# CS5430: System Security Programming Project (Spring 2026)

# Phase 3: Secure Channels

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

**Due Dates:**
Part 1 Submit design document. 3/27 (Friday) 11:59pm on CMS
Part 2 Submit code for running system. 4/17 (Wednesday) 11:59pm on CMS

The key-value store we have been implementing uses a network to link the client and the server. So that system can be subverted by a Dolev-Yao attacker that might insert, modify, or monitor the request messages sent by the client or the response messages sent by the server. Phase 3 is concerned with eliminating this class of vulnerabilities. You will (i) design protocols to resist Dolev-Yao attackers and (ii) implement those protocols by extending your Phase 2 code. So, Phase 3 is concerned with creating secure channels. These secure channels will enable all communication between the client and the server and vice versa to be **authenticated, secret,** and **integrity-protected**.

In your key-value store, interactions (except for `REGISTER`) between the client and server are already partitioned into sessions, with at most one session active at any time. Each session starts with a `LOGIN` operation and ends with a `LOGOUT` operation. Prudent security engineering would have each session use a separate secure channel, involving fresh secrets. That way, compromising the key(s) used for one session would not help an attacker to compromise traffic sent during a previous session or a later session. The secure channel associated with each session would be established in response to a user `u` submitting a `LOGIN` operation and the secure channel should be terminated in response to `u` submitting a `LOGOUT` request.

## Cryptographic Building Blocks

To implement secure channels, employ cryptography. For your convenience, we are providing the `CryptoUtils` class, located in the module `crypto.utils.module`. We believe that this library contains all of the cryptographic operations you could need for this project.

Recall, public-key encryption and decryption operations create longer messages and require significantly longer execution times when compared to shared-key cryptographic operations. Therefore, public-key cryptography should be used sparingly. Also, a sensible rule of thumb: A private key that was generated by a principal should not be communicated to any other principal.

A "real" networked system would have a certification authority (CA) to mediate distribution of the public keys to be used for encryption and to be used for digital signature verification. In such a system, the client would query the CA to learn those public keys for the server. Simulate the functionality of the CA by having a file `ServerPubK` that the server can write to and that the client can read from. The server "posts" public key(s) by writing to the file; the client learns the server's public key(s) by reading the contents of this file. We assume that Dolev-Yao attackers

can read this file but have no way to write to the file. We also assume that Dolev-Yao attackers are unable to corrupt the file contents and are unable to corrupt value being posted to the file by the server or delivered to a client that reads from this file.

## The Phase 3 Assignment

The Phase 3 project is split into two parts, with a different delivery date for each part. Your design and implementation is allowed to modify the client and the server but not allowed to modify anything else. Think carefully about what functionality needs to be added to the client and what needs to be added to the server. Duplication of effort is bad design.

> **Part 1**. Design the protocols to implement a channel that enables authenticated, secret, and integrity-protected communication between the client and the server and *vice versa*. The design is allowed to modify the client and the server, but not allowed to modify anything else.
>
> Document this design by providing a "protocol narration" (as we have been using in class) for the protocols used in communication between the client and server. Also provide a description of the properties your system is intended to enforce, and explain why you believe the properties are enforced. This design document should be only a few pages long.

> **Part 2**. (a) Download `phase3.zip`, which is available on CMS. This version of the system does not contain a file `Server.java` in the folder `server-module`. The version also includes the folder `crypto.utils.module` that contains the `CryptoUtils` class. As an aside, the parser in this system has been updated to use base 64 encodings for the serialization of requests and responses. This update is for consistency with the `CryptoUtils` class, which also uses base 64 encodings for various functions.
>
> (b) Implement the protocol you designed in Part 1 and submit that codebase. You may modify the client and you may modify your Phase 2 server. You may not modify any other code downloaded in `phase3.zip`. If necessary, revise the protocol design submitted in Part 1, explaining those revisions and the reasons they were necessary.

## Grading and Submissions

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

**For Part 1:** Submit a pdf file `protocolDesign.pdf` to CMS.

**For Part 2**: Submit files `Server.java` and `Client.java`, containing the code for an extended versions of the server and the client, and `Tests.zip`, containing test data (in one or

more files) that demonstrate correct operation of your system, to CMS. The .zip file `Tests.zip` should incorporate the following.

- `testRationale.txt` that explains how well your tests exercise the extended functionality of the system and don't compromise the initial functionality.

- `design.txt` which describes the design decisions behind your implementation, including a discussion of the expected performance of your code. Also describe any changes you made to your Phase 2 submission in order to correct problems that we found in that.

- `revisions.txt` which describes whether there were any revisions to your protocol proposal from part 1 and explain why changes had to be made.

**Grading**. Project grades will be based on the following rubric

- 5% -- Part 1 gives an appropriate list of properties to be supported
- 15% -- Part 1 write-up is clear and explains why properties follow
- 20% -- Part 1 protocols do not have vulnerabilities
- 20% -- Part 2 system operates correctly on tests provided by course staff to exercise functionality
- 15% -- how well your tests in `testRationale.txt` exercise the functionality of the system.
- 15% -- the quality of the explanations in `testRationale.txt`.
- 10% -- code quality, readability, and documentation in the form of comments.

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 your system on the `ugclinux` computers before you submit it.*