Title: Final Project Spec - Secret Sharing in Peerster
Author: Charles Jin
Class: CPSC 426: Decentralized Systems
Semester: Fall 2014
Group: Kayo Teramoto, Daniel Chiu

=== PART I

GOAL
To extend the Peerster application with a cryptographically secure secret sharing algorithm.

BACKGROUND
One of the biggest concerns of a decentralized system is security. In particular, without a central trustworthy authority, oftentimes it may seem that the safest course of action may be to trust no one.

There is a class of algorithms that seeks to alleviate this problem. Secret sharing employs redundancy to distribute portions of a secret to members of a group, without the assumption of trusting any single member. Rather, the algorithms generally rely on the assumption of a portion of the members being trustworthy, such that in a group of n players, it requires a group of at least t players to reconstruct the secret. Thus, the scheme is fault tolerant up to t-1 untrustworthy players, removing single-point vulnerabilities.

Several secret sharing algorithms have been proposed. In this project we will attempt to implement Shamir's Secret Sharing Algorithm. In the first case we will attempt to use the algorithm to transmit a secret key.

DESCRIPTION OF ALGORITHM
Shamir's algorithm is a (t,n)-threshold scheme, meaning that in any game of n players, at least t members need to collude in order to reconstruct the secret in a feasible amount of time.

Another desirable attribute of Shamir's algorithm is that any group of players of size < t do not gain any additional information by combining their secrets. A small tweak to the algorithm by taking arithmetic over a finite field yields this trait.

In short, the algorithm depends on the fact that it requires k points to uniquely determine a polynomial of degree k-1. Thus given the parameters k and n as above, with a secret S, the algorithm is as follows:

Randomly select k-1 numbers (a_1..a_{k-1}), and create a polynomial f(x) = S + a_1x + a_2x^2... . Construct n points from the polynomial (x, f(x)) and distribute them amongst the n participants.

Reconstruction with k numbers is elementary, as is the proof that k is the minimal number of elements for reconstruction.

IMPLEMENTATION
* Secret Share Generator

A node that wishes to spread a secret needs to have a way to construct the points (x, f(x)) as described above.

* Secret Regeneration
The nodes that hold secrets need to have a way to regenerate the secret as detailed above.

* Secret Message
A node needs a way to send a message to initialize the secret sharing algorithm.

* Reconstruction Message
A node needs a way to send a message to initialize the reconstruction of a secret.

* GUI elements - Secret Messages
A Peerster user needs a way to send messages to initialize secret sharing, as well as a way to initialize a reconstruction.

* GUI elements - Secrets
A Peerster user needs a way to see secrets it is currently the originator of, as well as secrets it holds a portion of.

EXTENSIONS
We would like to implement the algorithms over a finite field so as to be resistant to weakening.

DIVISION OF LABOR
* Charles:
Implement the three components of the secret sharing algorithm as described above.

* Daniel:
Implement the messages as described above.

* Kayo:
Implement the GUI elements as described above.

=== PART II

GOAL
To implement Vanish in Peerster—an implementation of self-destructing data leveraging a distributed system and Shamir's Secret Sharing.

BACKGROUND
Privacy of sensitive data is a very popular topic recently. Of particular interest is short-lived data. In theory, as soon as any data is transmitted across a network, someone has seen - and thus has had the opportunity to save - a copy of the data. Ideally, one could create data that persists only for a user-specified time period, after which all copies of the data self-destructing without fail.

Geambasu, et al. propose a system called Vanish that is built on top of a distributed hash table in their paper "Vanish: Increasing Data Privacy with Self-Destructing Data". Vanish makes headway toward our perfect solution by exploiting the churn of a distributed system. Data is encrypted using a key which is then split into secret shares using Shamir's algorithm. The shares are then "sprinkled" across the DHT, and depending on the natural churn of the DHT, the secret will be impossible to reconstruct after a certain amount of time, as the number of remaining secret shares fall below the threshold.

IMPLEMENTATION

* Join
Nodes must be able to join the hash table and broadcast this to update finger tables.

* Leave
Nodes must be able to leave the hash table and broadcast this to update finger tables.

* Death
Nodes must be able to detect an unexpected death in the hash table and adjust accordingly.

* Vanish
Shamir's Secret Sharing algorithm must layered on top of the DHT to provide the functionality required. The appropriate threshold must be chosen, the secret needs to be distributed, and interested parties must be able to reconstruct the secret by requesting shares be returned.

NOTES

* All DHT algos will be implemented in accordance with the CHORD protocol.

* Current implementation will be limited to splitting and reconstructing secrets. We are planning on selecting and implementing an application for the next section.

DIVISION OF LABOR

* Charles:
Implement all Vanish-related functions.

* Daniel:
Implement node leave and unexpected death.

* Kayo:
Implement node join and finger table data structure.