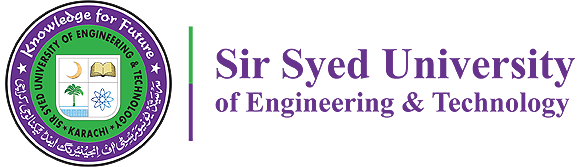
**LAB PROJECT REPORT**

****

**CE-408L: Cryptography & Network Security**

**AES-OTP-Auth APPLICATION**

**SECTION: B**

**Roll No:** 2021F-BCE-199, **Name:** Muhammad Khalil Ashraf

**Roll No:** 2021F-BCE-065, **Name:** Inshara Ahmed

**Roll No:** 2021F-BCE-013, **Name:** Ramsha Aftab

**Roll No:** 2021F-BCE-244, **Name:** Muhammad Huzaifa

Department of Computer Engineering

Sir Syed University of Engineering & Technology

Karachi Pakistan.

Department: **Computer Engineering** Program: **B.S (CE)**

**Lab Project Report**

**CE-408L Cryptography and Network Security**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | | Total Marks = 15 |
|  |  | Marks Obtained = | |

Teacher Name**: Dr. Rukaiya**

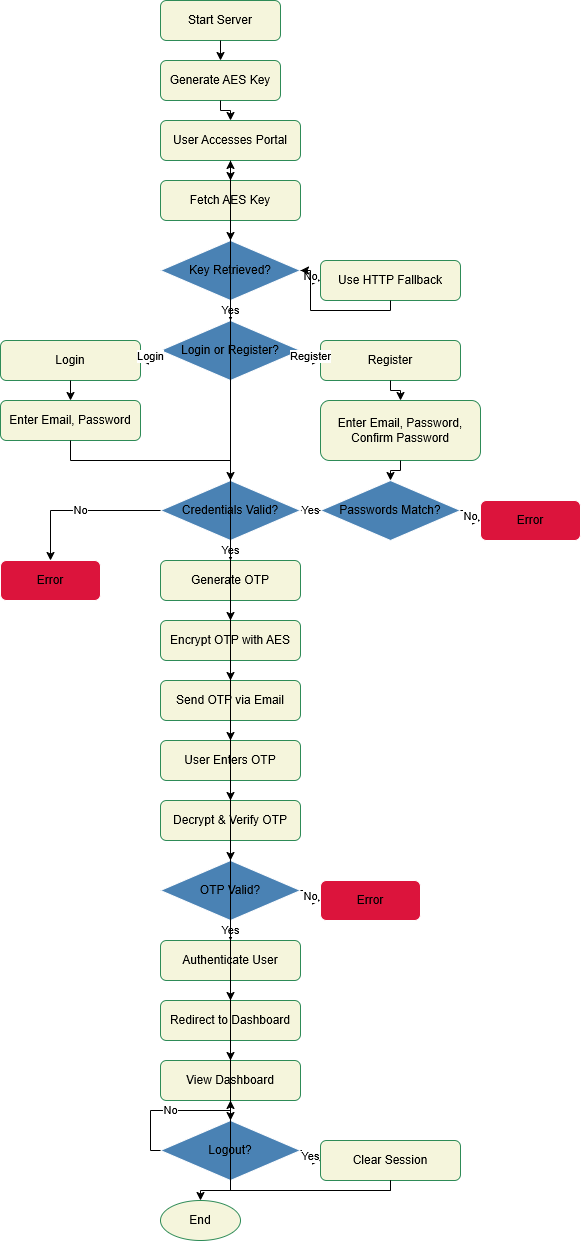
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Complex Engineering Activity** | | | | |
| **Sr. No** | **Course Learning Outcomes** | **Blooms Taxonomy** | **PLOs** | **Complex Problem Solving** |
| CLO\_1 | **Use** appropriate tools to apply different cryptographic algorithms  and network security | **C3**  (Applying) | **PLO\_3**  (Design/Development of Solutions) | WP1: Depth of knowledge required.  WP3: Depth of analysis required. |
|  | mechanisms. |  |  | WK8: Research |
|  |  |  |  |
| CLO\_2 | **Explain** emerging | **A4** | **PLO\_10** | Literature. |
|  | security issues, practices  or applications. | (Organization) | (Communication) | WA4: Investigation |

**OBJECTIVE:**

Design a two-factor authentication (2FA) mechanism where the user enter both a password and a code to gain access. The system validates both inputs using Cosine Similarity, allowing authentication if the inputs are sufficiently close to the stored values. Passwords must be securely hashed and stored in a file.

