication, other problems such as dropped packets could mean that one or more participants do not get all of the messages without any other participants knowing that the messages were lost.

**Dissent**
The goal of the Dissent project [4, 5] is to provide scalable, strong anonymity. Dissent expands upon the strong anonymity guaranteed by DC-nets, but makes the costs linear in the number of clients.

The current version of Dissent has two levels, clients which want to post messages anonymously (whistle-blowing journalists, for example) and servers hosted by anonymity providers. Dissent has an any-trust model, such that a client only needs to trust one of the servers in order for their anonymity to be preserved. Any nonce generation technique would preferably be able to depend only on this any-trust assumption and would not need Byzantine Fault Tolerance [6] assumptions, because it would be better not to weaken the powerful any-trust assumption of Dissent.

**Possible attacks on nonces**
In the context of Dissent and in general, there are several attacks that any nonce-generation protocol must guard against.

1. Replay attacks: In the current version of Dissent, it is possible for an adversary to make a replay attack, because the nonces just start at 0 and count up by round. Thus, a message from a previous round can be sent again just by using the nonce from that round.

2. Inconsistent nonce because of the adversary: Unless the protocol handles the nonce-generation carefully, it is possible that the adversary will be able to cause different servers to have different nonces, disrupting and possibly de-anonymizing communication via Dissent.

3. Adversary skewing the results: If servers can leave the nonce-agreement protocol in the middle, the adversary can remove specific nodes to change the resulting nonce to be biased in some way.

**Project proposal**
My goal is to design and implement new nonce protocols in Dissent. The servers will be the parts of Dissent that collaborate to determine a new nonce. The new nonce generation will be either every round or less frequently, to lower latency, if there is a secure way to either generate multiple nonces at once or generate new nonces from the initial nonce in a pseudo-random fashion.

The first step in adding new nonce protocols to Dissent is to add new pathways for alternative nonce protocols in Dissent. Then, I will implement a simple, perhaps unsafe solution and test it. Iteratively, then I will improve the solution, adding more security, likely via more complex cryptography.

Another I need to consider is what to do when a server gets disconnected from Dissent while the nonce generation protocol is in progress. One possibility is to abandon that attempt at generating a nonce, but that action could be used by an adversary to veto any nonce that it did not like. Another option is to wait until the server gets reconnected and just stall progress in the meantime, which is not good for the latency of the operations. A third option is to just compute the nonce without the randomness from the disconnected servers, but that allows the adversary to skew the resulting nonce. I will need to think more in detail about this scenario and handle it.
Evaluation of correctness
At the most basic level, I will need to make sure that Dissent works just as well before I add the new pathway for a nonce protocol and then new nonce protocols as after. It is also important to test the case in which a server gets disconnected for a period of time in the middle of the protocol, to ensure that the system behaves as designed.

For the iterations of the nonce that involve more cryptography, it will be necessary to provide proofs, or at least proof sketches, of their correctness, with the help of someone else who is knowledgeable about cryptography.

Deliverables
- Implementation of a pathway for different nonce protocols in Dissent
- Implementation of a simple nonce protocol in Dissent, and then as many further iterations as I have time for
- Written report

Bibliography


Namely, in the session code: [https://github.com/davidiw/Dissent/tree/master/src/Anonymity/Sessions](https://github.com/davidiw/Dissent/tree/master/src/Anonymity/Sessions) or as a portion of the Round itself, we should have a nonce generation round. The NullRound code may be a good starting place to figure out that. From there, you could take a peak at more complicated systems that use something called RoundStateMachine to help put together even more complex code...

A simple, but unsafe protocol would be the following:
A person participating in the protocol, pi, generates a number, ni, and produces a hash, hi.
pi shares hi with all other pj.
pi receives an hj from each pj.
pi shares ni with all other pj.
pi receives nj from each pj.
Then pi sets nonce = multi(n1 ... nk), where there are k participants.