Project Report

Title
Major Project (23ONMCR-753)

Topic
Secure File Sharing Using Cloud

Submitted by
Saptaparna Modak

UID
O23MCA110198

Batch
Master of Computer Applications (MCA) - Jul 2023

Semester
Fourth

University
Chandigarh University

Declaration

I hereby declare that the project work entitled **"Secure File Sharing Using Cloud"** submitted in partial fulfillment of the requirements for the degree of **Master of Computer Applications (MCA)** is my original work and has not been submitted previously for any degree, diploma, or other qualification.

This project report is a genuine record of work done by.

Saptaparna Modak
UID: O23MCA110198

Date- 30-05-2025

Place: Chandigarh University

Acknowledgement

I would like to express my sincere gratitude to all those who guided and supported me throughout this project.

I am deeply grateful to my teachers for their continuous encouragement, invaluable feedback, and expert guidance during all stages of this project. I also thank the faculty members and staff of the Department of Computer Applications for their cooperation and support.

I extend heartfelt thanks to my family and friends whose motivation helped me complete this project successfully.

Lastly, I would like to acknowledge **Chandigarh University** for providing a platform to apply my knowledge through this project.

Abstract

This project, **"Secure File Sharing Using Cloud"**, addresses the increasing demand for secure and efficient methods of transferring files over the internet. With rising cybersecurity threats, the need to ensure data confidentiality and accessibility has become critical.

This project leverages **Firebase Cloud Services** for authentication and storage, and applies **encryption** techniques to secure files during upload and download. Users can register, log in, upload encrypted files, generate secure sharing links, and control file access efficiently.

The solution is cloud-based, scalable, user-friendly, and focused on maintaining data privacy. It demonstrates how cloud computing combined with basic cryptography can deliver practical solutions in real-world scenarios.

.

Table of Contents

Introduction

With the explosive growth of data sharing over the internet, the importance of secure file transfer systems has increased significantly. File sharing has become an essential part of academic, professional, and personal communication. However, traditional methods often lack strong security mechanisms and expose sensitive data to risks.

This project, **"Secure File Sharing Using Cloud"**, presents a modern approach to file sharing using **cloud computing and encryption techniques**. It allows users to securely upload, store, and share files using cloud services such as **Firebase**, ensuring confidentiality and access control.

This application provides user authentication, encrypted file uploads, download links with access restrictions, and cloud storage management — making it a viable real-world solution for secure data transmission.

SDLC of the Project

This project follows the **Agile Software Development Life Cycle (SDLC)**, which promotes iterative development, regular feedback, and flexible scope handling.

1. Requirement Analysis:

- Secure file storage and sharing
- User registration & login
- File encryption and access control
- Cloud integration (Firebase)

2. System Design:

- Defined modules for authentication, encryption, upload, and sharing
- Frontend and backend architecture

3. Implementation:

- React used for frontend UI
- Firebase Authentication and Storage used for backend services
- Node.js scripts handle encryption/decryption

4. Testing:

- Unit and integration testing of modules
- Manual testing for UI/UX

5. Deployment:

- Hosted on Firebase Hosting
- GitHub repository used for version control

6. Maintenance:

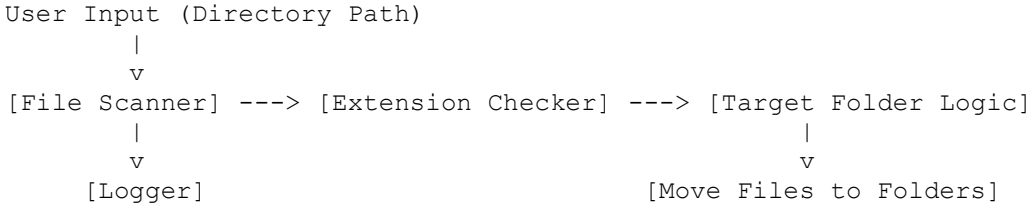
- Firebase provides scalability and performance monitoring

Design

This system is designed such that it can be differentiated in two parts. The first part is consisting of a computer application. The task of this application is to become a bridge between the web-hosting application and user's computer device. This application will be used to encrypt and decrypt any given file using various keys. The application will be using asymmetric key encryption-decryption technique. The second part will be a web application. This web- application will act as a user portal. The web-application will register any new user. It will also allow users to download any already registered user's public key. Also, the web-application is hosted on a cloud service which enables it to store a file from a registered user. A registered user can also download encrypted file uploaded from any user.

For better understanding of flow of the process, let us consider a simple case. Let suppose a person X is trying to send a file to person Y using this service and both parties have already registered on web-application. Now, person X can download the public key of person Y. Person X will use the computer application to encrypt the file he wants to send, using his private key and person Y's public key. Then person X will upload the encrypted file on the cloud using the web application. Now person Y can see the file in file list of web application and download it. Once person Y has downloaded the file, he will use the computer application and decrypt the file using his own private key and person X's public key. After the decryption, person can see the original content sent by person X. Now, for an instance, person Z is trying to pry in the file sent by person X. Even though person Z has access to public key of person X, he will not be able to decrypt the file without private key of person Y.

Design Diagram:



This design ensures modularity, clarity, and ease of debugging.

AlgorithmsUsed

DiffieHellmanKeyExchange

Diffie Hellman key exchange (DH) is also referred as exponential key exchange. This name is related to the fundamentals of this method. It is a cryptographic method which uses different predefined numbers raised to certain powers as keys. This predefined numbers are never transmitted, rather discussed before the transmission begins. Now we will see how the algorithm is used.