**Description of the Project:**

AES-OTP-Auth is a secure web-based authentication system featuring a Flask backend and Streamlit frontend, designed to provide robust user access control through two-factor authentication (2FA) with AES-128 encrypted One-Time Passwords (OTPs). It supports user registration, login, and a protected dashboard displaying organizational details, such as academic programs and research initiatives, within a custom-styled, user-friendly interface. The system employs a socket-based key exchange with an HTTP fallback for secure AES key distribution, filesystem-based session management, and resilient error handling to ensure reliable operation. While HTTP usage poses potential man-in-the-middle risks, AES-OTP-Auth demonstrates strong cryptographic practices and serves as an educational tool for secure software development, with opportunities for future enhancements like HTTPS and key rotation to strengthen security.

**Flow chart**

**Configurations/Commands Used/Pseudocode (mention all libraries used)**

The **AES-OTP-Auth** project requires specific configurations to ensure secure operation and compatibility across the client and server components. Below are the key configurations:

* **Python Environment**:
  + Python version: 3.13 or higher.
  + Virtual environment: Created using python -m venv .venv to isolate dependencies.
  + Dependencies listed in requirements.txt:
    - flask==2.3.3
    - flask-session==0.5.0
    - streamlit==1.38.0
    - requests==2.31.0
* **Flask Backend (server.py)**:
  + **Session Management**: Configured with Flask-Session using filesystem storage
  + Secret Key: Set for secure session signing.
  + Logging: Configured for debugging with timestamped logs
  + Streamlit Frontend (client.py, home.py):
  + Email Service (utils/email\_utils.py)
  + Encryption (utils/encryption\_utils.py):
  + **File Storage**:
    - User credentials stored in user\_data.txt (email, hashed password).
    - Session data stored in flask\_session/ directory.
* The following commands are used to set up, run, and manage the **AES-OTP-Auth** system:
  + **Setup Virtual Environment**:
    - python -m venv .venv
    - .venv\Scripts\activate # Windows
    - source .venv/bin/activate # Linux/macOS
  + Install Dependencies:
    - pip install -r requirements.txt
  + Run Flask Server:
    - python server.py
      * Starts the Flask API and socket server for key exchange.
      * Logs initialization and key generation.
  + Run Streamlit Frontend:
    - streamlit run client.py
      * Launches the portal at <http://localhost:8501>.
      * Logs key retrieval and authentication attempts.
* **Pseudocode**

// Server (server.py)

FUNCTION StartServer

GENERATE 16-byte AES key

INITIALIZE Flask app with session config

START socket server in separate thread

LOG "Server started"

WHILE running

HANDLE HTTP requests (/register, /login, /verify, /key, /logout, /shutdown)

ENDWHILE

END FUNCTION

FUNCTION Register(email, password, confirm\_password)

IF password != confirm\_password THEN

RETURN error "Passwords do not match"

ENDIF

HASH password

STORE email, hashed\_password in user\_data.txt

GENERATE OTP

ENCRYPT OTP with AES key (EAX mode)

SEND encrypted OTP via email

RETURN success "OTP sent"

END FUNCTION

FUNCTION Login(email, password)

READ user\_data.txt

IF email not found OR password hash mismatch THEN

RETURN error "Invalid credentials"

ENDIF

GENERATE OTP

ENCRYPT OTP with AES key (EAX mode)

SEND encrypted OTP via email

RETURN success "OTP sent"

END FUNCTION

FUNCTION VerifyOTP(email, otp, type)

DECRYPT OTP with AES key

IF decrypted OTP matches stored OTP AND type is valid THEN

CREATE session for email

RETURN success "Authenticated"

ELSE

RETURN error "Invalid OTP"

ENDIF

END FUNCTION

FUNCTION SendKey

IF socket connection THEN

SEND AES key via socket

ELSE

SEND AES key via HTTP (/key)

ENDIF

LOG "Key sent"

