**INTERNSHIP REPORT**

*Submitted by*

**PRITHIKA.S**(511822104034)

in partial fulfillment of the requirements for the Award of Degree

of

**BACHELOR OF ENGINEERING**

**IN**

**COMPUTER SCIENCE AND ENGINEERING**

**PODHIGAI COLLEGE OF EGINEERING AND**

**TECHNOLOGY**

(Approved by AICTE New Delhi and Affiliated to Anna University, Chennai)

**TIRUPATTUR-635601**

**ANNA UNIVERSITY: CHENNAI-600025**

JULY-2025

**DEPARTMENT OF COMPUTER SCIENCE AND**

**ENGINEERING**

**PODHIGAI COLLEGE OF ENGINEERING AND**

**TECHNOLOGY**

(Approved by AICTE New Delhi and Affiliated to Anna University, Chennai)

**TIRUPATTUR-635601**

**ANNA UNIVERSITY: CHENNAI-600025**

JULY-2025

**BONAFIDE CERTIFICATE**

This is to certify that summer internship report is the Bonafide work of **PRITHIKA.S(511822104034)** in partial fulfillment of the award of the Degree of **BACHELOR OF ENGINEERING IN COMPUTER SCIENCE AND ENGINEERING** during the year 2022-2026.

**MR.M.KESAVAN ,B.E., MR.RAJ GANESH,ME.,**

SUPERVISORHEAD OF THE DEPARTMENT

Submitted for the viva voce examination held on **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**COMPANY CERTIFICATE**

**DECLARATION**

I affirm that the Summer Internship report **DIYAN TECH SOLUTIONS Pvt Ltd** submitted in partial fulfillment for the award of **Podhigai College of Engineering and Technology-Tirupattur, Computer Science and Engineering** degree is the original work carried out by me.

(Signature of the Candidate)

**S.Prithika(511822104034)**

I certify that the declarations made above by the candidate are true.

(Signature of the Guide)

**Mr.KESAVAN,B.E.,**

**ACKNOWLEDGEMENT**

I wish to express my sincere thanks to all those who were involved in the completion of this project.

I also express my profound thank to our Chairman **Rtn.K.C.EZHILARASAN** for their support and encouragement.

I offer my sincere thank to our principal **DR.M.PRABHAKARAN ME.,Ph.D** for their valuable suggestions which edified us with zest to carry out the work. Their support and encouragement have motivated us to carry out the work with enthusiastm and dedication.

I offer my sincere thank **MR.A.RAJ GANESH ME.,** our beloved Head of the Department of Computer Science and Engineering, Podhigai College of Engineering& Technology, Tirupattur, for his encouragement.

I especially thank **MR.M.KESAVAN**.,our guide for assisting us with his valuable suggestions and technical guidance to complete this Internship.

I express my deepest gratitude to **DIYANTECH SOLUTOINS PVT LTD** for providing me with the opportunity to work as an intern.

My sincere thanks to my mentors and colleagues, whose guidance and support great contributed to the successful completion of this internship.

I also express my sincere thank to our Department Faculty for their valuable support. Last but not least we extend our thank to our beloved parents for their moral support and blessing.

**CONTENT**

|  |  |
| --- | --- |
| **1.** | **Introduction**   * 1.1 Purpose * 1.2 Overview * 1.3 Technologies Used |
| **2.** | **System Requirements**   * 2.1 Software Dependencies * 2.2 Python Libraries Used |
| **3.** | **User Interface**   * 3.1 Login Screen * 3.2 Registration Form * 3.3 Main Menu Dashboard * 3.4 File Dialogs and User Prompts * 3.5 GUI Theming with ttkbootstrap |
| **4.** | **User Management**   * 4.1 User Registration * 4.2 User Login * 4.3 Password Encryption with bcrypt * 4.4 Forgot Password Functionality |
| **5.** | **Encryption & Decryption**   * 5.1 Symmetric Encryption with Fernet * 5.2 Key Generation and Storage * 5.3 File Encryption Workflow * 5.4 File Decryption Workflow * 5.5 Error Handling for Invalid Keys**/**Files |

|  |  |
| --- | --- |
| **6.** | **Key Management**   * 6.1 Key File Storage (secret.key) * 6.2 Automatic Key Generation * 6.3 Shared vs. Per-User Key (Design Consideration) |
| **7.** | **Logging and Audit**   * 7.1 Activity Logging per User * 7.2 Log File Format and Storage (log.txt) * 7.3 Viewing Logs in GUI |
| **8.** | **Data Storage Structure**   * 8.1 users.json – Credential Storage * 8.2 log.txt – Action Logs * 8.3 Directory Structure (user\_data/) |
| **9.** | **Security Considerations**   * 9.1 Password Protection * 9.2 File Encryption Integrity * 9.3 Key Confidentiality * 9.4 Risks in Password Reset Logic |
| **10.** | **Future Enhancements**   * 10.1 Per-User Encryption Keys * 10.2 Email-Based Recovery * 10.3 Admin Dashboard * 10.4 Upload to Cloud/Drive Integration |
| **11.** | **FLOW DIAGRAM** |
| **12.** | **PROGRAM** |
| **13.** | **SAMPLE OUTPUT** |
| **14.** | **CONCLUSION** |

**🔐 Introduction**

In today’s digital age, the confidentiality, integrity, and availability of data are critical concerns. With the increasing risks of data breaches, unauthorized access, and cyberattacks, secure file storage solutions have become essential for protecting sensitive information.