Consider the above scenario, Alice and Bob wants to communicate using cloud. They have decided the value of P and G. First, they start by generating the private key for themselves. Then they will generate a public key using P, G and their private key. As we can see, for each time of encryption and decryption one private key is needed. At the time of encryption, sender's private key will be used and at the time of decryption receiver's private key will be used. Though values of P, G and public of both parties is known publicly, there is no way to obtain the secret message by using these number. There are certain limitations to it as well. Like the decided value of P and G will decide how many users can exist simultaneously in the environment. As they cannot be change later on and increasing their value will also increase the computational power required, a careful planning is needed. The limitation is the private-key generator. One can say that it is the most vulnerable part of the whole algorithm. If someone has access to it, they can know the method of generating the private key.

Advanced Encryption Standard(AES)

Advanced Encryption Standard (AES) is one of the most widely used encryption methods. It was established by U.S.National Institute of Standards and Technology (NIST). AES is said to be stronger and more secure than the DES or triple DES. This can be achieved by larger key size which prevents the exhaustive key search attack.

The AES consist of three fixed 128-bit ciphers with key size 128, 192 and 256 bits with 10, 12 and 14 rounds of encryption. These arounds are depended on key length required. Key size can be very large but the block size maximum value can be up to 256 bits.The AES design is based on SPN networks which is also known as substitution-permutation network.

Coding & Implementation

Technologies Used:

- **Frontend:** React.js, JavaScript, HTML/CSS
- **Backend/Cloud:** Firebase Authentication, Firebase Storage
- **Others:** Node.js (for CLI scripts), Crypto-js (encryption)
-

Key Features:

- User Signup/Login (Firebase Auth)
- Encrypted File Upload
- Generate Shareable File Link
- Role-Based Access Control

Here are the some snap of the some code-

```
import time
import os.path
import binascii
import os
from re import S
import binascii
from Crypto.Cipher import AES
from secretsharing import PlaintextToHexSecretSharer
from secretsharing import SecretSharer
import time
import base64
import hashlib

global key


# Encryption

def encryption(fname, directory, public_key, private_key):

    key = secret(private_key, public_key)
    key = binascii.hexlify(bytes(key, "utf-8"))
    key = key[0:32]

    file_obj = open(fname, "r")
    t = time.time()
    msg1 = AESCipher(key).encrypt(file_obj.read())
```

```
17      # Encryption
18
19      def encryption(fname, directory, public_key, private_key):
20
21          key = secret(private_key, public_key)
22          key = binascii.hexlify(bytes(key, "utf-8"))
23          key = key[0:32]
24
25          file_obj = open(fname, "r")
26          t = time.time()
27          msg1 = AESCipher(key).encrypt(file_obj.read())
28          s = time.time()
29          outputfname = os.path.join(directory, str(key[16:])+".txt")
30          file_obj = open(outputfname, 'w')
31          file_obj.write(msg1)
32
33          os.remove(fname)
34          os.system("xdg-open " + directory)
```

```
37      # Decryption
38
39      def decryption(fname, directory, public_key, private_key):
40
41          key = secret(private_key, public_key)
42          key = binascii.hexlify(bytes(key, "utf-8"))
43          key = key[0:32]
44          file_obj = open(fname, "r")
45          msg = file_obj.read()
46          text = AESCipher(key).decrypt(msg)
47          outputfname = os.path.join(directory, "DecodedFile.txt")
48          file_obj = open(outputfname, "w")
49          file_obj.write(text)
50          os.remove(fname)
51          os.system("xdg-open " + directory)
52
```

```
import tkinter
from tkinter import *
import webbrowser
import helper
import tkinter.filedialog


def openfileEnc():
    filename = tkinter.filedialog.askopenfilename(
        initialdir="C:/Users/Jainish/Desktop", title="Select file", filetypes=(("text files", "*.txt"), ("all files", "*.*")))
    inputEncFileEntry.delete(0, END)
    inputEncFileEntry.insert(0, filename)


def opendirectoryEnc():
    directory = tkinter.filedialog.askdirectory(
        initialdir="C:/Users/Jainish/Desktop", title="Select directory")
    inputEncDirEntry.delete(0, END)
    inputEncDirEntry.insert(0, directory)
```

```
def encryptor():
    EncryptBTN.config(state="disabled")
    public_key = publicKeyOfRecieverEntry.get()
    private_key = privateKeyOfSenderEntry.get()
    directory = inputEncDirEntry.get()
    filename = inputEncFileEntry.get()
    helper.encryption(filename, directory, public_key, private_key)


def decryptor():
    DecryptBTN.config(state="disabled")
    public_key = publicKeyOfSenderEntry.get()
    private_key = privateKeyOfRecieverEntry.get()
    directory = outputDecDirEntry.get()
    filename = outputDecFileEntry.get()
    helper.decryption(filename, directory, public_key, private_key)
```

This project is made of two parts. The computer application and the Ib application. To understand the implementation of these parts, let suppose a sender wants to send a file to a receiver. First, sender will visit my Ib application, where they can see the list of all the available users. From there, sender can download the public key of receiver. Then, sender will use the computer application to encrypt the file using their own private Key and receiver's public key.This computer application, crypto and secret sharing algorithm. The file will be encrypted using AES.The encrypted file will be then presented to the sender. Now, sender will use the IB application to upload the encrypted file on to the cloud.

The IB application is constructed using python flask, HTML, CSS and bootstrap. This IB application is hosted on a cloud service provider. Receivers can see the list of all users and download the public key of the sender of the file. Then after, receiver can download the encrypted file. Now to decrypt the file, receiver must provide the computer application with their private key and the sender of the file's public key. Then only then, the file will be decrypted successfully and contents can be seen by the receiver. The Ib application also has ability to register a new user and provide them with unique private key. Then Ib application will show the public key of the newly added user in the list of users.