END FUNCTION

FUNCTION Logout

CLEAR session

RETURN success "Logged out"

END FUNCTION

// Client (client.py)

FUNCTION Main

INITIALIZE Streamlit session state

DISPLAY login/register form

TRY

REQUEST AES key from server (socket or HTTP with retries)

STORE key in session state

CATCH failure

DISPLAY error "Key retrieval failed"

RETRY with HTTP fallback

ENDTRY

IF user selects "Register" THEN

CALL RegisterClient

ELSE IF user selects "Login" THEN

CALL LoginClient

ENDIF

END FUNCTION

FUNCTION RegisterClient

INPUT email, password, confirm\_password

SEND POST /register with email, password, confirm\_password

IF response is success THEN

INPUT OTP from user

DECRYPT OTP with AES key

SEND POST /verify\_registration with email, OTP

IF verification success THEN

SET session state authenticated = true

REDIRECT to dashboard

ELSE

DISPLAY error "Invalid OTP"

ENDIF

ELSE

DISPLAY error from response

ENDIF

END FUNCTION

FUNCTION LoginClient

INPUT email, password

SEND POST /login with email, password

IF response is success THEN

INPUT OTP from user

DECRYPT OTP with AES key

SEND POST /verify\_login with email, OTP

IF verification success THEN

SET session state authenticated = true

REDIRECT to dashboard

ELSE

DISPLAY error "Invalid OTP"

ENDIF

ELSE

DISPLAY error from response

ENDIF

END FUNCTION

// Dashboard (home.py)

FUNCTION Dashboard

IF session state authenticated THEN

DISPLAY dashboard with organizational details

IF user clicks "Logout" THEN

SEND POST /logout

CLEAR session state

REDIRECT to login page

ENDIF

ELSE

DISPLAY error "Unauthorized access"

