Cryptographically-Secure File Server ***by 240027249***

Starting It All Up

I am really sorry I couldn't figure out a way to start everything up in a simpler manner. Please navigate to the cs4203-p02 directory and run the following command -

pip3 install -r requirements.txt

This will set up all your dependencies with one command. VSCode may prompt you to set up virtual environment first, it is recommended in case you aren't using lab machines. I am not using any library that is relatively unknown.

To start the server, navigate to the **server** directory and run ***python3 server.py***
To start the client, navigate to the **client** directory and run ***python3 client.py***

Client should be started after the server (as you'd ofcourse know.)

I have made the client terminal intuitive enough for the interaction, please test it as you see fit (might have unhandled exceptions that have nothing to do with security side.)

Overview of Zero Knowledge Server

This is a python flask server. It is zero knowledge not in the sense of not storing any information, but that any information stored is efficiently encrypted and the server doesn't see any vital information even in the communication channels as often as possible. This is achieved via the following -

A. Public key storage – All users receive an asymmetric RSA public private key pair at registration. The private key is encrypted with the password chosen by the user during registration, meanwhile the public key is stored directly on the server. Public Key storage isn't a violation of zero knowledge as per me.

B. Any file uploaded is encrypted using Fernet (AES-128-CBC) on the client side itself. There are obviously much stronger cryptos available out there, but I chose Fernet because of its ease of use and the fact that it is good enough for commercial usage. The symmetric key generated is encrypted with the public key of each user who has permission to read the file. Storing symmetric keys with public key encryption such that the symmetric key can only be accessed by the users it is meant for with their password encrypted private key on client side isn't a violation of zero knowledge (as per me, again.) The server NEVER sees any of this data.

C. Use of TLS – The whole application is hosted on HTTPS. Obviously I couldn't register this app with an actual certification authority, so I created a CA that then issued the SSL certificates for my server. I have incorporated this CA.pem file on client side to mimic the actual HTTPS communication and how certificates interact.

D. Password – The only time server sees the password at all is during registration of the user. Once it sees the password, it immediately encrypts it using SHA-256 password hashing and stores it on the server. To ensure the server never sees the password after this once, we use the next point. Also, the getpass method is used to make sure password isn't visible while being typed.

E. Challenge-Response – Even with TLS, I didn't want sensitive information to be relayed on an open communication channel at all. So I used challenge response methodology such that password never really comes to the server. Instead, server sends a challenge that is worked upon in the same manner with the same local encryption by the client.

F. JWT and Authentication Decorator – JWT is used to authenticate sessions in the decorator. It is assigned only after successful login from client side, with an HS256 encryption with our server side secret key. This is how user logins are managed, and how upload or download options are verified in any given session.

G. 2FA – 2FA is enabled for all accounts during registration and logins.

H. All data encryption – Everything that touches the database is encrypted before, if not already encrypted. The username, for example, is deterministically encrypted before being stored in the database to ensure at rest security.

I. Coding Practices – User input has been sanitised wherever deemed necessary, and no sensitive information has been hardcoded at all.

J. Database – I have used SQLAlchemy which already protects against SQL injections. There are three tables – User, File and File Access. None of this data is painfully sensitive, as discussed. The only thing database is being primarily used for is Access Control and Support in Authentication.

Application Flow

Register

1. Email is used as username. Sanitised with the is_valid_email function.
2. Password is validated.
3. 2FA is triggered and verified on server side (this abstraction makes it more secure in my opinion)
4. Private key and public key pairs are generated. Private key is password encrypted. Username, Public key and password are sent to the server.
5. Server immediately encrypts the password and the username, and stores all of this information in the user table.

Login

1. Email is validated.
2. Email Authentication is triggered.
3. Password is stored locally on client side.
4. Challenge is requested from the server.
5. Response is calculated using the challenge and password encryption
6. Server checks this against its own work on server side, and allows access.

