A Documentation Report On

Secure File Storage System: Advanced Encryption and Decryption Tool

Submitted in partial fulfillment of the requirement for the award of the degree

Bachelor of Computer Applications (HONS.)

Academic Year 2024-25

Amaan Akbar Khojani

92100527008

Internal Guide

DR SUNIL BAJEJA

Marwadi University

Rajkot-Morbi Road, At & PO : Gauridad, Rajkot 360 003. Gujarat. India.

Marwadi University

Faculty of Computer Applications (FCA)

Certificate

This is to certify that the documentation work entitled

Secure File Storage System: Advanced Encryption and Decryption Tool

submitted in partial fulfillment of the requirement for the award of the degree of

Bachelor of Computer Applications (HONS.)

of the

Marwadi University

is a result of the bonafide work carried out by

Amaan Khojani 92100527008

during the academic year 2024-25

Faculty Guide	HOD	Dean

DECLARATION

I/We hereby declare that this project work entitled **Secure File Storage System: Advanced Encryption and Decryption Tool** is a record done by me.

I also declare that the matter embodied in this project is genuine work done by me and has been submitted to this University for the partial fulfillment of the requirement for the course of study.

Place : Rajkot

Amaan Khojani (92100527008) **Signature : _____**

TABLE OF CONTENTS

ABSTRACT

In today’s digital world, protecting personal and sensitive data is crucial. This project, Secure File Storage System, focuses on providing a secure and user-friendly way to encrypt and decrypt files while ensuring user credentials are stored safely.

The system uses AES (Advanced Encryption Standard) to securely encrypt files, ensuring confidentiality. For user authentication, Argon2 hashing is implemented to store passwords securely in the database, providing strong resistance against brute-force and cracking attempts.

The project features a login system with account creation, password reset options, and user-friendly error handling. Once logged in, users can encrypt and decrypt files directly through the interface, with encrypted files saved in the same location as the original. The goal of this project is to combine ease of use with reliable security measures, making it accessible for everyday users who want to protect their data without complex processes.

This report outlines the system's architecture, functionality, and the technologies behind its encryption and password management, demonstrating how the project achieves data security through practical implementation.

1. INTRODUCTION

In the digital age, protecting sensitive data has become a critical priority. With the rise of cyberattacks and data breaches, ensuring secure storage and transfer of files is essential for both individuals and organizations. This project, titled **"Secure File Storage System"**, is designed to provide a user-friendly yet robust encryption tool that enables users to safeguard their files with ease.

The system incorporates **AES (Advanced Encryption Standard)** — a widely trusted symmetric encryption algorithm — to securely encrypt and decrypt files. Additionally, **Argon2**, a highly regarded password hashing algorithm, is implemented to ensure secure storage of user credentials within the system's database. The combination of these technologies offers a strong foundation for maintaining data confidentiality and integrity.

This project not only aims to provide encryption functionality but also emphasizes a smooth and accessible user experience. The system features a graphical user interface (GUI) built with **PyQt5**, offering an intuitive layout that allows users to register, log in, and manage file encryption without needing advanced technical knowledge.

The **Secure File Storage System** is a practical solution for anyone looking to protect personal or professional files from unauthorized access — combining modern security standards with usability.

2. OBJECTIVES

The **Secure File Storage System** is designed with the following key objectives:

- Ensure Data Confidentiality:
  Protect sensitive files using **AES encryption** to prevent unauthorized access.

- User Friendly experience:
  Provide an intuitive **PyQt5** GUI with easy-to-navigate features like login, registration, file encryption, and decryption.

- Secure User Authentication:
  Implement **Argon2 password hashing** to securely store user credentials, reducing the risk of password breaches.

- Error Handling and Guidance:
  Include input validation, error messages, and help buttons to ensure a smooth user experience, even for non-technical users.

- Maintain File Integrity:
  Protect sensitive files using **AES encryption** to prevent unauthorized access.