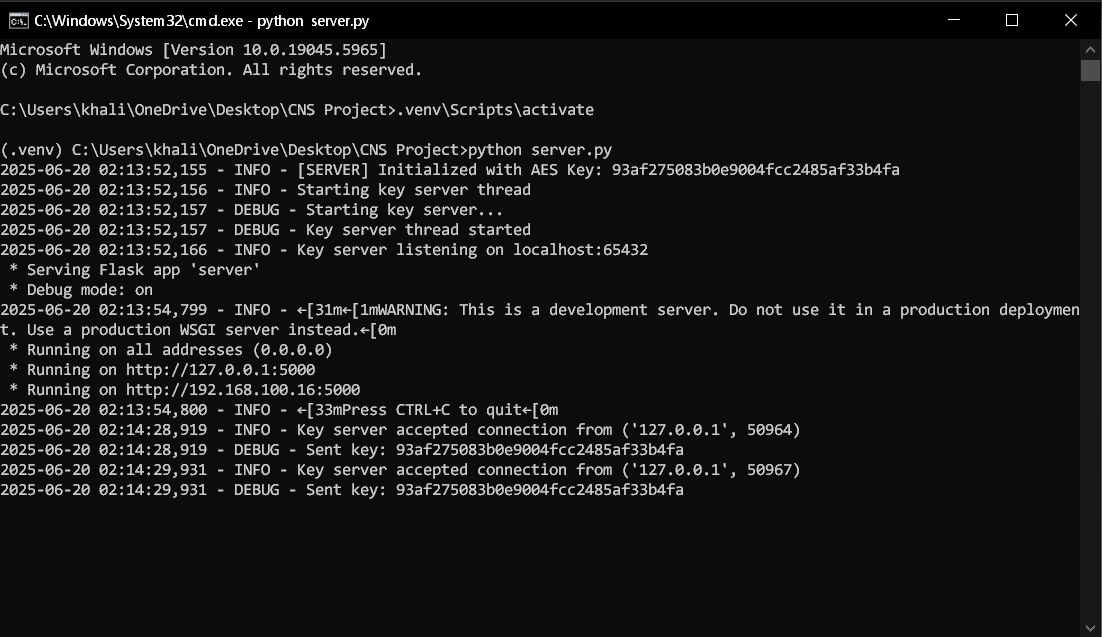
REDIRECT to login page

ENDIF

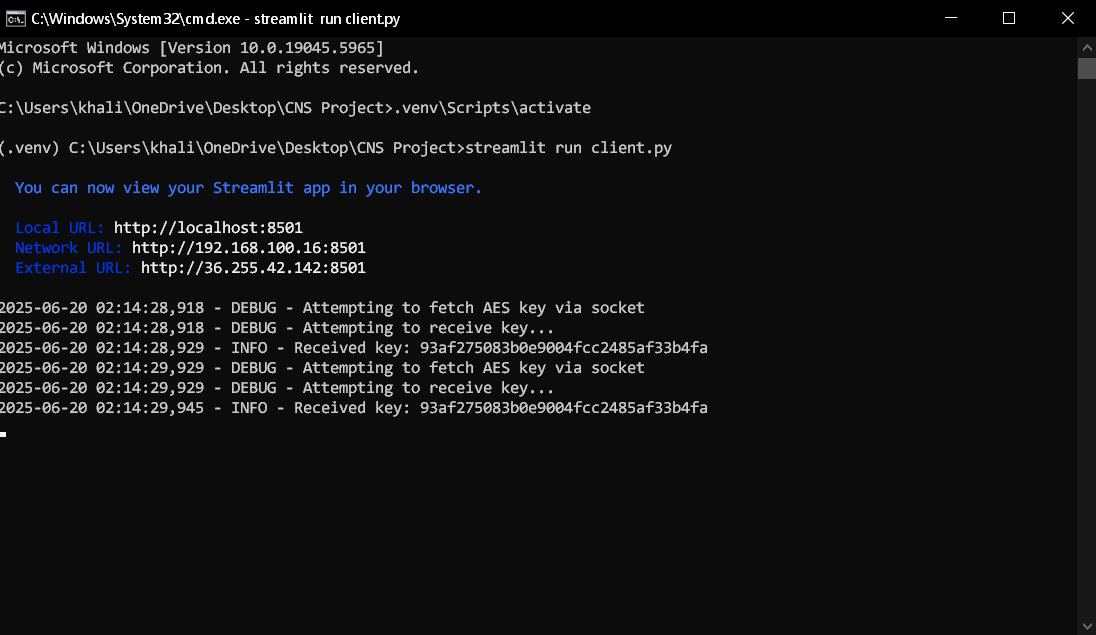
END FUNCTION

