Secure File Sharing System

A Project report submitted

in partial fulfillment of requirement for the award of degree

BACHELOR OF TECHNOLOGY

in

SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE

G.PRANATHI	2203A51049
K.JAHNAVI	2203A51050
P.JAYATHI	2203A51215

Under the guidance of

Dr. Santosh Kumar Henge

Associate Professor, School of CS&AI.

SR UNIVERSITY

SR University,

Ananthasagar, Warangal,Telangana-506371

SR UNIVERSITY

CERTIFICATE

This is to certify that this project entitled **“Secure File Sharing System** " is the bonafied work carried out by **G.PRANATHI,K.JAHNAVI,P.JAYATHI** as a Project for the partial fulfillment to award the degree **BACHELOR OF TECHNOLOGY** in **School of Computer Science and Artificial Intelligence** during the academic year 2024-2025 under our guidance and Supervision.

Dr. Santosh Kumar Henge	**Dr. M.Sheshikala**
Associate Professor	Professor& Head,
SR University	School of CS&AI,
Ananthasagar,Warangal	SR University
	Ananthasagar, Warangal.

ACKNOWLEDGEMENT

We owe an enormous debt of gratitude to our Major Project guide **Dr. Santosh Kumar Henge, Associate Professor** as well as Head of the School of CS&AI , **Dr. M.Sheshikala, Professor** and Dean of the School of CS&AI, **Dr.Indrajeet Gupta Professor** for guiding us from the beginning through the end of the Capstone Project with their intellectual advices and insightful suggestions. We truly value their consistent feedback on our progress, which was always constructive and encouraging and ultimately drove us to the right direction.

TABLE OF CONTENTS

1. ABSTRACT

This project describes a Secure File Sharing System that allows users to share files on an untrusted network and maintains confidentiality, integrity, and authentication. The system uses AES (Advanced Encryption Standard) for encryption, RSA (Rivest–Shamir–Adleman) for secure key exchange, and digital signatures for file authenticity verification.

The implementation is done with Python, together with the cryptography and Py Cryptodome packages. A user interface makes it possible for users to upload, encrypt, sign, share, receive, verify, and decrypt files.

This project showcases the manner in which the integration of symmetric and asymmetric cryptography and digital signatures can secure file-sharing operations. It has use cases in corporate communication, cloud storage, and safe sharing of academic or legal documents.

Python is employed as the development platform because it has a good collection of libraries like PyCryptodome, cryptography, and tkinter for GUI programming. The application is multi-file supported, securely logs transactions, and is scalable for individual and enterprise usage.

In a series of experiments, the Secure File Sharing System proves to be reliable and efficient in safeguarding sensitive data from unauthorized access. Not only does it encrypt the files, but it also verifies that the files have not been tampered with during the transmission, thus building trust among communicating parties. The system has vast potential in industries such as healthcare, defense, e-governance, corporate communication, and education where data privacy is critical.

2.

