CHAPTER 1

INTRODUCTION

Because it makes it possible for users to share private information store personal data and exchange documents instantly digital communication has emerged as one of the major significant facets of daily life. Additionally while digital interaction is comfortable there are significant security risks such as cyberattacks data breaches unauthorized access and data manipulation. Information security must be given the highest priority during both transmission and storage due to the rapid growth of digital data. Conventional encryption techniques ensure a certain level of security but they are frequently insufficient in terms of adaptability robustness and usability for the modern digital environment. As a result there is a huge market need for secure systems that are both effective and easy to use.

The project suggests a Secure File Storage and Transmission System that uses hybrid cryptography to guarantee the security of both text and files in order to meet this requirement. Because the system combines RSA for secure key management with AES for quick data encryption even large files can be swiftly deleted without compromising security. One of the most important enhancements is the addition of the encryption password feature which adds an extra degree of security by allowing the sender to select a password that the recipient must use to decrypt. Additionally Gmail integration makes the process more convenient and enjoyable for users by enabling secure communication between users and automatic notifications. Through password-based decryption controlled access authentication and secure storage Refrain from the fact that a certain part can fish one that doesn't just authorize users can access sensitive data. All things considered the project combines reliable cryptographic technologies with a user-centered design in a way that results in a reliable method of secure digital communication despite growing cyberthreats.

1.1 Problem Statement

The primary disadvantages of internet services however are the risks associated with using the internet to transmit documents and reveal personal information in the current digital era. These dangers include identity theft data breaches and illegal access. While symmetric encryption is faster and less computationally demanding it cannot ensure the security of key sharing. Conversely asymmetric encryption is more secure for managing keys but is not appropriate for large files because of its slowness rendering earlier encryption techniques insufficient. A safe and practical substitute is therefore required due to the challenges associated with encryption techniques particularly for the markets less tech savvy consumers. The most basic features such as password-based decryption secure communication and automatic email alerts are occasionally absent from the systems that are already in place. Therefore in order to make data storage and transfer safe and effective a comprehensive security solution that includes password-based decryption secure file storage Gmail notifications hybrid cryptography using both AES and RSA and an intuitive interface must be developed.

1.2 Project description

The project is about creating a platform that is both secure and easy to use which will enable the usage of hybrid cryptography for the protection of sensitive files and texts. The entire process uses RSA to securely handle the keys and AES to encrypt the data quickly ensuring that the data is kept private both in transit and at rest. When a user uploads a document or types something the platform encrypts the entire document using AESS before wrapping the AES key in RSA. Additionally because the sender can generate a unique encryption password that the intended recipient must provide in order to access the data the platform further strengthens security. An extra degree of security guarantees that the encrypted files can only be accessed with the right password even if they are discovered along the way.

1.3 Objectives

1. The projects primary goal is to create secure environments for text and file encryption and decryption. Consequently the greatest amount sensitive and classified information will always be safe both during transmission and storage.
2. AES and RSA will be used in hybrid cryptography AES handles quick and effective data encryption while RSA guarantees secure key management effectively keeping the data from being accessed by unapproved parties.
3. In addition to the current security measure a new feature for the encryption password function will be implemented allowing the sender to generate a unique password that the recipient will enter during the decryption process.
4. Users will thus be kept up to date on encryption file sharing and system updates thanks to the integration of Gmail support for automated notification or alert sending.
5. Only users who have been verified and granted permission will be able to view download or decrypt the files to do so guarantee total protection of data confidentiality integrity and access.
6. To develop an extremely user-friendly interface that will enable users of all technical skill levels to upload encrypt and retrieve their secure data with ease.

1.4 Scope

The assignment will entail the creation of a system, which will be both secure and user-friendly. The system will allow the users to encrypt and decrypt files and texts with the help of hybrid cryptography, which is a combination of AES and RSA. The system will also provide the option of custom encryption passwords as an additional safety measure and will use Gmail for notifying or alerting the users. The scope of the project will include user registration, login authentication, file uploading, text encryption, secure storage, and controlled data retrieval. The confidentiality, integrity, and access control are the main target of the project, and at the same time, a user-friendly interface that makes these advanced security measures easy to use for all customers is the main product. Besides, cloud integration, biometric authentication, and large-scale deployment are not available at the moment as they are still considered advanced features.

1.5 Purpose

The goal of this project aims to develop a safe and effective system that uses powerful hybrid cryptography techniques to safeguard both text and files. It seeks to give users a reliable platform that allows users to safely encrypt store and retrieve sensitive data without worrying about illegal access or data manipulation. The project maintains high performance while guaranteeing data confidentiality by combining the speed of AES with the secure key management of RSA. The system also prioritizes providing a seamless and intuitive user experience enabling encryption and decryption even for non-technical users. The ultimate goal of this project is to develop create a solid solution that improves data privacy fortifies digital security and helps users confidently manage their information.

CHAPTER 2

LITERATURE SURVEY

According to research in the area of information security as cyber threats grow so does the need for enhanced protection for digital data. Studies on symmetric encryption such as AES show that it is a very successful technique for securing large volumes of data but it has a problem with key sharing which makes it less secure. However asymmetric techniques like RSA provide a better key-management system at the expense of being slower and less effective for encrypting large volumes of data. Meanwhile hybrid cryptography which combines the security of asymmetric keys for handling with the symmetries of encryption is becoming increasingly popular in recent literature. The current literature continues to discuss integrity checks access control and authentication as the primary means of guaranteeing complete data protection. Therefore the project will employ a Hybrid AES-RSA model which will ensure a safe and effective encryption for texts and files in accordance with the suggested guidelines of contemporary security research.

[1] RashiDhagat, Purvi Joshi, "Cloud-Based Secure Storage for Group Data Sharing", Year 2016

The topic of this article is how a group of people can use the cloud as a storage medium to share and store their data in a secure way. Using group signatures and encryption is one of the writers recommendations. The main benefit of the suggested system is that data owners can submit their files while keeping their true identities hidden from other cloud users. PKA stands for Public Key Agreement.

[2] Bilal Habib, Bertrand Cambou, DuaneBooher, Christopher Philabaum, "A Secure and Addressable Public Key Infrastructure for Cloud Storage", Year 2017

This study offers a fresh and original approach to the implementation of public key infrastructure. The primary drawback of the PKI system is the preservation of the mathematical relationship between the public and private keys. The paper presents a novel PKA scheme with addressable features. The suggested approach uses addressable cryptographic tables to remove the mathematical relationship between public and private keys. Cloud storage uses key aggregation cryptography to enable safe data exchange.

[3] Shakeeba S. Khan, Prof. R. R. Tuteja, "A Multilevel Encryption Approach for Secure Cloud Data Exchange", Year – 2015

The Multilevel Encryption and Decryption algorithm is the suggested method which guarantees total data security for only authorized users. Layer by layer an intruder could decrypt the data but without the right key this is practically impossible. Additionally because there are numerous encryption and decryption processes involved it would be a very time-consuming process. Cryptographic techniques for safe data exchange in a cloud setting.

[4] Tulip Dutta, Amarjyoti Pathak, "Efficient Key Management for Encrypted Cloud Data Sharing", Year 2016

The sharing of a secret key among the users who will have access permissions is the primary topic of this paper. It also discusses the problem of using a single key to encrypt all data while assigning distinct keys to various files. In the paper the authors offer a solution that addresses the issue by using key aggregation. Key aggregation is a technique where a single aggregated key is used for the entire decryption process even though different keys that correspond to different data files are used. Javas key store data structure is used to implement the Advanced Encryption Standard (AES) encryption algorithm. The study discusses cloud security using identity-based encryption MD5 hashing and a third-party auditor.

[5] Mr. Rohit Barvekar, Mr. ShrajalBehere, Mr. Yash Pounikar, Ms. Anushka Gulhane, "Enhancing Cloud Security Through Efficient Cryptographic Methods", Year 2018

The security measures will first and foremost ensure that no one improperly uses confidential information at the implementation level thereby establishing system reliability. High speed: Compared to the standard procedure the suggested method will encrypt and decrypt data using appropriate keys at a rate that is many times faster. Cryptographic algorithms provide security in cloud computing.

2.1 Existing System

Asymmetric or symmetric encryption is the primary method employed for data security in many modern systems which has serious drawbacks. Although symmetric encryption techniques such as AES are quick and appropriate for big files they have unsafe key-sharing procedures that leave the data exposed in situations where the key is compromised. Although asymmetric techniques like RSA provide safer key management they are sluggish and ineffective when dealing with large files or volumes of text. These systems have trouble striking a balance between security and performance since they rely on a single encryption technique. Because of this current solutions frequently fall short of offering total security leaving private data vulnerable to tampering illegal access and data breaches.

2.1.1 Disadvantages of Existing System

1. Systems that only use symmetric encryption run the chance of key-sharing which might result in illegal access if the key is compromised.
2. Large files cannot be encrypted with asymmetric encryption alone which lowers system performance.
3. Threat actors can more easily infiltrate or benefit from the system when only one encryption technique is used.
4. Improper key management increases the likelihood of data loss and serious security breaches.
5. Performance problems and delays could result from the systems inability to manage big files effectively.
6. Sensitive data is vulnerable because the current method is unable to offer a balanced combination of speed and robust security.

2.2 Proposed System

By employing hybrid cryptography to handle both text and file encryption the suggested system offers a more effective and secure method. The system ensures greater security without compromising performance by combining RSA for secure key protection and AES for quick data encryption. The system automatically encrypts user-inputted text or uploaded files and safely stores it with the protected keys. While avoiding the slow performance issues related to using asymmetric encryption alone this hybrid model addresses the key-sharing issue present in symmetric encryption. Apart from its improved security the system prioritizes a seamless and intuitive user experience.

The interface makes it simple for users to register log in upload files encrypt text view stored data and retrieve decrypted material. Only authorized users can access sensitive data thanks to the backends management of encryption procedures secure storage and authentication. The suggested system offers a dependable way to protect text and file data in a contemporary digital environment by combining strong cryptographic techniques with transparent access control and easy interaction.

2.2.1 Advantages of Proposed System

1. RSA secure key protection is combined with AES fast encryption to provide extremely high security.
2. An extra degree of security during decryption is provided by the ability for users to select their own encryption passphrase.
3. This can include ongoing activities finished tasks and planned activities among other things.
4. Excellent performance and speedy processing even with the largest files are guaranteed. By using secure RSA-based key management the risk of key sharing is totally eliminated.
5. Both laypeople and tech-savvy people will find the design to be comfortable and accommodating.
6. According to the authorization code passwords and usernames must be supplied for logging granting only verified users access to the data.