Upload

1. Can only be accessed after login has been successful.
2. Path of the file that you want to upload is needed (Please copy path of untitled.txt in the client directory)
3. List of usernames is asked who this file needs to be shared with (All usernames are email based and hence unique. This is ensured in the registration process.)
4. File is encrypted using the symmetric key.
5. Public keys of all the mentioned users are fetched.
6. Symmetric key is encrypted for each of them using their public keys.
7. Encrypted file, encrypted symmetric key and other non-sensitive metadata is sent to server.
8. Logged in user's own access to the file is configured in case they haven't specified their email in the list of emails asked. THUS, THE PERSON UPLOADING ALWAYS HAS ACCESS TO THEIR FILE.
9. Server stores the already encrypted data appropriately for access control in file and file access tables.
10. File is specifically stored in a different directory, not in the DB. DB only has the path.
11. File ID is shared with the user for further download use.

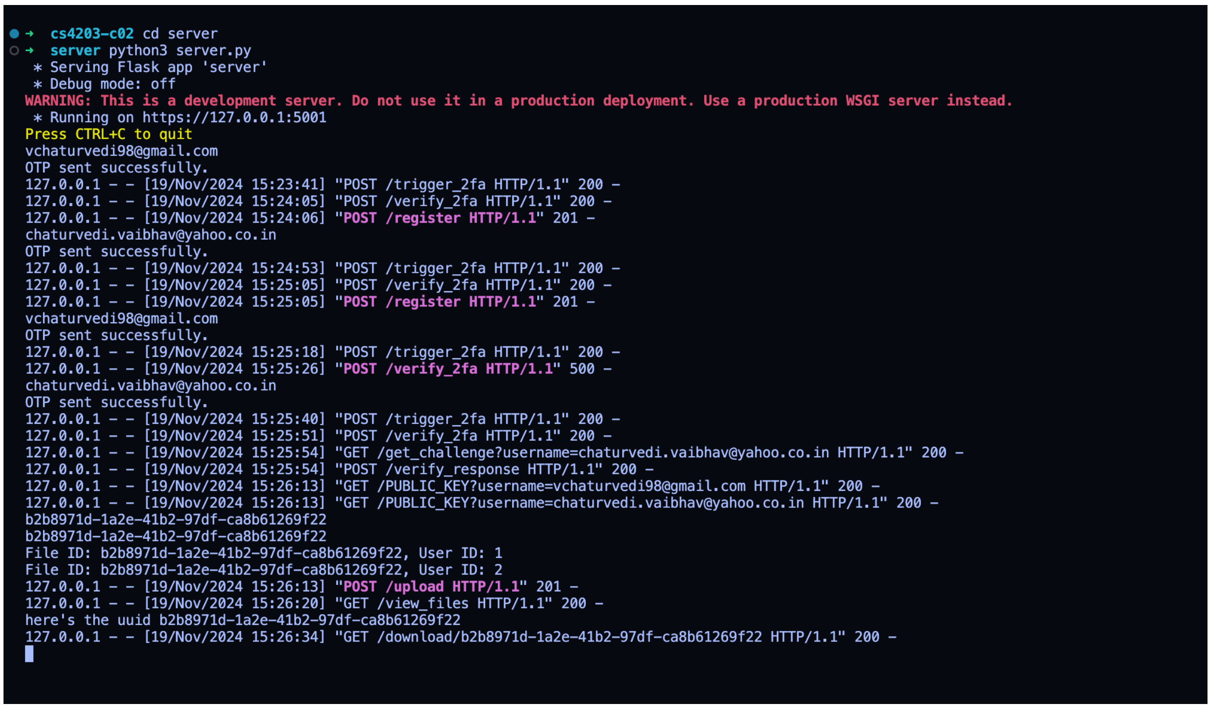
View Files

1. Session information is used to access the logged in username.
2. Based on file access table, list of file IDs that have been shared with you are shown.