This project, Secure File Storage using the AES (Advanced Encryption Standard) Algorithm, is designed to provide users with a robust and user-friendly platform to store, encrypt, and decrypt files securely. AES is a widely accepted symmetric encryption standard trusted by governments, enterprises, and cybersecurity professionals worldwide for its strong security and performance.

The system allows users to register and authenticate using hashed passwords and provides functionality to encrypt or decrypt files using a 128-bit, 192-bit, or 256-bit AES key. It also includes features like key management, activity logging, and a graphical user interface (GUI) to enhance usability and security.

By leveraging the AES encryption algorithm, this secure file storage application ensures that even if files are accessed by unauthorized individuals, the contents remain confidential and unreadable without the correct encryption key.

**🔍 Key Objectives:**

* Ensure secure file storage using AES encryption.
* Provide a simple GUI for users to interact with the system.
* Enable user authentication with hashed credentials.
* Support encryption and decryption of any file type.

Maintain logs for audit and traceability

**1.1 Purpose**

The primary purpose of this project is to develop a secure file storage system that protects user files through strong encryption. By incorporating AES (Advanced Encryption Standard) as the encryption algorithm, the system ensures that files are only accessible by authorized users. The application also offers secure user authentication, key management, and file operation logging, making it suitable for both personal and institutional use where confidentiality is critical.

Key objectives include:

* Protecting sensitive files from unauthorized access.
* Providing a secure and intuitive user interface for encryption and decryption.
* Ensuring user credentials are stored and verified securely.
* Tracking user activity for accountability and auditing.

**1.2 Overview**

This application is a desktop-based secure file storage system built using Python. It offers an intuitive GUI for users to register, log in, and securely encrypt/decrypt files using AES encryption. Passwords are hashed using bcrypt for secure authentication, while the AES key is generated and managed securely using Python's cryptography library. All user interactions such as logins, file encryption, and decryption are recorded in an activity log file for auditing purposes.

The application is structured into the following core modules:

* **Authentication System**: Handles user registration, login, and password reset with secure hashing.
* **Encryption/Decryption**: Uses AES (via Fernet) to encrypt or decrypt selected files.
* **Key Management**: Generates and stores encryption keys securely.
* **Logging**: Tracks all user actions in a persistent log file.
* **Graphical User Interface**: Built with tkinter and styled using ttkbootstrap for a modern look and feel.

**1.3 Technologies Used**

|  |  |
| --- | --- |
| **Technology / Library** | **Purpose** |
| Python | Core programming language |
| Tkinter | GUI creation |
| Tkbootstrap | Modern theming for the Tkinter interface |
| |  | | --- | | bcrypt |  |  | | --- | |  | | Securepassword hashing and verification |
| cryptography(Fernet) | Symmetric AES encryption for file protection |
| PIL **(**Pillow**)** | image handling for background images |
| Json | Reading and writing user data in JSON format |
| Os | File system path and directory management |
| tkinter.filedialog | File selection dialogs for encryption/decryption |
| tkinter.messagebox | Displaying alerts and messages to the user |

**🖥️ System Requirements**

This section outlines the hardware, software, and Python environment requirements necessary to run the Secure File Storage application successfully.

**2.1 Hardware Requirements**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **Component** | **Minimum** **Requirement** | | Processor | 1 GHz or faster (x86/x64) | | RAM | 2 GB (4 GB recommended) | | Storage | 200 MB free disk space (for files/logs) | | Display | 1024x768 resolution or higher | | input Devices | Keyboard, Mouse | |  |

|  |  |
| --- | --- |
| **Component** | **Requirement** |
| Operating system | Windows 10/11, macOS, or Linux |
| Python version | Python 3.8 or higher |
| Interpreter | CPython (default Python interpreter) |
| Admin Privileges | Not required |
| Internet Connection | Not required at runtime |

**2.2 Software Requirements**

|  |  |
| --- | --- |
| **Library** | **Purpose** |
| Tkinter | GUI framework (included with Python) |
| Ttkbootstrap | Modern theming for Tkinter UI |
| Bcrypt | Secure password hashing |
| Cryptography | AES-based encryption (via Fernet) |
| Pillow (PIL) | Forloading and displaying background images |
| Json | Built-in library for data storage |
| Os | File and directory operations |

**2.3 Python Libraries and Dependencies**

**🎨 3. User Interface**

The user interface of the Secure File Storage application is designed to be intuitive, clean, and user-friendly. Built using Python’s tkinter and enhanced with ttkbootstrap, the interface ensures a modern look while remaining lightweight and functional**.**

**3.1 Login Screen**

The login screen is the entry point of the application. It allows users to enter their username and password to access the secure storage functions.

* Input Fields: Username, Password
* Buttons: Login, Register, Forgot Password?
* Validations: Checks for empty fields and invalid credentials
* Styling: Custom background (if available),centered for background

**3.2 Registration Form**

If the user is new, they can use the registration function to create an account.

* Triggered from: Login screen (Register button)
* Input Fields: Username, Password
* Validations: Checks if username already exists and fields are not empty
* Backend: Password is hashed using bcrypt before saving
* Notifications: Success or error message via messagebox

**3.3 Main Menu Dashboard**

After successful login, the user is directed to the main dashboard, which offers core functionality in a well-organized layout.

