CSE451: Computer and Network Security

Secure Shared File Storage Using Hybrid Cryptography and FTP

Ahmed Mohamed Ahmed Aly	18P9313
Marawan Osama Mohamed Sherif	18P1416

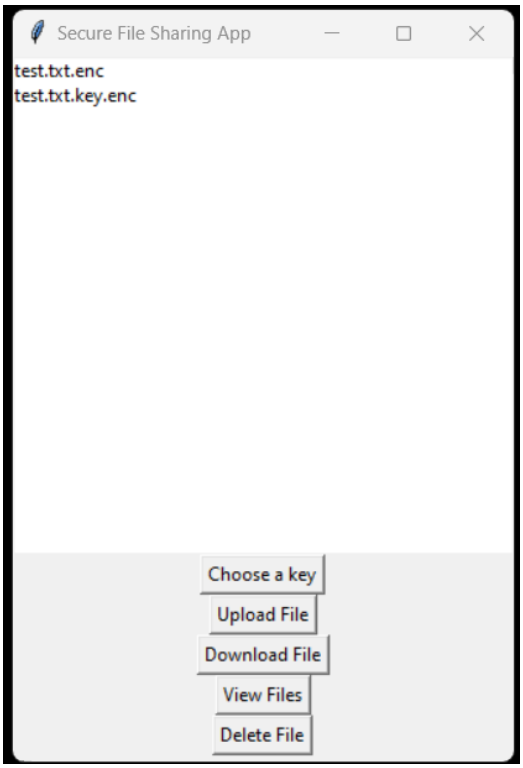
GitHub Link:

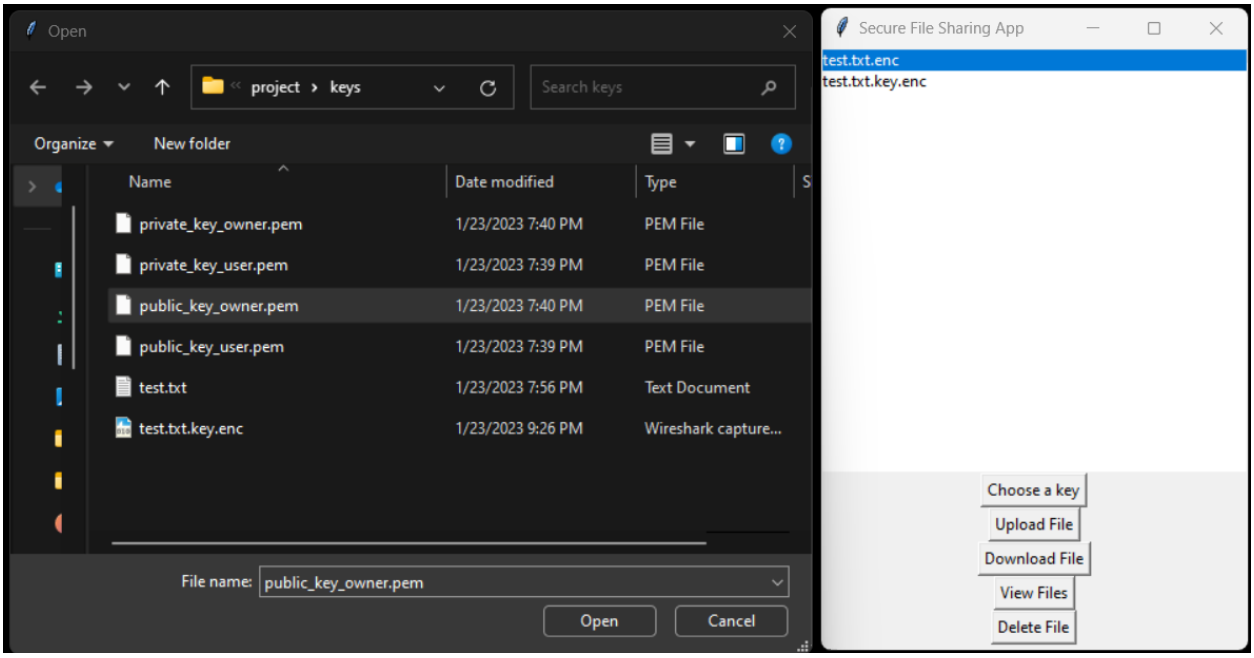
Link

Demo:

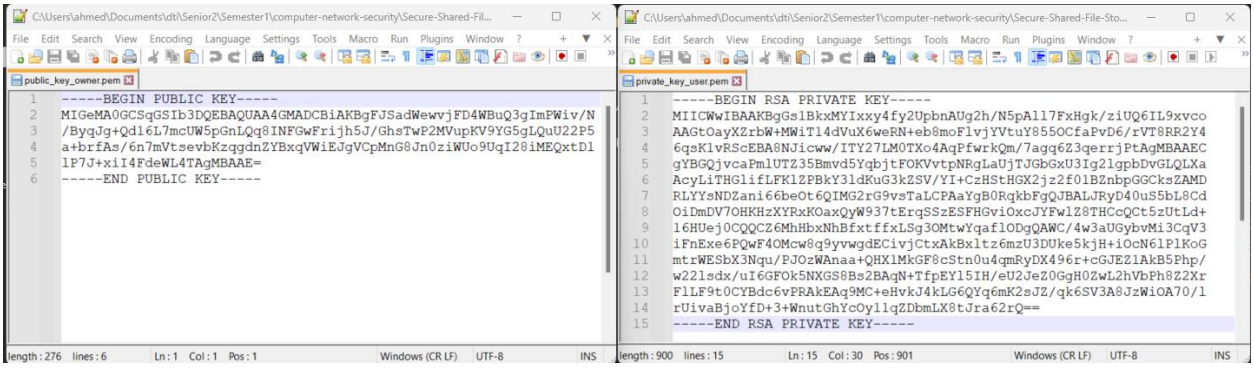
Video

Snippets:

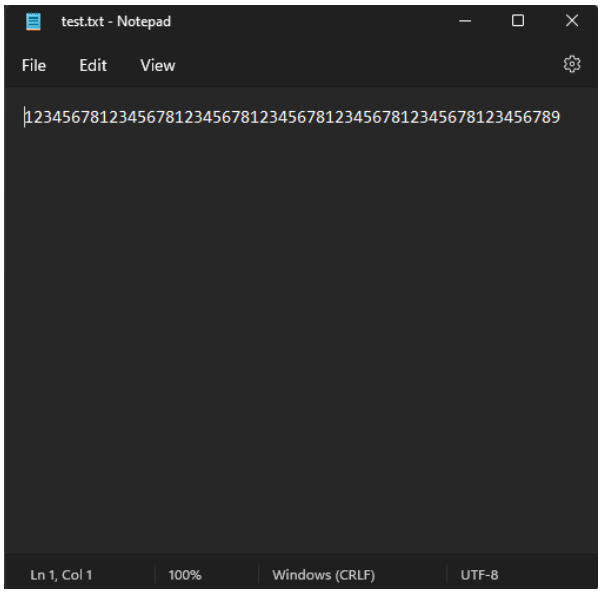




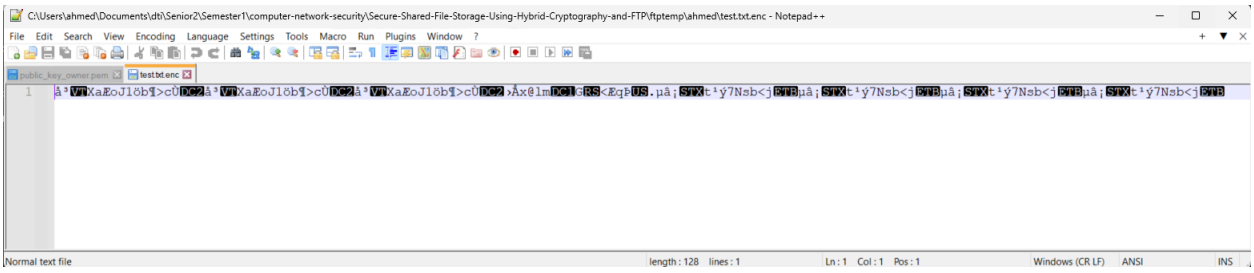
Owner’s public key, users private key:



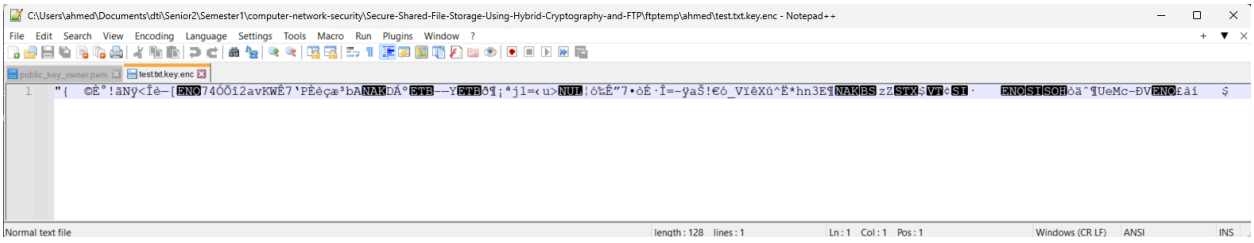
Text.txt



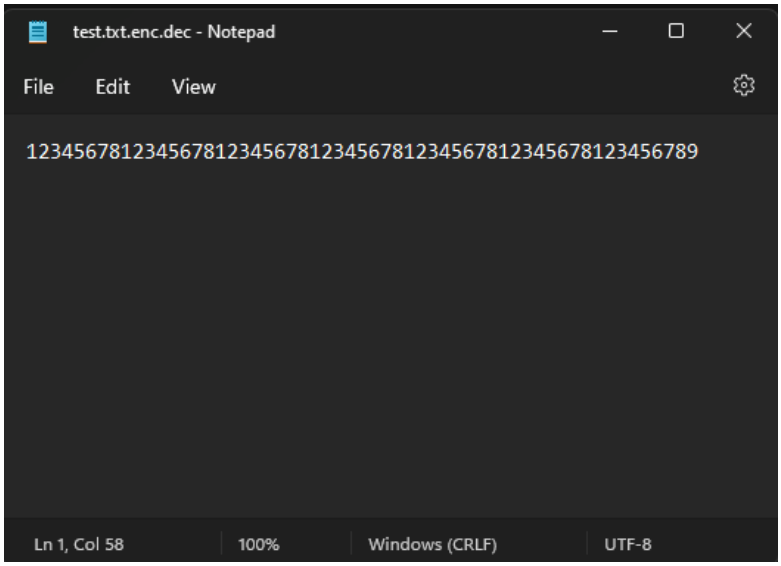
Text.txt.enc on ftp server



Text.txt.key.enc



Text.txt.enc.dec



Files hierarchy

Requirements

tkinter: This library is used to create the GUI for the app. It provides the widgets and functions needed to create a graphical user interface.

ftplib: This library is used to interact with an FTP server. It allows you to connect to an FTP server, upload and download files, and perform other operations.

pyftpdlib: This library is used to create an FTP server. It allows you to customize the FTP server's behavior and authenticate users.

Crypto: This library is used to encrypt and decrypt files. It provides various encryption algorithms such as AES, DES, Blowfish, RSA and other cryptographic tools.

Design

high-level design for the File Sharing App:

User authentication:

- Implement a login system to authenticate users before allowing them to access the app
- Use a database or a file to store the usernames and passwords

File management:

- Allow users to upload and download files to and from an FTP server
- Allow users to view the files on the server
- Allow users to delete files from the server

File encryption:

- Encrypt files using different algorithms such as AES, DES, and Blowfish
- Decrypt files using the same algorithms
- Allow users to select a key to encrypt or decrypt the files

Key management:

- Allow users to generate keys for encryption and decryption
- Allow users to import and export keys

User interface:

- Create a GUI for users to interact with the app
- Display a progress bar while uploading or downloading files to show the progress

FTP Server management:

- manage the setup and management of the FTP server

The design should consider how these different functionalities interact with each other and how they can be integrated into a cohesive user experience. It should also consider how the data will be stored, such as the keys, files, and user information.

Breaking down the code

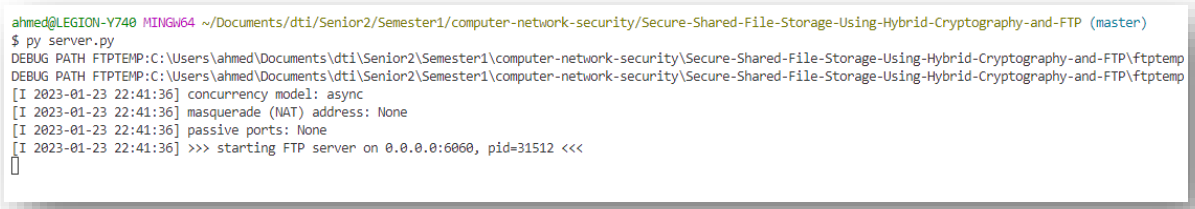
Server

AppFTPServer class uses the pyftpdlib library to handle the FTP functionality. It uses the DummyAuthorizer class to manage virtual users, the FTPHandler class to handle FTP requests and the FTPServer class to start the server. It also uses the os library to interact with the local file system.