**Results and Discussion (output Screenshot with details, datasets, and analysis)**

**Server.py**

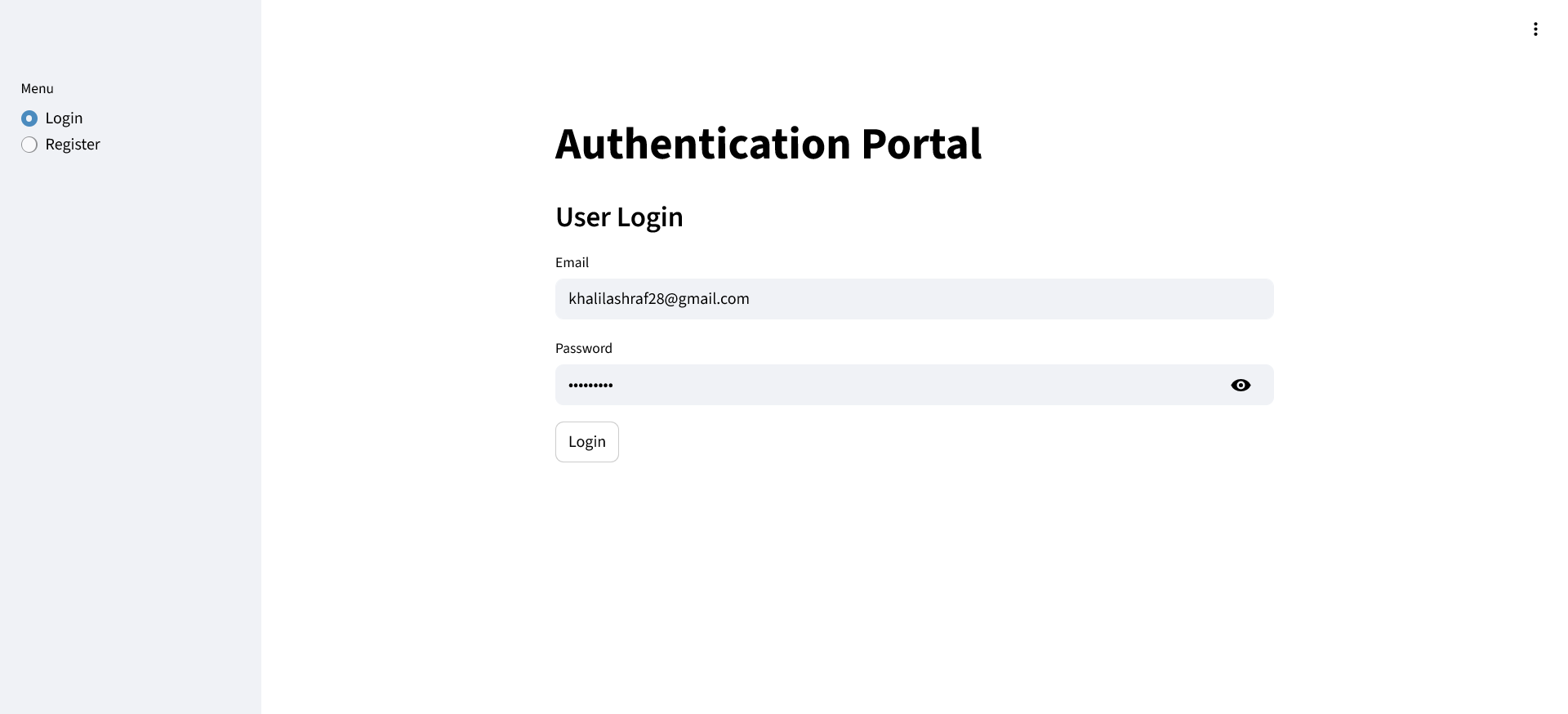