* Greeting: Dynamic welcome message with username
* Buttons:
  + 🔑 Generate Key
  + 🔒 Encrypt File
  + 🔓 Decrypt File
  + 🧾 View Logs
  + 🚪 Logout

 Each button launches a specific operation in a new window or prompt

 Background: Optional image (background\_menu.jpg) or fallback color

**3.4 File Dialogs and User Prompts**

To provide a smooth user experience, the app uses built-in dialogs for file selection and data entry.

* **File Dialogs**:
  + Open File for Encryption
  + Open File for Decryption
* **User Prompts**:
  + Password reset (via simpledialog.askstring)
  + Error and info messages (via messagebox)
  + Confirmation and feedback alerts

These dialogs simplify user interaction and reduce manual file path inputs.

**3.5 GUI Theming with ttkbootstrap**

The application uses the ttkbootstrap library to modernize the appearance of Tkinter widgets.

* **Theme Used**: morph (can be customized)
* **Themed Widgets**:
  + Styled buttons with bootstyle (e.g., PRIMARY, SUCCESS, DANGER)
  + Frames and labels use clean and consistent styling
* **Benefits**:
  + Professional, polished look
  + Better visual hierarchy
  + Enhanced readability and usability

**👤 4. User Management**

The Secure File Storage system includes a robust user management module that ensures only authorized individuals can access and perform encryption/decryption operations. It provides features for secure registration, login, and password recovery using industry-standard encryption techniques.

**4.1 User Registration**

New users can create an account directly from the login screen using the **Register** button.

* **Fields Required**: Username, Password
* **Validations**:
  + Fields must not be empty
  + Username must be unique
* **Action**:
  + Password is hashed using bcrypt before being stored
  + User data is saved in user\_data/users.json
* **Feedback**: Confirmation message on successful registration

**4.2 User Login**

Registered users can log in securely through the main login screen.

* **Fields Required**: Username, Password
* **Process**:
  + The system checks if the username exists
  + The entered password is verified using the stored bcrypt hash
* **On Success**: User is logged in and redirected to the main dashboard
* **On Failure**: Appropriate error message is displayed

**4.3 Password Encryption with bcrypt**

The system uses the bcrypt library for strong, one-way password hashing.

* **Hashing During Registration**:

bcrypt.hashpw(password.encode(), bcrypt.gensalt()).decode()

* **Verification During Login**:

bcrypt.checkpw(input\_password.encode(), stored\_hash.encode())

* **Advantages**:
  + Secure against brute-force attacks
  + Salting ensures unique hashes for identical passwords
  + Hashes are non-reversible

Passwords are **never stored in plain text**, ensuring high security.

**4.4 Forgot Password Functionality**

If a user forgets their password, they can reset it from the login screen using the **Forgot Password?** option.

* **Prompt**: Enter registered username
* **If Valid**: Prompt for a new password
* **Process**:
  + New password is hashed and replaces the old one in users.json
  + A log entry is recorded
* **Limitations**:
  + No identity verification (security enhancement recommended)

Top of Form

**🔐 5. Encryption & Decryption**

This module is the core of the Secure File Storage application, ensuring that files are encrypted and decrypted securely using symmetric cryptography. The system uses **Fernet**, a high-level interface provided by Python’s cryptography library, which implements **AES encryption in CBC mode** with PKCS7 padding, HMAC authentication, and built-in key management.

**5.1 Symmetric Encryption with Fernet**

* **Fernet** is built on **AES-128** and provides:
  + Confidentiality (data cannot be read without the key)
  + Integrity (tampering is detected via HMAC)
* It uses a **single key** (symmetric encryption), meaning the same key is used to both encrypt and decrypt data.

**Why Fernet?**

* Simplicity and security
* Includes encryption, signing, and timestamp verification

**5.2 Key Generation and Storage**

* **Key Generation**:
  + A new key is created using: python

python Fernet.generate\_key()

* **Storage**:
  + The key is saved to a file: user\_data/secret.key
  + If the key file doesn't exist, it is automatically generated when the application runs.

def generate\_key():

key = Fernet.generate\_key()

with open(KEY\_FILE, "wb") as f:

f.write(key)

return key

**5.3 File Encryption Workflow**

1. **User Action**: Clicks "🔒 Encrypt File"
2. **Select File**: File selected via file dialog
3. **Encrypt File**:
   * File is read as binary
   * Contents are encrypted using Fernet
   * Output file saved with .enc extension
4. **Confirmation**: Success message shown
5. **Log Entry**: Action is recorded

**Sample Code**:

decrypted = f.decrypt(data)

with open(new\_path, "wb") as file:

file.write(decrypted)

**5.5 Error Handling for Invalid Keys/Files**

* If decryption fails due to:
  + Incorrect key
  + Corrupted file
  + Wrong file type

... the user is shown an error message.

**Example Handling**:

except:

messagebox.showerror("Error", "Invalid key or corrupted file.")

**🔑 6. Key Management**

The encryption and decryption processes in this application rely on **symmetric key cryptography**, where a single key is used for both operations. Managing this key securely is essential to maintain the confidentiality and integrity of user files.

**6.1 Key File Storage (secret.key)**

* The encryption key is stored in a binary file named:

user\_data/secret.key

* This file contains a **Fernet key**, which is a URL-safe, base64-encoded 32-byte string representing an AES key (128-bit encryption by default).
* It is read each time the application performs encryption or decryption.