3. Background and Literature Review

3.1 Introduction to Data Security

With the increasing reliance on digital storage, securing sensitive files has become a crucial concern. Traditional file storage methods lack encryption, making them vulnerable to unauthorized access, data breaches, and cyberattacks. To mitigate these risks, encryption-based security solutions are widely adopted.

3.2 AES Encryption for Secure File Storage

The **Advanced Encryption Standard (AES)** is a widely accepted symmetric encryption algorithm known for its strong security and efficiency. It is used by governments and organizations worldwide to protect classified information. AES provides confidentiality by encrypting files using a secret key, ensuring that only authorized users can decrypt and access the stored data.

3.3 Argon2 for Secure Password Storage

To protect user credentials, this system implements **Argon2**, a key derivation function designed to securely hash passwords. Argon2 is resistant to brute-force and dictionary attacks due to its memory-hard design, making it highly effective for preventing unauthorized access to stored credentials.

3.4 Graphical User Interface (GUI) with PyQt5

Usability plays a significant role in security applications. This project employs **PyQt5** to create a user-friendly interface that allows users to encrypt and decrypt files with ease. The GUI ensures accessibility while maintaining robust security mechanisms in the background.

3.5 Existing File Security Solutions

Several existing file security applications, such as **VeraCrypt** and **AxCrypt**, provide encryption functionalities. However, these tools often require complex configurations or lack user-friendly interfaces. This project aims to bridge the gap by offering **strong encryption with a simple and intuitive design**, making file security accessible to all users.

4. System Design and Architecture

4.1 System Overview

The Secure File Storage System is designed to ensure user data is securely encrypted and accessible only to authorized users. It combines AES-256 encryption for files and Argon2 hashing for user passwords, providing a balance of strong security and performance.

The system follows a modular architecture, separating the user interface, authentication, encryption, and file management processes to improve maintainability and scalability.

4.2 Architectural Components

The system consists of the following key components:

- **User Authentication Module**:
  Handles user registration, login. It uses Argon2 hashing to securely store passwords and validate user credentials.

- **Encryption Module**:
  Uses AES-256 (Advanced Encryption Standard) to encrypt and decrypt files. The encryption key is derived securely using a combination of the user's password and a random salt.

- **File Manager Module**:
  Manages file selection, encryption, decryption, and saving of files. It supports drag-and-drop functionality to enhance user experience.

- **Graphical User Interface (GUI)**:
  Built with PyQt5, providing a clean, modern, and intuitive interface with dark mode, rounded buttons, and sidebar navigation.

4.3 System Workflow

1. **User Registration**:
    - A new user creates an account by entering a username, password, security question, and answer.
    - Passwords are hashed using Argon2, and user data is stored securely in the SQLite database.
2. **User Login**:
    - The user logs in using their registered credentials.
    - The entered password is hashed and compared against the stored hash.
    - If successful, the user gains access to the encryption/decryption features.
3. **File Encryption**:
    - The user selects a file.
    - The file is encrypted using AES-256 and saved with an .enc extension.

 4. **Logout**:

- The user can securely log out of the system to prevent unauthorized access.
- This clears the session and returns the user to the login screen.

 5. **Help Button**:

- A dedicated help button provides instructions on how to use the system.
- It ensures users understand how to register, log in, encrypt, decrypt, and log out without confusion.

5. Project Implementation

5.1 System Setup

The Secure File Storage system is developed in Python using the PyQt5 library for the graphical user interface (GUI) and SQLite for database management. The project folder structure includes the following key files:

- **app.py**: Manages the user interface and controls the login, registration, and password reset flows.
- **auth.py**: Handles user authentication, password hashing (using Argon2), and security question verification
- **file_manager.py**: Contains functions to encrypt and decrypt files using AES encryption.
- **crypto.py**: Implements AES encryption and decryption logic with secure key handling.
- **database.py**: Initializes the SQLite database and manages user and file data storage.

