Hacettepe University Computer Engineering Department Information Security Lab.

Homework 3

Subject: Public Key Encryption, Authentication
Due Date: 27.11.2024 23:59
T.A.: Ali Baran TAŞDEMİR, Sibel KAPAN

Introduction

In this assignment, you will develop a simple user portal where users can register and log in. The portal will have basic profile information as its only content. The assignment will focus on securely managing user credentials, storing them in a text-based database, and adding a hash chain-based OTP authentication step.

A One-Time Password (OTP) is a unique, temporary password that is valid for a single login session or transaction, adding an extra layer of security beyond traditional static passwords. OTPs are typically generated either through a time-based algorithm (TOTP) or a counter-based algorithm (HOTP), which ensures that each OTP is distinct and only usable for a short duration or a specific action. This makes it difficult for attackers to reuse a password even if they manage to intercept it, as it quickly expires or changes after use. OTPs are commonly used in two-factor authentication systems, sent to users via SMS, email, or generated through apps to verify identity and protect against unauthorized access.

Hash chain-based OTPs are a secure variant of one-time passwords generated by applying a hash function repeatedly to an initial secret or seed value. In this method, a sequence of hash values forms a "chain," where each OTP is derived from a hash of the previous one. To authenticate, a user provides the latest OTP in the chain, and the system verifies it by hashing it the correct number of times to match a stored value. As each OTP is only valid for a single use and cannot be reused, hash chain-based OTPs provide robust protection against replay attacks, where intercepted credentials might otherwise allow unauthorized access. This technique is often used in scenarios requiring lightweight but effective security, such as offline authentication systems or challenge-response protocols.

Key Features

You are expected to design a Client-Server simulation by utilizing the methods of public key encryption, hashing, and authentication. The following lines will state the key features of your assignment.

- **User Registration.** You will design a portal by using Python's Flask library. The first feature of the portal will be a simple user registration logic. Your portal should allow users to register by providing a username and a password. The username should not include any special characters and numbers. A password should be larger than 6 characters. You will provide a registration form from the Client side and send the registration information to the Server (The sent password will be hashed by using MD5

or SHA256). Also, the Client will create a hash chain-based OTP for the initialization (detailed information is shared in Section Hash chain-based OTP).

- **User Database.** All created users will be stored in a database file in a structured format. The database file will be accessed by only the Server. The client cannot access the file. For each user stored in the database, there will be a line (users separated by newlines). And for each user "username, hashed password, last OTP received) will be stored. This information is separated by ";" in the line. For the security of sensitive information, the database file is encrypted by the RSA algorithm.
- **User Login.** You will create a login page where the user can provide username and password information. The user will fill out the login form on the Client-side and the information is sent to the Server. The Server will check the login credentials first if the validation is successful, the client will send the OTP authentication hash to the Server. If the authentication token is validated the login will be successful and the user will be forwarded a welcome page. The welcome page will be a simple page that shows the "Welcome, {username}" message. The server will update the OTP information (also update the database file).

Implementation Details

Client

The Client is designed to serve as a front-end for the user. It will provide a register, login, and welcome screen. Besides providing simple screens, it will communicate with the Server for several tasks. You will implement the Client with Python's Flask library [1]. For cryptographic operations, you will use PyCryptoDome library [2], to install Flask and PyCryptoDome use;

pip install Flask, pycryptodome

- The Client should send the information of a registered user to the Server. It should check the validity of the username and the password before sending the information to the Server. The Client will send the hashed password to the server. To hash passwords you will use PyCryptodome's Hash functions (*Crypto.Hash.MD5* or *Crypto.Hash.SHA256*).
- The Client should send the information of a login attempt. It will collect the provided username and password information from a form, and send it to the Server. If the server validates the user, then it will start OTP operations. The client will send the OTP authentication token to the server. If the server validates the OTP token, the login will be successful.
- The Client shows a welcoming page, after the successful login for a user. The page only contains a simple welcome message in the format "Welcome, {username}".

You should implement corresponding functions, and front-end pages to the Client. You will run a Flask application for the Client. A simple setup for a Flask app;

[1] `https://flask.palletsprojects.com/en/stable/`
[2] `https://pycryptodome.readthedocs.io/en/stable/`

```
# Template for a simple Flask Server
from flask import Flask
from Server import Server

app = Flask(__name__)
server = Server()

@app.route("/")
def running():
   return "<p>Server is running</p>"

if __name__ == "__main__":
   app.run(host='0.0.0.0', port=5000, debug=True)
```

The example server is set to run on "localhost" at port 5000. The server has one functionality defined with "@app.route" at the home screen (defined by "/"). The server welcomes the user by rendering the "Server is running" message. You can add new functionalities by using @app.route to different URLs. For example, "@app.route("/link")" performs the described function below this decorator at the URL "http://localhost:5000/link".