**6.2 Automatic Key Generation**

* When the application runs for the first time, or if the key file is missing, a new encryption key is **automatically generated**.
* This prevents accidental failure due to missing keys and ensures that the system is always functional

def load\_key():

if os.path.exists(KEY\_FILE):

with open(KEY\_FILE, "rb") as f:

return f.read()

else:

return generate\_key()

def generate\_key():

key = Fernet.generate\_key()

with open(KEY\_FILE, "wb") as f:

f.write(key)

return key

**6.3 Shared vs. Per-User Key (Design Consideration)**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Shared Key (Current)** | **Per-User Key (Recommended for High Security)** |
| Ease of Use | Simpler, one key for all users | Requires key management for each use |
| Security Level | Moderate — all users use the same key | High — each user has a unique encryption key |
| Implementation | Single secret.key file | Separate key file per user (e.g., username\_key.key) |
| Risk | If key is leaked, all data is compromised | Compromise affects only one user's data |

Bottom of Form

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**🧾 7. Logging and Audit**

To ensure accountability and provide traceability of actions, the Secure File Storage system includes a basic **logging and audit** mechanism. This feature records user activities such as login, registration, file encryption, and decryption, helping administrators and users monitor system usage.

**7.1 Activity Logging per User**

* Each significant action performed by a user is logged with the username.
* Actions tracked include:
  + User registration
  + Login attempts
  + Password resets
  + Key generation
  + File encryption and decryption
* This ensures that every sensitive operation is recorded for future reference or investigation.

**Example Entry**:

[2025-07-18 14:32:10] alice: Encrypted file report.docx

**Logging Function**:

from datetime import datetime

def log\_action(username, action):

with open(LOG\_FILE, "a") as log:

timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

log.write(f"[{timestamp}] {username}: {action}\n")

**7.2 Log File Format and Storage (log.txt)**

**File Location:**

user\_data/log.txt

**Format:**

* Plain text file
* Each line represents one event
* Includes timestamp, username, and action description.

**Example Log Content**:

[2025-07-18 12:05:33] john: Registered

[2025-07-18 12:06:12] john: Logged in

[2025-07-18 12:08:44] john: Encrypted file budget.xlsx

**7.3 Viewing Logs in GUI**

* Logs can be viewed directly from the **main menu** via the 🧾 View Logs button.
* A new window opens showing the full content of log.txt in a scrollable Text widget.
* **Read-only**: Users can view but not edit logs from the interface.

**Key Code Snippet**:

def view\_logs\_ui():

if os.path.exists(LOG\_FILE):

with open(LOG\_FILE, "r") as f:

content = f.read()

log\_window = tk.Toplevel()

log\_window.title("Activity Logs")

text = tk.Text(log\_window, wrap="word")

text.insert("1.0", content)

text.pack(expand=True, fill="both")

else:

messagebox.showinfo("Logs", "No logs available.")

**🗂️ 8. Data Storage Structure**

The application uses local file-based storage to maintain user credentials, activity logs, and encryption keys. All related files are stored within a dedicated directory named user\_data/, making the system lightweight, portable, and easy to manage.

**8.1 users.json – Credential Storage**

* **Purpose**: Stores registered users’ usernames and securely hashed passwords.
* **Location**:

user\_data/users.json

**Format**: JSON key-value structure:

{

"alice": "$2b$12$uYxe...HashedPassword",

"bob": "$2b$12$KmP8...HashedPassword"

}

**Security**:

* Passwords are hashed using bcrypt.
* Only hashed versions are stored — never plain-text passwords.
* This file is accessed during login and password verification.

**8.2 log.txt – Action Logs**

* **Purpose**: Keeps a timestamped record of all critical user actions (login, encryption, etc.).
* **Location**:

user\_data/log.txt

**. Format**: Plain-text log entries with timestamps:

[2025-07-18 13:12:45] alice: Logged in

[2025-07-18 13:13:02] alice: Encrypted file resume.pdf

**Usage**:

* Read-only display through the GUI log viewer.
* Useful for auditing and tracking user behavior.

**8.3 Directory Structure (user\_data/)**

The user\_data/ folder serves as the central storage directory for all user-related data.

* **Created Automatically** at first launch using:

os.makedirs("user\_data", exist\_ok=True)

**Contents**:

|  |  |
| --- | --- |
| **File Name** | **Purpose** |
| users.json | Stores usernames and bcrypt-hashed passwords |
| log.txt | Stores all logged user actions |
| secret.key | Stores the encryption key (Fernet/AES) |

This folder can be backed up or moved to transfer the system between machines.

**🔐 9. Security Considerations**

Security is the foundation of the Secure File Storage application. While the system implements several best practices in cryptography and data protection, understanding its limitations and potential risks is essential for future enhancements and responsible use.

**9.1 Password Protection**

* **Hashing with bcrypt**:  
  All user passwords are hashed using the bcrypt algorithm before being stored in the users.json file. This makes it computationally expensive to reverse or brute-force the password hash.
* **Salting Built-in**:  
  Each password hash includes a unique salt, making identical passwords appear different.
* **Best Practices Followed**:
  + Passwords are **never stored in plain text**.
  + Hashing is done during **registration and password reset**.
  + Passwords are **verified** using bcrypt.checkpw().

**9.2 File Encryption Integrity**

* **AES-Based Encryption with Fernet**:
  + Files are encrypted using symmetric AES encryption via the Fernet module from the cryptography library.
  + Fernet includes **built-in HMAC authentication**, ensuring that:
  + Files cannot be decrypted without the correct key&Any tampering with encrypted data will cause decryption to fail.

