Hacettepe University

Computer Engineering Department

BBM465 Information Security Laboratory - 2024 Fall

Homework Assignment 3

November 27, 2024

Group ID: 28
Students name:
Mehmet Yiğit
Mehmet Oğuz Kocadere

Students Number:
b2210356159
b2210356021

1 Problem Definition

A user portal will be created where users can register and log in. The portal must securely manage user information and include a hash chain-based OTP (One-Time Password) verification step. The system will be designed with Flask to include a client (frontend) and a server (backend) configuration.

1. **User Registration** : Users can register with a valid username and a password longer than 6 characters. Usernames must not contain special characters or numbers. Passwords are hashed (using MD5 or SHA-256) before being sent to the server. A hash chain (100 iterations of SHA-256) is created by the client for OTP and sent to the server.

2. **User Database:** User data is stored in an encrypted database.txt file. Each user entry in the database is stored in the following format:

   username;hashedpassword;OTPToken;Counter

   The database is encrypted using RSA.

3. **User Login** : The client sends the username and hashed password to the server. If verification is successful, the OTP verification step is initiated. OTPs are validated using the hash chain method. (e.g., check if Sn == f(Sn-1)). Once the OTP is successfully verified, the user is redirected to a welcome page with the message:
   Welcome, username.

4. **Client:** Flask will be used to create the registration, login, and welcome screens. The PyCryptodome library will be used for encryption and hashing operations.

5. **Server**: The server will handle user verification and database management. The database will be accessible only by the server and encrypted using RSA.

6. **OTP Chain:** During registration, an OTP chain is generated, and the final value is sent to the server. The chain is updated after each successful login.

2 Client.py Implementations

This Flask application implements a user registration and login system with OTP (One-Time Password) authentication. Below is a detailed breakdown of the application's components and functionality:

2.1 Key Components

Libraries and Modules

Flask: Provides the web framework.

Flask, render_template, redirect, url_for, request, flash.

Server: A custom module (assumed to handle server-side logic like user registration and OTP validation).

Crypto.Hash: Used for hashing passwords and generating OTP chains.

MD5, SHA256.

re: For regular expression-based validation of usernames.

2.2 Functionality Overview

2.2.1 Password Hashing

The function **hash_password** hashes user passwords using either SHA256 (default) or MD5.

Inputs: password (str), algorithm (optional, default: SHA256).

Output: Hexadecimal hash of the password.

2.2.2 OTP Chain Generation

The function hash_otp generates a chain of OTPs using SHA256.

Inputs:

seed: The starting value for OTP generation.

n: Number of OTPs to generate (default: 100).

Output: A list of OTPs in reverse order for easier consumption.

2.2.3 Routes

/register (Registration)

Handles user registration. Supports GET and POST methods.

GET: Displays the registration form.
POST: Validates the username (only letters) and password (min. length: 6).

-Hashes the password using hash_password.

-Generates an OTP chain using hash_otp.

-Attempts to register the user via the Server module.

-On success: Redirects to the login page with a success message.

-On failure: Displays an error message.

/login (Login)

Handles user login. Supports GET and POST methods.

GET: Displays the login form.

POST:

-Verifies the username and hashed password via the Server module.

-Generates an OTP chain using hash_otp.

-Validates the OTP with the server.

-On success: Redirects to the welcome page.

-On failure: Displays an error message.

/welcome (Welcome Page)

Displays a welcome message after successful login.
GET: Renders the welcome.html template.

/ (Main Screen)

Redirects users to the login page.
GET: Redirects to /login.

2.2.4 Helper Functions

hash_password

Hashes the password based on the specified algorithm (SHA256 or MD5). It raises an error if an unsupported algorithm is provided.

hash_otp

Generates a sequence of OTPs using the SHA256 hash function.

Each OTP is derived by repeatedly hashing the previous hash value.

The final list is reversed for easier OTP retrieval.

2.2.5 Application Workflow

1. **Registration**
   - Users enter a username and password.

- Username is validated for alphabetic characters only.
- Password is validated for a minimum length of 6 characters.
- The server stores the hashed password and the first OTP in the chain.

2. **Login**

- Users enter their username and password.
- Password is hashed and verified with the server.
- An OTP is generated from the chain based on the user's OTP counter.
- The OTP is validated by the server.
- On success, the user is redirected to the welcome page.

3. **OTP Chain Management**

- OTPs are generated in advance and stored in reverse order.
- A global `index` variable tracks the user's current OTP in the chain.
- When the index reaches the chain's end, it resets.

2.2.6 Security Considerations

1.**Password Hashing**:

Use of SHA256 ensures secure storage of user passwords. MD5 is supported but not recommended due to vulnerabilities.

2. **OTP Validation**:

OTPs are generated from the hashed password for added security. Server-side validation ensures robust authentication.

3. **Flask Secret Key**: Used for session management and flash messages.

