# CS447 : Networks and Data Communications

**Programming Assignment #3 (P3)**

**Total Points: 150**

| | |
|---|---|
| **Assigned Date** | : Thursday, November 07, 2024 |
| **Due Date** | : Thursday, November 21, 2024 @ 01:59:59 p.m. (*hard deadline*) |

## Overview

Building upon the library book management system from P1, you will now enhance its security. This assignment focuses on implementing security mechanisms to protect the system from unauthorized access and ensure data confidentiality and integrity. You will implement Transport Layer Security (TLS) for secure communication, a secure login system, an authentication protocol, a secure password generator, and learn how to protect sensitive data at rest.

**Note:** This final assignment has a **hard deadline** and does not include the typical 48-hour late penalty period. Your learning objectives are:

a. Gain practical experience with OpenSSL for TLS implementation, secure password hashing, and data encryption.
b. Understand the importance of secure password management and implement techniques for generating, storing, and transmitting passwords securely.
c. Develop skills in securing network applications and protecting sensitive data in transit and at rest.

The nature of this last assignment will force you to do significant amount of unsupervised learning – online research, forum scans, trial-and-error, and troubleshooting. Most likely, you will also discover (on your own) that there are more than one library package implementation of `openssl` standard, especially for C/C++, which might further complicate your task. So, don't get frustrated but instead use this as a valuable learning opportunity.

## Back Story

Avast, ye mateys! The high seas be a treacherous place, not just for ships but for the digital kind too. Word's reached the ears of Captain Haddock that scurvy pirates be lurking in the digital ether, lookin' to plunder precious knowledge and disrupt the crew's peaceful book-borrowin' ways.

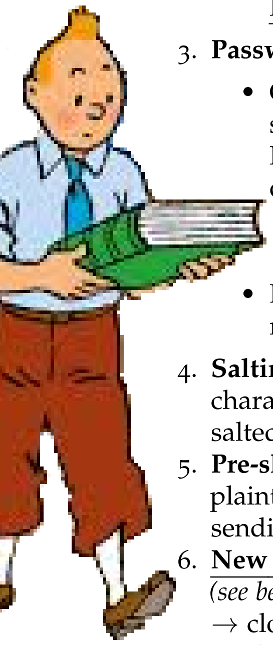
Professor Calculus, ever the vigilant guardian of the ship's library, has hoisted the cybersecurity sails. He's charted a course to reinforce the online book management system, makin' it as sturdy as a kraken's shell. But he needs a crew of savvy cybersecurity buccaneers to help him batten down the hatches.

That's where you fine folks come in. So swab the decks, hoist the colors, and prepare to defend the ship's library against the digital pirates!

## Technical Requirements

ALL P1 technical requirements fully apply to the P3. Additionally, following new requirements should be met.

1. **Secure Communication**
   - Establish **TLS 1.3** connections between the `clients` and the `server` for all communication using **OpenSSL 3.2.2+** (default on zone instances). This ensures confidentiality and integrity.
   - Use OpenSSL's **SSL_CTX** functions to configure the TLS context:
     - `SSL_CTX_set_min_proto_version()` and `SSL_CTX_set_max_proto_version()` to set the protocol version to **TLS 1.3**.
     - `SSL_CTX_set_ciphersuites()` to select strong cipher suites.
   - Generate *self-signed certificates* for testing using OpenSSL's command-line tools. Ensure your generated files are named **p3server.key** and **p3server.crt**
2. **Implement a Secure Login System**
   - Replace `HELO` command with `USER` → `PASS` command sequence.
   - `USER <username>`: Used to send the `username` to the `server`. The `server` checks if the received `username` is a known value. If *YES*, await for the `password`. Otherwise follow the New User Registration Protocol *(see below)*.
   - `PASS <password>`: Used to send user's `password` to the `server`. The `server` follows the Authentication Protocol *(see below)* upon receiving a `password`.