The FTP server listens on IP address 0.0.0.0 and port 6060 by default. The addUser method creates a directory called ftptemp, used as the root directory for the virtual users. The class is used to set up an FTP server that can be used to upload and download files. The class has the following methods:

`def __init__(self):` this initializes a new instance of the class and sets up the server by instantiating a dummy authorizer for managing 'virtual' users, adding users, instantiating an FTP handler, specifying server's settings such as max_cons and max_cons_per_ip, and starting the server.

`def addUser(self, username, password):` this method creates a directory "ftptemp" and adds a user with the given username and password, with the specified permissions.




```
class AppFTPServer:
    def __init__(self):
        # Instantiate a dummy authorizer for managing 'virtual' users
        self.authorizer = DummyAuthorizer()

        self.addUser('ahmed', 'password')
        self.addUser('marawan', 'password')
        # Instantiate FTP handler class
        handler = FTPHandler
        handler.authorizer = self.authorizer

        # Define a customized banner (string returned when client connects)
        handler.banner = "pyftpdlib based ftpd ready."

        # Specify a masquerade address and the range of ports to use for
        # passive connections.  Decomment in case you're behind a NAT.
        #handler.masquerade_address = '151.25.42.11'
        #handler.passive_ports = range(60000, 65535)

        # Instantiate FTP server class and listen on 0.0.0.0:2121
        address = ('0.0.0.0', 6060)
        server = FTPServer(address, handler)

        # set a limit for connections
```

```
        server.max_cons = 256
        server.max_cons_per_ip = 5

        # start ftp server
        server.serve_forever()

    def addUser(self, username, password):
        ftptemp = os.path.join(os.getcwd(), 'ftptemp')
        print('DEBUG PATH FTPTEMP:' + ftptemp)
        if not os.path.exists(ftptemp):
            os.mkdir(ftptemp)
        self.authorizer.add_user(username, password, ftptemp,
perm='elradfmwMT')


if __name__ == "__main__":
    server = AppFTPServer()
```

GUI created by this script has the following elements:

- A listbox that displays the files currently on the server.
- A button labeled "Choose a key" that, when clicked, prompts the user to select a key for encryption/decryption.
- A button labeled "Upload File" that, when clicked, prompts the user to select a file to upload (after a key has been selected)
- A button labeled "Download File" that, when clicked, prompts the user to select a file to download (after a key has been selected)
- A button labeled "View Files" that, when clicked, displays the files currently on the server in the list box.
- A button labeled "Delete File" that, when clicked, deletes the selected file from the server and updates the list box.

```
class FileSharingApp(tk.Tk):
    def __init__(self):
        self.ftp_client = client.FTPClient()
        tk.Tk.__init__(self)
        self.title("Secure File Sharing App")
        self.geometry("700x500")
        self.key_path = None
        self.master_key = None
        #list of files on the server
        self.file_list = tk.Listbox(self, height= 20, width= 60, border= 0)
        self.file_list.pack()
```

```
        choose_key_button = tk.Button(self, text="Choose a key",
command=self.choose_key_button)
        choose_key_button.pack()

        upload_button = tk.Button(self, text="Upload File",
command=self.upload_file)
        upload_button.pack()
        download_button = tk.Button(self, text="Download File",
command=self.download_file)
        download_button.pack()
        view_button = tk.Button(self, text="View Files",
command=self.view_files)
        view_button.pack()

        delete_button = tk.Button(self, text="Delete File",
command=self.delete_file)
        delete_button.pack()
        self.view_files()
```

Client:

FTPClient class uses the ftplib library to interact with the FTP server and os library to interact with the local machine. The class is used to interact with an FTP server. The class has the following methods:

`def __init__(self, host=FTP_HOST, port=FTP_PORT, user=FTP_USER, password=FTP_PASS):` this initializes a new instance of the class, connects to the specified FTP server (using the provided host, port, user and password), logs in to the server, creates a directory for the user if it does not already exist, changes the working directory to the user's directory, creates a downloads directory on the local machine if it doesn't exist and prints a message to indicate successful connection to the server.