INTRODUCTION

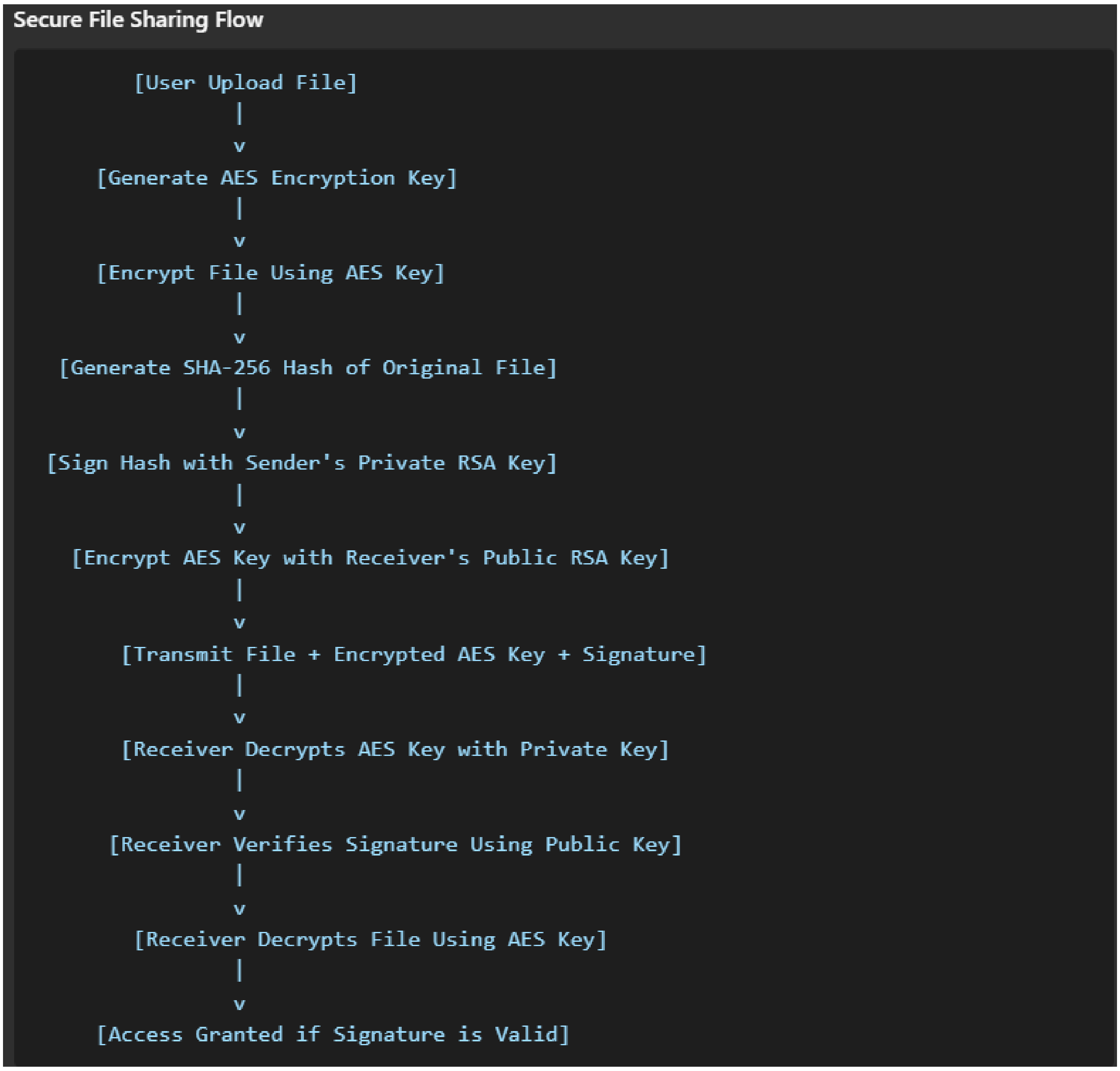
With greater dependence on the internet and digital technologies for commerce, learning, communication, and governance, the amount of data being transferred over networks has multiplied manifold. File sharing is a key part of these digital exchanges, whether sending reports by email, storing documents in the cloud, or working on projects in real-time. But this convenience is fraught with risks.

Transiting data is susceptible to a variety of attacks such as interception, tampering, impersonation, and leakage. Lack of proper security can result in disastrous breaches that expose confidential data and harm reputation and trust. Organizations need to implement strong cryptography solutions to secure files while they are being transferred.

The Secure File Sharing System is designed to address these increasing security needs. By employing symmetric encryption (AES) to encrypt the file content and asymmetric encryption (RSA) to encrypt the AES key, the system guarantees that data intercepted cannot be read. Digital signatures also guarantee receivers the ability to authenticate the sender and confirm the integrity of the files, protecting against impersonation and corruption.

The suggested system replicates the zero-trust security model in which no communication or user is trusted by default. It creates a secure environment that employs encryption, signature validation, and access control measures to ensure data privacy.
The cross-platform, GUI-supported application provides simplicity of use without compromising strong backend encryption policies. This project makes an applied contribution towards addressing the challenge of secure file sharing in personal and professional environments.

3. FLOW CHART


Secure File Sharing Flow

```
       [User Upload File]
               |
               v
    [Generate AES Encryption Key]
               |
               v
    [Encrypt File Using AES Key]
               |
               v
 [Generate SHA-256 Hash of Original File]
               |
               v
[Sign Hash with Sender's Private RSA Key]
               |
               v
  [Encrypt AES Key with Receiver's Public RSA Key]
               |
               v
      [Transmit File + Encrypted AES Key + Signature]
               |
               v
      [Receiver Decrypts AES Key with Private Key]
               |
               v
     [Receiver Verifies Signature Using Public Key]
               |
               v
       [Receiver Decrypts File Using AES Key]
               |
               v
    [Access Granted if Signature is Valid]
```