**Output Files**:

* Encrypted files use the .enc suffix.
* Decrypted files are saved with a \_decrypted suffix to prevent overwriting.

**9.3 Key Confidentiality**

* **Key File Storage**:
  + The encryption key is stored in a single file: user\_data/secret.key.
  + This key is **critical**—if leaked, all encrypted files can be decrypted.
* **Current Model**: Shared key for all users.
* **Risks**:
  + If secret.key is exposed, all users’ files are compromised.
  + The key is stored in **plain binary form** without encryption.

🔐 **Mitigation Suggestions**:

* Store **per-user keys** (e.g., alice.key, bob.key).
* Optionally, **encrypt the key** with a master password or derive it from the user's password using a key derivation function (KDF).

**9.4 Risks in Password Reset Logic**

* **Current Behavior**:
  + Anyone can reset a user’s password simply by knowing their username.
  + There is **no identity verification** during password reset.
* **Security Flaw**:
  + This opens the system to abuse or impersonation attacks.
  + An attacker could reset another user's password and gain unauthorized access.

**🚀 10. Future Enhancements**

To improve security, scalability, and usability, several features can be added in future versions of the Secure File Storage application.

These enhancements aim to make the system more robust for real-world or enterprise-level deployment.

**10.1 Per-User Encryption Keys**

**Current Limitation**:  
All users share a single encryption key (secret.key), which creates a single point of failure.

**Proposed Enhancement**:

* Generate and store **unique keys for each user**, e.g.:

user\_data/alice.key, user\_data/bob.key

* Keys can be encrypted using a **key derivation function (KDF)** based on the user's password.
* Improves:
* Confidentiality (user data is isolated)
* Granular access control
* Key rotation support

**10.2 Email-Based Recovery**

**Current Limitation**:  
Password reset is insecure and lacks user verification.

**Proposed Enhancement**:

* Add **email registration** during signup.
* Implement **email-based password recovery** using one-time tokens or verification codes.
* Integration with SMTP or third-party services (e.g., SendGrid, Gmail API) for sending recovery emails.

Benefits:

* Secure password recovery process
* Prevents unauthorized resets
* Supports modern account recovery standards

**10.3 Admin Dashboard**

**Purpose**:  
Introduce an **admin-level user interface** to manage accounts and monitor usage.

**Possible Features**:

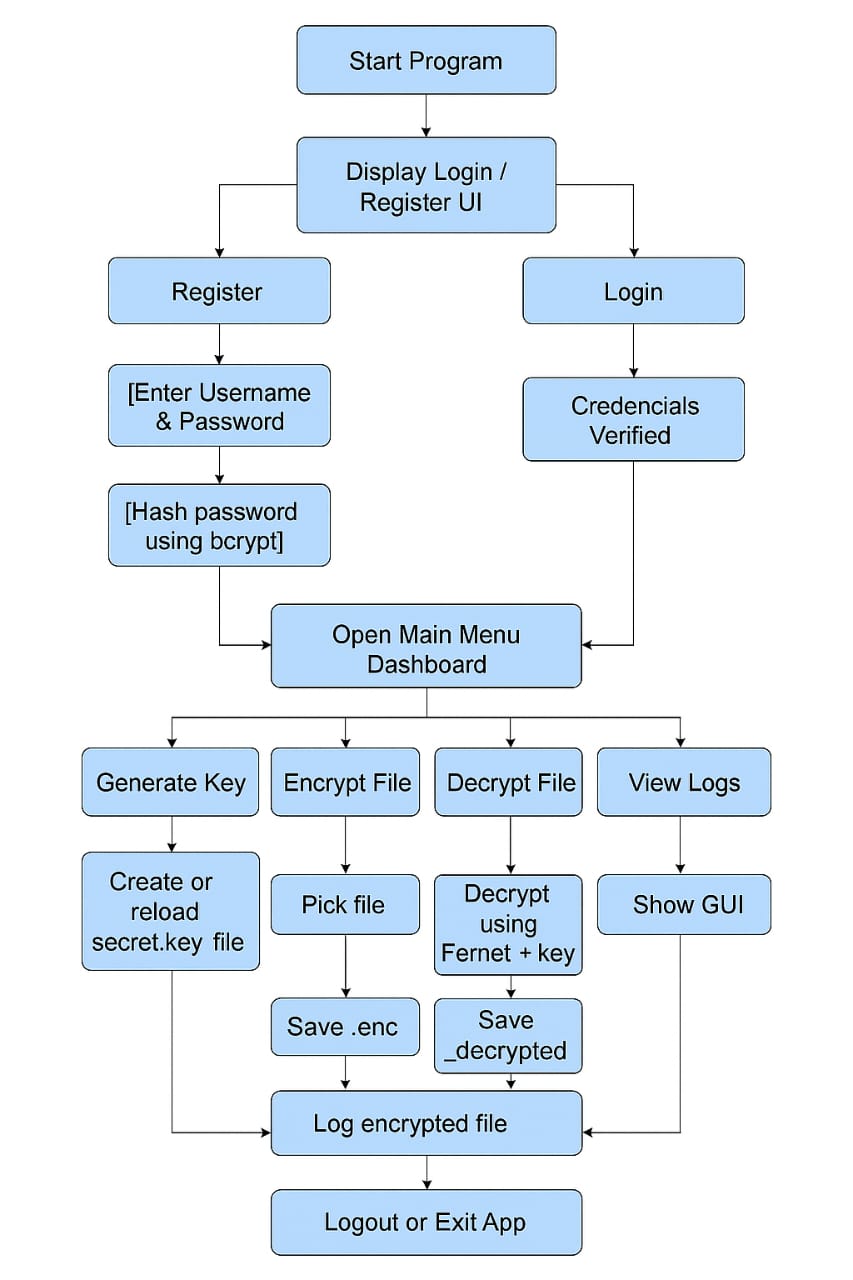
* View all registered users
* Reset or delete user accounts
* View full audit logs with filters (date, username, action)
* Manage encryption keys
* Export data or logs