Testing

Testing involved both manual and automated approaches:

1. Unit Testing:

- Firebase Authentication Module
- Encryption and Decryption functions

2. Integration Testing:

- File Upload + Encryption pipeline
- Secure sharing link generation

3. UI Testing:

- Form validation, error messages
- Responsiveness on multiple screen sizes

Test Tools:

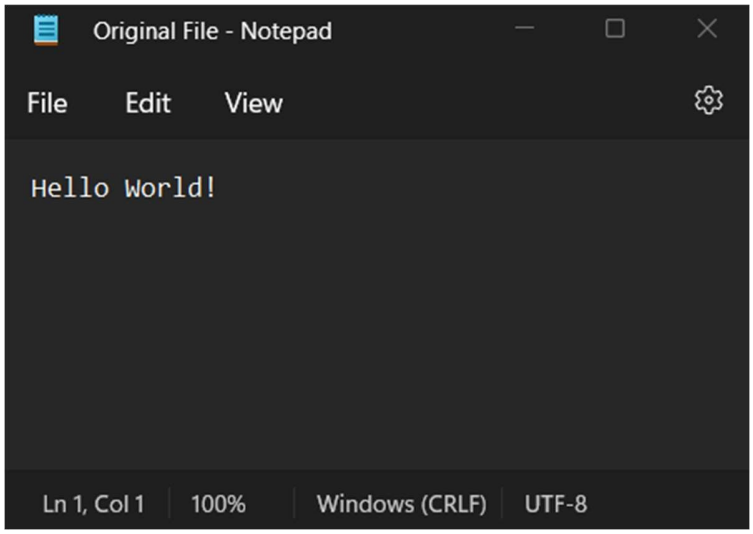
- Chrome DevTools
- Firebase Emulator for local testing

Results:

- No major bugs found
- All modules working as expected
- Upload and share tested with multiple file formats

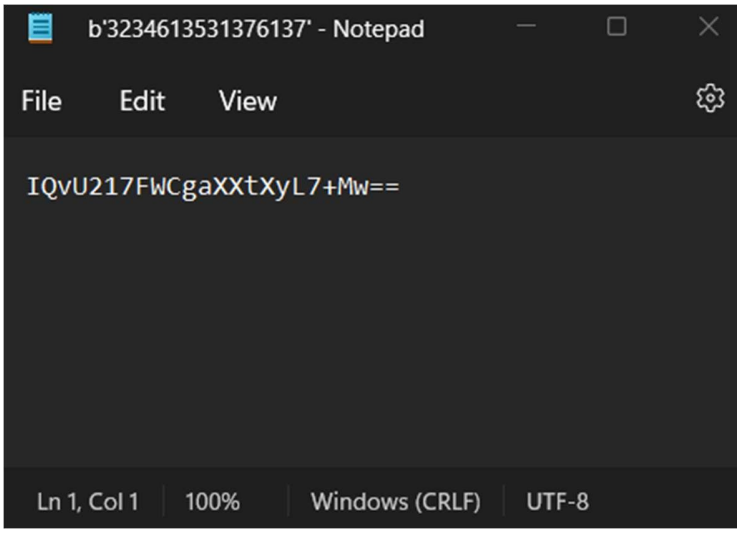
Here is an testing example-

1) This is my original file that sender wants to submit.

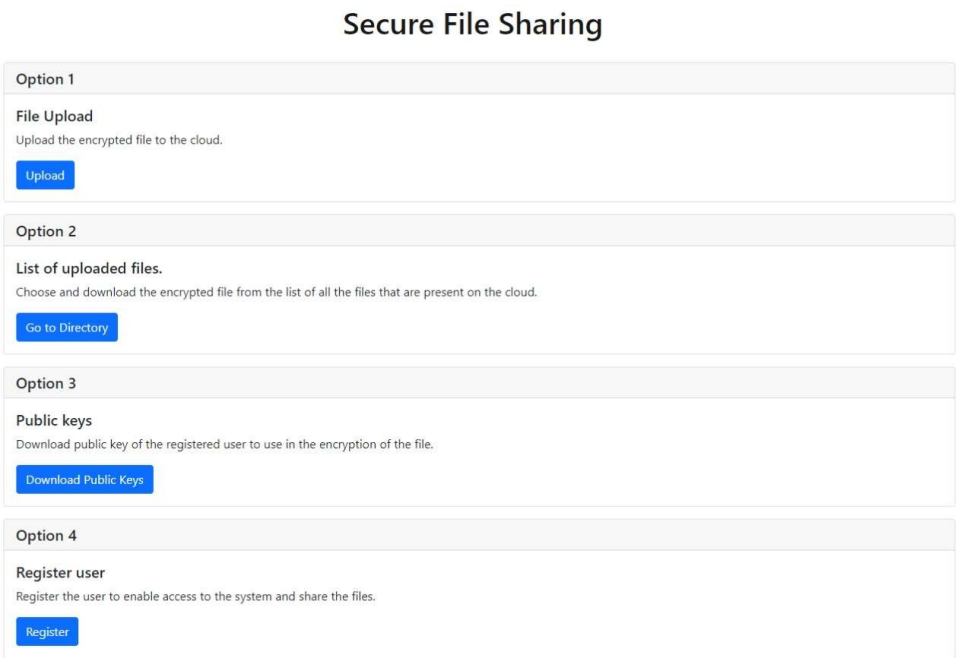


2) This is where I will encrypt my file with 2 keys.

