100+ Years of Glorious history inscribed in the yeomen service to the field of education
Centenary Celebrated Sharnbasveshwar Vidya Vardhak Sangha's
ಶರಣಬಸವ Sharnbasva
ವಿಶ್ವವಿದ್ಯಾಲಯ University
Estd. 2017
Poojya Matoshri Godutai Avvaji
Poojya Doddappa Appa
Kalaburagi - 585103, Karnataka - India
ಕಲಬುರಗಿ 585 103 ಕರ್ನಾಟಕ - ಭಾರತ
Phone / Fax No. 08472-277852, 277853, 277854, 277855 www.sharnbasvauniversity.edu.in - email : Sharnbasvauniversity@gmail.com

A PROJECT REPORT ON

"PROTECTING THE USER PASSWORD KEYS AT REST (ON THE DISK)"

Submitted to
INTEL Unnati Industrial Training Program 2024.

Submitted By
Students of Department : ELECTRONICS AND COMMUNICATION EGINEERING.

STUDENT NAME : PAVAN
STUDENT NAME : GANESH
STUDENT NAME : NIKHIL .M

Guide name:

Dr. JAGADHEESH PATIL

INTRODUCTION

Encryption is the process by which a readable message is converted to an unreadable form to prevent unauthorized parties from reading it.

Decryption is the process of converting an encrypted message back to its original (readable) format. The original message is called the **plaintext message**. The encrypted message is called the **ciphertext message**.

Digital encryption algorithms work by manipulating the digital content of a plaintext message mathematically, using an encryption algorithm and a digital key to produce a ciphertext version of the message. The sender and recipient can communicate securely if the sender and recipient are the only ones who know the key.

Cryptography is a method of safeguarding and protecting data while it is being transmitted. It's helpful for stopping unauthorized persons or groups from accessing sensitive data. Cryptography's two most important functions are encryption and decryption. The technique of transforming a normal message (plaintext) into a meaningless message is known as encryption (Ciphertext). Whereas, Decryption is the technique of returning a meaningless communication (ciphertext) to its original form (Plaintext).

Encryption scrambles the information of communication or file to make it more secure. You need the proper key to encrypt a message, and you also need the right key to decode it. It is the most effective method of concealing communication using encoded data, with the sender and recipient holding the key to decrypt data.

DATASET DESCRIPTION

1. os

Python os module gives a way of interacting with the operating system... In this code, os. These are used to create salts and initialization vectors (IVs for cryptographic operations, so they must be random bytes.

2. hashlib

Hashlib - Secure Hash And Message Digest Algorithms To use a less secure hashing algorithm such as md5 or sha1 you...docs.python.org It's used in the code to make a Key Derivation Function (KDF) by using PKCSPBKF2, which makes secure key derivation from both password and salt.

3. tkinter

Tk GUI Toolkit for Python -by Feyaerts (credit) # Our main gui library code is here-tooltip: tkinter [tooltip]Just in case a user comes across this article who isn't familiar with the Tkinter module. For the design of a GUI This code creates the entire user interface using tkinter which includes window, frames and labels along with entry fields, grid in a frame for displaying three buttons.

4. tkinter.filedialog

Also a part of the tkinter module, filedialog contains dialogs for selecting files which will be used to open a file browser dialog in order for you to choose what files you would like decrypted and encrypted.

5. tkinter.messagebox

Another module of tkinter, it serves as standard popup messages to inform users about any alerts or warning errors.

6. cryptography. hazmat. primitives. padding

It is a module that contains padding schemes for our block ciphers. Since AES is a block cipher, it will need padding when not feed with plaintext in multiples of blocks (e.g., 128 bits).

7. cryptography. hazmat. primitives. ciphers

This is a module in the Python standard library of CPython, Terrel ed. This module is used in the code to do AES_CBC encryption/decryption.