CHAPTER 3

SOFTWARE REQUIREMENTS SPECIFICATION

3.1 Requirement Analysis

The requirement analysis objective is to ascertain what the system needs to manage file and text encryption safely. It involves realizing that in order to protect sensitive data strong cryptographic techniques secure key management and reliable user authentication are essential. The system must maintain integrity and confidentiality while ensuring speedy processing secure storage and easy retrieval. Usability performance and scalability requirements are also considered in order to provide each user with a smooth and efficient experience.

3.1.1 Functional Requirements

1. User Verification

Registration sign-in and sign-out features must be included in a safe and reliable user authentication system. Before allowing access to any encryption or file-related operations the system will verify the unique credentials that each user is given. This requirement ensures accountability prevents unauthorized users and protects the user environment.

2. File Uploading

Users should have the ability to upload various types of files such as PDFs images and texts through a secure interface. Before the upload is permitted the application must verify the files size format and authenticity. The files will be uploaded processed securely and then encrypted to guarantee that no private information is revealed at any point.

3. Text Encryption

Users must be able enter plain text data into the application which will then use secure cryptography techniques to convert it into encrypted text. This feature is designed to essential for protecting private and sensitive communications and data from being intercepted while in transit or from unwanted access while being stored.

4. File Encryption Using AES

The Advanced Encryption Standard (AES) the only encryption technique unquestionably renowned for its strength and effectiveness will be applied to all uploaded files. Because AES encrypts and decrypts data quickly at the same time it is both robust and quick making it appropriate for data of any volume. Additionally, unyielding security by means of encryption is provided by AES; not easily breakable for both coded and non-coded attacks.

5. Encryption Using RSA Keys

To safely encrypt the AES key created for every file the system must employ RSA encryption. By combining the speed of symmetric encryption with the safe key management of asymmetric encryption this hybrid encryption technique makes ensure that encryption keys are securely shared with only those who are authorized.

6. Unique Password for Encryption

Furthermore the encryption keys the system needs a password so that the user specifies for both encryption and decryption. This password acts as an extra security precaution that restricts unauthorized users from accessing the encrypted data without the correct password even in the event that cryptographic keys are compromised.

7. File Download and Decryption

Only authorized users must be able to download encrypted files and safely decrypt them. The system must ensure that the decrypted file is identical to the original with no data loss or corruption and no modifications to the files format content or structure.

8. Text Decryption

By confirming the provided password or decryption key the system must enable safe text decryption. The encrypted text will be restored to its original readable form with data accuracy and confidentiality guaranteed once the user is granted authorization.

9. Gmail Notification Assistance

In order to inform users of important system activities such as successful file encryption file sharing login attempts and decryption completion the system must be able to automatically send email notifications through Gmail. These alerts will improve user awareness increase transparency and improve the identification of malicious activity.

10. Data Integrity Protection

The system should ensure that the stored data and encrypted files wont be accessed by unauthorized users and wont be changed in any way. Integrity verification techniques will be implemented to detect manipulation and ensure that the data is authentic throughout transmission and storage.

11. Dashboard Display

The system must provide a user-friendly an interactive dashboard that enables users to efficiently organize their encrypted files and texts. The dashboard should allow users to view upload history encrypt and decrypt files download files and check notifications making the process easy to use and system interaction effective.

3.1.2. Non Functional Requirements

1. Safety

Providing minimal security is supposed to be initiated in the process of processing data. The system should deliver strong security throughout the data handling process so that there is no risk of way to break into the system such as through unencrypted data or the exposure of cryptographic keys. Multiple layers of protection should be applied to the system aims to lower the risks related to unauthorized access data leakage tampering and even the worst-case scenario of malicious attacks where encrypted data is already compromised.

2. Actual Performance

The real performance. The platform must always be highly responsive because it must be operating at a very high level. Encryption and decryption processes must be finished quickly and file uploads and downloads shouldnt be significantly delayed. Even with large file sizes

the system should be capable of handling requests from multiple users at once without experiencing any slowdown.

3. Usability

The application interface should be designed with the highest level of transparency and simplicity so that even non-technical users can perform encryption and decryption tasks with ease. To reduce mistakes and boost satisfaction there should be labeling guiding messages and a smooth activity flow.

4. Trustworthiness

For users to be able to rely on the system for the safe storage and retrieval of their data it must operate faultlessly and reliably. It should be able to withstand crashes malfunctions and unforeseen errors providing users with the assurance that the encrypted data will always be available when needed.

5. Flexibility

In order to accommodate evolving needs over time the platform must be both flexible and scalable. Without requiring significant architectural changes or interruptions to current services the platform should be able to accommodate future growth including more users higher data volumes and additional security features.

6. Availability

The availability. Encryption decryption and file access should all be available whenever users need to access the system. To minimize service interruption and user inconvenience system maintenance should be performed infrequently and in a way that is extremely well-managed and communicated beforehand.

7. Maintenance

If one wants a piece of code to work with another program some documentation is necessary but too much documentation can be problematic. To have some maintainability code from even the simplest script must be reviewed. Long-term maintainability faster bug fixes and smooth integration of the improved security mechanisms are all made possible by this kind of design.

8. Integrity of Data

A mechanism that ensures the accuracy completeness and unaltered state of encrypted data will support its lifecycle. Integrity validation techniques will be used to identify unauthorized data modifications resulting in the conclusion that decrypted files and texts are identical to the originals.

9. Compatibility

The platform should function uniformly across major operating systems devices and web browser variations. It should be able to deliver the same performance and user experience regardless of the users surroundings making it usable and accessible to a large number of users.

10. Verifiability

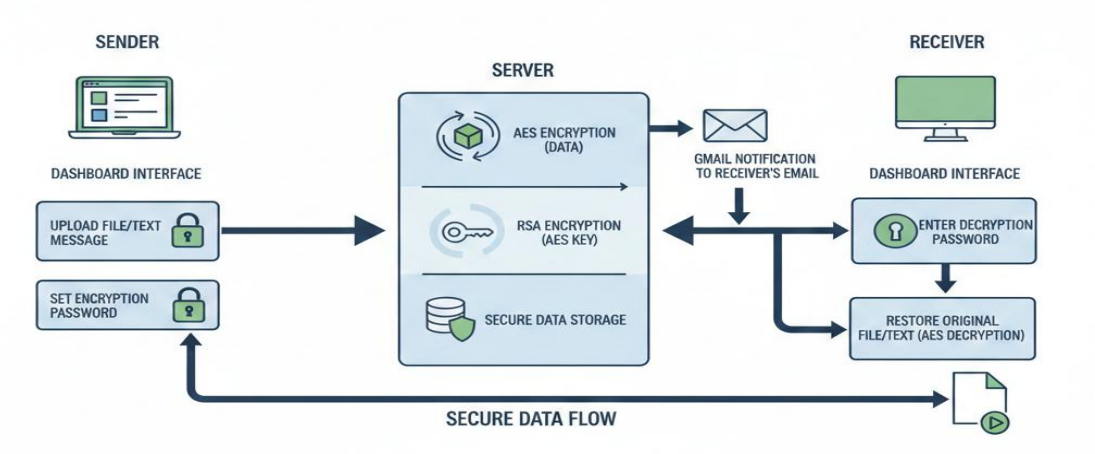
The system should create logs to document events such as file uploads downloads encryption and decryption and login attempts for verification purposes. These logs can be utilized for error tracking security analysis auditing and monitoring making it easier for administrators to identify questionable activity and increase system dependability.

3.2 System Design

The systems straightforward and modular design allows it to safely handle both file and text encryption. Users communicate through a web interface that enables users to enter text for encryption or upload files. After the input is submitted the backend handles authentication processes it encrypts it using the proper cryptographic algorithms and either stores or returns the encrypted output. To ensure secure access to both files and text decryption is only carried out after the user has been verified. The system is kept simple to maintain scalable and effective for safeguarding various kinds of data by keeping the interface processing and storage layers apart.

3.2.1 System Architecture

The system uses a client-server architecture in which all system operations take place on the server and users communicate via a web-based client. Users can register log in upload files and enter text for encryption and decryption on the client side. These requests are sent to the server where the backend manages text and file operations applies hybrid cryptography (AES for data encryption and RSA for AES key security) and handles authentication. While user information and logs are kept in the database encrypted files and any stored encrypted text along with their metadata are stored in secure storage. Before returning the chosen file or text to the client the server first verifies the user then uses RSA to decrypt the AES key. The system is safe scalable and simple to maintain thanks to this layered architecture which clearly divides the user interface application logic encryption modules and storage.



3.2.1.1 System Architecture

3.2.2 User Flow

The user begins by registering or logging into the system, following which users can access a simple dashboard to upload files or enter text for encryption. Once the user selects a file or types their message, they can choose to set a custom encryption password before submitting it. The system then validates the input, encrypts the data using AES, secures the AES key with RSA, and stores everything safely. A Gmail notification may be sent to the intended receiver to inform them that encrypted data is available. When the receiver logs in, they can view the encrypted item and, if required, enter the correct password to proceed with decryption. The system then retrieves the encrypted content, decrypts it securely, and provides the restored file or text back to the user. This flow ensures smooth navigation, strong security, and controlled access at every step.

3.2.3 Hardware Requirements and Software Requirements

Hardware Requirements

- Processor : Ryzen 7
- RMA : 8 GB
- Storage : 512 GB SSD

Software Requirements

- Operating System : Windows 11
- Programming Language : Python
- Development Environment : VS Code
- Libraries : Flask

3.3 Detailed Design

The system is modular in design with various parts managing data retrieval encryption storage and user interaction. While the backend handles authentication and encrypts data using AES and RSA for secure key protection users can register log in upload files or enter text through the interface. All text and encrypted files are safely stored for later access along with their metadata. To guarantee secure data handling decryption is only done following appropriate user verification. User data encrypted data references and activity logs are kept up to date in a central database facilitating seamless system operation and simple maintenance.