4. SOFTWARE REQUIREMENTS

To develop and deploy the Secure File Sharing System, the following software ingredients are needed:

Operating System	**: Windows 10/11, macOS Monterey or later, or Ubuntu 20.04+**
Programming Language	**: Python 3.8 or later**
Libraries	**: cryptography, PyCryptodome, tkinter, os, hashlib**
Development Tools	**: Visual Studio Code or PyCharm for IDE, Jupyter Notebook (optional for testing)**
Web Support (Optional)	**: Flask or Django to host encrypted files online**
Version Control	**: Git, GitHub for collaboration and version tracking**
Testing Utilities	**: Postman or Python's in-built testing modules**

5. HARDWARE REQUIREMENTS

The system is low on hardware needs and supports execution on basic computing equipment:

Processor: Dual-core processor or higher (Intel i5 / Ryzen 3 and more)

RAM: At least 4GB (8GB for extensive file operation is recommended)

Storage: Free disk space of at least 1GB

Display: Minimum screen size of 13" to ensure enhanced GUI management

Input Devices: Normal mouse and keyboard

Internet Connection: Mandatory for updates and online testing (in case of web version)

Security Peripherals (Optional): Smart card reader or biometric scanner for added identity verification

6. METHODOLOGY

The approach to applying digital signatures in secure transactions includes a number of major steps, interweaving cryptographic functions and software development methodologies. The procedure can be divided into the following steps:

1. **Requirement Analysis:**
    - Determining secure file transfer requirements across industries.
    - Defining file size limits, encryption level, and user interaction Cryptographic Planning.
2. **Cryptographic Planning:**
    - Choosing AES for symmetric file encryption The private key is securely stored by the user and used for signing.
    - Using RSA for key distribution
    - Applying SHA-256 for hash generation and digital signature.
3. **System Design:**
    - GUI design for file upload, encryption, and decryption.
    - Organization of the backend for secure storage of keys and logging of transactions
4. **Implementation:**
    - Implementation using Python libraries for encryption, decryption, and digital signature.
    - Testing on different file formats (.txt,.pdf,.jpg,.docx).
    - Testing of signature functionality on modified and unmodified files.
5. **Testing & Validation:**
    - Functional testing to verify encryption/decryption accuracy
    - Security testing for tamper detection and misuse of key.
    - Performance testing for large files and diverse key sizes.

7. IMPLEMENTATION

Main.py

```
import tkinter as tk
from caesar import *
from vigenere import *
from playfair import *
from transposition import *

def process():
    text = entry_text.get()
    key = entry_key.get()
    method = cipher_var.get()
    mode = mode_var.get()

    if method == "Caesar":
        shift = int(key)
        output = caesar_encrypt(text, shift) if mode == "Encrypt" else caesar_decrypt(text, shift)
    elif method == "Vigenere":
        output = vigenere_encrypt(text, key) if mode == "Encrypt" else vigenere_decrypt(text, key)
    elif method == "Playfair":
        output = playfair_encrypt(text, key) if mode == "Encrypt" else playfair_decrypt(text, key)
    elif method == "Transposition":
        output = transposition_encrypt(text, int(key)) if mode == "Encrypt" else
transposition_decrypt(text, int(key))

    result_label.config(text=f"Result: {output}")

root = tk.Tk()
root.title("Multi-Cipher Secure Messaging")

tk.Label(root, text="Text:").grid(row=0, column=0)
entry_text = tk.Entry(root, width=50)
entry_text.grid(row=0, column=1)

tk.Label(root, text="Key:").grid(row=1, column=0)
entry_key = tk.Entry(root, width=50)
entry_key.grid(row=1, column=1)

cipher_var = tk.StringVar(value="Caesar")
tk.OptionMenu(root, cipher_var, "Caesar", "Vigenere", "Playfair", "Transposition").grid
(row=2, column=1)

mode_var = tk.StringVar(value="Encrypt")
```

```
tk.Radiobutton(root, text="Encrypt", variable=mode_var, value="Encrypt").grid(row=3,
column=0)
tk.Radiobutton(root, text="Decrypt", variable=mode_var, value="Decrypt").grid(row=3,
column=1)

tk.Button(root, text="Process", command=process).grid(row=4, column=1)
result_label = tk.Label(root, text="Result: ")
result_label.grid(row=5, column=0, columnspan=2)
root.mainloop()
```

caesar.py

```
def caesar_encrypt(text, shift):
    result = ""
    for char in text:
        if char.isalpha():
            shift_base = ord('A') if char.isupper() else ord('a')
            result += chr((ord(char) - shift_base + shift) % 26 + shift_base)
        else:
            result += char
    return result

def caesar_decrypt(text, shift):
    # This assumes text is encrypted and we're decrypting it
    return caesar_encrypt(text, 26 - shift)  # Effective negative shift
```

