# CSCI 4900/6900 HW 4: Distributed Authorization Service

Assigned: Tuesday, Feb 20
Due: Tuesday, Mar 13, 11:00AM (LATE SUBMISSIONS NOT ACCEPTED)

## 1 Motivation

In the last three home work projects, we implemented a simple beacon service. First we located other beacons that are currently online and accessible. Next we implemented a simple beacon service that provided a file service. In the third home work, we provided a service to access files that are available in remote beacons. (yet, are not directly known to the current beacon).

Throughout the first three home work projects, beacons provided a mechanism for identifying themselves using a *name:key* pair. When a beacon logs into the system, it provided its *name:key* pair. On successful validation of this authentication key, a beacon is provided with an authorization token. This token was used for all further transactions. When a token was transferred across beacons, the tokens were assumed to be valid.

In this project, we will provide an simple authorization service. You will develop a key server that takes a user *name:key* to issue an authorization token. In order to prevent a single token from being reused, each token should be valid for a fixed time interval. For ease of testing, assume that the tokens are valid for 5 seconds. The key server will also be used to validate an authorization token. Your beacons will provide a new service that will utilize one or more key servers to validate authorization tokens. Such validation would be necessary if a beacon wants to provide service only to certain clients. Such validation would also allow the beacons to collect a fee for utilizing its services.

## 2 Description

The goal of this project is to provide a simple authorization service. For this project, first we will implement a key server. Next, we will extend the beacons to utilize this authorization key service.

### 2.1 Key Service

The key service will provide simple authentication service. The key server provides authentication functionality to other beacons and hence need not provide an API for end user interactions. You could implement the key server as a single server or utilize peer-to-peer technologies to provide a robust key service mechanism. The key service provides the following functionality (described using a C-style pseudo code):

- **authenticate(name, key)** The key server authenticates the *name:key* pair and returns an authorization token. Each token is valid for a fixed time interval (set it to 5 seconds for ease of testing).
- **validate(token)** The key server will validate the authentication token and return the *name* that was used to create this token. If the token had expired, then the key server will return an error message, indicating that the token is no longer valid.
- **revoke(token)** The beacons can use this service to invalidate a token even before its time expires. For example, when a beacon has received the file that it was looking for, it could revoke the token, freeing up precious server resources on the key service.

### 2.2 Beacon extensions

For this project, the beacons will be extended to provide the following service:

- **validate(token)** This service allows the user to validate if a token is valid. The beacon will utilize the key service to validate a token. The beacon will print the name of the user to whom this token was issued or an error message if the token is invalid. The token may be invalid either because it was an illegal token or if the token had expired.

You should also extend your beacon services (provided from Homework 2 and 3) to always validate a token (used in **GET**, **PUT**, **CLOSE**, **SEARCHGET** services) before using the token. Hence, services that used to work before might fail now because of an expired or an invalid token.

As usual, you are free to choose the exact technique to provide the service described above. You can use a single central server for key service or a distributed peer-to-peer approach for providing a robust service.

## 3 Submission

Please submit your project, along with a succinct report called **REPORT.txt** (plain text is fine) describing your approach, the merits of your approach and compilation instructions. You will turn in your complete project as a single tar file. On gemini, please use `/home/profs/surendar/bin/turnin hw4 <your tar file>` to submit your assignment. You can submit your assignment multiple times. I will only use the latest submission. To see the files that you had submitted, try `turnin hw4`. **Remember, I will randomly choose students who will be asked to explain their approach in person**. Issues that you might consider while developing and evaluating your system are:

1. **robustness:** How reliable is your system against failures? Does your beacon recognize forwarding loops? (wherein the requests are forwarded around in a loop without making progress towards the destination)
2. **scalability:** If your beacon suddenly becomes popular because of the files that your provide, how much load can you tolerate before your system crashes? Does your service degrade gracefully? Are you immune to denial-of-service attacks? (wherein, beacons repeatedly open connections to you to prevent you from servicing other, legitimate users)