Server

The Server is a simulation of a web server. But it will not work as a Flask server. Instead, you will implement all server-side computations inside this file.

- The server will create/update/read the database file. The database file has to be stored in encrypted format. You will use RSA to encrypt the user data. You will use PyCryptoDome's RSA implementation, the 'PKCS1_OAEP' [3] class in PyCryptodome. It is an asymmetric cipher based on the RSA algorithm that uses OAEP padding. When a user registers successfully, the database file will be updated with the new information.
- When the login operation is performed by the client, the server provides, first if the given user exists in the database, then if the credentials of the users are validated, and finally if the OTP token is validated.

Hash chain-based OTP

A hash chain OTP (One-Time Password) is a sequence of one-time passwords generated by repeatedly hashing an initial secret value to create a chain. Each password in the sequence can only be used once, adding an extra layer of security for authentication processes. The process is done by applying these steps:

- **Starting Value (Seed):** Start with a seed. The seed can be a random number. But for this assignment, we will use the password as the initial value.
- **Iterative Hashing:** Hash the starting value multiple times, n, to create a chain of hashes. The chain will look like this;

$$S_n = f(f(f(f(...f(S)...))))$$

[3] https://pycryptodome.readthedocs.io/en/latest/src/cipher/oaep.html

where S is the initial value and f is the hashing function. We will use SHA-256 for the hashing function. And for the n value, we will use 100.

- **Generate OTPs:** Each time a user logs in, an OTP token will be sent to the Server. The chain is used in reverse order. In other words, the last token in the chain S_n will be used first, then S_{n-1}, and finally S_1.
- **Verification Process:** When a user registers, the OTP chain is created by the client S_n is shared to the Server. The server stores the value as the OTP token for the user. When the user logs in, The Client sends the OTP token S_{n-1}. The server uses this token and checks if $S_n == f(S_{n-1})$. If validates the token, stores the new OTP token as $Sn - 1$ and increments the counter for the user.
- When the server reaches S_1 (the OTP chain is completed), the user creates a new chain. And all process starts again.

Database File

The database file is stored in a structured format with the name "database.txt". The format for the database file will be;

```
username1;hashedpassword;OTPToken;Counter
username2;hashedpassword;OTPToken;Counter
...
```

Note: The file will be stored as encrypted.

Important Notes

In this chapter, for the sake of clarification, some important notes about your implementation have been described.

- You will be provided with the *public.key* and *private.key* via the Piazza system.
- Your aim will be to design a project including these functionalities.
- Your code environment will be limited to 2 main files "Client.py" and "Server.py". Here the "Client.py" will be the front-end Flask app.
- The visual design of the pages will not be graded.
- You can only use the standard Python library, Flask, and PyCryptodome library. Do not use any additional libraries other than Python Standard Library. If you think that a certain library will be essential to the solution, please ask before using it.
- You must submit a detailed report to describe what you have done.

Notes

1. You should prepare a report involving your approach and details of the implementation you have coded. You must write down the names and ids of your teammates in the report in order to be evaluated correctly.

2. You must submit the homework in groups.
3. You should prepare a report that describes your approach to the problem with the details of your implementation. You must write down all group members' names and ids. Reports will be graded too.
4. You can ask your questions about the homework via Piazza. (`www.piazza.com/hacettepe.edu.tr/fall2024/bbm465`)
5. T.A. as himself has the right to partially change this document. However, the modifications will be announced in the Piazza system. In this case, it is your obligation to check the Piazza course page periodically.
6. You will submit your work via the submission system.
   (`submit.cs.hacettepe.edu.tr`)
   The submission format is given below:

→ <group id.zip>
→ src/
→ Client.py
→ Server.py
→ database.txt
→ public.pem
→ private.pem
→ templates/
→ login.html
→ register.html
→ welcome.html
→ report.pdf

Policy

All work on assignments must be done with your own group unless stated otherwise. You are encouraged to discuss with your classmates about the given assignments, but these discussions should be carried out in an abstract way. That is, discussions related to a particular solution to a specific problem (either in actual code or in pseudo-code) will not be tolerated. In short, turning in someone else's work(from the internet), in whole or in part, as your own will be considered a violation of academic integrity. Please note that the former condition also holds for the material found on the web as everything on the web has been written by someone else.