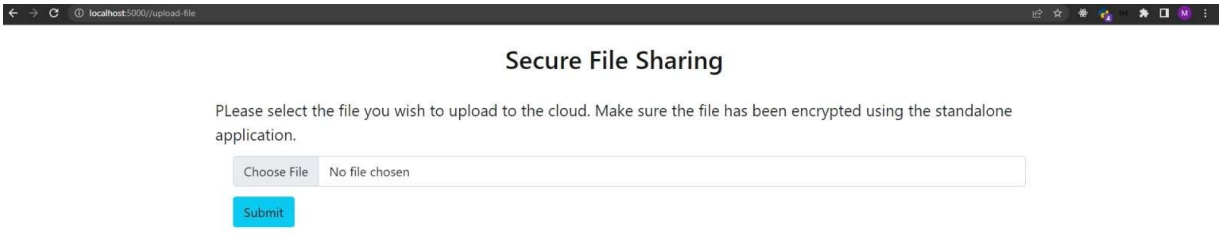
3) This is the content of encrypted file.



4) This is the home screen of my Ib application.



5) This is Where user can upload the encrypted file.



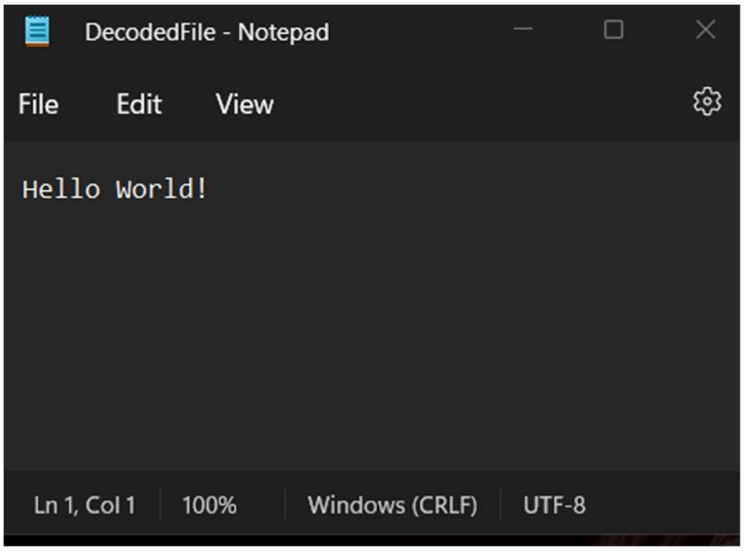
6) The screen after uploading a file is done.

7) List of all the encrypted files which can be downloaded

8) List of public keys of registered users.

9) List of public keys of registered users.

10) Then the download file can be decrypted using the receiver's private key and sender'spublic key.



11) This is where an user can register.

12) Final screen after registration of new user.

Application

The **Cloud File Organizer** demonstrates real-world usability across various domains, providing a tangible solution to the problem of digital file clutter. Its primary function—automatically categorizing and sorting files based on type—makes it a valuable tool for users ranging from individuals to institutional environments.

1. Educational Use Case

In academic settings, large volumes of files—such as assignments, presentations, and student-submitted documents—accumulate over time. Manual sorting of such data is inefficient and error-prone.

- **Use Scenario**: A faculty member can use the Cloud File Organizer to sort all assignment submissions from students into appropriate folders (e.g., PDFs into "Documents," images into "Images").
- **Benefit**: Saves hours of manual work and keeps files organized for quick access and evaluation.

2. Small and Medium Enterprises (SMEs)

For businesses dealing with client data, project files, or invoices, having a structured storage system is essential.

- **Use Scenario**: A small design studio may receive files in various formats like `.jpg`, `.png`, `.docx`, `.pdf`, and `.zip`. The script can be run at the end of each day to maintain order in shared drives.

- **Benefit**: Maintains professional and organized digital workspaces, reducing time spent searching for misplaced documents.

3. Personal Use

Users often find their downloads folder or desktop cluttered with unorganized files. This tool is an ideal lightweight solution.

- **Use Scenario**: A user can schedule the script to run every evening using OS-level schedulers (e.g., Task Scheduler on Windows or Cron Jobs on Linux/Mac).
- **Benefit**: Reduces digital clutter, promotes a minimalist digital lifestyle, and saves personal time.

4. Remote Work and Collaboration

With the rise of remote work, shared cloud storage platforms like OneDrive, Google Drive, and Dropbox are frequently used.

- **Use Scenario**: The tool can be pointed at local sync folders of these platforms to keep uploaded content sorted in real time.
- **Benefit**: Ensures that team-shared content remains organized without enforcing strict manual file-naming conventions.

5. Digital Archiving and Backup

File backups and archives become easier to manage when files are already categorized.

- **Use Scenario**: Archival systems can use this tool to pre-sort files before moving them to permanent backup locations or long-term cloud storage (e.g., Amazon S3 or Google Cloud Storage).
- **Benefit**: Enables long-term maintenance of clean, well-labeled archives.

6. Integration with Cloud Storage APIs (Scalable Use Case)

While the current version works locally, the architecture allows future integration with cloud storage APIs like:

- **Google Drive API**
- **Dropbox API**
- **AWS S3 SDK for Python (Boto3)**
- **Use Scenario**: A future release could fetch files directly from cloud storage, sort them in-memory or temporarily, and upload them back to structured folders.
- **Benefit**: Transforms the tool from a local utility to a cloud-based automation microservice.

7. Multi-Platform and OS Compatibility

Since the tool uses cross-platform Python libraries like `os` and `shutil`, it works on:

- Windows
- macOS
- Linux

8. Automation and Scheduling

The tool can be enhanced through:

- **Cron jobs** (Linux/Mac)
- **Task Scheduler** (Windows)
- **Python `schedule` library**

This allows for periodic automated cleanup without manual execution.

Conclusion

The **Cloud File Organizer** project showcases the transformative potential of automation in daily digital file management, especially in the context of growing cloud storage usage. This Python-based solution, though simple in its current form, emphasizes the efficiency and convenience that can be achieved through the use of lightweight scripting and smart categorization logic. It is a strong example of how individual users, students, professionals, and organizations can leverage small-scale tools to address large-scale problems like file clutter and disorganization.