`def upload_file(self, file_path):` this method takes a file path as an argument and uploads the file to the server. It prints a message indicating that the file has been uploaded.

`def download_file(self, file_path):` this method takes a file path as an argument and downloads the file from the server to the local machine. It prints a message indicating that the file has been downloaded.

`def delete_file(self, file_path):`this method takes a file path as an argument and deletes the file from the server. It prints a message indicating that the file has been deleted.

`def view_files(self):` this method returns a list of all the files in the user's directory on the server

`def downloads_dir(self):` this method create the download dir in the local machine

```
class FTPClient:

    def __init__(self, host=FTP_HOST, port=FTP_PORT, user=FTP_USER,
password=FTP_PASS):
        self.user = user
        self.server = ftplib.FTP()
        self.server.connect(FTP_HOST, FTP_PORT)
        self.server.login(user, password)
        print('DEBUG `__init__` PATH SERVER_DIR:' + self.server.pwd())
        self.client_dir = os.path.join(self.server.pwd(), self.user)
        print('DEBUG `__init__` PATH CLIENT_DIR:' +self.client_dir)
        print('DEBUG `__init__` SERVER DIR LIST:' + str(self.server.nlst()))
        if(self.user not in self.server.nlst()):
            self.server.mkd(self.client_dir)
            print(f'Directory {self.client_dir} has been created on the
server.')
        self.server.cwd(self.client_dir)
        self.downloads_dir()
        print("Connected to the server.")

    def downloads_dir(self):
        downloads = os.path.join(os.getcwd(), 'downloads')
        if not os.path.exists(downloads):
            os.mkdir(downloads)
            print(f"Directory /downloads has been created locally.")


    def upload_file(self, file_path):
        abs_file = os.path.split(file_path)[1]
        with open(file_path, "rb") as f:
            self.server.storbinary(f'STOR {abs_file}', f, 1024)
            pass
        print(f"File {abs_file} has been uploaded to the server.")

    def download_file(self, file_path):

        abs_file = os.path.split(file_path)[1]
        with open(file_path, "wb") as f:
            self.server.retrbinary(f'RETR {abs_file}' , f.write, 1024)
        print(f"File {file_path} has been downloaded from the server.")
```

```
    def delete_file(self, file_path):
        self.server.delete(file_path)
        print(f"File {file_path} has been deleted from the server.")

    def view_files(self):
        files = self.server.nlst()
        return files
```

Encryption:

Encryption class uses the Crypto library to perform the encryption and decryption. It uses various encryption algorithms like AES, DES, Blowfish, RSA and PBKDF2. The class also uses the os library to interact with the local file system and the client class to interact with the FTP server. The class is used to encrypt and decrypt files. The class has the following methods:

`def __init__(self, file_path, key_path, master_key_path=None):`

this initializes a new instance of the class with the file path of the file to be encrypted or decrypted, the path of the key to be used for encryption/decryption and an optional path of the master key if the file is to be encrypted.

It also initializes some other variables used in the class such as a list of ciphers, the encrypted file, the encrypted master key, the RSA key and the decrypted file.

`def generate_key(self, cipher_type):` this method generates a key for the specified cipher type. It uses the PBKDF2 algorithm from the Crypto.Protocol.KDF library to generate the key.

`def padding_file(self):` this method pads the last chunk of the file to be encrypted if its size is not a multiple of 128 bytes.

`def divide_file(self):` this method divides the file to be encrypted into N parts of 128 bytes each.

`def get_keys(self):` this method generates keys for AES, DES and Blowfish ciphers and stores them in the key_list variable.

`def encrypt_file(self):` this method encrypts the file using AES, DES and Blowfish ciphers in a rotating manner.

`def upload_file(self):`this method uploads the encrypted file to the server.