3. **Password & Salt Generation**
   - Generate a 5-character[†] random password and a 6-character random salt for each user. The password should include at least one uppercase letter, one number, and one symbol (from the set `!@#$%&*`). Passwords may begin with an alphanumeric character but not with special symbols. Salts should consist of printable characters excluding whitespace.
     - Thought not strictly necessary, consider employing OpenSSL's `RAND_bytes()` to generate passwords and salts as it's readily available.
   - Ensure the generated passwords are strong and random. Implement a password strength checker and regenerate passwords if they do not meet the complexity criteria
4. **Salting**: Use the interleaving method to salt the password, i.e, alternate salt characters and password characters staring with the first salt character. (e.g., if the salt is `'S1a2tX'` and the password is `'P4s5w'`, the salted password would be `'SP14as25twX'`).
5. **Pre-shared key**: A secret symmetric-key shared between the `clients` and `server`. This key is used to encrypt plaintext passwords any time they need to be exchanged between `clients` and the `server`. This will prevent sending plaintext passwords over-the-wire. The pre-shared key for this assignment is **F24447TG**
6. **New User Registration Protocol**: Generate a secure random password → store it securely with the `username` *(see below)* → encrypt the password using a pre-shared key → transmit the encrypted password to the `client` → close the connection.
7. **Authentication Protocol**: Decrypt the received string using the pre-shared key → salt it → generate a hash → compare the generated hash against the stored hash for the user.
8. Your client code should decrypt the received server-generated password and print it to standard output before closing the connection.
9. **Storing Passwords and Salts**
   - Salt passwords using a unique salt for each user and hash salted passwords using **SHA-512** to protect against rainbow table attacks
   - As a homage to how Linux store passwords (in **/etc/shadow**), store the username, salt, and the salted password hash in a hidden file called **.book_shadow**. The format of this file should mimic the following:

```
username:salt:salted_password_hash
```

[†]NOTE: Typical passwords should be longer than 5-characters to avoid possible brute force attacks. We are using a shorter one simply for assignment purposes. It's definitely not a recommendation

## Functional Requirements

**ALL** P1 functional requirements fully apply to the P3. Additionally, following new requirements should be met.

1. Fix any lingering issues in P1 before starting on P3 tasks.
2. Multiple concurrent client functionality will be fully tested in P3. Thus, consider moving away from shared memory-based solutions (if you used one in P1) to a file-based solution.
3. It's not realistic to assume that your users know numerical **`book_ids`**. Thus, modify following P1 command syntax to avoid implementation confusions. You should, however, consider using an id value (or a hash table based approach) for efficient indexing purposes.
    - **`DETAILS`** ~~<*book_id*>~~ <*book_title*>
    - **`CHECKOUT`** ~~<*book_id*>~~ <*book_title*>
    - **`RETURN`** ~~<*book_id*>~~ <*book_title*>
    - **`RATE`** ~~<*book_id*>~~ <*book_title*> `<rating>`

4. The user enters the `password` (for the `PASS` command) in plaintext. Your `client` should encrypt it before sending to the `server`.
5. You must provide proof of secure communication. To do this, run `Wireshark` with and without TLS separately, and provide appropriately annotated screenshots in your report. Failure to provide sufficient proof of secure communication will be considered as an indication of not meeting project requirements.
    - Two terminal equivalents for `wireshark` are `tcpdump` and `tshark`. You can save your capturing session as a `.pcap` file, which can be later opened in `wireshark` for easier analysis. The `wireshark` manual explains how to use both. See here `https://www.wireshark.org/docs/wsug_html_chunked/AppToolstcpdump.html` and here `https://www.wireshark.org/docs/man-pages/tshark.html`

## Extra Credit

- For an additional level of obscurity, encode passwords in **`base64`** before encrypting them for transmission over-the-wire. Correspondingly, decode strings at the receiving end before other performing any other actions on the received data. You may also consider encoding anything else that you deem *"sensitive"* but make sure to provide an explanation as to why with appropriate proof of work (in screenshot form) in your report [**10 points**].

- **Only** take this next extra credit option if you have ample time, your core implementation is complete, and (to a lesser extent) you have some prior experience programming **`ncurses`**. Submissions that meet (or exceed) at least 80% of project objectives are eligible for up to **15% extra credit** by integrating a **`ncurses`** based text-based user interface (TUI) `https://www.gnu.org/software/ncurses/`. Consider incorporating features like a login screen, menu-driven navigation for browsing and searching books, and interactive forms for user input.

