Dissent: Group Management
by Nikola Chonkov
December 8, 2011
Advisor: Bryan Ford

Abstract:
Dissent is a protocol for accountable anonymous distribution of information among members of a decentralized group. Dissent groups are not static - one should be able to create, modify and maneuver them in a Dissent network, while trying to preserve the consistency of the group versions used by the nodes pertaining to a certain group. The contents of a group are vital for the correct execution of the protocols providing anonymity and accountability. Even though member groups are a central part of Dissent, no group management interface has been implemented to enable their manipulation. The group manager library described in this report covers the requirements mentioned, and enables the creation of new groups and group proposals, the storage of group data on permanent memory devices, and the retrieval of various information about the group content and the group version history used by nodes participating in a Dissent network.

1. Background:

There are many examples of information, that people are willing to share only if their complete anonymity is guaranteed – secret votes, 'Wikileaks' type of documents, military intelligence and so on. In most cases, these people want not only to remain anonymous, but also to be absolutely sure that they are sharing their data with a well known subset of the network. Dissent is a protocol that allows its users to create a network, in which they can share information anonymously with a well known set of other members. Anonymity is guaranteed by a combination of the Brickell-Shmatikov decentralized shuffle algorithm and a dining cryptographer network. A Dissent network operates in rounds. Every member of the group that comprises the network can send an encrypted message to the rest of the nodes at the beginning of a round. The broadcast data can be decrypted only at the end of the round, when every participant's input has been received. In this way, group members are protected against traffic and other analysis attacks. Dissent also utilizes a blame mechanism, which allows for the identification of misbehaving nodes while preserving the anonymity of well-behaved ones.

2. Design:

While designing the Group Management module, I had to consider some high level requirements that were provided by the Preliminary Architecture Specification for Dissent. The specification presents a brief overview of the type of functionality the module should provide, and suggests the idea of representing the version history of a group as a traversable graph. My goal was to identify some problems this additional module could help us solve and decide on how I should design and implement it so that it would be easy to integrate with the already large Dissent code base.

2.1. Highly isolated code:

The first design decision that I took was to make the Group Manager as modular as possible. Dissent is a complex project that is being actively worked on by many people. The code base is constantly being extended and changed, so isolating the Group Manager module’s code seemed like a good initial
approach. The files containing the Group Manager are VersionNode.hpp/cpp and VersionGraph.hpp/cpp.

Integrating some parts of the Group Manager’s functionality within the Session and Group modules seemed to be an attractive option, as this is where group versions would be most heavily used. However, in the long run this approach would make working on these modules cumbersome for both me and the other developers. In addition, the code would be much harder to keep track of and test.

2.2. Serializable objects:

Since Dissent nodes form a decentralized network, the group management data has to be easily transferable over a network. This implies that every class that it contains has to be easily serializable to binary data, and deserializable from such. I considered using the Boost library to do the serialization, but after talking with David Wolinsky - one of the main contributors to the project, we decided that it would pose too much unnecessary overhead, and it would be wiser for us to do the serialization ourselves.

The external modules that I had to include in the Group Manager module were the AsymmetricKey, CppPublicKey and CppPrivateKey from the Crypto++ library, and the Id, and Group modules. Conversion to and from binary data was already provided for the elements of the Crypto++ library. I wrote the serialize and deserialize functions of the Id and the Group modules. Some of these contain states implemented as containers of pointers to objects, which are not so trivial to serialize. To deal with this issue, I implemented the serialization function so that it makes deep copies of the objects to which it has a pointer and stores the dereferenced data in a contiguous byte array. Upon deserialization, intervals of this byte array are converted into the corresponding objects, and are made to populate the necessary array with pointers to segments from the new memory space. The container classes’ serialization features were provided by the Qt library.

The ability to easily serialize the Group Management data also made saving it to and loading it from a flat file on a permanent storage device trivial.

