COMP3334 - Project

Section 1: Overview

Online storage is a popular application in our daily life. With online storage, a user can upload its files to a server and access them when the user wants. The security of uploaded content is important because it may contain the sensitive information of users.

In this project, you and your teammates should design a secure online storage system, which contains various functionalities, such as user authentication, access control, file encryption and activity auditing, etc.

Section 2: Deadlines

1. Team Registration: 11:59 PM, March 6th, 2025.
2. Submission of required materials: 11:59 PM, April 6th, 2025.
    a. The materials include your report, codes and demonstration video.

Section 3: Team Requirements

Students should participate in this project in teams. Each team should have a voluntary coordinator for administrative purposes. The coordinator should fill in a form (https://forms.office.com/r/c8VaKumiMG, needs PolyU Connect account) to register his/her team before 11:59 PM March 6th, 2025.

You may use the discussion board in Blackboard to find your teammates.

To avoid high workload and free riders, each team should contain 3 or 4 students (4 is recommended).

If there are any students who are not in a team after the deadline, they will be organized as several teams randomly. We will try to keep the size of teams within 3~4 students. However, in extreme cases, it may not follow the regular guidelines.

Section 4: Threat Models

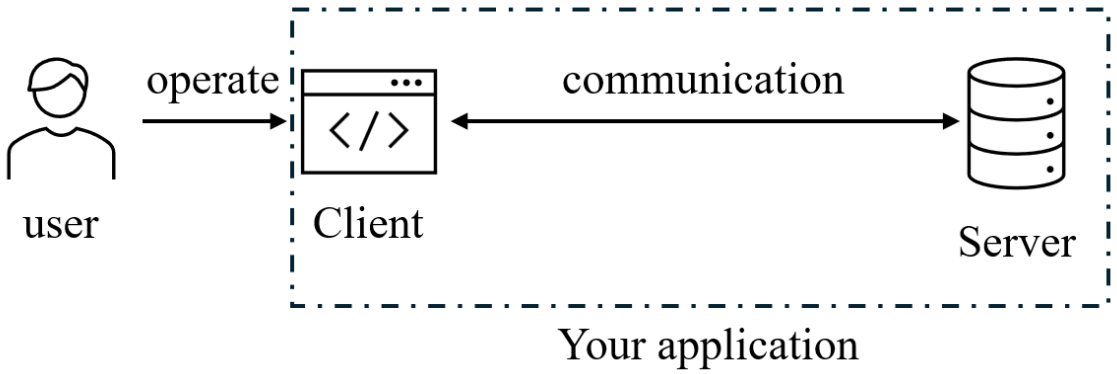
Your application should contain two sub-programs, Client and Server.

Client program helps a user upload its files and access them when the user wants.

Server program receives the uploaded files and manages the users.

A user operates a Client program to use your application.

Client and Server are communicated via network connections.



We assume that machine that runs Server is a **passive adversary**. It executes your program **honestly** but monitors communication and the stored data from a Client and wants to decrypt this Client's uploaded files. That means, the machine that runs Server does not perform active attacks, such as altering the messages, returning fake content, etc. It only READ the messages from a client program and wants to decrypt files based on the read messages.

We also assume that there is a passive adversary who is an unauthorized user. This unauthorized adversary may use a legitimate user's computer to try to access the online files of that legitimate user.

The security measures in your application should be able to prevent such adversaries.

Section 5: Functionality

The **CORE** functionalities of your application are listed below:

1. User Management:
    a. Register a user by username and password.
        i. The username must be unique.
        ii. The password must be hashed by a proper algorithm.
    b. Log in
        i. Check whether the password is identical to the password in registration.
    c. A user should be able to reset its password.
2. Data Encryption:
    a. Upload

        i. When a user uploads a file, the client should encrypt the file using an appropriate cryptosystem, with the key securely generated and stored locally.
        ii. Server should not be able to read the file in plaintext.
    b. Download
        i. When a user downloads a file, the client should decrypt the file and return the plaintext to the user.
3. Access Control
    a. A user can only add/edit/delete its own files.
    b. A user can share its files with designated users. The designated users should be able to read the shared files via their Clients.
    c. An unauthorized user should not be able to access the file content of other users.
4. Log Auditing
    a. The critical operations, such as logging in, logging out, uploading, deleting, sharing, should be recorded.
        i. A user should not be able to repudiate it.
    b. The administrator account of your application should be able to read logs.
5. General Security Protection
    a. File name must be valid. Some file names can be used to attack. For example, the file name "../file.txt" (without quotes) can be used to access *file.txt* in the parent folder.
    b. Your application should also consider the security threats on accounts, e.g., SQL injections.

