CS 1653: Applied Cryptography and Network Security
Spring 2010

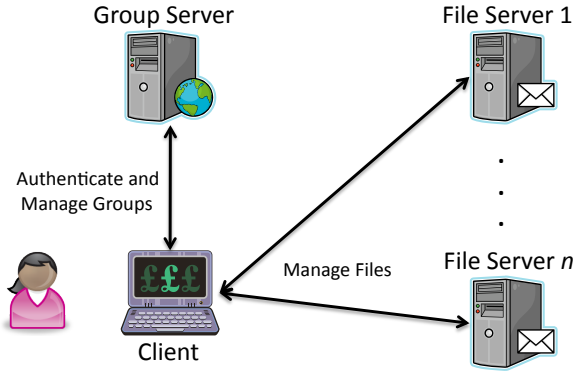
Term Project, Phase 2

Assigned: Tuesday, January 22 **Due:** Sunday, February 9 11:59 PM

---

1 Background

Over the course of this semester, we will experiment with and apply the security concepts that are covered in the classroom by developing a group-based file sharing application that is secure against a number of different types of security threats. In this phase of the project, you will implement a bare-bones system that provides the necessary functionality *without* any security features whatsoever. Recall that our system will consist of three main components: a single group server, a collection of file servers, and some number of clients.



The *group server* manages the users in the system and keeps track of the groups to which each user belongs. Any number of *file servers* can be deployed throughout the network, and will rely on the group server to provide each legitimate user with an authentication and authorization token that answers the question *"Who are you, and what are you permitted to do?"* Users within the system make use of a networked *client application* to log in to the system and manage their groups (via the group server), as well as upload, download, modify, and delete files stored in the system (via the file servers).

2 Trust Model

In this phase of the project, we are going to focus on implementing the *core functionality* of the file sharing service, and will defer the implementation of *security features* to later phases of the project. The reason for this is twofold. From a pragmatic perspective, you have yet

to learn the security "tricks and tools" that you will need to properly secure this type of application. Furthermore, having a fully-functional system in hand will simplify the later phases of the project, as you will only be debugging security features, not security features *and* core functionality. When implementing this phase of the project, you may make the following assumptions about the participants in the system:

- **Group Server.** The group server is entirely trustworthy. In this phase of the project, this means that the group server will behave *exactly* as specified in its interface (see `GroupClientInterface.java`).
- **File Servers.** The file servers are all entirely trusted. In this phase of the project, this means that each file server will behave *exactly* as specified in its interface (see `FileClientInterface.java`). For example, this means that you need not worry about a file server modifying a user's token, or making unauthorized use of a user's token (e.g., using the token itself or disclosing the token to another principal).
- **Clients.** All clients will behave in a trustworthy manner. This means that clients will not share their tokens with one another, nor will they make any attempt to modify their tokens.
- **Other Principals.** You may assume that the only principals in the system are the group server, file servers, and a single client. In particular, this means that you do not need to worry about the threats of passive attackers stealing information that is transmitted between the client and servers, or active attackers altering the data that is transmitted between the client and servers.

In short, you may assume that your code will be run by trustworthy principals in a closed environment.

3 What do I need to do?

In this phase of the project, you will be building two pairs of client/server applications: a group server and client application, and a file server and client application. For those of you who have never written a client/server application in Java, you might want to check out the file `server_sample.tgz` on the course web page. It contains a simple chat client/server application that could provide you with some helpful hints.

Although much of the code needed for this project is provided in the `project2src.tgz` tarball, we now explain the premise behind this code in a little bit more detail.

3.1 The Group Server

The primary purpose of the group server is to manage user-to-group bindings. In particular, the group server is a central point at which (i) an administrator can create or delete users; (ii) any user can create groups, delete groups, and add other users to the groups that they create; and (iii) a user can obtain a *token* that can be used to prove that they are members of certain groups. Once a user obtains a token from the group server, they can log into one or more file servers to upload, download or delete files.

