Using Collective Signing Towards Collective Consensus in a Distributed Network based on Anonymous Communication

Iulia Tamas

Abstract

The problem of achieving Collective Consensus is of great importance in the field of Distributed and Decentralized Systems as it allows groups of nodes to store information reliably and make progress. In the context of Dissent, a group communication system which guarantees the anonymity of users within their communication group, achieving Collective Signing becomes a very important sub-part of achieving Collective Consensus. The anonymity and security guarantees of Dissent crucially depend on the accuracy, stability, scalability and efficiency of Collective Signing. This paper details the work that will be done towards achieving an infinitely scalable Collective Signing protocol between a set of independent nodes, in a decentralized system.

I. Introduction

The Dissent research project aims to create a group communication system that offers to the participants anonymity within their group. Moreover Dissent offer strong cryptographic guarantees of disruption resistance, message integrity, proportionality, and location hiding.

Collective Signing is important for the Dissent Project, as it is the first step towards Collective Consensus. One reason why we need a Collective Consensus protocol is to facilitate building a fully decentralized reliable directory service for Dissent, which will maintain a list of all the volunteer Dissent servers. To achieve reliability, we need to be able to properly keep track of the nodes, to make sure that the right nodes are participating in the protocol, and that they are up to date and complying with the protocol. Moreover we would like to be able to effortlessly be able to track down failed, misbehaving or out of date nodes. The Collective Signing Protocol is what will help us achieve all these goals, by enabling all participating nodes to generate a new collective signature for every round of communication, and sign in their names the message or piece of information that is being passed around or decided on for the round.

II. Personal Contribution

Being part of the Yale DeDis Research group, my contributions to Dissent will be done as part of a group effort. I will be working on the Collective Signing Protocol in direct collaboration with at least one other Yale undergraduate, and in indirect collaboration with the DeDis Research group members. As developing a scalable, secure and efficient protocol for achieving Collective Signing in a group of independent nodes as part of a decentralized system is in itself a complex project, it can split in several sub-parts. A few of them would be:

- implement and integrate with cryptography algorithms and tools
- implement a way for the independent nodes to dynamically self organize in a

\[1\] More information about project and contributors can be found at: http://dedis.cs.yale.edu/dissent
Senior Thesis Proposal • January 2015 • Iulia Tamas

- implement the infrastructure needed to reliably store, and access at various moments in time the information being signed on during successive rounds
- implement an abstract network layer to eliminate dependency on a particular connection type
- implement the 4 phases of the Collective Signing protocol

Having all these sub-projects allows each of the team members to focus and do more independent work on at least one or two sub-parts, while also collaborating on integrating all the sub-parts. As such, I will be first focusing on implementing the Collective Signing Protocol itself, and on integrating in the ElGamal Signature Scheme. The sub-projects I will continue with after are described in the Next Steps section.

### III. Methods

#### I. Assumptions

A set of assumptions was chosen as to keep the problem of Collective Signing separated from other tangentially related problems: the set of servers(nodes) is well defined and stable, each server has agreed on a public key the other servers are aware of, the servers are organized in the form of a tree with a node being able to connect to its parent node and its children nodes.

As the project advances and moves more towards the general Collective Consensus problem, some of the assumptions listed above will be reconsidered and dealt with.

#### II. ElGamal Signature

A basic ElGamal Signature can be used if during a communication between two nodes/ servers, one of the servers needs from the other one a proof of its identity. As a first step, server 1 asks server 2 for a commit point. Server 2 generates a random secret value, v, and chooses a commit point, V, based on it to send to server 1. Then, server 1 computes a challenge $c = \text{Hash}(V^2)$ and sends it to Server 2. Server 2 now responds with $r=v-c*x$, where x is Server 2’s private key. Knowing Server 1’s public key, X, and V, Server 1 can check that the messages were all received and sent back by Server 2, by seeing that $g^r * X^c$ is equal to V.