**10.4 Upload to Cloud/Drive Integration**

**Proposed Feature**:  
Enable **cloud storage integration** for secure backup and access from any device.

**Possible Options**:

* Google Drive (via pydrive or google-api-python-client)
* Dropbox API
* OneDrive or AWS S3.

****

**12.PROGRAM**

import os

import json

import bcrypt

import tkinter as tk

from tkinter import messagebox, simpledialog, filedialog

from PIL import Image, ImageTk

from cryptography.fernet import Fernet

import ttkbootstrap as ttk

from ttkbootstrap.constants import \*

USER\_DATA\_FILE = os.path.join("user\_data", "users.json")

KEY\_FILE = os.path.join("user\_data", "secret.key")

LOG\_FILE = os.path.join("user\_data", "log.txt")

os.makedirs("user\_data", exist\_ok=True)

if not os.path.exists(USER\_DATA\_FILE):

    with open(USER\_DATA\_FILE, "w") as f:

        json.dump({}, f)

def load\_users():

    with open(USER\_DATA\_FILE, "r") as f:

        return json.load(f)

def save\_users(users):

    with open(USER\_DATA\_FILE, "w") as f:

        json.dump(users, f)

def hash\_password(password):

    return bcrypt.hashpw(password.encode(), bcrypt.gensalt()).decode()

def verify\_password(password, hashed):

    return bcrypt.checkpw(password.encode(), hashed.encode())

def generate\_key():

key = Fernet.generate\_key()

    with open(KEY\_FILE, "wb") as f:

        f.write(key)

    return key

def load\_key():

    if os.path.exists(KEY\_FILE):

        with open(KEY\_FILE, "rb") as f:

            return f.read()

    else:

return generate\_key()

def log\_action(username, action):

    with open(LOG\_FILE, "a") as log:

        log.write(f"{username}: {action}\n")

class LoginApp:

    def \_\_init\_\_(self, master):

        self.master = master

        master.title("Secure File Storage - Login")

        master.geometry("520x500")

        master.resizable(False, False)

try:

            bg\_image = Image.open("background.jpg").resize((520, 500), Image.Resampling.LANCZOS)

            self.bg\_photo = ImageTk.PhotoImage(bg\_image)

            bg\_label = ttk.Label(master, image=self.bg\_photo)

            bg\_label.place(x=0, y=0, relwidth=1, relheight=1)

        except:

            master.configure(bg="white")

        form\_frame = ttk.Frame(master, padding=25, bootstyle="light")

        form\_frame.place(relx=0.5, rely=0.5, anchor="center")

ttk.Label(form\_frame, text="🔒 Secure Login", font=("Segoe UI", 22, "bold"), bootstyle="dark")\

            .pack(pady=(0, 20))

        ttk.Label(form\_frame, text="Username", font=("Segoe UI", 11, "bold")).pack(anchor="w")

        self.username\_entry = ttk.Entry(form\_frame, width=32, font=("Segoe UI", 10))

        self.username\_entry.pack(pady=(0, 12))

        ttk.Label(form\_frame, text="Password", font=("Segoe UI", 11, "bold")).pack(anchor="w")

        self.password\_entry = ttk.Entry(form\_frame, show="\*", width=32, font=("Segoe UI", 10))

        self.password\_entry.pack(pady=(0, 18))

btn\_frame = ttk.Frame(form\_frame)

        btn\_frame.pack()

        ttk.Button(btn\_frame, text="Login", width=15, bootstyle=SUCCESS, command=self.login).pack(side="left", padx=5)

        ttk.Button(btn\_frame, text="Register", width=15, bootstyle=PRIMARY, command=self.register).pack(side="left", padx=5)

ttk.Button(form\_frame, text="Forgot Password?", width=34, bootstyle=SECONDARY, command=self.forgot\_password).pack(pady=(18, 5))

    def login(self):

        username = self.username\_entry.get().strip()

        password = self.password\_entry.get().strip()

        users = load\_users()

        if username not in users:

            messagebox.showerror("Error", "User not found.")

            return

        if verify\_password(password, users[username]):

            log\_action(username, "Logged in")

            self.master.destroy()

            open\_main\_menu(username)

else:

            messagebox.showerror("Error", "Incorrect password.")

    def register(self):

        username = self.username\_entry.get().strip()

        password = self.password\_entry.get().strip()

        if not username or not password:

            messagebox.showwarning("Warning", "Please fill in all fields.")

            return

        users = load\_users()

        if username in users:

            messagebox.showerror("Error", "User already exists.")

            return

        users[username] = hash\_password(password)

        save\_users(users)

        log\_action(username, "Registered")

        messagebox.showinfo("Success", "User registered successfully.")