2.3. Group versioning:

The main reason why a Group Manager is needed in Dissent is that groups are not designed to be static objects. Users should be able to create new groups and propose and consider changes to the ones they already participate in. In the same time, all members should be able to follow the evolution of the groups they are a part of, regardless of whether they are online at the time the modifications are taking place.

Inconsistencies in the group contents among the members of a given group could break the Shuffled Send and Bulk protocol, cause loss of anonymity and other security problems. In order to deal with this issue, the Preliminary Architecture Specification for Dissent suggests using a group versioning system, where any change to a given group results in the creation of a new group version. In this way, all the modifications made to a group can be represented through a directed acyclic graph (DAG).

2.4. The Version Node design:

The nodes in a version DAG are immutable instances of the VersionNode class. Each node stores a set of parent nodes, corresponding to the versions from which the group it represents was derived. A node’s states also include the group documented by the version, its policy, and a cryptographic hash of all the information stored in the version node. This cryptographic hash serves as a unique logical identity for the version as a whole. Changes to any state of the version will result in a different hash, and therefore in a different identity.
2.5. The Version Graph:

The directed acyclic graph formed by the nodes described above is represented as an adjacency list in the VersionGraph module. This adjacency list is implemented as a hash table, where each version node is identified by its cryptographic hash. Thus, version look-ups can be done in amortized constant time and the addition of new versions is trivial. One problem with this design is that since each node in the graph is aware only of its parents, graph traversal is unnecessarily cumbersome. To get around this problem, we make the graph appear to be undirected in memory by storing a list of the children nodes for every node in the graph. This approach allows us to preserve the immutable nature of the nodes, and also makes traversing the graph trivial and as efficient as possible.

The VersionGraph module also stores a state representing the version of the group, that the user’s application believes is used by the rest of the members in the group. This state is necessary, because it allows Dissent members to “remember” which version from the VersionGraph they need to use, once they deserialize it.

The final piece of data that the VersionGraph module contains is a confirmation database. This database represents a multiset, where each node is a version of the group, and each edge is a “confirmation” by a user from a “from” version, expressing their satisfaction with the contents of the “to” version.

3. Functionality

3.1. Maintaining a permanent group version database:

The Group Management library allows Dissent users to store their group version database as a file on some suitable per-node permanent storage. The database can be saved as a file containing a binary representation of the VersionGraph class related to a certain group. Any Dissent node can later recreate and use the graph data stored in this file. While this last feature could have some practical uses - such as enabling a member to transfer their group membership to different Dissent-running devices, it could also pose a security risk if the group version database file comes into the possession of a malevolent Dissent user.

3.2. Group Creation:

The Group Manager enables any node in a Dissent network to create a new group. In order for a new group to be created, a list of members’ cryptographic identities and their corresponding public keys should be provided. A Dissent group could initially be created with many members, or with only one member, and then incrementally expanded through the addition of new members. As of now, a Dissent node creating a new group does not need to ask the members for permission to add them. This makes it possible for Dissent members to become parts of groups that they have not been aware of before. The Architecture Specification suggests calling such nodes “honorary members”.

3.3. New Group Version Proposals:

The library allows any Dissent user to create new versions of a group that they are a part of. New
version proposals are created when a group member sets up a new group with a new roster or policy and then creates a VersionNode which includes the new group, the cryptographic hashes of one or more existing versions, representing the new version’s parent(s), and the group’s policy. Additional group members or administrators may then confirm the proposed version, indicating that they accept it as the new, “legitimate” successor to the indicated parent versions. Only after the version proposal has been confirmed by a certain quorum, specified in the group policy, can the new group take the place of the old one. Thus members cannot make arbitrary changes to the group unless a subset of the rest of the group’s members have agreed to them.

3.4. Confirm Version Proposals:

When a new version proposal is made, it has to be confirmed by a subset of the members of the currently active group. The size of this subset is determined by the group’s policy. This functionality is provided by the Group Management library through the “confirm” function. The arguments that it takes are a “from” and a “to” group, and a member’s public key. Members can confirm a group only once, and only if they are a part of the “from” group, from which we want to migrate. As mentioned in section 5.2 of the original Dissent paper, this serves as a protection against colluding members wishing to limit the anonymity of some honest members. Besides the requirement for a quorum of agreement in order for the change to take place, this approach also allows Dissent group members to analyze the new proposal themselves, and choose whether they want to be a part of it even if it gets accepted.