The **EXTENDED** functionalities of your application are listed below:

1. Multi-Factor Authentication (MFA): FIDO2, One-Time Password (OTP), email/phone verification code, etc.
2. Efficient update on files: Suppose you are editing a file that has already been saved online. If you want to modify a part of this file, find a method that Client does not need to encrypt the entire file and submit it again.
3. Other security designs that you think are necessary.

Your application should implement at least **ALL** of the **CORE** functionalities.

Your application should implement at least **ONE** of the **EXTENDED** functionalities.

The implementations on **EXTENDED** functionalities will be considered in grading. (However, please do not add too many functionalities to your applications.)

To reduce your workload, your application does not need a Graphical User Interface (GUI). Running in command line is enough. However, you should at least provide a menu (in command line) to assist your user to use your application.

Section 6: Programming Languages and Potential Needed Tools

You may use any programming languages you are familiar with. However, it is recommended to use Python due to its low difficulty.

In the design of Server, you may need a database to host the user information. It is recommended to use SQLite, which is a lightweight database system.

Python has already provided some cryptography libraries. You can refer to our Tutorial 1.

If you are using C/C++, it is recommended to use *OpenSSL*, which is a popular and comprehensive cryptography library in C/C++.

It is recommended to use the existed cryptography libraries as building blocks, because your own implementation may not consider all security concerns.

However, you are not allowed to call **all-in-one** libraries to build your application.

Here is an example, which is simply called an existed library as your application.

```
import xxx_library

server = xxx_library.storage_server()

server.start()
```

As long as your implementation involves reasonable details for solving this problem, then it is fine. Unless it is too obvious, we will be very moderate when deciding if implementation is solely based on all-in-one libraries, i.e., **let us see your efforts**.

Section 7: Report File

Your report should be within 10 pages. More pages do not lead to higher grades.

- Include your team's name, your names and student IDs in the report.
- A contribution table indicating your percentage of contributions, in total 100%.
    - Grades will be adjusted accordingly.
- Abstract
- Introduction
    - Background

- Threat Models
    - Who are adversaries?
    - What are the abilities of adversaries?
    - etc.
- Algorithms you designed to implement functionalities
    - For each functionality requirement, what your theoretical design is.
        - Which building blocks (algorithms, tools, etc.) you used.
        - How you used them to design a workflow that meets the requirement.
    - To implement your theoretical design, what the technical details are.
        - Which libraries you used.
        - Are there any technical challenges? If yes, how you encountered them.
- At least 2 Test Cases
    - To verify whether your design can resist attacks.
    - Examples: Whether the files uploaded by users can be read by unauthorized users or not, SQL Injection Attacks, and whether unauthorized users can get the secret keys or not, …
- Future Works
- Reference

Section 8: Demonstration Video

A team should record a 10-min demonstration video to demonstrate the designed functionalities with necessary description.

Section 9: Code

Your code must contain all the source codes, a file that can be imported to SQL database and **a step-by-step document about how to deploy and use your application**.

This document must be able to guide a person to deploy and run your application from a **clear** Windows 11.0 OS (i.e., no assumptions on pre-installed software/libraries), i.e., your document should guide a person to install the needed software/libraries and use your application.

If you are using Python solely, it is recommended to export all your dependencies to a `requirements.txt` file when you are done.

Your code should be well documented that is comprehensive comments and is readable.

Section 10: Submission Guidelines

- Create a folder with the name *TeamName*
    - Put all your code in a folder with the name *code*
    - Rename your report with the name *report* (with the extension name, such as *pdf*)
    - Rename your video with the name *video* (with the extension name, such as *mp4*)
    - Put *code*, *report* and *video* in the folder *TeamName*
    - You should replace *TeamName* with your actual team's name, which will be released after registration period.
- Compress this folder as one zip file.
- Follow the example below to name your zip file by replacing *TeamName* with your actual team's name:
    - *TeamName.zip*
- Your submission should be submitted by your **TEAM COORDINATOR** before the deadline.

Section 11: Grading Rubrics

- Code (30%)
- Report (50%)
- Demonstration (20%)

Outcome Presentations	%	**A+/A/A-**	**B+/B/B-**	**C+/C/C-**	**D+/D**	**F**
Code	30%	Programs are well-organized, making good use of whitespace and comments. Variables have helpful names.	Programs are well-organized, easy to read and understand.	Programs can be read and are in a logical order.	Programs are runnable but barely readable.	Absent
Report	50%	Excellent, comprehensive and in-depth analysis with concrete facts/evidence	Clear analysis with good analysis supported by plenty of facts/evidence	Basic analysis with some level of facts/evidence	Barely relevant analysis with minimal facts/evidence	Absent
Demonstration	20%	Very clear and logical	Good, easy to follow	Understandable, structured	Barely understandable	Absent