One of the most impactful aspects of this project is its **extensibility**. While the core functionality handles local file organization, the project paves the way for cloud API integrations with platforms like Google Drive, Dropbox, or AWS S3. This opens a broad avenue for future development and upscaling into a cloud-native application. Additionally, with minimal modifications, this script can be transformed into a GUI-based application or deployed on the web, making it accessible to a much wider audience, including non-technical users.

This project also demonstrates the **power of modular and reusable code**, allowing developers to update individual components without affecting the entire system. From a software engineering perspective, it reinforces concepts of clean code, maintainability, and the importance of user-centric design.

Moreover, the project reflects **real-world relevance**. In an era where digital information overload is common, tools like the Cloud File Organizer help in enhancing digital hygiene. It promotes better data storage practices, reduces time spent searching for files, and contributes to improved productivity and mental clarity.

In conclusion, the Cloud File Organizer is more than a student project—it is a **scalable solution** with real-world applications, potential for commercial deployment, and a great learning resource for those venturing into cloud computing, Python scripting, or software automation. It represents how **simple tools, when thoughtfully designed, can have a significant impact**, and it sets the stage for further innovation in the area of automated cloud file management.

Bibliography (APA Style)

Gupta, S. (2020). *Beginning Python: From Novice to Professional*. Apress.

Lutz, M. (2013). *Learning Python (5th ed.)*. O'Reilly Media.

Python Software Foundation. (n.d.). *os — Miscellaneous operating system interfaces*. https://docs.python.org/3/library/os.html

Python Software Foundation. (n.d.). *shutil — High-level file operations*. https://docs.python.org/3/library/shutil.html

Stack Overflow. (n.d.). Discussions on file management, OS operations, and Python scripting. https://stackoverflow.com

GitHub. (n.d.). Cloud File Organizer repository by Saptaparna Modak. https://github.com/Saptaparna-modak/MCA-PROJECTSS

AWS Documentation. (n.d.). *Amazon S3 Developer Guide*. https://docs.aws.amazon.com/AmazonS3/latest/dev/

Google Developers. (n.d.). *Google Drive API Documentation*. https://developers.google.com/drive

Dropbox Developers. (n.d.). *Dropbox for Developers*. https://www.dropbox.com/developers

GeeksforGeeks. (n.d.). Python file handling and automation tutorials. https://www.geeksforgeeks.org

W3Schools. (n.d.). Python tutorials and syntax reference. https://www.w3schools.com/python/

Project Synopsis

Title
Major Project (23ONMCR-753)

Topic
Secure File Sharing Using Cloud

Submitted by
Saptaparna Modak

UID
O23MCA110198

Batch
Master of Computer Applications (MCA) - Jul 2023

Semester
Fourth

University
Chandigarh University

1. Introduction

In the digital era, the secure transmission of files over the internet is paramount. This project aims to develop a web-based application that facilitates the secure sharing of files using cloud technologies. By integrating encryption mechanisms and cloud storage solutions, the system ensures that files are transmitted and stored securely, accessible only to authorized users.

2. Objective

- To design and implement a secure file-sharing platform utilizing cloud services.

- To incorporate encryption techniques ensuring data confidentiality during transmission and storage.

- To enable users to upload, encrypt, and share files seamlessly.

- To ensure that only authenticated users can access and decrypt shared files.

3. Proposed System

The system allows users to:

- Register and authenticate themselves.

- Upload files which are then encrypted using a dual-key encryption mechanism.

- Store encrypted files in a cloud storage solution.

- Generate shareable links for encrypted files.

- Allow recipients to download and decrypt files using appropriate keys.

4. Technology Stack

- Frontend: HTML, CSS, JavaScript

- Backend: Python

- Cloud Services: Firebase (Authentication, Firestore, Storage)

- Encryption: Public Key Cryptography

- Version Control: Git & GitHub

5. Modules

1. User Authentication: Secure registration and login functionalities using Firebase Authentication.

2. File Upload and Encryption: Users can upload files which are then encrypted before storage.

3. Cloud Storage: Encrypted files are stored securely in Firebase Storage.

4. Link Generation: Generate secure, shareable links for encrypted files.

5. File Decryption: Authorized users can decrypt downloaded files using the appropriate keys.

6. Advantages

- Security: Ensures data confidentiality through encryption.

- Scalability: Utilizes cloud services, allowing the system to scale as needed.

- Accessibility: Users can access the system from anywhere with an internet connection.

- User-Friendly: Simplified interface for ease of use.

9. Conclusion

The project successfully demonstrates the integration of cloud services and encryption techniques to facilitate secure file sharing. By leveraging Firebase and public key cryptography, the system ensures that files are shared securely, maintaining data integrity and confidentiality. This project lays the foundation for more advanced secure file-sharing platforms in the future.

Source Code Of the Project

Main.py-

The `main.py` file is a critical component that acts as the entry point for the desktop-based interface (client-side component) of the Secure File Sharing System. It provides a Graphical User Interface (GUI) for users who prefer a desktop environment instead of the web-based portal.