To implement the required functionality, you will need to develop a client application that implements *all* of the functionality described in `GroupClientInterface.java`, which is contained in the `project2src.tgz` tarball on the project web page. Most of the functionality enabling these client/server operations is encapsulated in `GroupClient.java`, `GroupServer.java`, and their associated classes. We now briefly describe the functionality that your group server and client need to implement:

- `boolean connect(String server, int port)`
  Your implementation of this interface method is responsible for establishing a connection to the group server. No other method provided by your group client application should be able to work until this connection is established!
- `void disconnect()`
  This method is responsible for cleanly tearing down the connection to the group server. It should be invoked when exiting the client application, or any other time that the user wishes to disconnect from a particular group server.
- `UserToken getToken(String username)`
  This method is used to obtain the token describing the group memberships of a particular user. Given that we are working under a completely open trust model, the client does not need to "log in" to obtain this token: simply providing the user name is good enough for this phase of the project.
- `boolean createUser(String username, UserToken token)`
  This method creates a new user at the group server. The `token` parameter is used to identify the user who is requesting the creation of the new account. This should only succeed if the requesting user is a member of the administrative group "ADMIN". That is, not all users should be allowed to create other user accounts.
- `boolean deleteUser(String username, UserToken token)`
  This method deletes a user account, and again, should only succeed when run by a user in the "ADMIN" group. Deleting a user account should also remove that user from all groups to which they belong.
- `boolean createGroup(String groupname, UserToken token)`
  This method allows the owner of `token` to create a new group. The owner of token should be flagged as the owner of the new group. Any user can create a group.
- `boolean deleteGroup(String groupname, UserToken token)`
  This method allows the owner of `token` to delete the specified group, provided that they are the owner of that group. After deleting a group, no user should be a member of that group.
- `boolean addUserToGroup(String user, String group, UserToken token)`
  This method enables the owner of `token` to add the user `user` to the group `group`. This operation requires that the owner of `token` is also the owner of `group`.
- `boolean deleteUserFromGroup(String user, String group, UserToken token)`
  This method enables the owner of `token` to remove the user `user` from the group `group`. This operation requires that the owner of `token` is also the owner of `group`.

- `List<String> listMembers(String group, UserToken token)`
  Provided that the owner of `token` is also the owner of `group`, this method will return a list of all users that are currently members of `group`.

The `UserToken` interface plays an important role in most of the above methods, as it represents the binding between a user and the groups that he or she belongs to. Given the centrality of this interface, it would be prudent for your project group to design your implementation of this interface *before* tackling any other coding tasks.

3.2 The File Server

In our scenario, there will only be one group server, but there may be many file servers. After obtaining a token from the group server, a user can make use of *any* file server to manage his or her files. To implement the required functionality, you will need to develop a client application that implements *all* of the functionality described in `FileClientInterface.java`, which is largely encapsulated by `FileClient.java`, `FileServer.java`, and their associated classes. We now briefly describe the functionality that your file server client needs to implement:

- `boolean connect(String server, int port)`
  Your implementation of this interface method is responsible for establishing a connection to the indicated file server. No other methods provided by your file client application should be able to work until this connection is established!
- `void disconnect()`
  This method is responsible for cleanly tearing down the connection to the currently connected file server. It should be invoked when exiting the client application, or any other time that the user wishes to disconnect from a particular file server.
- `List<String> listFiles(UserToken token)`
  This method generates a list of all files that can be accessed by members of the groups specified in `token`. The user should not be able to see any files that are shared with groups to which he or she does not belong.
- `boolean upload(String sourceFile, String destFile, String group, UserToken token)`
  This method allows the owner of `token` to upload a file to be shared with members of `group`, provided that he or she is also a member of `group`. The `sourceFile` parameter points to a file on the local file system, while `destFile` is simply a name used to identify the file on the server.
- `boolean download(String sourceFile, String destFile, UserToken token)`
  This method allows to owner of `token` to download the specified file, provided that he or she is a member of the group with which this file is shared.
- `boolean delete(String filename, UserToken token)`
  This method allows the owner of `token` to delete the specified file on the server, provided that he or she is a member of the group with which this file is shared.

You *do not* need to implement support for traversing directories locally or remotely in this phase of the project. To simplify the coding tasks at hand, it is acceptable to assume that each file server exports a single directory for users to upload to and download from.

3.3 Client Functionality versus Client Interfaces

After completing instances of the `GroupClientInterface` and `FileClientInterface` interfaces and the associated servers, your group must also develop a client application (or pair of client applications) that allows a human user to access this functionality. Exactly what form the client application(s) take is entirely up to your group. For instance, one group might argue that a command-line based client makes the most sense, as it can easily be run from within scripts without requiring a human in the loop. Another group might argue that a text-based client makes the most sense, as you might *want* to require a human in the loop while still allowing the client application to be run via an SSH session. Finally, another group might argue that a user-friendly graphical interface makes the most sense. As long as your client application (or applications) provides a way for the user to execute *all* client operations, it will be sufficient for this phase of the project.