3.5. Find heads, find Union, Intersection, Symmetric Difference:

Different members of a Dissent group might want to change the currently used group version in different ways. This would result in multiple version proposals, descending from the same parent. In the graph representation of the group’s history, these version proposals might further be extended by other proposals, which derive from them. Therefore, it would be useful to be able to find all the “heads”, or version nodes descending from a given version that have no children. The method provided by the Group management library that allows this functionality is called “getHeads.” Having all head nodes at one’s disposal can be useful for a number of different purposes.

First, it enables the extraction of a “definite group” - a group composed of the set of members equal to the intersection of the sets of the members of all version heads that we have found. If we are able to find such a group with a size equal to the minimum size required for anonymity to be preserved, then we can ensure that there exists a group on the contents of which everybody would agree even if no consensus was reached for any of the official new version proposals.

We could also deduce the symmetric difference of the membership sets of all version heads. We can create a group comprised of the members which cause the lack of consensus. Such a group would be useful if we wanted to, for example, create a “ban list” of members to be avoided. These two methods are implemented as getIaSD in the VersionGraph module.

We could also take the union of all heads to see what all possible members of a hypothetical new group could be.

3.6. Check if a given version belongs to the group's version history:

The Group Management library allows Dissent members to quickly verify if a given group
version’s cryptographic hash belongs to the group’s history. Since in most cases only a subset of a group needs to confirm the transition to a new group version, we can expect some Dissent nodes to miss a group transition, because they were offline for example. Once they try to rejoin what they believe to be the current group version, some of the updated members will need to figure out that an outdated node is attempting to rejoin the network. Since the version graph database allows for fast look-up by cryptographic identity, this check can be easily done.

4. Further improvements:

4.1. Decentralized Versioning System:

The Group Manager’s key component in this report is the versioning system. The versioning method described is specifically tailored towards group management, but if redesigned correctly, it could be used to version any kind of data in a decentralized system - encryption keys, transmitted data, etc.

4.2. Improve modularization:

As mentioned earlier, one of the design decisions that I took when I started working on this project was to keep the Group Management module as separate from the rest of the code as possible. As I was finishing, however, I realized that it would be a good idea to move the group policy state and the methods that find the intersection and symmetric differences of groups to the Group module. This would make their implementation much simpler, and from an architectural perspective, it seems to me that this is where they really belong.

4.3. Consensus Problems:

What is to be done if there are multiple new version proposals for a given group, and none of them has a confirmation quorum and no definite group can be found? What makes for a good least number of member confirmations that should be considered sufficient and safe? Even though the original Dissent paper proposes some solutions to these problems, I believe they can still be considered to be open.

4.5. Versioning optimization:

Right now, each new version of a group contains a changed copy of its derivatives. We can see that version maintenance projects such as “git” often store only the differences between the old and the new data that they track. This approach could greatly reduce the overhead of the versioning system described in this report, especially if it turns out that in most cases members apply only small changes to groups.
4.6. Improve Version database permanent storage:

The Preliminary Architecture Specification implies that we should expect moderate sized groups, so I believe data compression mechanisms will not be necessary, at least initially. However, it might be a good idea to encrypt the saved group data in order to prevent security risks related to tampering or theft of the file in which the versioning graph is stored, etc.

5. Conclusion:

Dissent is the first large research project that I have worked on. By being a part of the group that develops it, I learned a lot about computer networks, encryption schemes, decentralized systems, data versioning techniques, performance measurement and metrics, and many other technical topics. However, I believe the most valuable lessons I learned came from the group’s discussions, where we designed the approaches that we need to take to achieve our goals.

6. References:

2. Ford, B., Dissent: Preliminary Architecture Specification