3.3.1 Data flow diagram

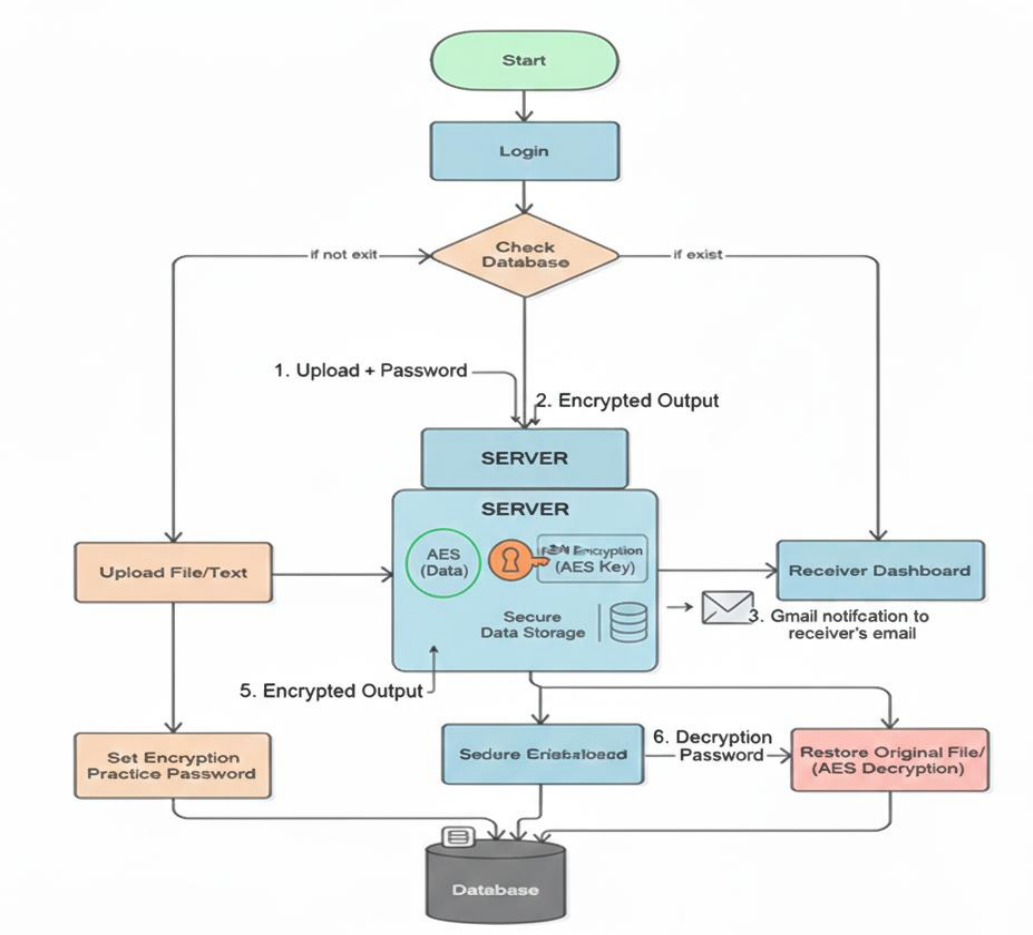


Figure 3.3.1.1 Data Flow Diagram

1. **User Interaction**

   Interaction between users. When a user logs in or registers on the system the process starts. This guarantees that the platform can be retrieved only by authenticated users.

2. **Dashboard Access**

   The dashboard which serves as the main control panel is accessed by the end user following successful authentication. They can then choose from a range of options including text encryption file uploads and stored data downloads.

3. **Text Encryption Process**

   The user inputs plain text into the system after choosing the text encryption option. Before being saved or shown the text is securely transformed into encrypted form by the encryption module.

4. **File Encryption Process**

   The process of encrypting files. When a user uploads a file it is processed by the system and sent to the encryption module. The file content is encrypted using AES and secure key management is ensured by using RSA to encrypt the AES key.

5. **Hybrid Cryptography Module**

   Module on Hybrid Cryptography. The hybrid cryptography module combines file and text encryption functions. This module is in charge of implementing AES + RSA protection ensuring that keys are handled securely and data is kept private.

6. **Secure Storage**

   Storage that is safe. After encryption the data (text or file) and its encrypted key are stored in safe location. This keeps the data safe even in the event that storage is compromised and stops unwanted access.

7. **Download & Decryption**

   Download then decrypt. The system retrieves the encrypted file when a user requests to download it decrypts it using the encrypted key that has been stored and then returns the original data to the user. To ensure security decryption doesnt happen until the users identity has been confirmed.

3.3.2 Use Case Diagram

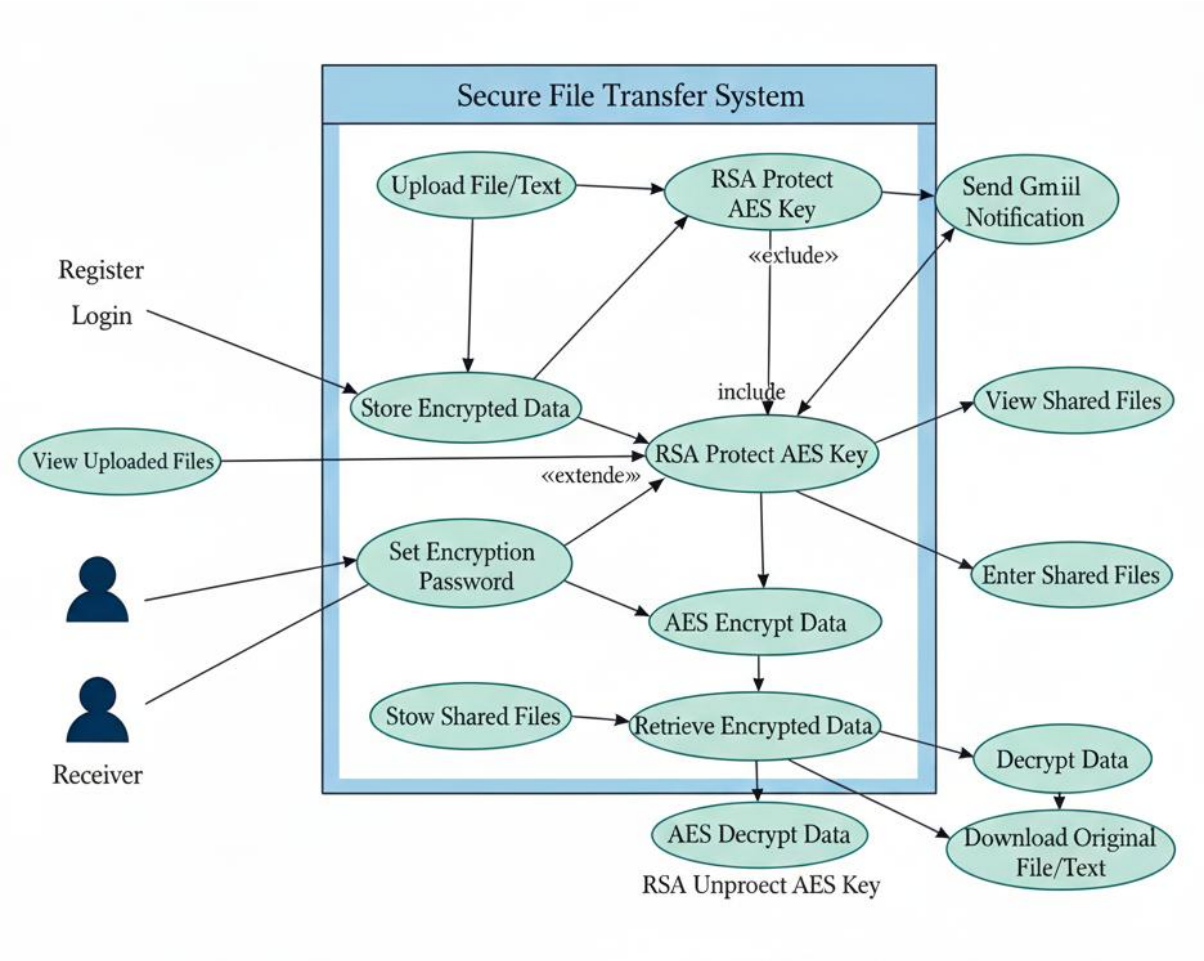


Figure 3.3.2.1 Use Case Diagram

The users interaction with the text and file encryption system is depicted in the use case diagram. To access the platform securely the user must first register and log in. They can carry out crucial tasks like uploading files for encryption and encrypting text after they have been verified. The central text and file encryption module manages all of these tasks and uses hybrid cryptography to safeguard user data. The primary functions provided by the system and the roles of the user are highlighted in this diagram.

1. User Interaction

Every action on the platform is initiated by the user who is the main actor. When the user opens the system and proceeds to sign in or register the interaction starts. These procedures establish a secure basis for all subsequent operations by authenticating the user and ensuring that only those who are authenticated are permitted to use the encryption and decryption features.

2. Dashboard Access

Following a successful authentication the user will be taken to the dashboard which serves as the primary control panel for all operations. The user can encrypt text upload files check notifications and download previously saved files here. Additionally the Gmail option is available to notify the recipient when new encrypted data is shared.

3. Method of Text Encryption

When a user chooses to encrypt text they enter their plain text in the designated input field. After receiving this text the system sends it to the encryption module where it is securely converted to ciphertext. The user has the option to notify the intended recipient via Gmail that the encrypted text is available.

4. File Encryption Procedure.

After the user uploads a file the encryption module receives it after the system verifies its authenticity. AES encrypts the file content and RSA encrypts the AES key for efficient key management. Following the encryption the system may notify the recipient via Gmail that an encrypted file has been uploaded.

5. Module on Hybrid Cryptography

All text and file encryptions are handled by the hybrid cryptography module. This module applies AES + RSA protection which guarantees their confidentiality and safe handling of their keys. Additionally Gmail notifications can be set up automatically if any of the process steps require a user to verify or distribute information.

6. Safekeeping

After that the encrypted text or file and the protected key will be stored securely by the system. As a result even if data is accessed incorrectly it wont be lost or compromised. Additionally users may receive notifications from Gmail about the safe storage of their data.

3.4 Algorithm used

1. AES (Advanced Encryption Standard)

Almost all contemporary security systems use AES a very potent symmetric encryption algorithm to safeguard sensitive information. Because it uses the same secret key for encryption and decryption it can handle big files and data streams quickly and with minimal resource usage. The highest key provides the best defense against brute-force attacks and AES permits the use of 128- 192- and 256-bit keys. Substitutions permutations mixing and key addition are among the transformation rounds that follow the techniques division of the data into blocks of a preset size. Without the correct key these operations are so successful that the attackers have very little chance of decrypting the ciphertext. Because AES combines speed reliability and a high level of security it was chosen for this projects text and file encryption. Because of its capacity to process large amounts of data it is the best primary encryption technique for the system.

2. RSA (Rivest–Shamir–Adleman)

Two mathematically related keys a public key for encryption and a private keys are used for decryption in the well-known RSA public-key encryption algorithm. RSA is good for safely exchanging encryption keys but its slow computation makes it unsuitable for encrypting large files. Large prime numbers are difficult to factor using RSA which makes it extremely secure and nearly impenetrable given the state of computing power. To safeguard the AES secret key your project specifically uses RSA. After the users data has been encrypted using AES the AES key is encrypted with an RSA procedure before being stored. This guarantees that the data cannot be decrypted without the private RSA key even if someone manages to access the stored files. As a result RSA strengthens the encryption process overall by adding an essential layer of secure key management that stops unauthorized access.