Download

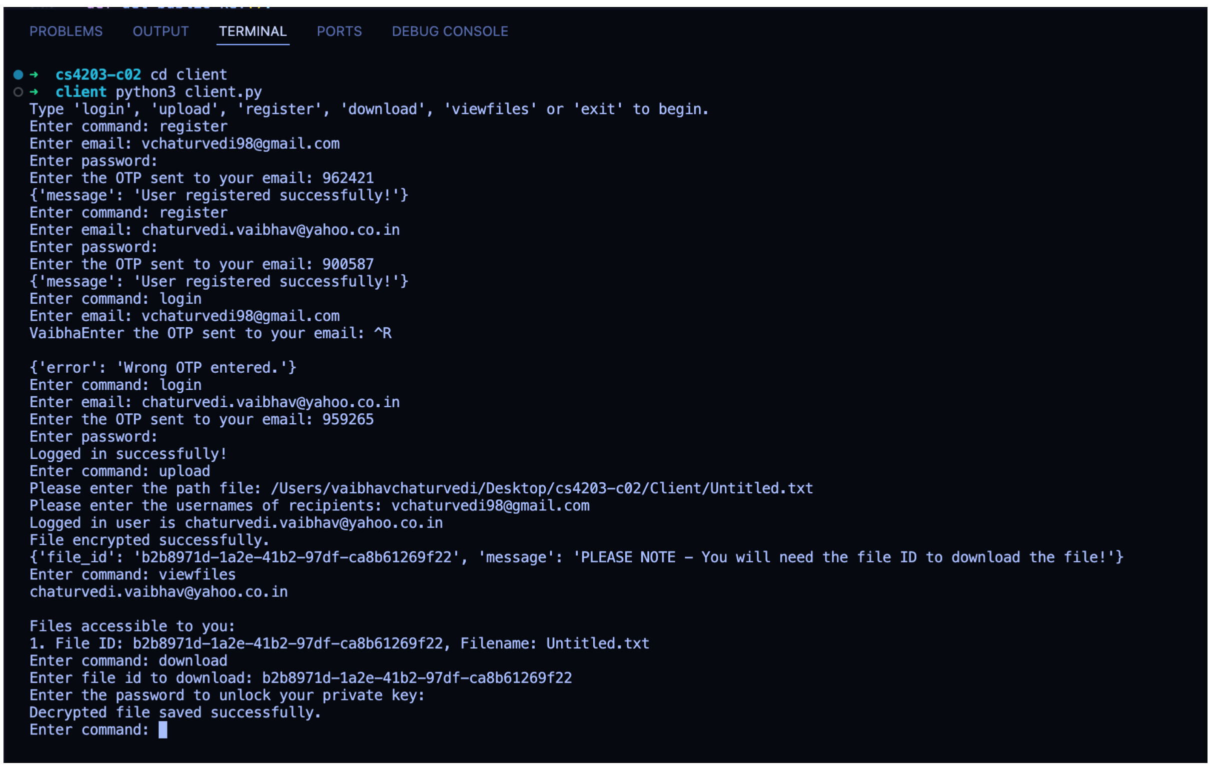
1. Access is checked in the server from the file access table.
2. If access is there, the encrypted file and the encrypted symmetric key are shared to the client.
3. Client's private key is used to decrypt the symmetric key encryption.
4. Once the symmetric key is obtained, it is used to decrypt the file.
5. File content is saved in downloads directory.