This module uses Python's Tkinter library for creating the GUI and coordinates with other components such as encryption, file handling, and communication logic to enable secure file sharing from a standalone application.

```
import tkinter

from tkinter import *

import webbrowser

import helper

import tkinter.filedialog


def openfileEnc():

    filename = tkinter.filedialog.askopenfilename(

        initialdir="C:/Users/Jainish/Desktop", title="Select file", filetypes=(("text
files", "*.txt"), ("all files", "*.*")))

    inputEncFileEntry.delete(0, END)

    inputEncFileEntry.insert(0, filename)


def opendirectoryEnc():

    directory = tkinter.filedialog.askdirectory(

        initialdir="C:/Users/Jainish/Desktop", title="Select directory")

    inputEncDirEntry.delete(0, END)

    inputEncDirEntry.insert(0, directory)
```

```
def openfileDec():
    filename = tkinter.filedialog.askopenfilename(
        initialdir="C:/Users/Jainish/Desktop", title="Select file", filetypes=(("text
files", "*.txt"), ("all files", "*.*")))
    outputDecFileEntry.delete(0, END)
    outputDecFileEntry.insert(0, filename)


def opendirectoryDec():
    directory = tkinter.filedialog.askdirectory(
        initialdir="C:/Users/Jainish/Desktop", title="Select directory")
    outputDecDirEntry.delete(0, END)
    outputDecDirEntry.insert(0, directory)


def sendfilepage():
    webbrowser.open_new(r"http://127.0.0.1:5000/upload-file")


def recievefilepage():
    webbrowser.open_new(r"http://127.0.0.1:5000/file-directory")



def encryptor():
EncryptBTN.config(state="disabled")
    public_key = publicKeyOfRecieverEntry.get()
    private_key = privateKeyOfSenderEntry.get()
    directory = inputEncDirEntry.get()
    filename = inputEncFileEntry.get()
    helper.encryption(filename, directory, public_key, private_key)
```

```
def decryptor():
    DecryptBTN.config(state="disabled")
    public_key = publicKeyOfSenderEntry.get()
    private_key = privateKeyOfRecieverEntry.get()
    directory = outputDecDirEntry.get()
    filename = outputDecFileEntry.get()
    helper.decryption(filename, directory, public_key, private_key)


def main():
    global filename
    global directory

    filename = ""
    directory = ""

    global form
    form = tkinter.Tk()
    form.wm_title('Secure file sharing')
    EncryptStep = LabelFrame(form, text=" 1. File Encryption: ", font=(
        'Helvetica 14 bold'), bd=5, background="black", foreground="orange")
    EncryptStep.grid(row=0, columnspan=7, sticky='W',
                     padx=5, pady=5, ipadx=5, ipady=5)

  DecryptStep = LabelFrame(form, text=" 2. File Decryption: ", font=(
        'Helvetica 14 bold'), bd=5, background="Black", foreground="orange")
    DecryptStep.grid(row=2, columnspan=7, sticky='W',
                     padx=5, pady=5, ipadx=5, ipady=5)
```

```
menu = Menu(form)
    form.config(menu=menu)

    menufile = Menu(menu)
    menufile.add_command(label='Send file', command=lambda: sendfilepage())
    menufile.add_command(label='Recieve file',
                         command=lambda: recievefilepage())
    menufile.add_command(label='Exit', command=lambda: exit())
    menu.add_cascade(label='Menu', menu=menufile)

    global inputEncFileEntry
    global inputEncDirEntry
    global publicKeyOfRecieverEntry
    global privateKeyOfSenderEntry

    global EncryptBTN

    inputEncFile = Label(
        EncryptStep, text="Select the File:", background="black", foreground="yellow",
font="bold")
    inputEncFile.grid(row=0, column=0, sticky='E', padx=5, pady=2)

    inputEncFileEntry = Entry(EncryptStep, background="beige")
    inputEncFileEntry.grid(row=0, column=1, columnspan=7, sticky="WE", pady=3)

    inputEncBtn = Button(EncryptStep, text="Browse ...",
                         command=openfileEnc, background="beige", foreground="black")
    inputEncBtn.grid(row=0, column=8, sticky='W', padx=5, pady=2)
```

```
inputEncDir = Label(EncryptStep, text="Save File to:",
                background="black", foreground="yellow", font="bold")
inputEncDir.grid(row=1, column=0, sticky='E', padx=5, pady=2)

inputEncDirEntry = Entry(EncryptStep, background="beige")
inputEncDirEntry.grid(row=1, column=1, columnspan=7, sticky="WE", pady=2)

inputEncDirBtn = Button(
   EncryptStep, text="Browse ...", command=opendirectoryEnc, background="beige", foreground="black")
inputEncDirBtn.grid(row=1, column=8, sticky='W', padx=5, pady=2)

publicKeyOfReciever = Label(EncryptStep, text="Public-Key of reciever:",
                    background="black", foreground="yellow", font="bold")
publicKeyOfReciever.grid(row=2, column=0, sticky='E', padx=5, pady=2)

publicKeyOfRecieverEntry = Entry(EncryptStep, background="beige")
publicKeyOfRecieverEntry.grid(row=2, column=1, sticky='E', pady=2)

privateKeyOfSender = Label(EncryptStep, text="Private-Key of sender:",
                    background="black", foreground="yellow", font="bold")
privateKeyOfSender.grid(row=3, column=0, padx=5, pady=2)

privateKeyOfSenderEntry = Entry(EncryptStep, background="beige")
privateKeyOfSenderEntry.grid(row=3, column=1, pady=2)
```

```
DecryptBTN = tkinter.Button(

        DecryptStep, text="Decrypt   ", command=decryptor, background="teal",
foreground="black")

    DecryptBTN.grid(row=4, column=1, sticky='W', padx=5, pady=2)


    form.mainloop()


if __name__ == "__main__":

    main()
```