`def download_file(self):` this method downloads the encrypted file from the server and decrypts it.

```
from Crypto.PublicKey import RSA
from Crypto.Cipher import AES, DES, Blowfish, PKCS1_OAEP
from Crypto.Util.Padding import pad, unpad
from Crypto.Random import get_random_bytes
from Crypto.Protocol.KDF import PBKDF2
import os
import client


class Encryption:
    def __init__(self, file_path, key_path, master_key_path=None):
        self.N = 0
        self.key_list = []
        self.chunks = []
        self.file_path = file_path
        self.file_size = 0
        self.ciphers = ["AES", "DES", "Blowfish"]
        self.encrypted_file = []
        self.encrypted_master_key = None
        self.key_path = key_path
        self.RSA_key = None
        self.master_key_path = master_key_path
        self.master_key = None
        self.ftp_client = client.FTPClient()
        self.decrypted_file = []

    def generate_key(self, cipher_type):
        password = b"CSE451 - Computer and Network Security"
        # DES key is of length 8 bytes --> 64 bits
        if cipher_type == "DES":
            salt = get_random_bytes(8)
            key = PBKDF2(password, salt, dkLen=8)
            self.key_list.append(key)
        # both AES and Blowfish keys are of length 16 bytes --> 128 bits
        else:
            salt = get_random_bytes(16)
            key = PBKDF2(password, salt, dkLen=16)
            self.key_list.append(key)
        return

    def padding_file(self):
        # get file size
        self.file_size = os.path.getsize(self.file_path)
        # get the size of the last chunk
        last_chunk_size = self.file_size % 128
        print("DEBUG padding_file last_chunk_size: " + str(last_chunk_size))
```

```
        # get the padding size
        if not last_chunk_size == 0:
            padding_size = 128 - last_chunk_size
            # get the padding
            padding = b" " * padding_size
            # add the padding to the last chunk
            with open(self.file_path, "ab") as f:
                f.write(padding)
        return

    def divide_file(self):
        # padding logic for the last chunk
        self.padding_file()
        # get file size
        self.file_size = os.path.getsize(self.file_path)
        # get the number of parts
        num_parts = self.file_size // 128
        print("DEBUG divide_file num_parts: " + str(num_parts))
        # divide the file into N parts of 128 bytes if i
        with open(self.file_path, "rb") as f:
            for i in range(num_parts):
                if i % 3 == 0:
                    chunk = f.read(128)
                else:
                    chunk = f.read(64)
                self.chunks.append(chunk)
        return
    def get_keys(self):
        # generate keys
        for cipher in self.ciphers:
            self.generate_key(cipher)
        print("DEBUG get_keys key_list: " + str(self.key_list))
        return
    def encrypt_file(self):
        # encrypt the file
        for i in range(len(self.chunks)):
            # if i % 3: encrept with AES
            # if i % 3 == 1: encrypt with DES
            # if i % 3 == 2: encrypt with Blowfish
            if i % 3 == 0:
                cipher = AES.new(self.key_list[0], AES.MODE_ECB)
            elif i % 3 == 1:
                cipher = DES.new(self.key_list[1], DES.MODE_ECB)
            elif i % 3 == 2:
                cipher = Blowfish.new(self.key_list[2], Blowfish.MODE_ECB)
```

```
            print()
            ciphertext = cipher.encrypt(self.chunks[i])
            print(
                "DEBUG encrypt_file ciphertext: "
                + str(self.ciphers[i % 3])
                + " "
                + str(ciphertext)
            )
            self.encrypted_file.append(ciphertext)

        return

    # encrypt the keys with the public key
    def encrypt_keys(self):
        with open(self.key_path, "rb") as f:
            RSA_key = f.read()
            self.RSA_key = RSA.import_key(RSA_key)
        cipher = PKCS1_OAEP.new(self.RSA_key)

        print(b"".join(self.key_list))

        with open(f'{self.file_path.split("/")[-1]}.key', "wb") as f:
            f.write(b"".join(self.key_list))

        self.encrypted_master_key = cipher.encrypt(b"".join(self.key_list))
        print(
            "DEBUG encrypt_keys encrypted_master_key: " +
str(self.encrypted_master_key)
        )
        return

    def upload_file(self):

        # gets generated keys
        self.get_keys()
        # encrypt the keys
        self.encrypt_keys()
        # divide the file into N parts
        self.divide_file()
        # encrypt the file
        self.encrypt_file()

        # upload the file

        with open(f'{self.file_path.split("/")[-1]}.enc', "wb") as f:
            f.write(b"".join(self.encrypted_file))
```

```
        with open(f'{self.file_path.split("/")[-1]}.key.enc', "wb") as f:
            f.write(self.encrypted_master_key)

        self.ftp_client.upload_file(f'{self.file_path.split("/")[-1]}.enc')
        self.ftp_client.upload_file(f'{self.file_path.split("/")[-1]}.key.enc')
        os.remove(f'{self.file_path.split("/")[-1]}.enc')
        os.remove(f'{self.file_path.split("/")[-1]}.key.enc')

    def download_file(self):
        # download the file
        # get test.txt.key.enc
        # get private key of the user
        # decrypt key.enc with public key of the user --> key
        # decrypt file.enc with key

        # download the file
        currdir = os.getcwd()
        if not os.path.exists(os.path.join(currdir, "downloads")):
            os.mkdir(os.path.join(currdir, "downloads"))

        download_dir = os.path.join(currdir, "downloads")
        os.chdir(download_dir)

        self.ftp_client.download_file(f"{self.file_path}")

        with open(self.master_key_path, "rb") as f:
            self.master_key = f.read()

        with open(self.key_path, "rb") as f:
            RSA_key = f.read()
            self.RSA_key = RSA.import_key(RSA_key)
        cipher = PKCS1_OAEP.new(self.RSA_key)

        print(self.master_key)
        decrypted_master_key = cipher.decrypt(self.master_key)
        print("DEBUG download_file decrypted_master_key: " +
str(decrypted_master_key))

        # divide the file into N parts
        # get file size
        self.file_size = os.path.getsize(self.file_path)
        # get the number of parts
        num_parts = self.file_size // 128
        with open(self.file_path, "rb") as f:
            for i in range(num_parts):
```

```
                if i % 3 == 0:
                    chunk = f.read(128)
                else:
                    chunk = f.read(64)
                self.chunks.append(chunk)

        # decrypt the file
        AES_key = decrypted_master_key[:16]
        DES_key = decrypted_master_key[16:24]
        Blowfish_key = decrypted_master_key[24:]
        for i in range(len(self.chunks)):
            # if i % 3: encrept with AES
            # if i % 3 == 1: encrypt with DES
            # if i % 3 == 2: encrypt with Blowfish
            if i % 3 == 0:
                cipher = AES.new(AES_key, AES.MODE_ECB)
            elif i % 3 == 1:
                cipher = DES.new(DES_key, DES.MODE_ECB)
            elif i % 3 == 2:
                cipher = Blowfish.new(Blowfish_key, Blowfish.MODE_ECB)
            print()
            ciphertext = cipher.decrypt(self.chunks[i])
            print(
                "DEBUG encrypt_file ciphertext: "
                + str(self.ciphers[i % 3])
                + " "
                + str(ciphertext)
            )
            self.decrypted_file.append(ciphertext)
        print("DEBUG download_file decrypted_file: " +
str(self.decrypted_file))
        # write the file
        with open(f"{self.file_path}.dec", "wb") as f:
            f.write(b"".join(self.decrypted_file).replace(b" ", b""))
        os.chdir(currdir)
        return
```