Evidence of Work



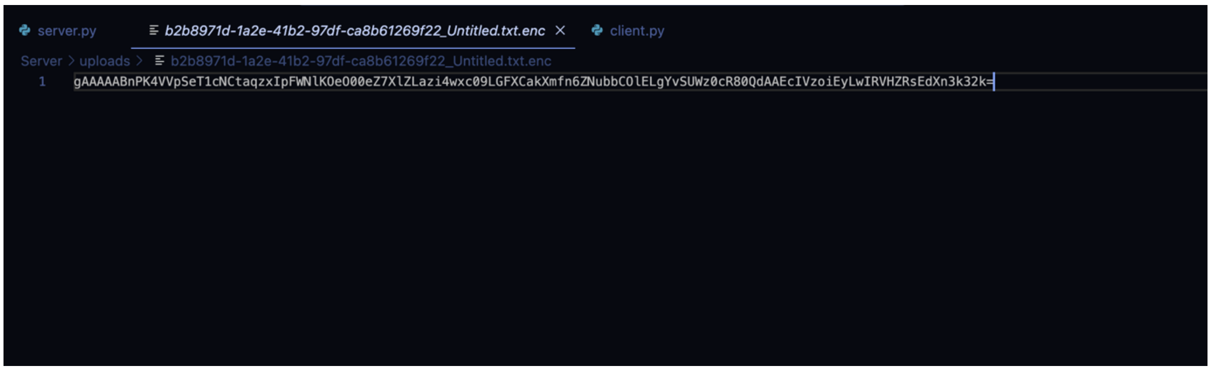
```
➜  cs4203-c02 cd server
➜  server python3 server.py
 * Serving Flask app 'server'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on https://127.0.0.1:5001
Press CTRL+C to quit
vchaturvedi98@gmail.com
OTP sent successfully.
127.0.0.1 - - [19/Nov/2024 15:23:41] "POST /trigger_2fa HTTP/1.1" 200 -
127.0.0.1 - - [19/Nov/2024 15:24:05] "POST /verify_2fa HTTP/1.1" 200 -
127.0.0.1 - - [19/Nov/2024 15:24:06] "POST /register HTTP/1.1" 201 -
chaturvedi.vaibhav@yahoo.co.in
OTP sent successfully.
127.0.0.1 - - [19/Nov/2024 15:24:53] "POST /trigger_2fa HTTP/1.1" 200 -
127.0.0.1 - - [19/Nov/2024 15:25:05] "POST /verify_2fa HTTP/1.1" 200 -
127.0.0.1 - - [19/Nov/2024 15:25:05] "POST /register HTTP/1.1" 201 -
vchaturvedi98@gmail.com
OTP sent successfully.
127.0.0.1 - - [19/Nov/2024 15:25:18] "POST /trigger_2fa HTTP/1.1" 200 -
127.0.0.1 - - [19/Nov/2024 15:25:26] "POST /verify_2fa HTTP/1.1" 500 -
chaturvedi.vaibhav@yahoo.co.in
OTP sent successfully.
127.0.0.1 - - [19/Nov/2024 15:25:40] "POST /trigger_2fa HTTP/1.1" 200 -
127.0.0.1 - - [19/Nov/2024 15:25:51] "POST /verify_2fa HTTP/1.1" 200 -
127.0.0.1 - - [19/Nov/2024 15:25:54] "GET /get_challenge?username=chaturvedi.vaibhav@yahoo.co.in HTTP/1.1" 200 -
127.0.0.1 - - [19/Nov/2024 15:25:54] "POST /verify_response HTTP/1.1" 200 -
127.0.0.1 - - [19/Nov/2024 15:26:13] "GET /PUBLIC_KEY?username=vchaturvedi98@gmail.com HTTP/1.1" 200 -
127.0.0.1 - - [19/Nov/2024 15:26:13] "GET /PUBLIC_KEY?username=chaturvedi.vaibhav@yahoo.co.in HTTP/1.1" 200 -
b2b8971d-1a2e-41b2-97df-ca8b61269f22
b2b8971d-1a2e-41b2-97df-ca8b61269f22
File ID: b2b8971d-1a2e-41b2-97df-ca8b61269f22, User ID: 1
File ID: b2b8971d-1a2e-41b2-97df-ca8b61269f22, User ID: 2
127.0.0.1 - - [19/Nov/2024 15:26:13] "POST /upload HTTP/1.1" 201 -
127.0.0.1 - - [19/Nov/2024 15:26:20] "GET /view_files HTTP/1.1" 200 -
here's the uuid b2b8971d-1a2e-41b2-97df-ca8b61269f22
127.0.0.1 - - [19/Nov/2024 15:26:34] "GET /download/b2b8971d-1a2e-41b2-97df-ca8b61269f22 HTTP/1.1" 200 -
```