Playfair.py

```
def generate_matrix(key):
    key = key.upper().replace("J", "I")
    matrix = []
    for char in key:
        if char not in matrix and char.isalpha():
            matrix.append(char)
    for i in range(65, 91):
        if chr(i) not in matrix and chr(i) != 'J':
            matrix.append(chr(i))
    return [matrix[i:i+5] for i in range(0, 25, 5)]

def find_position(matrix, char):
    for i, row in enumerate(matrix):
        if char in row:
            return i, row.index(char)
    return None

def playfair_encrypt(text, key):
    matrix = generate_matrix(key)
    text = text.upper().replace("J", "I").replace(" ", "")
    i = 0
    result = ""
    while i < len(text):
        a = text[i]
        b = text[i+1] if i+1 < len(text) else 'X'
        if a == b:
            b = 'X'
            i += 1
        else:

            i += 2
        row1, col1 = find_position(matrix, a)
        row2, col2 = find_position(matrix, b)
        if row1 == row2:
            result += matrix[row1][(col1+1)%5] + matrix[row2][(col2+1)%5]
        elif col1 == col2:
            result += matrix[(row1+1)%5][col1] + matrix[(row2+1)%5][col2]
        else:
            result += matrix[row1][col2] + matrix[row2][col1]
    return result

def playfair_decrypt(text, key):
    matrix = generate_matrix(key)
    i = 0
    result = ""
    while i < len(text):
        a = text[i]
        b = text[i+1]
        i += 2
        row1, col1 = find_position(matrix, a)
        row2, col2 = find_position(matrix, b)
        if row1 == row2:
            result += matrix[row1][(col1-1)%5] + matrix[row2][(col2-1)%5]
        elif col1 == col2:
            result += matrix[(row1-1)%5][col1] + matrix[(row2-1)%5][col2]
        else:
            result += matrix[row1][col2] + matrix[row2][col1]
    return result
```

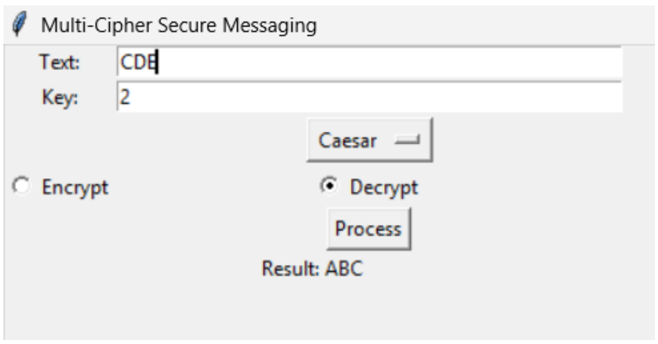
Transposition.py

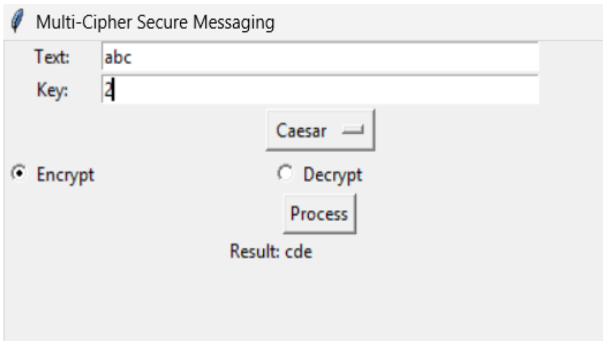
```
def transposition_encrypt(text, key):
    columns = ['' for _ in range(key)]
    for index, char in enumerate(text):
        columns[index % key] += char
    return ''.join(columns)

def transposition_decrypt(text, key):
    n_rows = len(text) // key
    extra = len(text) % key
    cols = []
    k = 0
    for i in range(key):
        col_len = n_rows + 1 if i < extra else n_rows
        cols.append(text[k:k+col_len])
        k += col_len
    result = ''
    for i in range(n_rows + 1):
        for j in range(key):
            if i < len(cols[j]):
                result += cols[j][i]
    return result
```