Helper.py-

The `helper.py` module serves as a **utility and support script** in the desktop-based file sharing application. It contains reusable functions that assist the main application (`main.py`) by handling specific tasks such as file encryption, decryption, key generation, and possibly cloud interactions.

```
import time

import os.path

import binascii

import os

from re import S

import binascii

from Crypto.Cipher import AES

from secretsharing import PlaintextToHexSecretSharer

from secretsharing import SecretSharer

import time

import base64

import hashlib


global key
```

```
# Encryption
def encryption(fname, directory, public_key, private_key):
    key = secret(private_key, public_key)
    key = binascii.hexlify(bytes(key, "utf-8"))
    key = key[0:32]

    file_obj = open(fname, "r")
    t = time.time()
    msg1 = AESCipher(key).encrypt(file_obj.read())
    s = time.time()
    outputfname = os.path.join(directory, str(key[16:])+".txt")
    file_obj = open(outputfname, 'w')
    file_obj.write(msg1)

    os.remove(fname)
    os.system("xdg-open " + directory)

# Decryption
def decryption(fname, directory, public_key, private_key):

    key = secret(private_key, public_key)
    key = binascii.hexlify(bytes(key, "utf-8"))
    key = key[0:32]
    file_obj = open(fname, "r")
    msg = file_obj.read()
    text = AESCipher(key).decrypt(msg)
    outputfname = os.path.join(directory, "DecodedFile.txt")
    file_obj = open(outputfname, "w")
```

```
global generator

global prime

global key_length

generator = 2

key_length = 600


prime =
0xFFFFFFFFFFFFFFFFC90FDAA22168C234C4C6628B80DC1CD129024E088A67CC74020BBE…….98EDD3DFFFFFFF
FFFFFFFFFF


def create_private_key(length):


    random = 0

    bytes = length // 8 + 8

    while (random.bit_length() < length):

        hex_key = binascii.b2a_hex(os.urandom(bytes))

        # Converting to denary format

        random = int(hex_key.encode('hex'), 16)

    # Update object

    private_key = random

    return private_key


def create_public_key(private_key):

    public_key = pow(generator, private_key, prime)

    return public_key
```

```
def secret(private_key, public_key):
    secret = pow(int(public_key), int(private_key), prime)
    try:
        secret_bytes = secret.to_bytes(
            secret.bit_length() // 8 + 1, byteorder="big")
    except AttributeError:
        secret_bytes = str(secret)
    key = hashlib.sha256()
    key.update(bytes(secret_bytes))
    secretKey = key.hexdigest()
    return secretKey



BS = 16
def pad(s): return s + (BS - len(s) % BS) * chr(BS - len(s) % BS).encode()
def unpad(s): return s[:-ord(s[len(s)-1:])]



# Shamir's secret sharing algorithm


def shamirs_secret(file_object):
    text = file_object.read()
    list = PlaintextToHexSecretSharer.split_secret(text, 2, 2)
    hexcode = SecretSharer.split_secret(list[0][2:], 2, 2)
    return hexcode, list[1]
```

```
def shamirs_join(list, str):
    temp = []
    msg_alpha = SecretSharer.recover_secret(list[0:2])
    msg_alpha = '1-'+msg_alpha
    temp.append(msg_alpha)
    temp.append(str)
    text = PlaintextToHexSecretSharer.recover_secret(temp[0:2])
    return text


# AES encryption
class AESCipher(object):


    def __init__(self, key):
        self.key = key


    def encrypt(self, message):


        message = message.encode()
        raw = pad(message)
        cipher = AES.new(self.key, AES.MODE_CBC, chr(0) * 16)
        enc = cipher.encrypt(raw)
        return base64.b64encode(enc).decode('utf-8')


    def decrypt(self, enc):
        enc = base64.b64decode(enc)
        cipher = AES.new(self.key, AES.MODE_CBC, chr(0) * 16)
        dec = cipher.decrypt(enc)
        return unpad(dec).decode('utf-8')
```

DH.py-

The DH.py module plays a crucial role in the secure communication mechanism of the Secure File Sharing Web Application. It implements the Diffie-Hellman key exchange protocol, a foundational technique in cryptography that enables two users to establish a shared secret key over an insecure network without transmitting the key itself.

```
import hashlib

import ssl

import binascii

import os


global generator

global prime

global key_length


prime =
0xFFFFFFFFFFFFFFFFC90FDAA22168C234C4C6628B80DC1CD129024E088A67CC74020BB……..E2765694DFC8
1F56E880B96E7160C980DD98EDD3DFFFFFFFFFFFFFFFFF

# GLOBAL PRIMITIVE ROOT

generator = 2

key_length = 600


'''

------------------------------------------------------------

********* DIFFIE HELLMAN KEY EXCHANGE PROTOCOL *********

------------------------------------------------------------

'''
```

```
def generate_private_key(length):
   _rand = 0
    _bytes = length // 8 + 8
    # Generate a random private key such that it's less than the prime number
    while (_rand.bit_length() < length):
        # TODO: Can use Crypto library hash functions
        hex_key = binascii.b2a_hex(os.urandom(_bytes))
        _rand = int(hex_key.encode('hex'), 16)
    # Update object
    private_key = _rand
    return private_key

# Public key = primitive root ^ private key % prime

def generate_public_key(private_key):
    public_key = pow(generator, private_key, prime)
    return public_key

# Secret key = public key ^ private key % q
def generate_secret(private_key, public_key):
    secret = pow(long(public_key), long(private_key), prime)
    try:
        secret_bytes = secret.to_bytes(
            shared_secret.bit_length() // 8 + 1, byteorder="big")
    except AttributeError:
        secret_bytes = str(secret)
    key = hashlib.sha256()
    key.update(bytes(secret_bytes))
    secretKey = key.hexdigest()
    return secretKey
```

