**SecureCrypt: Secure File Sharing Using FTP and Hybrid Cryptography Instructions**

FTP Server Setup

1. **Start FTP Server:**
   * Navigate to the main directory: **secure-shared-file-storage**.
   * Run the server command:

**python -m python\_ftp\_server -u "username" -p "password" --ip 0.0.0.0 --port 6060 -d "/Users/koushik/Documents/MSCS/Fall 2023/Advanced Cryptography/secure-shared-file-storage/Home/temp"**

* + Note the IP address **0.0.0.0**, port **6060**, username **username**, and password **password**.

FTP Client Usage

1. **Launch Client Application:**
   * Open a new terminal.
   * Execute: **python run.py**.
2. **Uploading Files:**
   * Connect to the FTP server with the provided IP address and port.
   * Log in using the username and password.
   * Specify the file path for upload (e.g., **/Users/koushik/Documents/MSCS/Fall 2023/Advanced Cryptography/secure-shared-file-storage/Home/temp/text.txt**).
   * Generate a Public/Private Key pair using **ssh-keygen**. Save the key to: **/Users/koushik/Documents/MSCS/Fall 2023/Advanced Cryptography/secure-shared-file-storage/client/key**.
   * Enter the Public Key path during upload.
   * Confirm upload success via the application's "Uploaded file->Path" message.
3. **Downloading Files:**
   * Enter the name of the file to download (e.g., **text.txt**).
   * Provide the previously generated Private Key.
   * Click download and wait for the confirmation message: “Downloaded and decrypted file: filename.txt”.
   * The decrypted file (e.g., **filename.txt.dec**) will be available in the main folder (**secure-shared-file-storage**).

Notes:

* Ensure the correct file paths are used for both upload and download processes.
* Maintain the security of your Private Key at all times.

**Understanding the Code and Its Working**

1. **run.py and main.py**:
   * These files serve as the entry point to your application.
   * **run.py** simply calls the **main** function from **main.py**.
   * In **main.py**, the MVC (Model-View-Controller) pattern seems to be implemented:
     + **Model**: **FTPConnectionModel** manages the data, logic, and rules of the application.
     + **View**: **FTPClientGui** handles the graphical user interface.
     + **Presenter**: **FTPClientPresenter** acts as an intermediary between View and Model.
   * This structure facilitates a separation of concerns, making your code more organized and maintainable.
2. **file\_handler/file\_cryptographer.py**:
   * This module is responsible for the encryption and decryption of files using a key.
   * The **encryptFile** method encrypts a file using hybrid encryption and saves the encrypted data and keys.
   * The **decryptFile** method decrypts the file using the provided keys.
   * It demonstrates the application's core functionality in securing file data.
3. **cipher/hybrid\_cipher.py and Related Cipher Modules (AES.py, RSA.py, blowfish.py, DES.py)**:
   * **hybrid\_cipher.py** implements hybrid encryption, which is a mix of symmetric (AES, DES, Blowfish) and asymmetric encryption (RSA).
   * It divides a file into parts and encrypts each part using a round-robin fashion with different algorithms.
   * The keys for each algorithm are generated randomly and used for both encryption and decryption.
   * Each cipher class (**AESCipher**, **RSACipher**, **BlowfishCipher**, **DESCipher**) implements specific cryptographic algorithms, highlighting the use of polymorphism and the principle of encapsulation.

**Utility and Support Files Overview**

1. **src/presenter.py (FTP Client Presenter):**
   * This module acts as the 'Presenter' in the MVP pattern, mediating between the view (**src/view.py**) and the model (**src/model.py**).
   * It contains methods to handle various user interactions like connecting, logging in, uploading, and downloading files.
   * Use of decorators like **\_newServerResponseEntry** for handling server response updates is a good design choice, illustrating the application's robust structure.
2. **src/view.py (FTP Client GUI):**
   * The 'View' component of the MVP pattern, responsible for the graphical user interface.
   * It uses **customtkinter** for creating a modern-looking GUI.
   * The separation of different GUI components into methods like **buildConnectSection** and **buildResponseSection** demonstrates a clean, modular approach to UI design.
3. **src/model.py (FTP Connection Model):**
   * The 'Model' in the MVP pattern, responsible for handling all operations related to the FTP server.
   * It uses **ftplib** for FTP operations and raises custom exceptions (**FTPError**, **UnableToConnect**, **NotAuthorized**) to handle different error scenarios.
   * This module encapsulates all the FTP server interaction logic, keeping it separate from the view and presenter.
4. **Configuration Files (.pylintrc, mypy.ini, .gitignore):**
   * These files are critical for maintaining code quality and consistency.
   * **.pylintrc** and **mypy.ini** are used for linting and type-checking, ensuring adherence to Python best practices.
   * **.gitignore** is configured to exclude unnecessary files from version control, which is crucial for a clean repository.