server.py logs



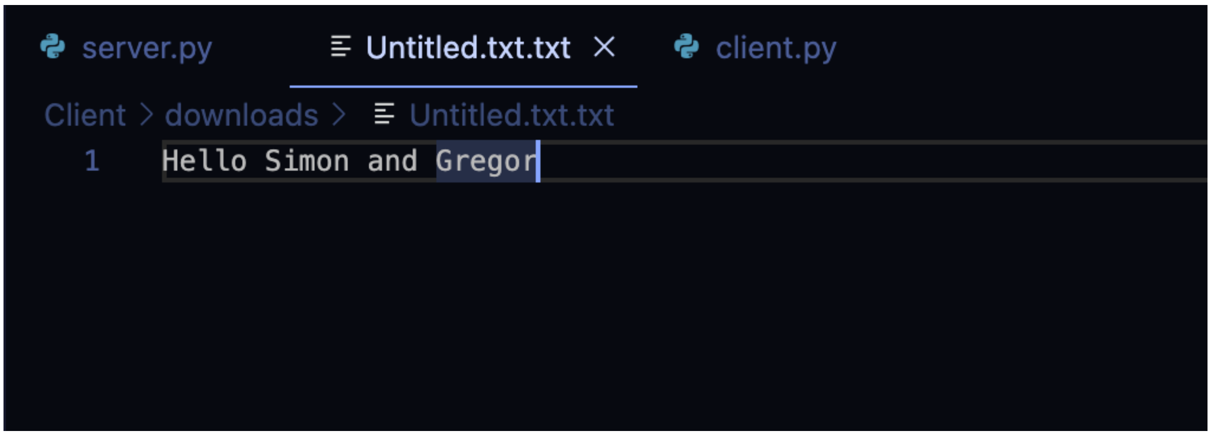
```
➜  cs4203-c02 cd client
➜  client python3 client.py
Type 'login', 'upload', 'register', 'download', 'viewfiles' or 'exit' to begin.
Enter command: register
Enter email: vchaturvedi98@gmail.com
Enter password:
Enter the OTP sent to your email: 962421
{'message': 'User registered successfully!'}
Enter command: register
Enter email: chaturvedi.vaibhav@yahoo.co.in
Enter password:
Enter the OTP sent to your email: 900587
{'message': 'User registered successfully!'}
Enter command: login
Enter email: vchaturvedi98@gmail.com
VaibhaEnter the OTP sent to your email: ^R

{'error': 'Wrong OTP entered.'}
Enter command: login
Enter email: chaturvedi.vaibhav@yahoo.co.in
Enter the OTP sent to your email: 959265
Enter password:
Logged in successfully!
Enter command: upload
Please enter the path file: /Users/vaibhavchaturvedi/Desktop/cs4203-c02/Client/Untitled.txt
Please enter the usernames of recipients: vchaturvedi98@gmail.com
Logged in user is chaturvedi.vaibhav@yahoo.co.in
File encrypted successfully.
{'file_id': 'b2b8971d-1a2e-41b2-97df-ca8b61269f22', 'message': 'PLEASE NOTE - You will need the file ID to download the file!'}
Enter command: viewfiles
chaturvedi.vaibhav@yahoo.co.in

Files accessible to you:
1. File ID: b2b8971d-1a2e-41b2-97df-ca8b61269f22, Filename: Untitled.txt
Enter command: download
Enter file id to download: b2b8971d-1a2e-41b2-97df-ca8b61269f22
Enter the password to unlock your private key:
Decrypted file saved successfully.
Enter command:
```

client.py logs


server.py
b2b8971d-1a2e-41b2-97df-ca8b61269f22_Untitled.txt.enc
client.py
Server > uploads > b2b8971d-1a2e-41b2-97df-ca8b61269f22_Untitled.txt.enc

```
gAAAAABnPK4VVpSeT1cNCtaqzxIpFWNlKOeO00eZ7XlZLazi4wxc09LGFXCakXmfn6ZNubbCOlELgYvSUWz0cR80QdAAEcIVzoiEyLwIRVHZRsEdXn3k32k=
```

Encrypted file in the upload directory of the server



```
Hello Simon and Gregor
```

Downloaded file from the server on the client side