CS5430: System Security Programming Project (Spring 2026)

Phase 4: User Authentication

General Instructions. Work together in a group of 2 or 3 students from the class.

Due Date: May 4 at 11:59pm on CMS.

The secure channels implemented in Phase 3 provide no assurance to the server about the identity of the human user who is entering commands at the client during a session. Phase 4 addresses that by authenticating the identity of the human user who invokes a `LOGIN` command. In general, we might authenticate human users by checking what they know, what they have, or what they are. The scheme you implement in Phase 4 is based on what the user knows. It involves: (i) designing a password scheme for authenticating a human user who is attempting to begin a session and (ii) incorporating that scheme into the authentication protocols delivered for Phase 3.

Implementation. Phase 4 modifies the `REGISTER` command to require an additional triple that gives a password *pass* for any user-id *user-id* being added to the registry.

`op = REGISTER; uid =` *user-id*`; password =` *pass*`;`

As before, if *user-id* is already in the registry then the command should fail. The output for this command should be a single triple giving the appropriate value for `status`.

Phase 4 also modifies the `LOGIN` command to require giving an additional triple that provides a password *pass* for the user-id *user-id*.

`op = LOGIN; uid =` *user-id*`; password =` *pass*`;`

A `LOGIN` operation fails if the value *pass* that is provided does not match the password value currently associated with *user-id*.

Finally, with the Phase 4 version of the system, human users may change passwords by invoking a new command `CHANGE_PASS`. This command makes *newpass* into the password for user-id *user-id* if *user-id* already is a registered user-id and if the human user who has invoked `CHANGE_PASS` shows knowledge of currently registered password *pass* for user-id *user-id*.

`op = CHANGE_PASS; uid =` *user-id*`; oldpass =` *pass*`; newpass =` *newpass*`;`

`CHANGE_PASS` may be executed from within a session or from outside of a session. The command fails if the human user who invoked `CHANGE_PASS` does not provide the correct currently registered password for *pass*.

Various designs are possible for this kind of authentication scheme. A good design is informed by what is being assumed about the threat. You may assume that the client is more likely to be attacked successfully than the server. Also assume that, over time, the same client computer might be used by a series of human users who do not trust each other.

Here are some questions to ponder for your design:

1. What are the requirements, if any, for a string that will serve as a password? The latest version (April 2026) of NIST Special Publication 800-63B will be helpful.
2. Where are passwords and related information stored?
3. What information should be stored when a user is registered?
4. What is the appropriate means for storing passwords in a secure way? Here is some suggested reading to consult about password hashing: https://en.wikipedia.org/wiki/Key_derivation_function#Password_hashing.
5. What information is sent over the network. In what format?
6. Since a `CHANGE_PASS` command is not necessarily invoked during a session, how should secure communication be achieved?
7. What restrictions should be enforced to help frustrate attackers who are attempting to guess passwords?

The Phase 4 Assignment:

Design the protocols for password-based authentication of users. Document this design by giving the answers to questions 1 – 7 above. Then implement the design and submit that codebase, along with suitable test cases. If necessary, revise the protocols implemented for Phase 3 and explain those revisions, along with the reasons for them. Also, if necessary, add to `CryptoUtils.java` any cryptographic functions that you need to support storing passwords (but you may not change the functions that are already present in `CryptoUtils.java`).

Grading and Submissions

This programming assignment should be implemented in Java version 25, which is available on the `ugclinux` computers. Submit the following files to CMS.

- Modified files for `Server.java`, `Client.java`, and `CryptoUtils.java` that implement Phase 4.
- `Tests.zip` that contains test data (in one or more files) demonstrating correct operation of your system.
- `testRationale.txt` that explains how well your tests exercise the extended functionality of the system and don't compromise the initial functionality.

- `design.txt` that describes the design decisions behind your implementation, including answers to questions 1 – 7 above, a discussion of the expected performance of your code, and an explanation of changes you made to your Phase 3 submission.
- `revisions.txt` that describes any revisions to your protocol from Phase 3 and explains why changes were made.

Grading. Project grades will be based on the following rubric

- 35% -- A good design, accompanied by compelling justifications.
- 15% -- Correctly repaired problems found and reported regarding your Phase 3.
- 30% -- System operates correctly on tests provided by course staff to exercise functionality.
- 10% -- How well the tests provided in `testRationale.txt` exercise the functionality and security of the system.
- 10% -- The quality of the explanations in `testRationale.txt`.

Automatic deduction of up to 50% if the program does not compile using `build.sh` or does not run using `run.sh` on the `ugclinux` computers. *Be smart: Try to rebuild your system on the `ugclinux` computers before you submit it.*