Presentation:

**Slide 1: Project Name and You name**

“Hello everyone, I am Nikitha. I am here to discuss an increasingly critical topic in our digital world - secure file sharing. Secure file sharing is an integral part of our daily lives, as we frequently exchange documents and data for personal and professional purposes. It's important to ensure that such transfers are conducted safely to protect sensitive information from unauthorized access or breaches. Our focus is SecureCrypt, a secure document sharing platform we've developed to address this need.”

**Slide 2: Introduction**

"So, what exactly is SecureCrypt? It's our answer to the growing demand for a secure, efficient, and user-friendly way to share files online. Using the File Transfer Protocol, or FTP, combined with a hybrid approach to cryptography, SecureCrypt stands out as a robust solution for both businesses and individuals.

But why is this important? Well, as we all know, the digital landscape is evolving rapidly. With this comes great convenience, but also significant risks. Data breaches and unauthorized access are more common than ever, and traditional file sharing methods just aren't cutting it anymore. They lack the necessary security measures, leaving sensitive information vulnerable. This is where SecureCrypt comes in, bridging this gap and offering a secure way to share files without sacrificing ease of use.

**Slide 3: Problem Definition**

" SecureCrypt emerged from a pressing need for safer, simpler file sharing in our hyper-connected world. Here’s the thing – sharing files is as common as sending a text, but it’s also full of risks like data theft and privacy invasion. Traditional ways to share files often leave them wide open for attacks. That’s scary, right?

That's why we created SecureCrypt. It's our answer to those complex, clunky security methods that feel like cracking a code. With SecureCrypt, we’re making sure that even your most confidential files travel safely across the web, without making you jump through hoops. Imagine sending a secret message that only the right person can read – that's what SecureCrypt does with your files. It's like having an armored truck for your data, ensuring it reaches its destination without any unwanted pit stops. So, whether you're a business keeping trade secrets or an individual protecting personal photos, our aim is To ensure that your private stuff doesn’t become someone else’s business.

**Slide 4: System Overview**

"Hello, I am Koushik. Coming to System Architecture of SecureCrypt, our system is crafted to ensure high security without compromising on user experience.

At the core of SecureCrypt is the Model-View-Presenter, or MVP pattern, an architecture that enhances both the manageability and scalability of our application. In this setup, the Model handles data and business logic, including the crucial FTP operations and file processing. The View, designed with user interaction in mind, facilitates file uploading/downloading and displaying information. Bridging these two is the Presenter, handles the input from UI, processes it and updates the View with necessary changes.

Now, let's focus on the key components of SecureCrypt. The FTP Server, our central hub, hosts the files and manages file transfers. The Client Application is where users interact with our system, designed to be intuitive and straightforward. And then, the highlight of our application, the Encryption/Decryption Modules, employ a hybrid encryption system combining AES, DES, Blowfish, and RSA for robust security. Together, these components form a cohesive system, providing a secure yet accessible platform for document sharing."

**Slide 5: Encryption Techniques in SecureCrypt**

"In Slide 5, we delve into the individual encryption techniques that form the backbone of SecureCrypt. Our system integrates four distinct encryption algorithms: AES, DES, Blowfish, and RSA. Each plays a vital role in our encryption strategy.

AES (Advanced Encryption Standard) is renowned for its strength and efficiency, making it a popular choice in various security applications. DES (Data Encryption Standard), though older, provides a simpler yet effective encryption layer. Blowfish is another symmetric key algorithm, known for its speed and compactness, especially suitable for encrypting large files.

Then we have RSA, an asymmetric encryption algorithm. RSA is critical in our system for encrypting the keys generated by AES, DES, and Blowfish. This ensures that even if a data breach occurs, the files remain securely encrypted.

These algorithms are not used in isolation. SecureCrypt employs a round-robin cipher technique. This method involves dividing a file into chunks and encrypting each part using a different algorithm in a cyclic manner. This hybrid approach enhances security, making it more challenging for unauthorized entities to decrypt the files without access to the specific set of keys."