3. Password-Based Key Derivation Function (KDF)

Because the system supports user-defined encryption passwords the users password is converted into a secure cryptographic key using a custom script or a KDF like PBKDF2 or Argon2. By using hashing salting and repeated processing the KDF protects the password from dictionary and brute-force attacks preventing even short or weak passwords from being compromised. The AES key is then encrypted using the derived key or it is used as an additional

encryption layer. This suggests that even if the encrypted file and the RSA key are somehow obtained the encrypted content cannot be decrypted without the correct password. The projects use of a KDF makes it extremely difficult for attackers to breach the password-based encryption that already surrounds the area.

3.5 Tools and Technologies

1. Python

Python is the primary programming language designed to work on the entire system and manage the backend operations. Implementing encryption and decryption logic file handling procedures user authentication and system module communication are just a few of the numerous ways it is used. Python is a great option to develop secure applications since it offers a huge variety of web and cryptography libraries. Its modular programming and readable syntax also make the whole process of maintenance, future upgrades and debugging of the system easier in the long run.

2. Flask Framework

Flask is a web framework which is utilized for the development the applications backend that is not just lightweight but also very flexible. It takes care of user sessions management, authentication, form submissions, HTTP requests and routing URLs. Furthermore, Flask allows the smooth and efficient transaction of databases and the use of cryptographic algorithms. Flask idles on the edge, ready to accommodate and throw off spikes of users and incoming data; it trundles on through the heavy load, understandably and visibly, all the while one lowering the overhead.

3. HTML, CSS, and JavaScript

In order to create an interactive and user-friendly interface the systems frontend was developed using HTML CSS and JavaScript. HTML is in charge of outlining and organizing the web pages CSS is in charge of the interfaces appearance and adaptability on different devices and JavaScript offers interaction such as by validating inputs sending alerts and enabling real-time interaction. Regardless of their level of technical expertise users can access and navigate the system with ease thanks to the combination of these technologies.

4. Database (SQLite)

SQLite is used as the backend database to store user information, encrypted file metadata, encryption parameters, and activity logs. It provides structured and reliable data storage with minimal configuration requirements. SQLite ensures efficient data retrieval, integrity, and access control, making it suitable for secure applications with moderate data storage needs.

5. Gmail SMTP Service

SMTP service for Gmail. Gmail SMTP has been used to automatically send email notifications about system activity and file uploads and downloads. Furthermore, said informing the end-users about various major sight scenes, also increases communication integrity under adverse environment conditions. Using secure email protocols ensures notification messages are delivered safely without disclosing any private system information.

6. Encrypted Storage Environment

storage environment that is encrypted. In a secure storage area on the server where access is restricted the application encrypts files and texts. Since only encrypted data is kept even in the event of unauthorized access to secure storage sensitive data cannot be compromised. The possibility of data breaches is reduced and data protection is elevated beyond application-level encryption thanks to these multi-layered security measures.

7. Development Tools (Visual Studio Code)

The primary environment for the projects development and testing will be Visual Studio Code. Python and web development are made much simpler by robust extensions with built-in debugging tools and version control features. The advantages of these benefits make it possible for the code to be written by the developers in a structured manner, for errors to be found quickly and for the project flow to be managed easily.

3.6 System Implementation

3.6.1 Python code

```python
import os
import secrets
import secrets
from io import BytesIO
from datetime import datetime
from dotenv import load_dotenv

load_dotenv()  # Load environment variables from .env file

from flask import (
    Flask,
    render_template,
    request,
    redirect,
    url_for,
    flash,
    send_file,
    flash,
    send_file,
    session,
)
from flask_mail import Mail, Message
from flask_sqlalchemy import SQLAlchemy
from flask_login import (
    LoginManager,
    login_user,
    logout_user,
    login_required,
    current_user,
    UserMixin,
)
from werkzeug.security import generate_password_hash, check_password_hash

# Cryptography
from cryptography.hazmat.primitives import hashes, padding as sym_padding
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC
from cryptography.hazmat.backends import default_backend

# App & DB setup

BASE_DIR = os.path.abspath(os.path.dirname(__file__))

app = Flask(__name__)
app.config["SECRET_KEY"] = "change-this-secret-key"  # change for production
app.config["SQLALCHEMY_DATABASE_URI"] = "sqlite:///" + os.path.join(BASE_DIR, "secure_storage.db")
app.config["SQLALCHEMY_TRACK_MODIFICATIONS"] = False

# Flask-Mail Configuration (Use environment variables or defaults for testing)
app.config["MAIL_SERVER"] = os.environ.get("MAIL_SERVER", "smtp.gmail.com")
app.config["MAIL_PORT"] = int(os.environ.get("MAIL_PORT", 587))
app.config["MAIL_USE_TLS"] = os.environ.get("MAIL_USE_TLS", "true").lower() == "true"
app.config["MAIL_USERNAME"] = os.environ.get("MAIL_USERNAME", "your-email@example.com")
app.config["MAIL_PASSWORD"] = os.environ.get("MAIL_PASSWORD", "your-email-password")
app.config["MAIL_DEFAULT_SENDER"] = os.environ.get("MAIL_DEFAULT_SENDER", app.config["MAIL_USERNAME"])


mail = Mail(app)

# Check for default email configuration
with app.app_context():
    if app.config["MAIL_USERNAME"] == "your-email@example.com" or \
       app.config["MAIL_USERNAME"] == "your-email@gmail.com" or \
       app.config["MAIL_PASSWORD"] == "your-email-password" or \
       app.config["MAIL_PASSWORD"] == "your-app-password":
        print("\n" + "="*60)
        print(" WARNING: Email credentials are still set to defaults!")
        print(" Email notifications will NOT work until you update .env")
        print(" Edit c:\\Users\\ULLAS\\OneDrive\\Desktop\\project\\secure_file_storage\\.env")
        print("="*60 + "\n")

db = SQLAlchemy(app)

login_manager = LoginManager(app)
login_manager.login_view = "login"

# Models

class User(db.Model, UserMixin):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(120), nullable=False)
    email = db.Column(db.String(120), unique=True, nullable=False)
    password = db.Column(db.String(255), nullable=False)
    role = db.Column(db.String(20), default="sender")  # "sender" or "receiver"

    # RSA keys are no longer stored or used

class EncryptedData(db.Model):
    __tablename__ = "encrypted_data"

    id = db.Column(db.Integer, primary_key=True)
    filename = db.Column(db.String(255))
    file_data = db.Column(db.LargeBinary)           # AES-encrypted file bytes
    text_encrypted = db.Column(db.LargeBinary)      # AES-encrypted text message

    # Password-based encryption fields
    aes_key_wrapped = db.Column(db.LargeBinary)     # AES key wrapped with derived key
    salt = db.Column(db.LargeBinary)                # Salt for KDF
    wrapping_iv = db.Column(db.LargeBinary)         # IV for wrapping the key

```

```python

    iv = db.Column(db.LargeBinary)                  # AES IV for file/text encryption
    created_at = db.Column(db.DateTime, default=datetime.utcnow)

    sender_id = db.Column(db.Integer, db.ForeignKey("user.id"))
    sender = db.relationship("User", backref="encrypted_items")

# Login manager

@login_manager.user_loader
def load_user(user_id):
    # SQLAlchemy 2.x-compatible
    return db.session.get(User, int(user_id))

# Crypto helpers

def derive_key(password: str, salt: bytes) -> bytes:
    """
    Derive a 32-byte (256-bit) key from the password using PBKDF2-HMAC-SHA256.
    """
    kdf = PBKDF2HMAC(
        algorithm=hashes.SHA256(),
        length=32,
        salt=salt,
        iterations=100000,
        backend=default_backend()
    )
    return kdf.derive(password.encode())

def aes_encrypt(plaintext: bytes, key: bytes, iv: bytes) -> bytes:
    """
    AES-CBC with PKCS7 padding.
    plaintext -> ciphertext
    """
    padder = sym_padding.PKCS7(128).padder()
    padded = padder.update(plaintext) + padder.finalize()

    cipher = Cipher(algorithms.AES(key), modes.CBC(iv))
    encryptor = cipher.encryptor()
    ciphertext = encryptor.update(padded) + encryptor.finalize()
    return ciphertext

def aes_decrypt(ciphertext: bytes, key: bytes, iv: bytes) -> bytes:
    """
    AES-CBC with PKCS7 unpadding.
    ciphertext -> plaintext
    """
    cipher = Cipher(algorithms.AES(key), modes.CBC(iv))
    decryptor = cipher.decryptor()
    padded = decryptor.update(ciphertext) + decryptor.finalize()

    unpadder = sym_padding.PKCS7(128).unpadder()
    try:
        plaintext = unpadder.update(padded) + unpadder.finalize()
        return plaintext
    except ValueError:
        # Padding error usually means wrong key
        raise ValueError("Invalid padding or incorrect key")

def send_email(subject, recipients, body):
    """
    Helper function to send simple emails.
    Returns (success, error_message)
    """
    if isinstance(recipients, str):
        recipients = [recipients]

    try:
        msg = Message(subject, recipients=recipients, body=body)
        mail.send(msg)
        print(f"Email sent to {recipients}: {subject}")
        return True, None
    except Exception as e:
        print(f"Failed to send email to {recipients}: {e}")
        return False, str(e)

# Routes

@app.route("/")
@login_required
def index():
    """
    After login, redirect based on role.
    """
    if current_user.role == "receiver":
        return redirect(url_for("receiver_dashboard"))
    else:
        return redirect(url_for("sender_dashboard"))

# --------------- Register ------------------------------

@app.route("/register", methods=["GET", "POST"])
def register():
    if request.method == "POST":
        name = request.form.get("name", "").strip()
        email = request.form.get("email", "").strip().lower()
        password = request.form.get("password", "")
        role = request.form.get("role", "sender").strip().lower()