## Instructions

- **Start early & backup often:** This assignment requires careful planning and implementation and remember to save your progress frequently.
- **Don't procrastinate!!** As mentioned earlier, this assignment involves a significant amount of self learning secure network programming and cybersecurity concepts. Allow yourself plenty of time to absorb new knowledge.
- Maintain clean, readable code. Adhere to Google's C++ Style Guide or another established standard. Use one of Google's coding standard found here `https://google.github.io/styleguide/`, if you don't already follow one.
- Your code must compile and run flawlessly on a standard Linux environment. Test it thoroughly using command-line tools.

- Implement your solution in C++. Submit a `.tgz` file containing only your source code, a `README` with compilation instructions, and your report.
- Handling parallelism is not trivial. Start with a single-client version, then gradually scale up.
- Focus on core C++ socket and I/O functionalities. Avoid external libraries unless explicitly permitted.
- DEADLINE: **Thursday, November 21, 2024 @ 01:59:59 p.m. (*hard deadline*)** through Moodle. Email submissions are not honored.

## Deliverables

A complete solution comprises of:

1. Report (**`pdf`**):
   - Introduction: Describe your learning objectives from a cybersecurity and secure network programming perspective. Explain what you knew about secure programming techniques before starting P3 and what you hope to learn by completing P3.
   - Design: Explain your overall system design, key choices you made, and specific challenges you faced.
   - Sample Run: Include screenshots showcasing a typical interaction with your system. These can help illustrate functionality and potentially earn partial credit if certain features aren't fully implemented.
   - Proof of secure communication, password specification conformity, base64 encoding (if applicable, etc).
   - Summary and Issues: Summarize your achievements and what you learned from a cybersecurity perspective. Explicitly list any unimplemented or buggy functionality.
2. Compressed Tarball (**`siue-id-p3.tgz`**):
   - Source Code Directory: Include all your C++ source code. No need to include any `*.config` files. Also, no need to send your self-signed certificates either (as long as they following the naming standard specified earlier).
   - `README`: Provide clear instructions on how to compile and run your code.
   - `Makefile`: A Makefile is required, especially if your project involves multiple executables or compilation flags.
   - Your `.pcap` file as proof of secure communication.

Use the following command to create the compressed tarball:

```
tar -zcvf siue-id-p3.tgz p3/.
```

(Replace `siue-id` with your real siue email id and `p3/` with the name of your source code directory)

Your code should be a testament to your abilities, not a copy of someone else's work. Collaborate, don't copy! Learning from peers is great, but copying code (from classmates or online) is strictly prohibited. Cite any external resources you use for ideas, and then implement those ideas in your own way.

This assignment is a challenge designed to push your boundaries and foster growth. Embrace the learning process and don't hesitate to seek help when needed. Remember, the goal is to master these concepts, not just complete the assignment. Plagiarism, whether from classmates, online sources, or AI tools, stifles learning and compromises academic integrity. The instructor actively uses MOSS `http://theory.stanford.edu/~aiken/moss/` to check for software similarity. Plagiarism has severe consequences including and will result in a failing grade. No exceptions.

## Some Useful Resources

- Linux Man pages – found in all Linux distributions
- Beej's Guide to Network Programming – A pretty thorough free online tutorial on basic network programming for C/C++ `https://beej.us/guide/bgnet/`.
- Linux Socket Programming In C++ – `https://tldp.org/LDP/LG/issue74/tougher.html`
- The Linux HOWTO Page on Socket Programming – `https://www.linuxhowtos.org/C_C++/socket.htm`

- Learn Makefiles With the tastiest examples – https://makefiletutorial.com/
- Fedora Security Team – Defensive Coding
  https://docs.fedoraproject.org/en-US/Fedora_Security_Team/1/html/Defensive_Coding/index.html
- OpenSSL Wiki – Simple TLS Server
  https://wiki.openssl.org/index.php/Simple_TLS_Server
- The Illustrated TLS 1.3 Connection – https://tls13.xargs.org/
- A Readable Specification of TLS 1.3 – https://www.davidwong.fr/tls13/
- Creating a Self-Signed SSL Certificate – https://linuxize.com/post/creating-a-self-signed-ssl-certificate/
- Ncurses Programming Guide – http://jbwyatt.com/ncurses.html