5.2 User Registration

1. **Input Validation:**
    - Users provide a username, password, security question, and security answer.
    - Both the password and security answer are hashed using the Argon2 algorithm for secure storage.
    -
2. **Database Integration:**
    - The registration data is inserted into the SQLite database, ensuring each username is unique.

3. **Encryption Key Setup:**
    - A unique AES encryption key is generated and securely linked to the user for file protection.

5.3 User Login

1. **Password Verification:**
    - The entered password is compared against the stored Argon2 hash to authenticate the user.

2. **Access Control:**
    - If login is successful, the system retrieves the user's AES encryption key, enabling file encryption and decryption.

5.4 File Encryption

1. **File Selection:**
    - Users select a file from their system using the GUI.

2. **AES Encryption:**
    - The file's content is encrypted using AES in EAX mode (providing both encryption and integrity checks).
    - An encrypted file is saved with the original file name plus a .enc extension.

3. **Status Feedback:**
    - The system displays a confirmation message upon successful encryption.

5.5 File Decryption

1. **File Selection:**
    - Users select an encrypted .enc file from their system.

2. **Password Authentication:**
    - The user is authenticated, ensuring only authorized access to the original content.

3. **Decryption Process:**
    - The AES cipher decrypts the file and restores the original content, saving it with the .decrypted extension.

4. **Status Feedback:**
    - A success or failure message is displayed based on the decryption result.

5.6 Logout

- Users can securely log out of the system, which clears any session data and returns to the login screen.

5.7 Help Section

- A help button is provided in the GUI to guide users on encryption, decryption procedure.

6. Results and Testing

6.1 System Functionality Testing

The Secure File Storage system underwent rigorous testing to ensure smooth functionality across all features. Each module was tested independently and then integrated to evaluate the complete system workflow. Below are the key results:

6.1.1 User Registration and Login

✅ **Test Case:** Registering a new user with valid details.
Result: Successfully created the user and stored the hashed password and security answer securely in the SQLite database.

✅ **Test Case:** Logging in with the correct username and password.
Result: User authenticated successfully, granting access to file encryption and decryption features.

✅ **Test Case:** Attempting login with an incorrect password.
Result: Login denied with an error message ("Incorrect password").

✅ **Test Case:** Attempting to register with an existing username.
Result: Registration denied with an error message ("Username already exists").

6.1.2 File Encryption

✅ **Test Case:** Selecting a file and performing encryption.
Result: File successfully encrypted and saved with a .enc extension.

✅ **Test Case:** Attempting to encrypt the same file multiple times.
Result: Each encrypted file produced a unique output due to random AES nonce generation.

✅ **Test Case:** Attempting to encrypt a non-existent or unsupported file.
Result: System gracefully handled errors, displaying a "File not found" message.

6.1.3 File Decryption

✅ **Test Case:** Decrypting a file with the correct password.
Result: File successfully decrypted, restoring the original content with a .decrypted extension.

✅ **Test Case:** Attempting to decrypt a file with the wrong password.
Result: Decryption failed with an error message ("Decryption failed: Incorrect password").

✅ **Test Case:** Trying to decrypt a non-encrypted file.
Result: System detected the invalid file and displayed an error.

6.2 Performance Testing

The system was tested with files of different sizes and types to evaluate performance:

File Type	File Size	Encryption Time	Decryption Time
Text File	14 KB	0.2s	0.2s
Image (PNG)	2.5 MB	0.8s	0.9s
PDF Document	4 MB	1.1s	1.2s
Video (MP4)	50 MB	8.4s	8.6s

✅ **Observation:** The system maintained consistent performance and handled files efficiently without significant delays.

6.3 Security Evaluation

The system was also assessed for resilience against common security threats:

- **Password Storage:** Argon2 hashing ensures secure, irreversible password storage.
- **File Integrity:** AES encryption in EAX mode guarantees both confidentiality and data integrity.
- **Unauthorized Access:** Decryption requires the correct user credentials, ensuring only authorized users can access data.