**Client.py**

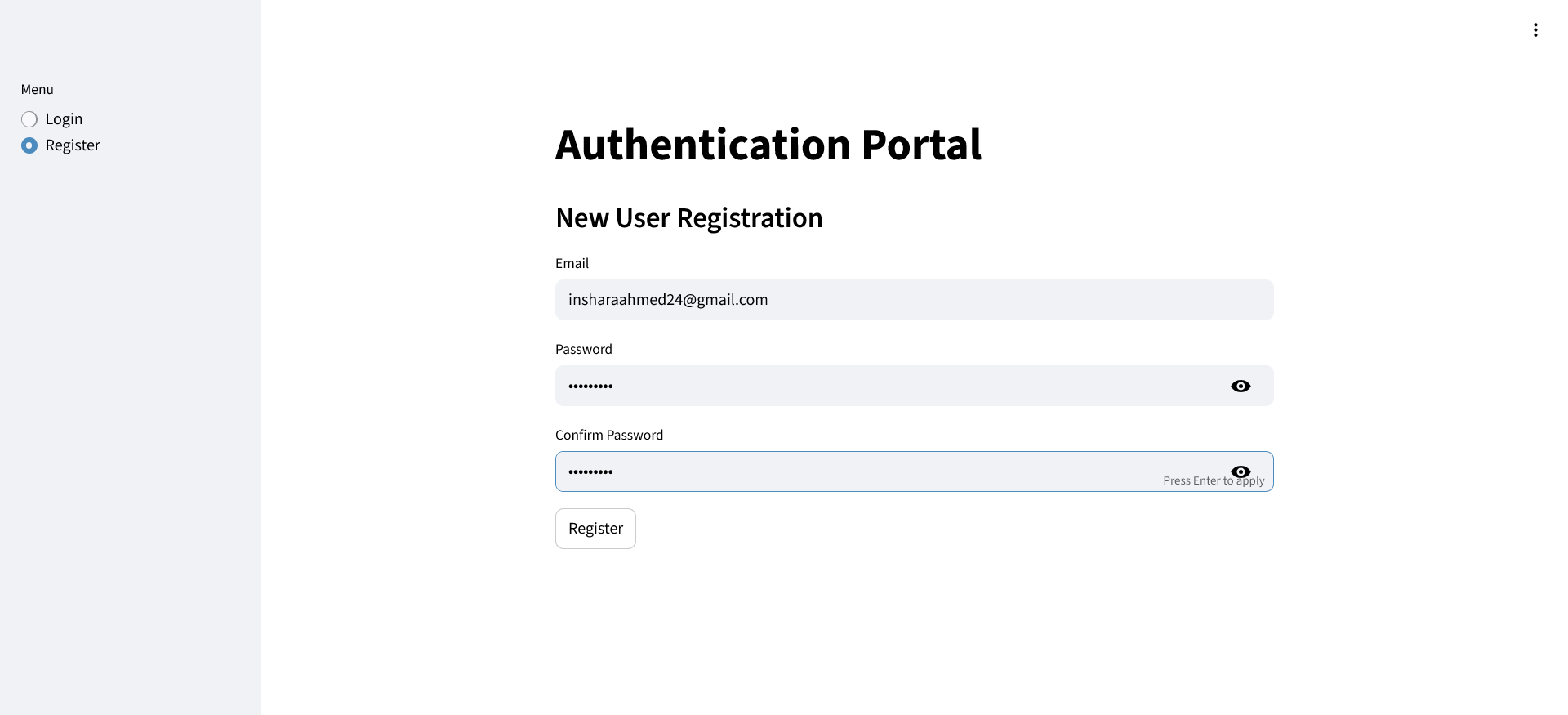


After run the program we will show the above window.