#### III. Collective Signing Protocol

A round of the Collective Signing Protocol, which is enough to ensure the collective signing of one piece of information by all servers, is composed of 2 round trips from the the root of the tree of servers down to all its descendants.

**First** The Announcement Phase. The root signs a message and sends it down to its children. The children propagate it downward to their children, recursively, until all nodes in the tree have received the announcement. The announcement serves the purpose of informing the nodes that a new Collective Signing Round is beginning.

**Second** The Commitment Phase. Upon receiving an announcement message, a node enters the Commit Phase, which leads to a Commitment Message being sent upwards to a node’s parent. The commit phase consists of a node generating a round-lasting secret($\nu_i$), and a commit point based on that secret($V_i$). These two values, are the Elliptic Curve Cryptography equivalents of a general Public Key, Private Key pair. Before doing anything with these values, a node waits for its children to send it their Commitment Messages consisting of 2 values: $V_i$ and $\hat{V}_i$. $V_i$ is the commit point for this round generated by child i. $\hat{V}_i$ is $\prod_{j:\text{child}(i)} V_j$. After receiving all commitment messages from its children, a node can compute its own $\hat{V}$ value, pack it with

---

2Realistically we would also hash here a message that the two servers are trying to sign
its $V$ value in a Commitment Message, and send it upwards to its parent.

**Third** The Challenge Phase. After the root receives all Commitment Messages from its children, it multiplies them together to get $\hat{V}_0$. It then generates a challenge, equal to a hash of $\hat{V}_0$, and some piece of information that the nodes are collectively trying to sign. The root node then packs this challenge in a Challenge Message and broadcasts it down to its children, as to reach all of its descendants, just as in the case of the Announcement Messages.

**Fourth** The Response Phase. Upon receiving a Challenge Message, a node enters the Response Phase, which leads to a Response Message being sent upwards to a node’s parent. The Response Phase consists of a node generating a response, $r = v - xc$, as described in the ElGamal section. Before doing anything with $r$, a node waits for its children to send it their Response Messages consisting of 2 values: $r_i$ and $\hat{r}_i$. $r_i$ is the response for this round generated by child $i$. $\hat{r}_i$ is defined as $r + \sum_{j:\text{child}(i)} r_j$.

After receiving all commitment messages from its children, a node can compute its own $\hat{r}$ value, pack it with its $r$ value in a Response Message, and send it upwards to its parent.

After the 2 round trips are done, the root node has an aggregate response $\hat{r}0$ with which it can verify the integrity of the signature by performing a check equivalent to the one described in the ElGamal section. First compute $T = g^{\hat{r}} \cdot \hat{X}^c$. If the hash function used in the Challenge Phase, when used on $T$ and the piece of information the nodes are trying to sign generates a value $c_2$, equal to the challenge $c$, then the ElGamal Signature is complete, correct, and verified. Moreover, each of the non-root nodes is also capable of verifying the correctness of the partial ElGamal signature generated by the nodes within its sub-tree. In order to verify it, a node computes $T$ based on its own $\hat{r}$ and $\hat{X}$ and checks that it is equal to its $V$.

**IV. Next Steps**

After completing Collective Signing I will focus on the information that the nodes are interested in signing. We want the decentralized network of servers to participate in creating a scalable and disruption-resistant logging and timestamp service. Each server will keep a log structure and all servers will agree on a time rate $t$. During every round of time $t$, the servers will collect from their local clients the items that are to be logged and timestamped during that time. Then, at the end of each period, the servers will aggregate the data in a large, decentralised Merkle Tree, and they will collectively sign the aggregated data and append the signature at the head of their log.

Other undergraduates will primarily focus on other issues such as dynamic reconfiguration of the server tree, and life insurance policies from servers to their clients via their group of trustees. We will discuss and collaborate, however, on achieving our many common goals as described in this paper, and we will integrate our sub projects as to contribute to a larger, closer to completion Dissent.

---

3 $\hat{X}$ is the product of the public keys of all the nodes’ descendants which we assume the node can easily compute as all public keys are public.