```

```
        if not name or not email or not password:
            flash("Please fill in all fields.")
            return redirect(url_for("register"))

        if role not in ["sender", "receiver"]:
            role = "sender"

        existing = User.query.filter_by(email=email).first()
        if existing:
            flash("An account with this email already exists.")
            return redirect(url_for("register"))

        # No RSA key generation anymore

        hashed_password = generate_password_hash(password)

        user = User(
            name=name,
            email=email,
            password=hashed_password,
            role=role,
        )
        db.session.add(user)
        db.session.commit()

        flash("Registration successful. Please log in.")
        return redirect(url_for("login"))

    return render_template("register.html")

# --------------- Login ---------------------------------
@app.route("/login", methods=["GET", "POST"])
def login():
    # If already logged in, send them to their dashboard
    if current_user.is_authenticated:
        if current_user.role == "receiver":
            return redirect(url_for("receiver_dashboard"))
        else:
            return redirect(url_for("sender_dashboard"))

    if request.method == "POST":
        name = request.form.get("name", "").strip()
        email = request.form.get("email", "").strip().lower()
        password = request.form.get("password", "")

        user = User.query.filter_by(email=email).first()

        if not user:
            flash("Invalid email, name, or password.")
            return redirect(url_for("login"))

        # Validate name (case-insensitive)
        if not user.name or user.name.strip().lower() != name.lower():
            flash("Invalid name for this account.")
            return redirect(url_for("login"))

        # Validate password
        if not check_password_hash(user.password, password):
            flash("Invalid email, name, or password.")
            return redirect(url_for("login"))

        # Login OK
        login_user(user)

        next_page = request.args.get("next")
        if user.role == "receiver":
            return redirect(next_page or url_for("receiver_dashboard"))
        else:
            return redirect(next_page or url_for("sender_dashboard"))

    return render_template("login.html")

# --------------- Logout ---------------------------------

@app.route("/logout")
@login_required
def logout():
    logout_user()
    # clear any decrypted info from session
    session.pop("decrypted_text", None)
    session.pop("last_decrypted_item_id", None)
    session.pop("aes_key_hex", None)
    flash("You have been logged out.")
    return redirect(url_for("login"))

# --------------- Sender Dashboard ----------------------

@app.route("/sender_dashboard", methods=["GET", "POST"])
@login_required
def sender_dashboard():
    if current_user.role != "sender":
        return redirect(url_for("receiver_dashboard"))

    if request.method == "POST":
        file = request.files.get("file")
        text = request.form.get("text", "")
        encryption_password = request.form.get("encryption_password", "")

        if not file or not text or not encryption_password:
            flash("Please provide file, message, and an encryption password.")
            return redirect(url_for("sender_dashboard"))
```

```
    if request.method == "POST":
        file = request.files.get("file")
        text = request.form.get("text", "")
        encryption_password = request.form.get("encryption_password", "")

        if not file or not text or not encryption_password:
            flash("Please provide file, message, and an encryption password.")
            return redirect(url_for("sender_dashboard"))

        file_bytes = file.read()

        # 1. Generate Info for File Encryption
        aes_key = secrets.token_bytes(32)  # 256-bit key for the file
        file_iv = secrets.token_bytes(16)  # 128-bit IV for the file

        # 2. Encrypt file + text
        file_cipher = aes_encrypt(file_bytes, aes_key, file_iv)
        text_cipher = aes_encrypt(text.encode("utf-8"), aes_key, file_iv)

        # 3. Derive Key Encryption Key (KEK) from password
        salt = secrets.token_bytes(16)
        kek = derive_key(encryption_password, salt)

        # 4. Wrap the AES key using the KEK
        wrapping_iv = secrets.token_bytes(16)
        aes_key_wrapped = aes_encrypt(aes_key, kek, wrapping_iv)

        # 5. Store in DB
        record = EncryptedData(
            filename=file.filename,
            file_data=file_cipher,
            text_encrypted=text_cipher,
            aes_key_wrapped=aes_key_wrapped,
            salt=salt,
            wrapping_iv=wrapping_iv,
            iv=file_iv,
            sender=current_user,
        )
        db.session.add(record)
        db.session.commit()

        flash("File and message encrypted and stored successfully.")

        #      Email Notification to Receivers
        try:
            # Find all users with role 'receiver'
            receivers = User.query.filter_by(role="receiver").all()
            receiver_emails = [r.email for r in receivers if r.email]

            if receiver_emails:
                subject = "New Secure File Available"
                body = (
                    f"Hello,\n\n"
                    f"A new file has been uploaded by {current_user.name} ({current_user.email}).\n"
                    f"Filename: {file.filename}\n\n"
                    f"Please log in to your dashboard to decrypt and view it.\n"
                )
                # Send email notification
                success, err_msg = send_email(subject, receiver_emails, body)
                if not success:
                    flash(f"Warning: Email notification failed. {err_msg}")

        except Exception as e:
            print(f"Error sending upload notification: {e}")
            flash(f"Error checking receivers for notification: {e}")

        return redirect(url_for("sender_dashboard"))

    return render_template("sender_dashboard.html")

#                   Receiver Dashboard

@app.route("/receiver_dashboard", methods=["GET", "POST"])
@login_required
def receiver_dashboard():
    if current_user.role != "receiver":
        return redirect(url_for("sender_dashboard"))

    if request.method == "POST":
        decryption_password = request.form.get("decryption_password", "")

        if not decryption_password:
            flash("Please enter the decryption password.")
            return redirect(url_for("receiver_dashboard"))

        # Get latest encrypted record
        item = EncryptedData.query.order_by(EncryptedData.created_at.desc()).first()
        if not item:
            flash("No encrypted data found.")
            return redirect(url_for("receiver_dashboard"))

        try:
            # 1. Derive KEK using the stored salt and provided password
            kek = derive_key(decryption_password, item.salt)

            # 2. Unwrap the AES key
            aes_key = aes_decrypt(item.aes_key_wrapped, kek, item.wrapping_iv)

            # 3. Decrypt text message
            plaintext_bytes = aes_decrypt(item.text_encrypted, aes_key, item.iv)
            plaintext = plaintext_bytes.decode("utf-8", errors="replace")

            # Save info to session (NOT full file bytes; only small data)
            session["decrypted_text"] = plaintext
            session["last_decrypted_item_id"] = item.id
            session["aes_key_hex"] = aes_key.hex()

            flash("Decryption successful.")

            #      Email Notification to Sender
            try:
                # Notify the original sender that their file was decrypted
                sender = item.sender
                if sender and sender.email:
                    subject = "Your File Was Decrypted"
                    body = (
                        f"Hello {sender.name},\n\n"
                        f"Your file '{item.filename}' was successfully decrypted by {current_user.name} ({current_user.email}).\n"
                        f"Time: {datetime.utcnow()}\n"
                    )

                    success, err_msg = send_email(subject, sender.email, body)
                    if not success:
                        flash(f"Warning: Email notification failed. {err_msg}")
                except Exception as e:
                    print(f"Error sending decryption notification: {e}")
                    flash(f"Error notifying sender: {e}")
            except Exception as e:
                print("Decryption error:", e)
                session.pop("decrypted_text", None)
                session.pop("last_decrypted_item_id", None)
                session.pop("aes_key_hex", None)
                flash("Decryption failed. Incorrect password or data corrupted.")
                return redirect(url_for("receiver_dashboard"))

    decrypted_flag = "decrypted_text" in session
    return render_template("receiver_dashboard.html", decrypted=decrypted_flag)

#                   Download decrypted file

@app.route("/download_decrypted_file")
@login_required
def download_decrypted_file():
    """
    Decrypt the file on the fly using AES key stored in session,
    and send it as a download.
    """
    item_id = session.get("last_decrypted_item_id")
    aes_key_hex = session.get("aes_key_hex")

    if not item_id or not aes_key_hex:
        flash("No decrypted file available. Please decrypt first.")
        return redirect(url_for("receiver_dashboard"))

    # Get encrypted record
    item = db.session.get(EncryptedData, int(item_id))
    if not item:
        flash("Encrypted data not found.")
        return redirect(url_for("receiver_dashboard"))

    try:
        aes_key = bytes.fromhex(aes_key_hex)
        decrypted_file_bytes = aes_decrypt(item.file_data, aes_key, item.iv)

        mem = BytesIO()
        mem.write(decrypted_file_bytes)
        mem.seek(0)

        filename = item.filename or "decrypted_file"
        return send_file(
            mem,
            as_attachment=True,
            download_name=filename,
            mimetype="application/octet-stream",
        )
    except Exception as e:
        print("File decrypt error:", e)
        flash("Unable to decrypt file for download.")
        return redirect(url_for("receiver_dashboard"))

# App start

if __name__ == "__main__":
    with app.app_context():
        db.create_all()
    app.run(debug=True)
```

3.6 System Testing

system testing verifies the platforms overall robustness by closely examining the modules for authentication encryption file handling storage decryption and notifications simultaneously. The primary goal is to guarantee that users can safely upload encrypt and retrieve data without any issues as well as that the systems functional and non-functional requirements are satisfied. To make sure that the system proved dependable under all circumstances testing was executed in such a way that involved participating in various scenarios such as valid and invalid logins encryption password correctness large file uploads and decryption failure. To guarantee appropriate communication between AES encryption RSA key management password-based security and Gmail notifications each feature was tested handled independently before being added to the entire workflow. In order to guarantee that encrypted files remain secure and that attempts at unauthorized access are appropriately blocked error handling data integrity and security measures received substantial attention. In conclusion system testing attests to the applications stability security ease of use and ability to perform all encryption and decryption tasks with extreme accuracy.

Various types of Software testing

1. Unit Testing

The main objective of unit testing is to confirm the accuracy of the smallest software components which are usually function classes or particular code segments. Before integration with other modules each component is tested to make sure it functions properly. This projects unit tests could verify that the database appropriately stores the metadata validate the creation of RSA keys or verify the outcome of AES encryption.

2. Integration Testing

The process of assessing different components and how well they function together when combined is called integration testing. Even if each unit functions well independently integration may still present some challenges. This testing ensures the seamless interoperability of the Gmail notification system storage layer encryption engine authentication module and all other components.

3. System Testing

System testing is the stage in which the entire program is tested in the final operating environment. It shows that every system feature including file upload user login hybrid encryption password validation and file retrieval is carried out correctly from start to finish. Therefore the features that would be expected if the system were used by actual users are considered to be present in the system under test.

4. Functional Testing

To make sure the program keeps all of its features functional testing is used. It verifies successful registration in accordance with specifications tests decryption using the correct password and checks the uploaded files correct processing.

5. Security Testing

Applications that rely on encryption must undergo security testing. The procedure evaluates the systems resilience to attacks like brute-force or injection ensures secure password hashing verifies the security of encrypted files and assesses the systems resistance to unwanted access.

6. Performance Testing

To get the most out of the system performance testing is conducted under various workloads. It gives the information about the system response times, encryption/decryption speed, and the capability to work with large files or multiple user requests at the same time. This will also highlight the areas that require improvement while maintaining the applications speed and responsiveness.

7. Usability Testing

The aim of usability testing services is to represent system users experiences. It assesses whether the encryption and decryption procedures are clearly defined whether the interface is easy to use and whether users can complete their tasks without getting lost or making mistakes. Users will accept and trust the system if they have a positive experience.