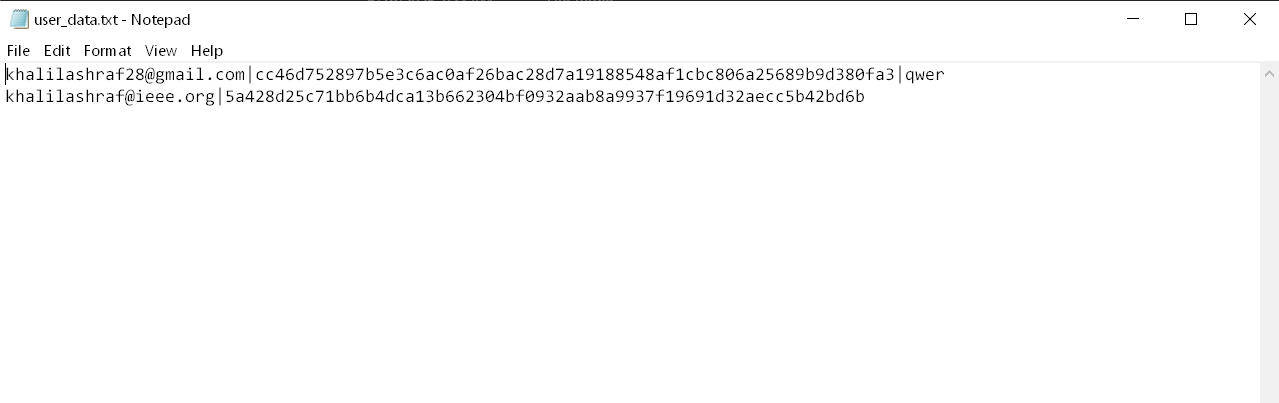
**Login Portal**



**Registration Portal**



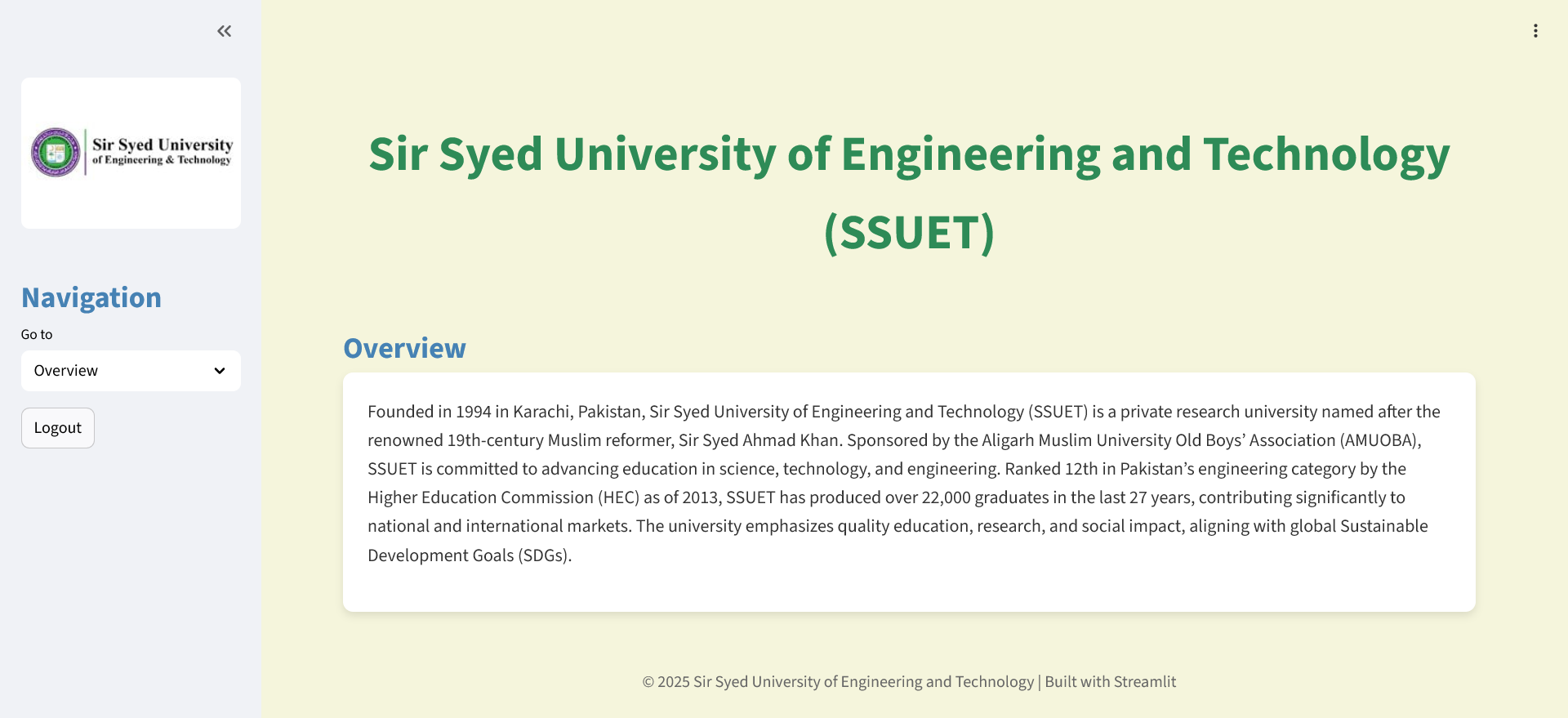
User Data save in Txt file format in Hashed Form



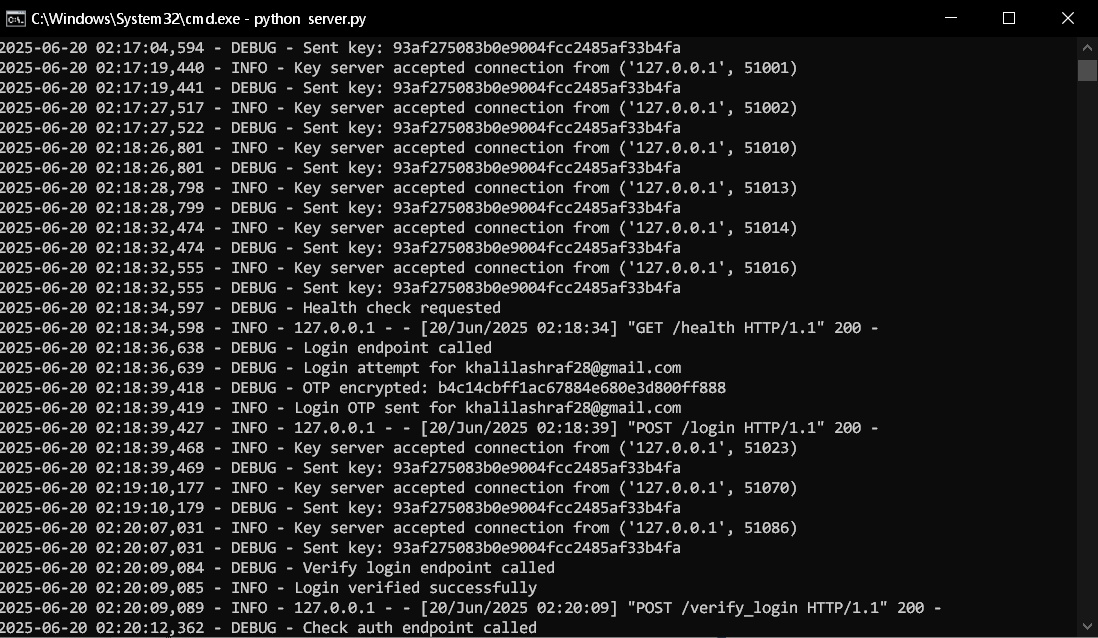
OTP Received on email via Gmail for 2 Factor Authentication System