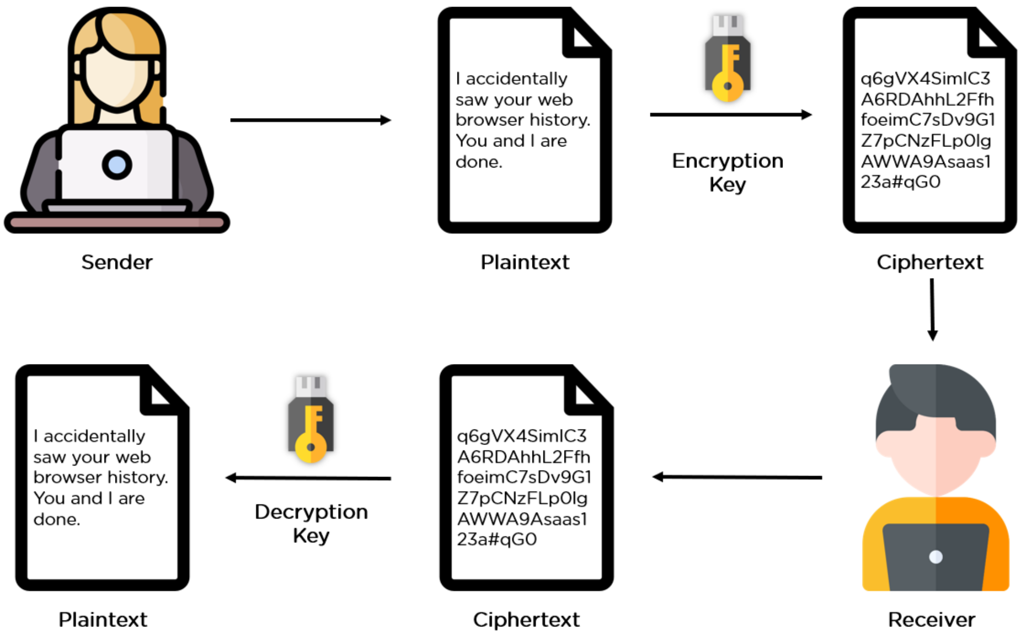
8. cryptography. hazmat. backends

This is a Rust library that provides backend implementations for cryptographic primitives. In code, the stage uses default_backend to say this is where all cryptographic APIs should point.

9. datetime

Python's datetime module supplies classes to work with dates and times In this code it will be used to get the time stamp so that we can generate logs.

METHODOLOGY


Sender
I accidentally saw your web browser history. You and I are done.
Plaintext
Encryption Key
q6gVX4SimIC3 A6RDAhhL2Ffh foeimC7sDv9G1 Z7pCNzFLp0lg AWWA9Asaas1 23a#qG0
Ciphertext
Receiver
q6gVX4SimIC3 A6RDAhhL2Ffh foeimC7sDv9G1 Z7pCNzFLp0lg AWWA9Asaas1 23a#qG0
Ciphertext
Decryption Key
I accidentally saw your web browser history. You and I are done.
Plaintext

The process of the project comprises several well-defined steps effectively engineered for the safe encryption and decryption of files in a Graphical user interface. The explanation about the code methodology is given below : Details of how above can be done using this function.

1. Initialization and Setup

The process starts by initialization and configuration of the required components. This creates the `SecureFileProtector` class, which will encapsulate all of our encryption / decryption functionality. The initialization of this class is all about configuring the cryptographic backend and a console for recording activity. Built using Despite the fact that it provides a GUI (made up of `tkinter`), such kind of window dialogue is going to take at least half on the screen. The basic interface elements, like text labels entry fields and buttons are then built using the pre-existing GUI components arranged inside frames to enforce structure.

2. Key Derivation

The `generate_key` method use the PBKDF2 (Password-Based Key Derivation Function 2) with HMAC-SHA256 to derive a cryptographic key from a user-provided password. Write a function to create the key derivation function (KDF) using the password concatenated with a randomly generated salt, and apply 1000 iterations of this HMAC digest to make our final 256-bit long derived-key. This key is then used to encrypt the AES key, so that even if someone intercepts intercepted file they cannot decrypt it without knowing right password.