    def forgot\_password(self):

        username = simpledialog.askstring("Forgot Password", "Enter yourusername:")

        users = load\_users()

        if username not in users:

            messagebox.showerror("Error", "User not found.")

            return

        new\_pass = simpledialog.askstring("Reset Password", "Enter new password:", show="\*")

        if not new\_pass:

            messagebox.showwarning("Warning", "Password reset cancelled.")

            return

        users[username] = hash\_password(new\_pass)

        save\_users(users)

        log\_action(username, "Password reset")

        messagebox.showinfo("Success", "Password reset successfully.")

def open\_main\_menu(username):

    menu\_win = ttk.Window(title="Main Menu", themename="morph")

    menu\_win.geometry("620x580")

    menu\_win.resizable(False, False)

    try:

        bg\_image = Image.open("background\_menu.jpg").resize((620, 580), Image.Resampling.LANCZOS)

        bg\_photo = ImageTk.PhotoImage(bg\_image)

        bg\_label = tk.Label(menu\_win, image=bg\_photo)

        bg\_label.image = bg\_photo

        bg\_label.place(x=0, y=0, relwidth=1, relheight=1)

    except:

        menu\_win.configure(bg="#e3e3e3")

    ttk.Label(menu\_win, text=f"Welcome, {username}!", font=("Segoe Script", 22, "bold"),

              bootstyle="dark").place(relx=0.5, y=50, anchor="center")

    y\_start = 130

    spacing = 70

    btn\_width = 30

    font\_size = ("Segoe UI", 12, "bold")

    ttk.Button(menu\_win, text="🔑 Generate Key", width=btn\_width, bootstyle="warning",

               command=lambda: gen\_key\_ui(username), style="TButton").place(relx=0.5, y=y\_start, anchor="center")

    ttk.Button(menu\_win, text="🔒 Encrypt File", width=btn\_width, bootstyle="primary",

               command=lambda: encrypt\_file**\_**ui(username)).place(relx=0.5, y=y\_start + spacing, anchor="center")

    ttk.Button(menu\_win, text="🔓 Decrypt File", width=btn\_width, bootstyle="success",

               command=lambda: decrypt\_file\_ui(username)).place(relx=0.5, y=y\_start + 2 \* spacing, anchor="center")

    ttk.Button(menu\_win, text="🧾 View Logs", width=btn\_width,bootstyle="info",

               command=view\_logs\_ui).place(relx=0.5, y=y\_start + 3 \* spacing, anchor="center")

    ttk.Button(menu\_win, text="🚪 Logout", width=btn\_width, bootstyle="danger",

               command=menu\_win.destroy).place(relx=0.5, y=y\_start + 4 \* spacing + 10, anchor="center")

    menu\_win.mainloop()

def gen\_key\_ui(username):

    generate\_key()

    messagebox.showinfo("Key Generated", "New encryption key generated.")

    log\_action(username, "Generated encryption key")

def encrypt\_file\_ui(username):

    key = load\_key()

    f = Fernet(key)

    path = filedialog.askopenfilename(title="Select file to encrypt")

    if path:

        with open(path, "rb") as file:

data = file.read()

        encrypted = f.encrypt(data)

        with open(path + ".enc", "wb") as file:

            file.write(encrypted)

        messagebox.showinfo("Encrypted", "File encrypted successfully.")

        log\_action(username, f"Encrypted: {os.path.basename(path)}")

def decrypt\_file\_ui(username):

    key = load\_key()

    f = Fernet(key)

    path = filedialog.askopenfilename(title="Select file to decrypt")

    if path:

        with open(path, "rb") as file:

            data = file.read()

        try:

            decrypted = f.decrypt(data)

            new\_path = path.replace(".enc", "\_decrypted")

            with open(new\_path, "wb") as file:

                file.write(decrypted)

            messagebox.showinfo("Decrypted", f"Saved as: {new\_path}")

            log\_action(username, f"Decrypted: {os.path.basename(path)}")

        except:

            messagebox.showerror("Error", "Invalid key or corrupted file.")

def view\_logs\_ui():

    if os.path.exists(LOG\_FILE):

        with open(LOG\_FILE, "r") as f:

            content = f.read()

        log\_window = tk.Toplevel()

        log\_window.title("Activity Logs")

        text = tk.Text(log\_window, wrap="word")

        text.insert("1.0", content)

        text.pack(expand=True, fill="both")

    else:

        messagebox.showinfo("Logs", "No logs available.")

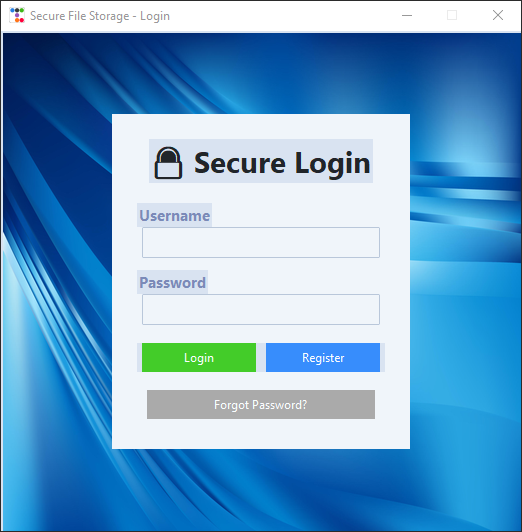
if \_\_name\_\_ == "\_\_main\_\_":