Test cases

1.Test uploading a file:

- Choose a key
- Select a file to upload
- Check that the file is encrypted and uploaded to the server
- Check that the file appears in the file list on the app

2.Test downloading a file:

- Select a file from the file list
- Choose a master key
- Check that the file is decrypted and downloaded to the downloads folder

3.Test deleting a file:

- Select a file from the file list
- Click the delete button
- Check that the file is deleted from the server and it no longer appears in the file list

4.Test invalid key:

- Try to upload a file without choosing a key
- Check that an error message is displayed and the file is not uploaded

5.Test invalid master key:

- Try to download a file without choosing a master key
- Check that an error message is displayed and the file is not downloaded

6.Test large file upload:

- Upload a file larger than 128 bytes
- Check that the file is split into chunks and encrypted with different ciphers before uploading to the server
- Verify that the whole file can be downloaded and decrypted correctly

7.Test invalid file:

- Try to upload or download a file that does not exist
- Check that an error message is displayed and the file is not uploaded or downloaded

8.Test ftp server's security:

- Try to access the ftp server with wrong credential
- Check that the access is denied

Resources

Pycryptodome's documentation. Welcome to PyCryptodome's documentation – PyCryptodome 3.15.0 documentation. (n.d.). Retrieved January 23, 2023, from https://pycryptodome.readthedocs.io/