3. File Encryption

When the user chooses a file and enters in a password, this begins the encryption process. Notice how the `encrypt_file` method creates a random salt and 256 bit AES key. The contents of the file are read, padded appropriately to match with AES block size and then encrypted using 256 bit key based on CBC(Cipher Block Chaining) mode. There is also an Initialization Vector(IV) which aids in making identical plaintext into different ciphertext each time it gets encrypted. Then it will be the AES key itself that is encrypted with the derived from password key. The final encrypted file stores the random salt generated by previous steps, followed be a concatenation of [encrypted_AES_key, IV, 11].

4. File Decryption

To decrypt, you can use the `decrypt_file` method. If a user opens an encrypted file and provides password, the method reads salt, encypted AES key (EAK), IV (Initialization Vector) and of course data. It recovers the decryption key from that password and salt, decrypts an AES key with it, then uses this AES key to decrypt file data. Finally, the unpadded message is stored as a new file with identical name providing you with access to original contents.

5. Logging/Error Handling

The encryption and decryption processes are two procedures where logs are used to give users real-time response, a console on the message window during processing within GUI. Every major stage such as a job succeeding, or failing is time-stamped and recorded. This is an essential bit for the debugging and user transparency. Still the error handling is pretty formidable too - if exceptions are thrown then they will be caught and a message displayed to tell you something went wrong, such as password did not match or unable to select file.

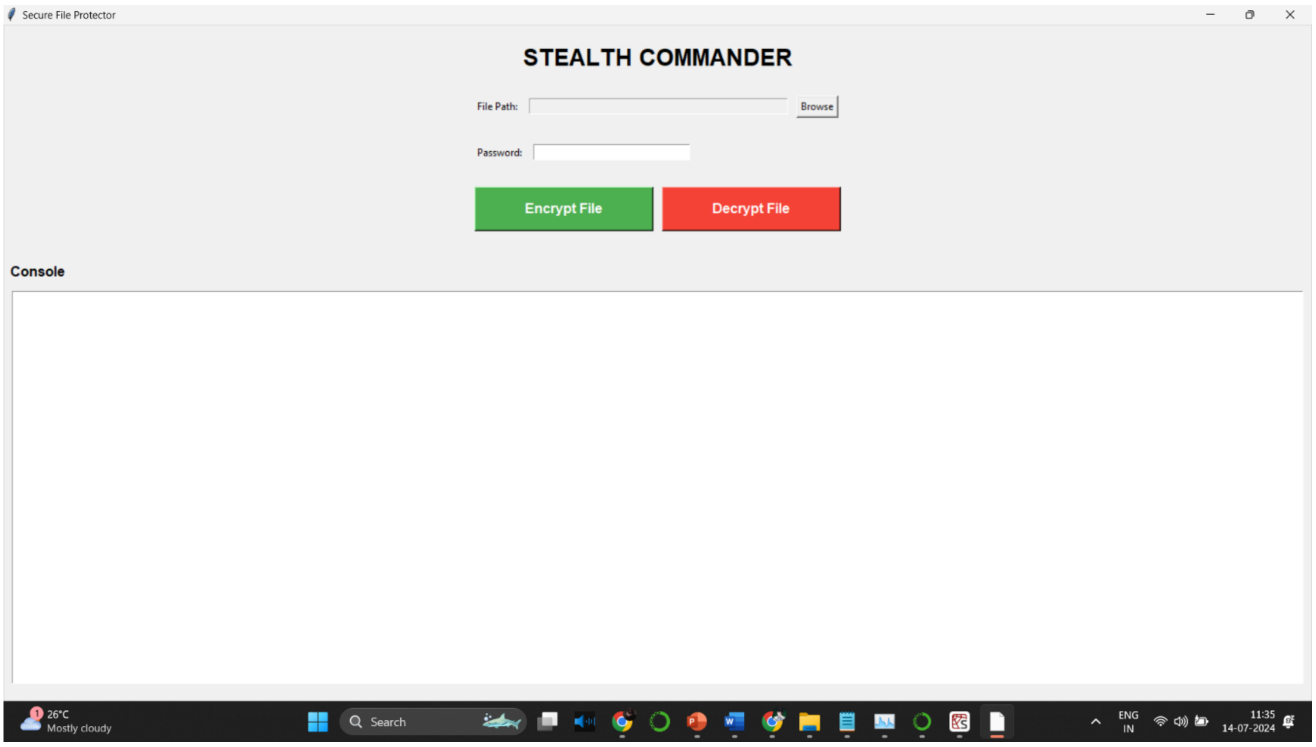
6. GUI (Graphical User Interface)