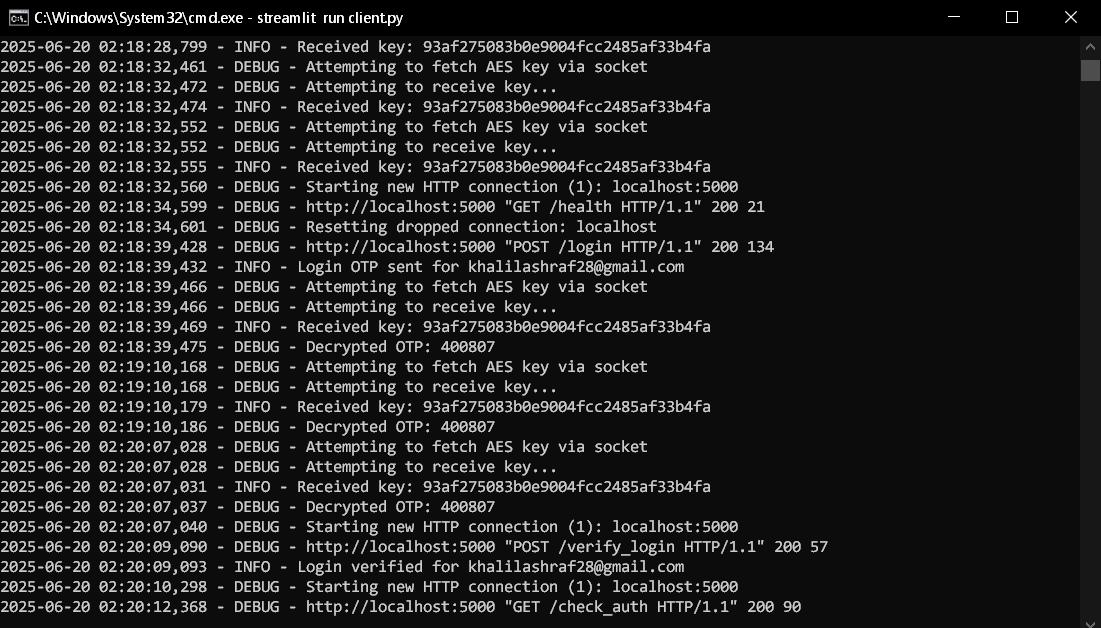


**Home.py**



Communication between Server.py and Client.py





**Conclusions:**

The AES-OTP-Auth project successfully delivers a secure, web-based authentication system, integrating two-factor authentication with AES-128 encrypted OTPs, a Flask backend, and a Streamlit frontend. By implementing robust key exchange, session management, and error handling, it provides a reliable and user-friendly portal for accessing protected resources like the dashboard. Despite limitations, such as HTTP vulnerabilities, the project demonstrates effective cryptographic practices and serves as an educational tool for secure software development. Future enhancements, including HTTPS adoption and key rotation, could further strengthen its security, making AES-OTP-Auth a valuable foundation for scalable authentication solutions.

**References:**

[1] R. Armin, Flask: Web development, one drop at a time. Flask Documentation, 2023. [Online]. Available: <https://flask.palletsprojects.com/en/2.3.x/>

[2] Streamlit Inc., Streamlit: A faster way to build and share data apps. Streamlit Documentation, 2024. [Online]. Available: <https://docs.streamlit.io/>

[3] Python Software Foundation, Python Standard Library. Python Documentation, 2025. [Online]. Available: <https://docs.python.org/3/library/>