3.7.1 Test Cases

Test ID	Module	Test Case	Input	Expected Output	Status
TC01	User Registration	Register with valid details	Name, valid email, strong password	Account created, confirmation saved in DB, success message shown	Pass
TC02	User Registration	Register with an email already in use	Existing email, password	Registration rejected, friendly error shown, no duplicate DB record	Pass
TC03	User Login	Login with correct credentials	Registered email, correct password	Login success, session/JWT issued, access to dashboard	Pass
TC04	User Login	Login with invalid password	Registered email, wrong password	Login denied, error shown, failed attempt logged, rate-limit counter incremented	Pass
TC05	File Upload	Upload a supported file type within size limit	PDF/image/text file under limit	Upload accepted, file stored temporarily, encryption begins, metadata saved	Pass
TC06	File Download & Decryption	To ensure encrypted files are correctly decrypted on download	A previously uploaded encrypted file	The user receives a decrypted file identical to the original	Pass
TC07	Text Encryption	Encrypt a short and long text (special chars included)	Plain text with punctuation and unique code	Encrypted ciphertext returned/stored; original cannot be read from storage	Pass

TC08	Custom Encryption Password	Sender sets a password, receiver decrypts using correct password	Password during upload, same password during decrypt	Password-based unwrap succeeds; file/text decrypts correctly	Pass
TC09	Wrong Encryption Password	Attempt decryption with incorrect password	Wrong password at decrypt step	Decryption denied, error shown; attempt logged, no plaintext returned	Pass
TC10	Unauthorized Access Attempt	To verify system security against non-logged-in users	Trying to access upload or download pages without logging in	The system should block access and redirect to login	Pass
TC-11	Gmail Notification Sent	Trigger notification after upload / share	Upload + share action with recipient email	Gmail API/SMTP called; email delivered notification status saved	Pass
TC12	Storage Integrity	To ensure stored encrypted files/text remain unchanged	Retrieve stored encrypted data	Data should match the stored version without modification	Pass
TC13	Logout process	Logout button	Click logout button	User session cleared, return to login page	Pass

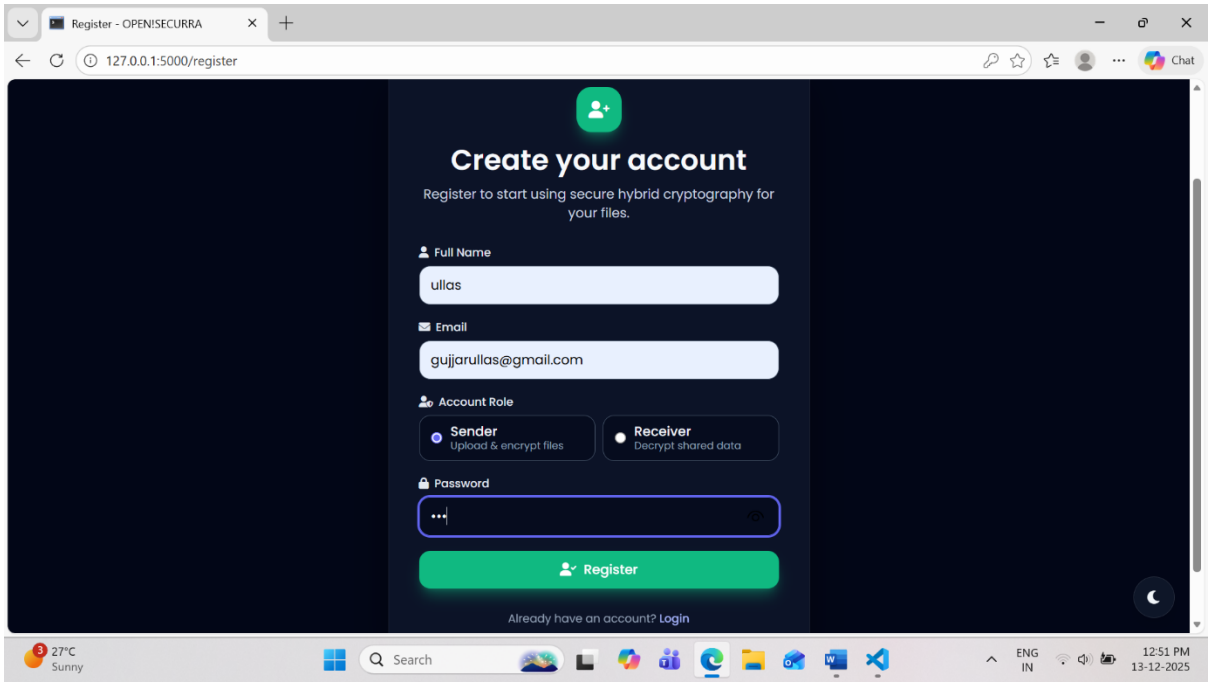
Table 3.7.1 Test Cases

CHAPTER 4

RESULTS

4.1 Output Screenshots

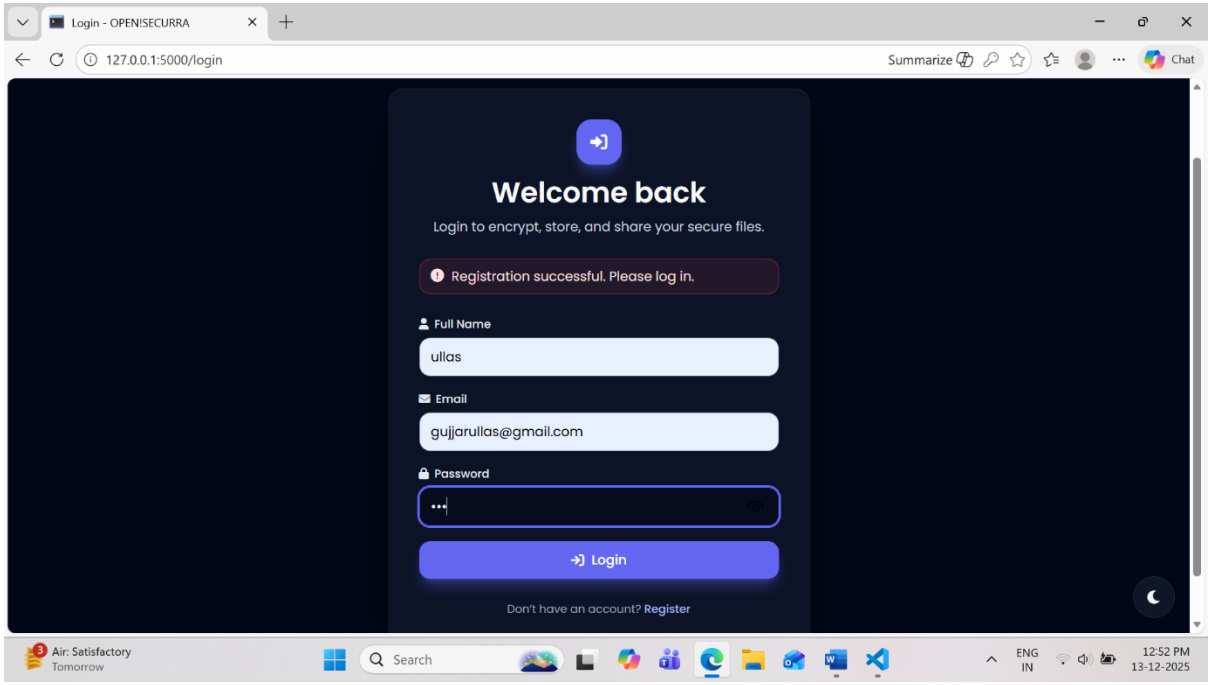
Registration Page



4.1.1 Registration Page

New users are able to register by submitting their name and email address and password on the registration page. The system validates the information and prevents duplicate registrations. Once registration is completed, users can log in securely and access the full system.

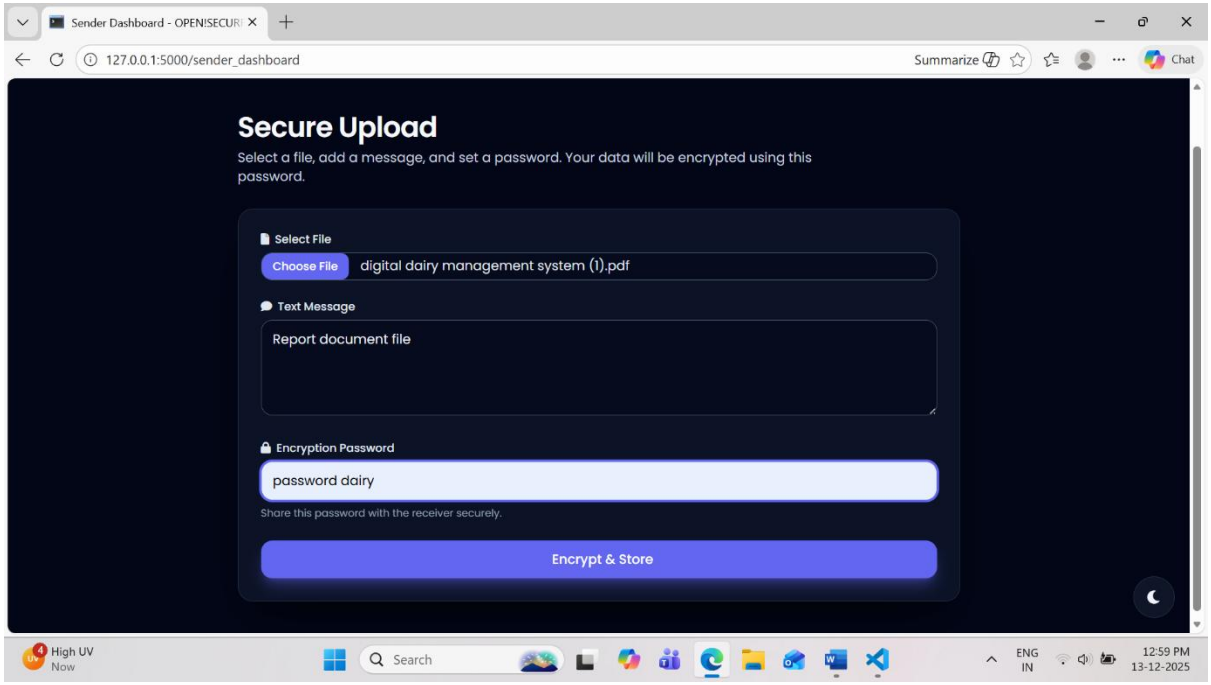
Sender Login Page



4.1.2 Sender Login Page

By submitting a form requesting their previously used email address and password users can log into the system. The user is presented with the option of a dashboard once the credentials have been verified.