7. Conclusion

The Secure File Storage system demonstrates a practical, user-friendly, and secure approach to protecting sensitive data. By combining AES encryption for file security and Argon2 hashing for password protection, the system ensures that both stored files and user credentials remain safeguarded against unauthorized access.

The project successfully implemented core functionalities such as:

- User authentication with secure password hashing.
- File encryption and decryption using AES, ensuring data confidentiality.
- Password reset via security questions, enhancing user account recovery.
- Error handling and user guidance to ensure a smooth experience, even during incorrect operations.

The system's performance proved efficient across various file types and sizes, demonstrating its ability to handle real-world use cases. Furthermore, the security evaluation confirmed resilience against common attack scenarios such as password theft and unauthorized file access.
This project bridges the gap between cybersecurity principles and user accessibility, making secure file management more approachable without compromising safety.

---

8. Future Improvements
While the current system meets its objectives, several enhancements could elevate its functionality, usability, and security:

8.1 Enhanced Security Measures

- Two-Factor Authentication (2FA): Implementing an OTP-based or email verification system to provide an extra layer of user authentication.
- Multi-Algorithm Support: Introducing other encryption algorithms (e.g., RSA or ChaCha20) for more flexible encryption options.

- Secure File Deletion: Implementing secure file shredding to ensure deleted files are irrecoverable.

8.2 User Experience Improvements

- Drag-and-Drop Support: Allowing users to drag files directly into the interface for quicker file selection.
- Progress Bars: Displaying encryption/decryption progress for large files to improve user feedback.
- Dark/Light Mode Toggle: Enhancing customization options for user preference.

8.3 Cross-Platform Compatibility

- Linux and macOS Support: Adapting the system for wider usability across different operating systems.
- Mobile Version: Creating a lightweight mobile version for file encryption on the go.

8.4 Backup and Cloud Integration

- Automatic Backups: Allowing users to store encrypted backups securely in a separate directory.
- Cloud Storage Support: Integrating secure upload options to platforms like Google Drive or OneDrive, ensuring encrypted files are stored remotely and safely.

8.5 Advanced Error Handling and Logging

- Detailed Error Logs: Capturing more detailed logs for failed attempts and errors to help with debugging and potential forensic analysis.
- Auto-Recovery for Failed Operations: Ensuring interrupted encryption/decryption tasks resume without corrupting files.

This project has built a strong foundation for secure file storage. With these potential upgrades, the system can evolve into an even more robust, versatile, and accessible solution for protecting data in a digital world.

9. References

1. **PyQt5 Documentation** — *Building GUI Applications with PyQt5*.
    - Retrieved from: https://www.riverbankcomputing.com/software/pyqt/intro

2. **SQLite Documentation** — *SQLite Database Engine Reference*.
    - Retrieved from: https://sqlite.org/docs.html

3. **PyCryptodome Library** — *Python Cryptography Toolkit Documentation*.
    - Retrieved from: https://pycryptodome.readthedocs.io

4. **Argon2 Documentation** — *Password Hashing with Argon2*.
    - Retrieved from: https://github.com/P-H-C/phc-winner-argon2

5. **GitHub** — *Version Control and Project Management*.
    - Retrieved from: https://github.com

10. GitHub Repository

To ensure transparency and easy access for future improvements, the entire project — including source code, database setup, and encryption modules — is hosted on GitHub. This repository serves as both a demonstration of the system's functionality and a foundation for potential enhancements.

The project repository can be accessed at:

https://github.com/threathawk05/Secure-File-Storage-System

The repository includes:

- **Complete Source Code:** All Python files (app.py, auth.py, file_manager.py, crypto.py, database.py).
- **Database Structure:** SQLite database schema for user credentials and secure storage.
- **README.md:** A brief guide on how to set up, run, and extend the project.
- **Screenshots:** Demonstrations of the GUI, encryption, and decryption process.

This ensures the project remains accessible for further development and evaluation by professors and potential collaborators.