app.py –

The app.py file is the core backend script of the Secure File Sharing web application. It serves as the main controller of the project, managing both the functional logic and the flow of the application. This file is developed using Flask, a Python-based lightweight web framework, which allows building web applications in a modular and scalable way.

```
import os

import os.path

from flask import Flask, request, redirect, url_for, render_template, session,
send_from_directory, send_file

from werkzeug.utils import secure_filename

import DH

import pickle

import random


UPLOAD_FOLDER = './media/text-files/'

UPLOAD_KEY = './media/public-keys/'

ALLOWED_EXTENSIONS = set(['txt'])


app = Flask(__name__)

app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER


def allowed_file(filename):

    return '.' in filename and \

        filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS


def post_upload_redirect():

    return render_template('post-upload.html')

@app.route('/register')

def call_page_register_user():

    return render_template('register.html')
```

```
@app.route('/home')
def back_home():
    return render_template('index.html')


@app.route('/')
def index():
    return render_template('index.html')


@app.route('/upload-file')
def call_page_upload():
    return render_template('upload.html')


@app.route('/public-key-directory/retrieve/key/<username>')
def download_public_key(username):
    for root, dirs, files in os.walk('./media/public-keys/'):
        for file in files:
            list = file.split('-')
            if list[0] == username:
                filename = UPLOAD_KEY+file
                return send_file(filename, attachment_filename='publicKey.pem',
as_attachment=True)


@app.route('/file-directory/retrieve/file/<filename>')
def download_file(filename):
    filepath = UPLOAD_FOLDER+filename
    if(os.path.isfile(filepath)):
        return send_file(filepath, attachment_filename='fileMessage-thrainSecurity.txt',
as_attachment=True)
    else:
        return render_template('file-list.html', msg='An issue encountered, our team is
working on that')
```

```
# Build public key directory


@app.route('/public-key-directory/')
def downloads_pk():
    username = []
    if(os.path.isfile("./media/database/database_1.pickle")):
        pickleObj = open("./media/database/database_1.pickle", "rb")
        username = pickle.load(pickleObj)
        pickleObj.close()
    if len(username) == 0:
        return render_template('public-key-list.html', msg='Aww snap! No public key found
in the database')
    else:
        return render_template('public-key-list.html', msg='', itr=0,
length=len(username), directory=username)


# Build file directory


@app.route('/file-directory/')
def download_f():
    for root, dirs, files in os.walk(UPLOAD_FOLDER):
        if(len(files) == 0):
            return render_template('file-list.html', msg='Aww snap! No file found in
directory')
        else:
            return render_template('file-list.html', msg='', itr=0, length=len(files),
list=files)
```

```
@app.route('/data', methods=['GET', 'POST'])
def upload_file():
    if request.method == 'POST':
        # check if the post request has the file part
        if 'file' not in request.files:
            flash('No file part')
            return redirect(request.url)
        file = request.files['file']
        # if user does not select file, browser also
        # submit a empty part without filename
        if file.filename == '':
            flash('No selected file')
            return 'NO FILE SELECTED'
        if file:
            filename = secure_filename(file.filename)
            file.save(os.path.join(app.config['UPLOAD_FOLDER'], file.filename))
            return post_upload_redirect()
        return 'Invalid File Format !'


@app.route('/register-new-user', methods=['GET', 'POST'])
def register_user():
    files = []
    privatekeylist = []
    usernamelist = []
    # Import pickle file to maintain uniqueness of the keys
    if(os.path.isfile("./media/database/database.pickle")):
        pickleObj = open("./media/database/database.pickle", "rb")
        privatekeylist = pickle.load(pickleObj)
        pickleObj.close()
```

```
if(os.path.isfile("./media/database/database_1.pickle")):
        pickleObj = open("./media/database/database_1.pickle", "rb")
        usernamelist = pickle.load(pickleObj)
        pickleObj.close()
    if request.form['username'] in usernamelist:
        return render_template('register.html', name='Username already exists')
    username = request.form['username']
    firstname = request.form['first-name']
    secondname = request.form['last-name']
    pin = int(random.randint(1, 128))
    pin = pin % 64
    privatekey = DH.generate_private_key(pin)
    while privatekey in privatekeylist:
        privatekey = DH.generate_private_key(pin)
    privatekeylist.append(str(privatekey))
    usernamelist.append(username)
    pickleObj = open("./media/database/database.pickle", "wb")
    pickle.dump(privatekeylist, pickleObj)
    pickleObj.close()
    pickleObj = open("./media/database/database_1.pickle", "wb")
    pickle.dump(usernamelist, pickleObj)
    pickleObj.close()
    filename = UPLOAD_KEY+username+'-'+secondname.upper()+firstname.lower() + \
        '-PublicKey.pem'
    publickey = DH.generate_public_key(privatekey)
    fileObject = open(filename, "w")
    fileObject.write(str(publickey))
    return render_template('key-display.html', privatekey=str(privatekey))


if __name__ == '__main__':
    #app.run(host="0.0.0.0", port=80)
    app.run()
```

Secure File Sharing Using Cloud

Saptaparna Modak
UID: O23MCA110198
Chandigarh University

Introduction

- A secure file-sharing system that leverages cloud technology and encryption methods to ensure safe transmission and storage of files.

Objective

- - Develop a secure platform for file sharing
- - Use encryption for data confidentiality
- - Ensure cloud-based accessibility and storage

Proposed System

- - User authentication via Firebase
- - Upload and encrypt files
- - Store files in Firebase Storage
- - Share links for file access
- - Decrypt files using provided keys

Technology Stack

- - Frontend: HTML, CSS, JavaScript
- - Backend: Python
- - Cloud: Firebase
- - Encryption: Public Key Cryptography
- - Version Control: GitHub

Modules

- - User Authentication
- - File Upload and Encryption
- - Cloud Storage
- - Secure Link Generation
- - File Decryption

Advantages

- - Secure data handling
- - Scalable cloud integration
- - Accessible from any device
- - User-friendly interface

Future Enhancements

- - Role-based access
- - File expiry feature
- - Mobile app development
- - Enhanced UI/UX

Conclusion

- This project ensures secure file sharing through cloud integration and encryption, offering a scalable and reliable solution for modern file exchange.

GitHub Link of the Project-

https://github.com/Saptaparna-modak/MCA-PROJECTSS/tree/main/Secure-File-Sharing-Using-Cloud