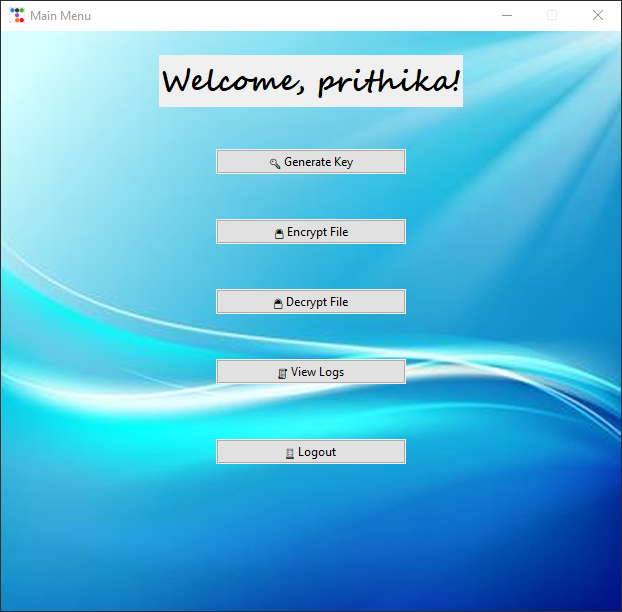
    root = ttk.Window(themename="morph")

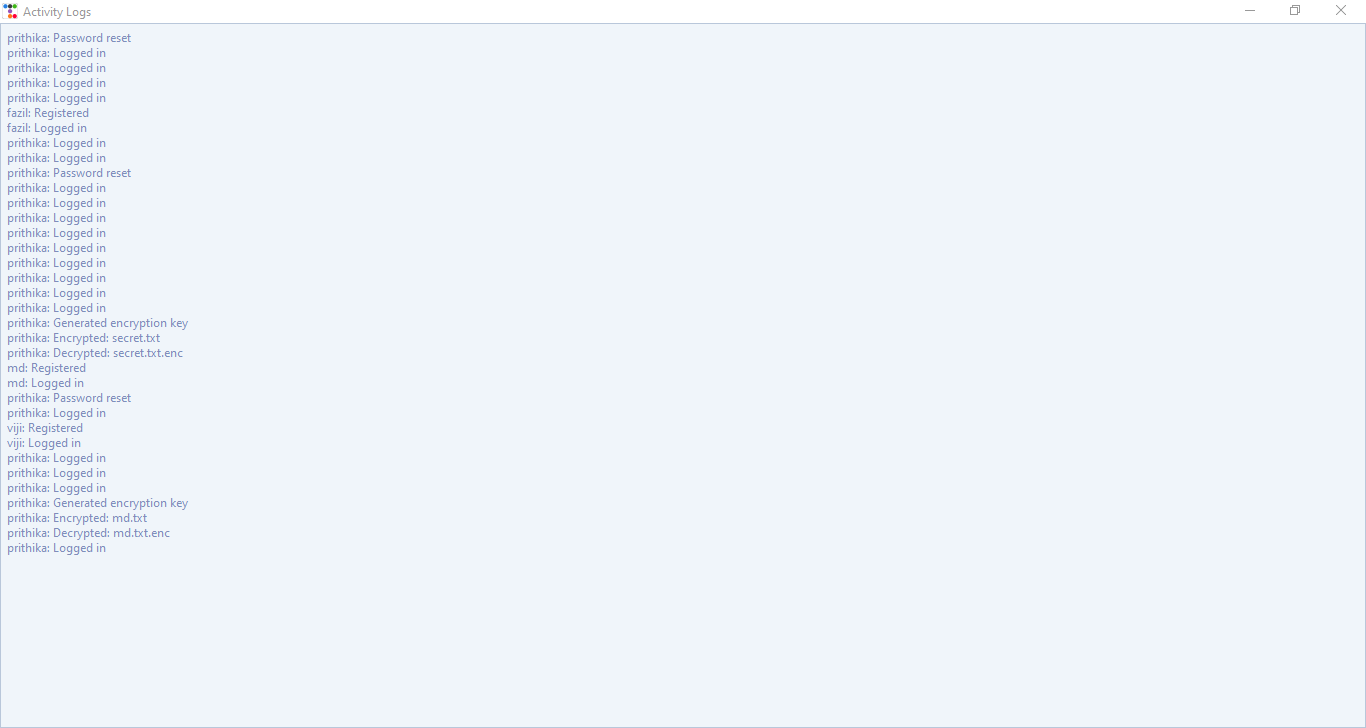
    app = LoginApp(root)

    root.mainloop()

**13.SAMPLE OUTPUT**







**14. Conclusion**

The **Secure File Storage System** provides a user-friendly yet robust approach to protecting sensitive files through secure authentication and encryption techniques. By combining Python's simplicity with powerful cryptographic libraries such as **bcrypt** for password hashing and **Fernet (AES-based)** for file encryption, the system ensures both data confidentiality and integrity.

This application emphasizes:

* Strong **user authentication** via hashed credentials.
* Reliable **file encryption and decryption** using symmetric keys.
* Secure **key management** and activity **logging** for accountability.
* An intuitive **GUI** built with ttkbootstrap for better user experience.

While the current version is effective for local and personal use, future enhancements—like per-user encryption keys, email-based password recovery, and cloud integration—will make it more scalable, secure, and enterprise-ready.

In summary, this project demonstrates how modern cryptography and Python can be leveraged to develop secure, efficient, and maintainable storage solutions suitable for real-world applications