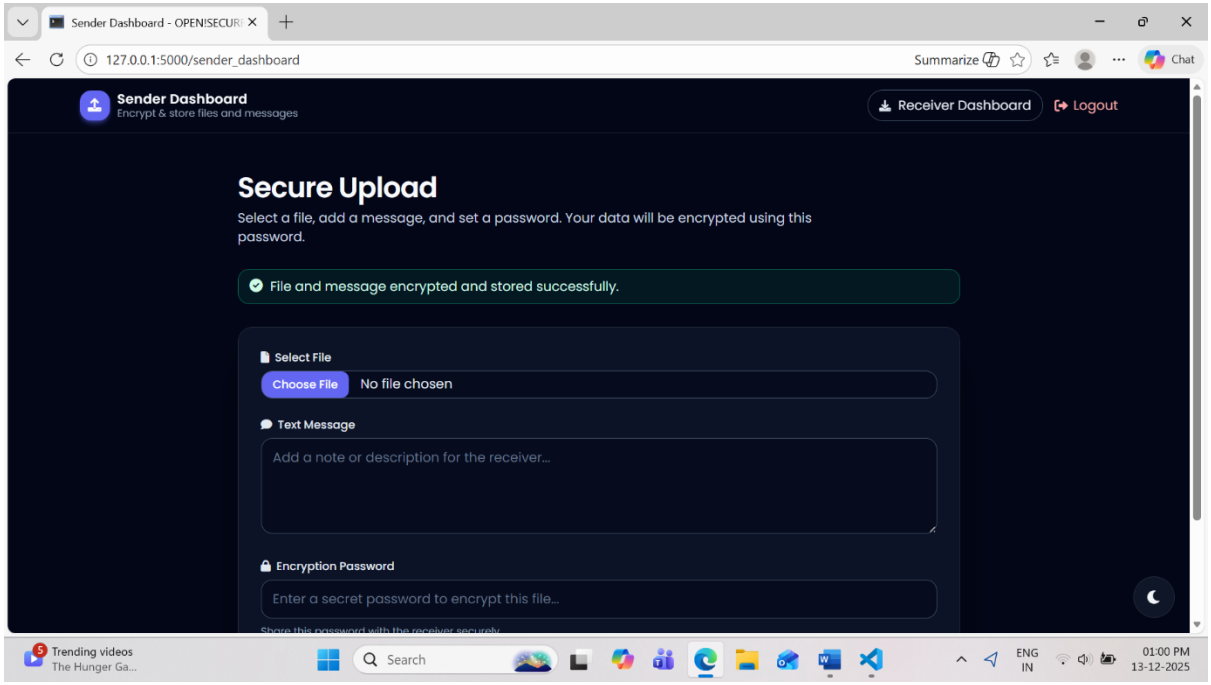
File Selection, Message Input, and Password Encryption



4.1.3 File Encryption Section

The files are secured because users are the only ones who can upload the document along with a text message and a password that is used for encryption. The dashboards straightforward layout facilitates quick data encryption and protection prior to transmission to the recipient.

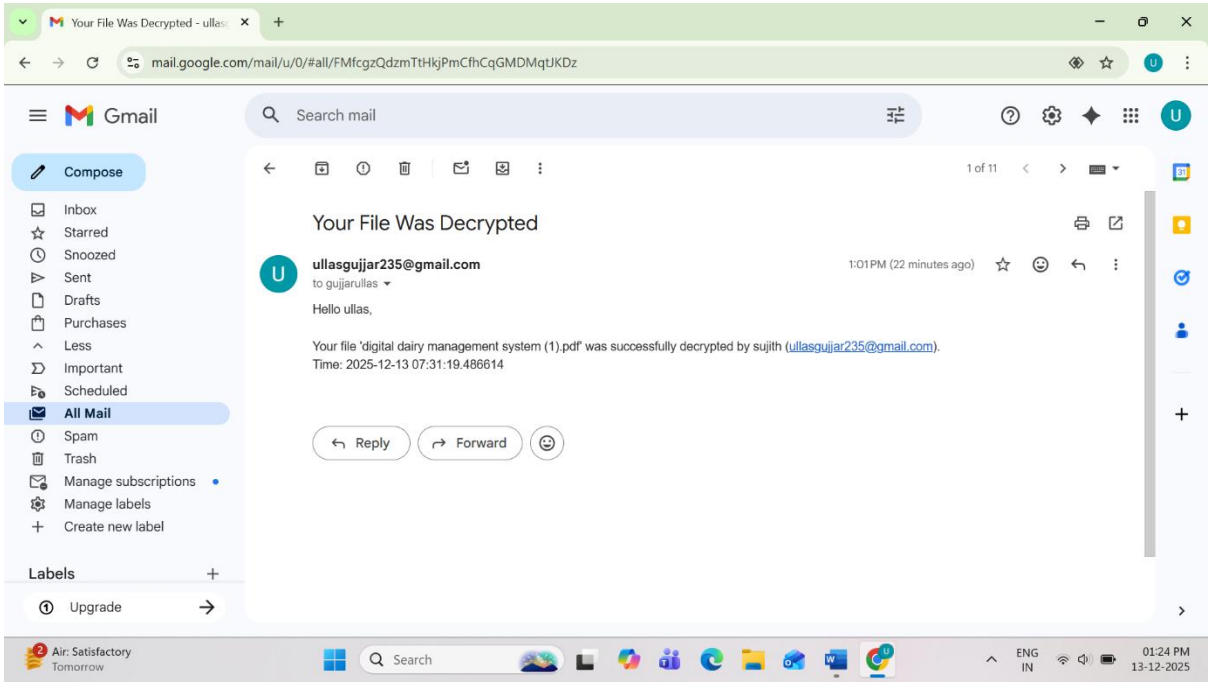
File, and Message Encrypted



4.1.4 File, and Message Encrypted

Before the data is saved the sender chooses a file writes a message and inputs the encryption password on this Secure Upload page. The success message informs him or her that the file has been stored and encrypted. It offers a user-friendly interface for secure file preparation for the recipient.

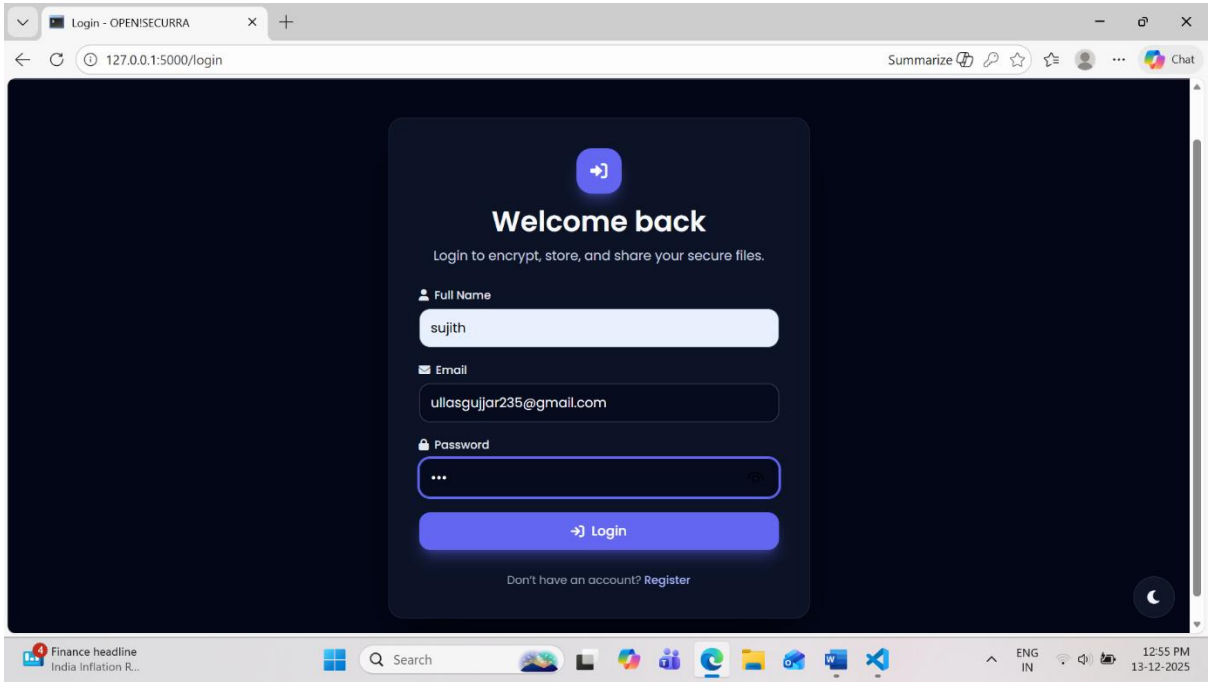
Receiver Email Notification for Secure File Access



4.1.5 Receiver Email Notification for Secure File Access

The recipient receives a notification via email that a new encrypted file has been shared. By providing details about the sender and the file the notification encourages the recipient to log in and safely decrypt the content.

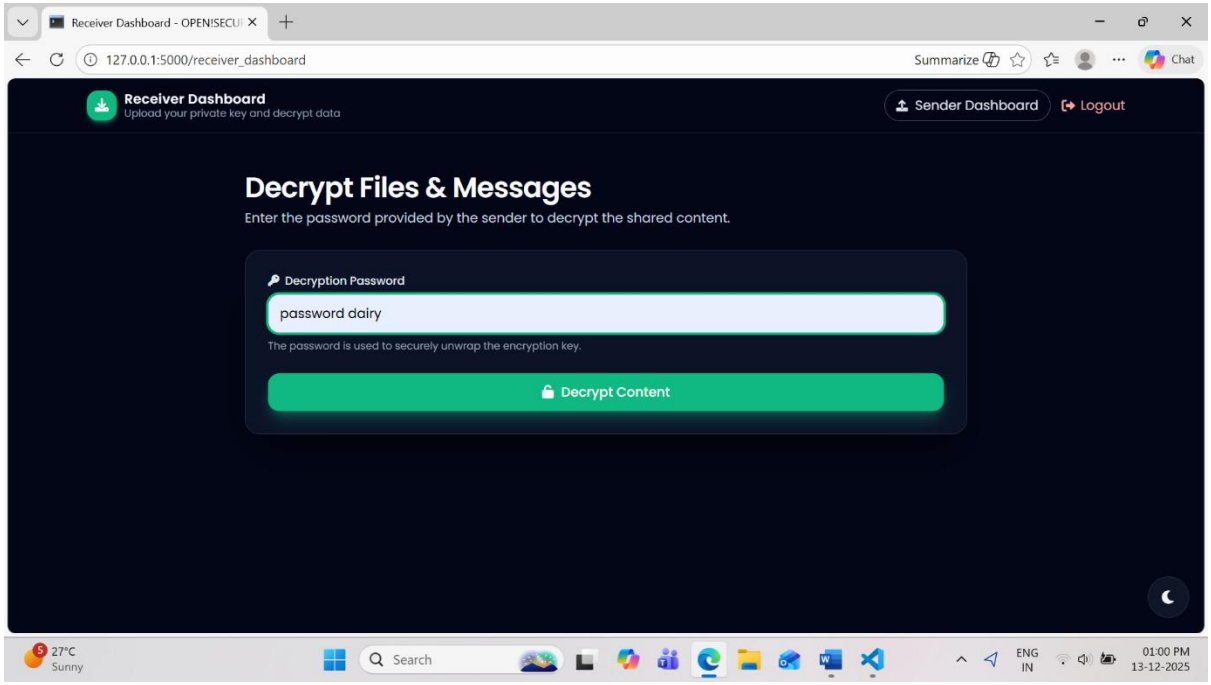
Receiver Login Page



4.1.6 Receiver Login Page

By submitting their registered email address along with the password on the Receiver Login Page users can safely access the shared files. It is guaranteed that the encrypted content can only be viewed and understood by the legitimate recipients.