2.2.7 Running the Application

To run the application, use the following command:

```
// python Client.py
```

The application will be accessible at http://0.0.0.0:5000.

3 Server.py Implementation

The Server class is a secure backend implementation designed for managing user authentication and database operations using RSA encryption and OTP (One-Time Password) mechanisms. It supports functionality to encrypt and decrypt sensitive data with RSA public and private keys, ensuring confidentiality of user information. The class includes methods for loading and saving a database file with encrypted records, allowing secure storage of user credentials. It also provides a user registration method to add new users, along with OTP-based mechanisms for authentication and verification. Additionally, hashed OTPs are generated and validated to enhance security during login processes, ensuring only legitimate users can access the system. This implementation is designed to protect sensitive information while maintaining robust user management and authentication.

3.1 Key Components

Crypto.PublicKey.RSA and **Crypto.Cipher.PKCS1_OAEP**: Used for RSA key management and encryption decryption operations.
Crypto.Hash.SHA256: Ensures secure hashing for OTP and password operations

3.2 Server Attributes

database_path: Path to the encrypted database file (database.txt).
public_key: RSA public key for encrypting data.
private_key: RSA private key for decrypting data.
index: Tracks the current OTP chain position.
register_otp: Stores the initial OTP seed for new registrations.
otp_mod: OTP modulus to reset the chain after reaching the end.

3.3 Functionality Overview

3.3.1 Database Management

load_database()

Purpose: Decrypt and load user data from the encrypted database file.
Workflow:
Reads encrypted data line by line.
Decrypts each line using RSA private key.
Parses decrypted lines into a structured list.
Output: List of user entries [username, hashed_password, OTP_token, Counter].
save_database()

Purpose: Encrypt and save user data securely to the database file.
Workflow:
Converts user data to a semicolon-separated string.
Encrypts each line using RSA public key.
Writes the encrypted data with its size to the file.

3.3.2 User Management

register_user()

Purpose: Register a new user.
Input: username: Must be unique. hashed_password: Securely hashed password. otp_token: Initial OTP token derived from the OTP chain.
Output: Returns True if registration succeeds, otherwise False.

verify_login()

Purpose: Authenticate a user using their username and hashed password.
Input: username: The user's registered username.
hashed_password: The user's hashed password.
Output: Returns True if credentials match a database entry, otherwise False.

3.3.3 OTP Verification

register_user()
Purpose: Register a new user.
Input:
username: Must be unique.
hashed_password: Securely hashed password.
otp_token: Initial OTP token derived from the OTP chain.
Output: Returns True if registration succeeds, otherwise False.

verify_login()

Purpose: Authenticate a user using their username and hashed password.
Input:
username: The user's registered username.
hashed_password: The user's hashed password.
Output: Returns True if credentials match a database entry, otherwise False.

3.3.4 Encryption and Decryption

validate_otp()

Purpose: Validate the OTP submitted by a user during login.

Input:
username: The user's username.
client_otp: OTP submitted by the client.
Workflow:
Validates the current OTP against the server's record.
Updates the OTP chain and counter upon successful validation.
Output: Returns True if OTP is valid, otherwise False.

hash_one_time_otp()

Purpose: Hash a single OTP value using SHA256.
Input: client_otp in hexadecimal format.
Output: Returns the hashed OTP in hexadecimal format.

3.4 Application Workflow

3.4.1 Registration

User provides a valid username and password.
Password is hashed using SHA256.
An OTP chain is generated, and the last OTP is stored as the OTP_token.
Data is encrypted and saved in the database.

3.4.2 Login

User submits username and hashed password.
Credentials are verified against the database.
User submits an OTP derived from their local chain.
The server validates the OTP by comparing the hash of the client OTP with the stored OTP.
Upon success, the user is authenticated and redirected.

3.4.3 OTP Chain Management

The OTP chain resets when the index reaches otp_mod.
Each OTP is generated by hashing the previous value.

3.5 Security Considerations

3.5.1 Password Security:

SHA256 is used for hashing passwords.
MD5, while supported, is not used.

3.5.2 OTP Validation:

Securely derived using SHA256 hash chain.
Server-side validation ensures the integrity of the authentication process.

3.5.3 Database Encryption:

RSA encryption secures user data in transit and at rest.

3.5.4 Data Access:

Encrypted database accessible only by the server.

4 Summary

This system ensures:

Confidentiality: Through RSA-encrypted databases.
Integrity: OTP validation with secure hash chains.
User Authentication: Robust mechanisms for verifying credentials and OTPs.
Modularity: Separation of frontend and backend with Flask and custom server logic.

The design integrates modern cryptographic standards, ensuring secure and efficient user authentication in a web-based environment.

References

- BBM465 Lecture 4 Hash Functions.
- BBM465 Lecture 5 Number Theory.
- BBM465 Lecture 6 Authentication.
- BBM465 Lecture 6 Authentication.
- https://www.geeksforgeeks.org/rsa-algorithm-cryptography/