You are *strongly* encouraged to separate this user interface from your implementations of the `GroupClientInterface` and `FileClientInterface` interfaces. For instance, say that your group develops the `MyGroupClient` and `MyFileClient` classes that implement the above interfaces. You could then implement a class `MyClientApp` that provides the user interface. This clean separation provides you with several benefits. First, testing your client code is simplified, as the core functionality can be exercised in isolation. Second, your group can focus solely on implementing the client functionality before worrying about the user interface. This will allow you to build the best user interface possible, without worrying about doing so at the cost of not finishing the core functionality of the system. Lastly, you can implement more than one user interface! For instance, you may develop a text-based client to begin with, but later decide to add GUI functionality to simplify usage/testing in later phases of the project.

3.4 Development Tasks

The majority of the core functionality needed for developing (insecure) group servers, file servers, and clients is provided for you in `project2src.tgz`. Your job in this phase of the project is to complete the development of this system. In particular, you must do the following:

1. The code provided is nearly 2,000 lines of sparsely commented Java code. Your first task should be to read and understand the interactions between *all* included components. Do not put off this task!
2. You will need to develop an implementation of the interface described in `UserToken.java`.
3. You will need to implement the `connect()` method in the `Client.java` class. If you are unfamiliar with client/server programming in Java, the sample code provided on the course web page could prove helpful in this task.

4. The file server implementation is nearly complete, but does not include the ability to list the files that a particular user is allowed to see. Edit the file server to include this functionality.
5. The group server implementation includes handlers for all user-related tasks (creation, deletion, etc.). However, the handlers needed to carry out group-related tasks are just stubs. You will need to complete the group server implementation to support all operations required by `GroupClientInterface.java`.
6. You will need to develop a user-interface capable of connecting to group and file servers. This interface should allow the human user to carry out *all* functionality described in previous sections of this assignment.

3.5 An Important Hint

Note that your implementation of the `UserToken` interface is the *only* connection between the group client/server and the file client/server. It is in the best interest of your group to implement your instantiation of the `UserToken` interface before undertaking any other coding tasks. Once you have done this, you can work on implementing the group client/server and the file client/server in parallel. Given that your group contains at least two people, this could save you a considerable amount of time!

4 Extra Credit

In this phase of the project, you have the opportunity to earn up to 5% extra credit. Should you happen to complete the required portions of the project early, consider adding in extra functionality in exchange for a few extra points (and a more interesting project). Any extra features that you add will qualify, but here are a few ideas to get you started:

- **Graphical client applications.** Although a text-based or command line interface to the group and file servers is sufficient, adding a graphical interface to your code would make it easier to use and flashier to look at.
- **Support for the principle of least privilege.** There many be times when a user does not want to have access to all of their groups at once. Adding support for the principle of least privilege could be useful in these types of situations.
- **Support for server-side directories.** Directories exist for a reason: putting too many files in one place is messy! Adding server-side support for directories would be a nice touch to this project.

If you opt to do any extra credit, be sure to include a brief description of it when you submit the project (see Section 5).

5 What should I turn in?

To submit your project, first create a tarball containing the following items:

- **Full source code.** Include *everything* that we need to compile your code from scratch. Please do not include any JAR or class files, as we will be rebuilding your code before we evaluate it. Note that we will be grading your assignment using the machines in SENSQ 6110. It is in your best interest to test your code in that room prior to the submission deadline!
- **Compilation instructions.** In a text file named `compile.txt`, please write detailed instructions for compiling your code. If you opt to use a makefile or other script to build the code, include instructions here on how to run your build script.
- **Usage instructions.** In a text file named `usage.txt`, include explicit instructions on how to use your system. In particular, this should explain how to start your group server and file server, as well as how to start your client application(s). For each operation supported by your client application(s), provide a short statement of how to invoke this operation.
- **Extra credit.** If you have gone above and beyond the requirements of this project, please explain what you have done in a text file called `extra_credit.txt`.

Your project is due before 11:59 PM on Sunday February 9th. Late assignments **will not** be accepted! To prevent naming conflicts during submission, please include the netids of your group members in the name of your tarball. Hand in instructions will be posted on the course web page prior to this due date.

In addition, *each student* should send a brief email to Professor Lee (adamlee@cs.pitt.edu) and the TA (bill@cs.pitt.edu) that indicates his or her assessment of each group member's contribution to this phase of the project (e.g., *Bill did 40% of the work, and Mary did 60% of the work*).