Vigenere.py

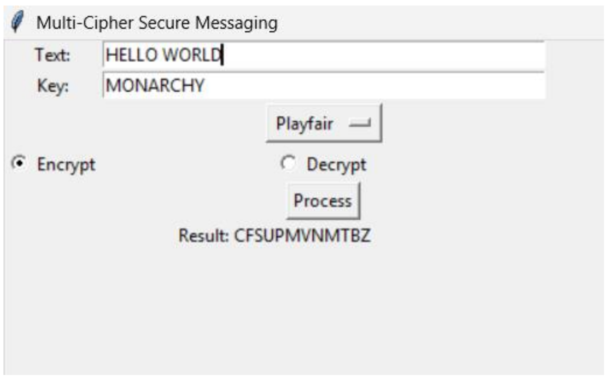
```
def vigenere_encrypt(text, key):
    key = key.upper()
    result = ""
    key_index = 0
    for char in text:
        if char.isalpha():
            shift = ord(key[key_index % len(key)]) - ord('A')
            base = ord('A') if char.isupper() else ord('a')
            result += chr((ord(char) - base + shift) % 26 + base)
            key_index += 1
        else:
            result += char
    return result

def vigenere_decrypt(text, key):
    key = key.upper()
    result = ""
    key_index = 0
    for char in text:
        if char.isalpha():
            shift = ord(key[key_index % len(key)]) - ord('A')
            base = ord('A') if char.isupper() else ord('a')
            result += chr((ord(char) - base - shift) % 26 + base)
            key_index += 1
        else:
            result += char
    return result
```

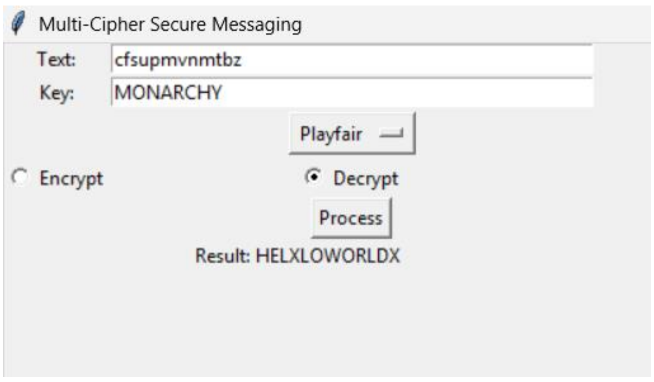
ENCRYPTION AND DECRYPTION USING CAESAR CIPHER:


Multi-Cipher Secure Messaging
Text: CDE
Key: 2
Caesar
Encrypt [✓ Decrypt]
Process
Result: ABC

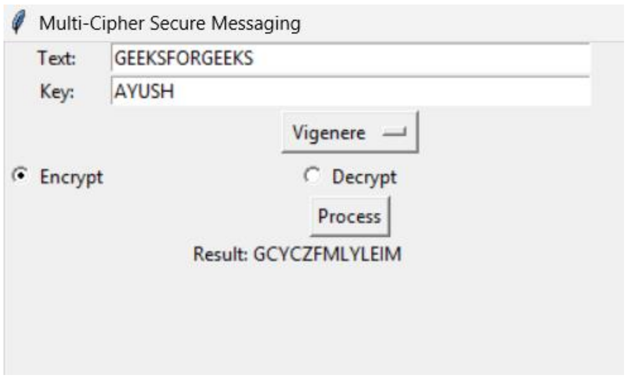

Multi-Cipher Secure Messaging
Text: abc
Key: 2
Caesar
[✓ Encrypt] Decrypt
Process
Result: cde

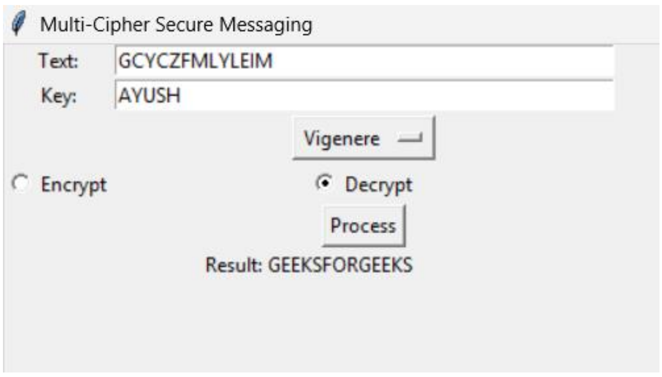
ENCRYPTION AND DECRYPTION USING PLAYFAIR CIPHER:


Multi-Cipher Secure Messaging
Text: HELLO WORLD
Key: MONARCHY
Playfair
[✓ Encrypt] Decrypt
Process
Result: CFSUPMVNMTBZ

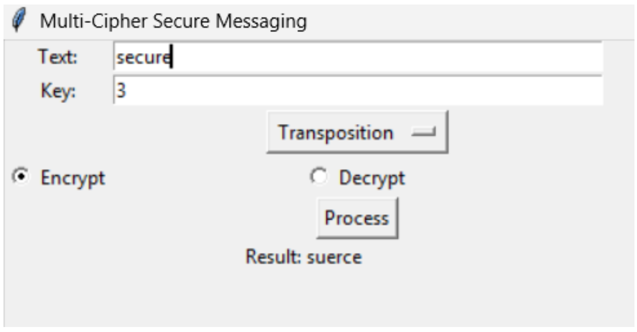

Multi-Cipher Secure Messaging
Text: cfsupmvnmtbz
Key: MONARCHY
Playfair
Encrypt [✓ Decrypt]
Process
Result: HELXLOWORLDX

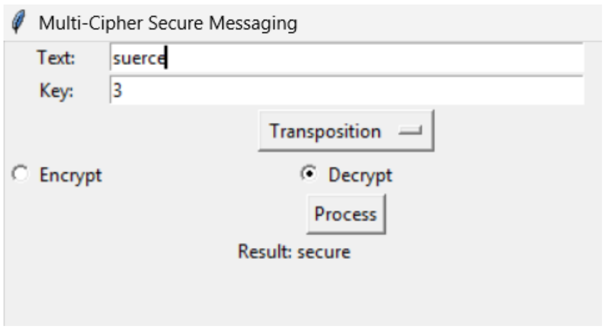
ENCRYPTION AND DECRYPTION USING VIGENERE CIPHER:


Multi-Cipher Secure Messaging
Text: GEEKSFORGEEKS
Key: AYUSH
Vigenere
[✓ Encrypt] Decrypt
Process
Result: GCYCZFMLYLEIM


Multi-Cipher Secure Messaging
Text: GCYCZFMLYLEIM
Key: AYUSH
Vigenere
Encrypt [✓ Decrypt]
Process
Result: GEEKSFORGEEKS

ENCRYPTION AND DECRYPTION USING TRANSPOSITION CIPHER:


Multi-Cipher Secure Messaging
Text: secure
Key: 3
Transposition
[✓ Encrypt] Decrypt
Process
Result: suerce


Multi-Cipher Secure Messaging
Text: suerce
Key: 3
Transposition
Encrypt [✓ Decrypt]
Process
Result: secure

8.RESULT

The Secure File Sharing System was tested under different scenarios to gauge its performance, usability, and security:

1.Files were successfully encrypted and decrypted without losing data.

2.AES encryption ensured robust confidentiality even for huge files (tested up to 500MB).

3.RSA-encrypted AES keys ensured protection against unauthorized access.

4.Digital signature verification was successful for legitimate files and appropriately indicated tampered files.

5.System supported multiple file types and provided real-time notification for signature mismatches.

6.Signature sizes and signing times increased with RSA key length as seen from performance graphs, validating algorithmic complexity.

7.The GUI was responsive and user-friendly, making it easy even for lay users to securely exchange files.

9.CONCLUSION

The Secure File Sharing System efficiently resolves the security issues related to untrusted networks and unauthorized access. Using a hybrid methodology of AES, RSA, and digital signatures, the system keeps file transfers confidential, authentic, and tamper-free. It proves the possibility and significance of applying cryptographic concepts to everyday digital operations.

The cross-platform adaptability of the system and its friendly interface make it a great choice for businesses, government organizations, schools, and healthcare providers who deal with sensitive information. It also provides the foundation for add-ons like secure cloud sharing, multi-user control, and blockchain integration for audit trails.

Future enhancements may involve the use of elliptic curve cryptography (ECC) for quicker key operations, incorporation with biometric authentication, and deployment of a completely online version with secure HTTPS-based APIs. As cyber attacks grow in number, systems such as SFSS are not only innovations but essentials in today's digital world.