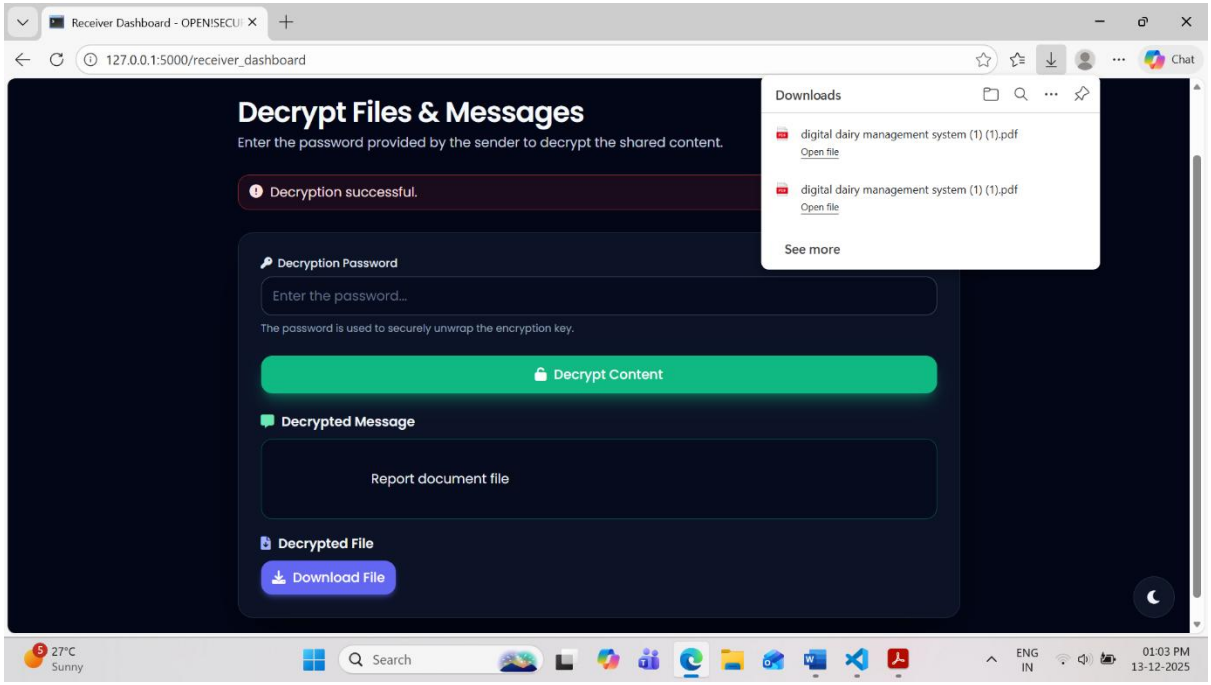
Receiver Decryption Dashboard



4.1.7 Receiver Decryption Dashboard

In order to access the encrypted data or message the user must enter the password that was sent by the sender using the Dashboard of Receiver Decryption. The system decodes the content and makes it available for download once the correct password is entered.

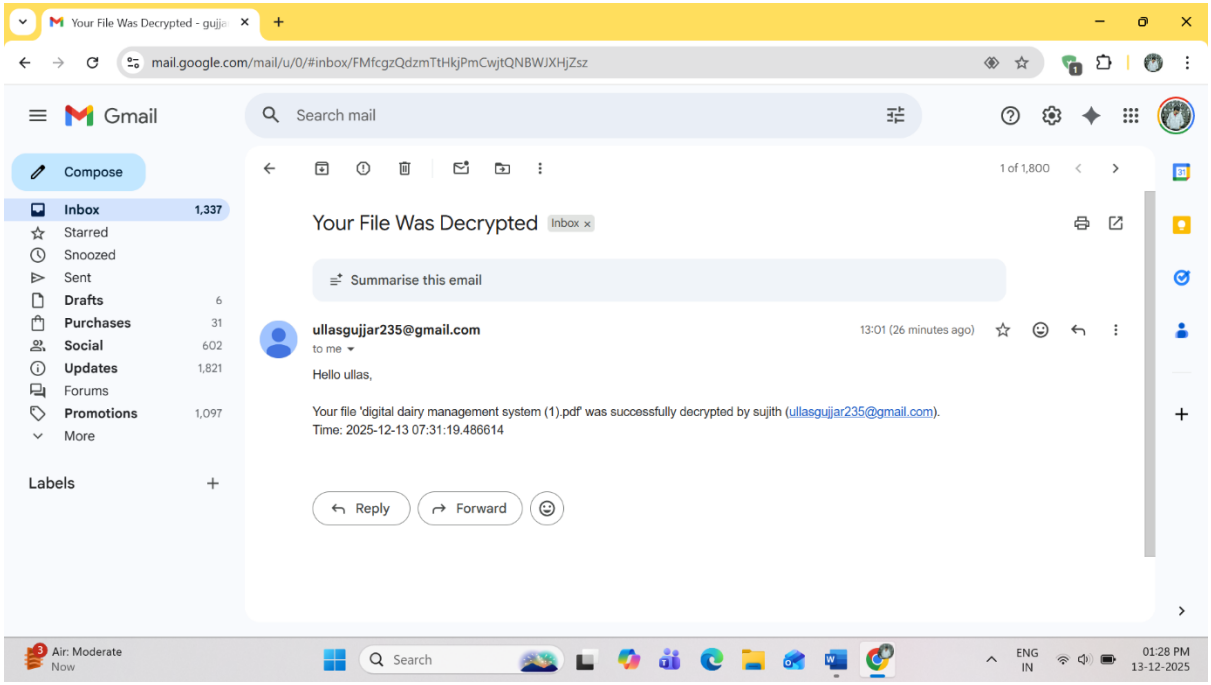
Receiver Decryption Results



4.1.8 Receiver Decryption Results

This page displays the results of receiver decryption and the encrypted data has been fully unlocked by entering the password. After displaying the first message the system offers the option to save the recovered file. It verifies the decryption procedure and enables the recipient to safely access the file and text that the sender has sent.

File Decryption Confirmation Email



4.1.9 File Decryption Confirmation Email

When the recipient successfully decrypted the shared document this particular page displays the Gmail alert that was sent to the sender. The file name the recipients email address and the precise moment of decryption are all included in the message. It informs the sender that their encrypted file has been safely accessed and decrypted.

CHAPTER 5

CONCLUSION AND FUTURE ENHANCEMENT

5.1 Conclusion

This project shows how a hybrid cryptography approach can provide a robust useful and approachable way to safeguard private digital data. The system protects text and file data from common cyber threats tampering and unauthorized access by combining the speed of AES with the secure key handling of RSA. The platform demonstrates that sophisticated security can still be easily accessible by enabling users to register log in encrypt data and retrieve it with ease. The system demonstrated dependability accuracy and efficiency in managing various data types without sacrificing performance through meticulous testing and validation.

Beyond its technical implementation this project demonstrates how secure data management tools are progressively becoming important in daily digital activities. It highlights how user trust and data security can be greatly increased with robust encryption appropriate authentication and organized storage. All things considered the system offers a comprehensive and trustworthy solution for safe communication and file protection as well as a base upon which more sophisticated features can be added later on.

5.2 Future Enhancement

In order to satisfy the increasing demands of users the system can be further enhanced by incorporating more sophisticated security and usability features. Integrating cloud storage support is one potential improvement that would enable users to securely access their encrypted text and files from any device. Extra security layers like biometric login or two-factor authentication could improve security and lower the risk of unwanted access. Additionally the system might be improved to better manage bigger files and enable real-time alerts for user activity. Offering a mobile application version is another improvement that increases the accessibility of encryption and decryption while on the go. Future upgrades would increase the platforms adaptability user-friendliness and capacity to manage a greater variety of real-world situations.

REFERENCES

[1] RashiDhagat, Purvi Joshi, "Cloud-Based Secure Storage for Group Data Sharing", Year 2016

[2] Bilal Habib, Bertrand Cambou, DuaneBooher, Christopher Philabaum, "A Secure and Addressable Public Key Infrastructure for Cloud Storage", Year 2017

[3] Shakeeba S. Khan, Prof. R. R. Tuteja, "A Multilevel Encryption Approach for Secure Cloud Data Exchange", Year 2015

[4] Tulip Dutta, Amarjyoti Pathak, "Efficient Key Management for Encrypted Cloud Data Sharing", Year 2016

[5] Mr. Rohit Barvekar, Mr. ShrajalBehere, Mr. Yash Pounikar, Ms. Anushka Gulhane, "Enhancing Cloud Security Through Efficient Cryptographic Methods", Year 2018

[6] Shaikh, S., & Vora, D. (2016). "Secure cloud auditing over encrypted data." 2016 International Conference on Communication and Electronics Systems (ICCES).

[7] Gajendra, B. P., Singh, V. K., & Sujeet, M. (2016). "Achieving cloud security using third party auditor, MD5, and identity-based encryption". 2016 International Conference on Computing, Communication, and Automation (ICCCA), 1304–1309.

[8] Bhandari, A., Gupta, A., & Das, D. (2016). "Secure algorithm for cloud computing and its applications". 6th International Conference on Cloud System and Big Data Engineering (Confluence), 188–192.

[9] Taha, A. A., Elminaam, D. S. A., & Hosny, K. M. (2018). "An improved security schema for mobile cloud computing using hybrid cryptographic algorithms". Far East Journal of Electronics and Communications, 18(4), 521–546.

[10] Kranthi Kumar, K., & Devi, T. (2018). "Secured data transmission in cloud using hybrid cryptography". International Journal of Pure and Applied Mathematics, 119(16), 3257–3262.

PUBLICATIONS

SI.NO	Paper	Author	Publication
1	Secure Cloud Auditing over Encrypted Data	Shaikh S., Vora D.	International Conference on Communication and Electronics Systems (ICCES), IEEE
2	Secured Data Transmission in Cloud Using Hybrid Cryptography	Kranthi Kumar K., Devi T.	International Journal of Pure and Applied Mathematics
3	An Improved Security Schema for Mobile Cloud Computing Using Hybrid Cryptographic Algorithms	Taha A. A., Elminaam D. S. A., Hosny K. M.	Far East Journal of Electronics and Communications