**Slide 6: Implementing Hybrid Encryption in SecureCrypt**

"In Slide 6, we focus on how SecureCrypt implements its hybrid encryption system, a combination of symmetric and asymmetric encryption techniques.

The hybrid encryption process begins by dividing the file into N segments of 16 bytes each. For each segment, a key is generated randomly for AES, DES, and Blowfish. These segments are then encrypted using these algorithms in a round-robin fashion. The resulting encrypted data is a blend of segments encrypted with different algorithms.

What makes this hybrid approach unique is the use of RSA encryption for the symmetric keys. Once the file is encrypted using AES, DES, and Blowfish, the keys for these algorithms are grouped and then encrypted using RSA.

During decryption, the reverse process is followed. The RSA-encrypted keys are first decrypted to retrieve the symmetric keys. Then, using these keys, the file segments are decrypted in the reverse order of the round-robin process.

This hybrid method ensures that SecureCrypt not only protects data during transmission but also secures the keys used for encryption, providing comprehensive security for digital file sharing."

**Slide 7: User Interface and Interaction**

"In this slide, we delve into the user interface and interaction aspects of SecureCrypt, focusing on our GUI design and its functionality. Our FTPClientGui, shown in the screenshots, offers a clean layout with a sidebar for mode switching like switching between light and dark modes, enhancing user experience. In the main interface, you'll notice distinct sections for different operations. There's a straightforward connection setup, where users can enter the server IP and port. The login section, designed for simplicity, allows users to quickly enter their credentials. Once logged in, the control section becomes the hub of activity. Here, users can upload or download files, create or delete directories, all with a few simple clicks.The server's real-time feedback keeps you updated, making the process transparent and user-friendly."

**Slide 8 – Demonstration**

"For this part of the demonstration, I'll start by launching the SecureCrypt client application. I have already set up the FTP server due to time constrants. Here we go. Once the application is open, the first step is to connect to our FTP server. I'll enter the IP address and port number that we configured earlier. Next, I'll log in using the username and password we set for the FTP server.

Now, let's dive into the core functionality of SecureCrypt - secure file sharing. I'm going to upload a file to our server. I'll select a file here. SecureCrypt takes over from this point, automatically encrypting the file using the public key I generated previously. The file is then uploaded in its encrypted state, ensuring its security during transit and on the server.

Next up, let's download a file from the server. I'll choose a file from our server's directory and enter the private key generated previously. Upon clicking download, SecureCrypt decrypts the file using the private key that corresponds to the public key used for encryption. And there you have it - the file is now available in its original, unencrypted form."

**Slide 9 – Code Quality and Standards**

"In this slide, we'll briefly touch upon how we uphold code quality and standards in SecureCrypt through key configuration files.

The **.pylintrc** file guides our coding standards, ensuring that our code is consistent, readable, and maintainable.

The **mypy.ini** file enforces type checking, an essential practice for early bug detection and enhancing the robustness of our application.

And the **.gitignore** file, though often overlooked, plays a vital role in keeping our repository clean and focused by excluding unnecessary files from version control.

**Slide 10: Challenges and Lessons Learned**

"I would like to highlight the challenges we faced and the lessons we learned along the way.

In developing SecureCrypt, one of our most significant challenges was constructing the hybrid encryption system. This involved intricately weaving together AES, DES, Blowfish, and RSA algorithms into a cohesive and secure framework. The complexity lay in ensuring seamless integration where each algorithm complemented the others, enhancing overall security without sacrificing performance. Particularly, orchestrating the round-robin encryption process with these varied algorithms required meticulous attention to detail and a deep understanding of cryptographic principles. Through this challenge, we learned the critical importance of algorithm compatibility and the delicate balance between cryptographic strength and system efficiency.

**Slide 11: Conclusion**

"As we conclude, Our journey has been filled with learning, from developing a sophisticated encryption system to designing an intuitive interface. We've risen to the challenge, ensuring that our platform stands strong on security without compromising user experience.Looking forward, SecureCrypt aims to expand its horizons. The immediate goal is deployment for a wider audience, which means establishing a database for key and user management. We're also looking to enhance our encryption with cutting-edge techniques and digital signatures. Cloud integration is on the horizon too, promising scalability and global access, solidifying our commitment to a secure and user-friendly platform."

**Slide 12: Q&A Session**

* **Invite Questions from the Audience**
* **Be prepared to clarify technical aspects, design choices, etc.**