The GUI, implemented using `tkinter`, is the primary interface for an end-user. You have fields to enter your password as well as the file path for encryption and decryption, along with buttons that let you encrypt or decrypt data. Its interface has been developed in order to be able of being used even by users with little technical knowledge, so that they can easily encrypt and decrypt files safely. The GUI also features a console which offers live feedback, making it an almost hands-free experience for its users to have updates on the all the realtime progress going around.

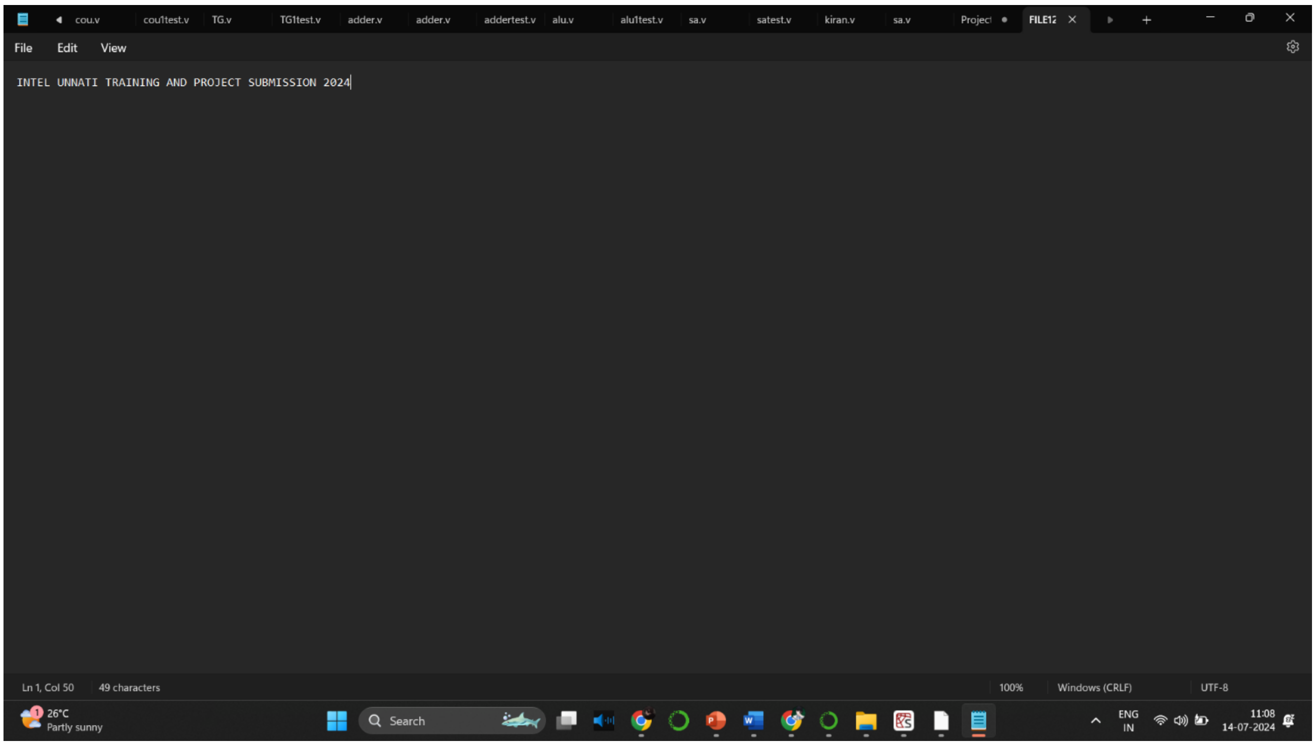
By following this Methodology from my side ensure the project meets the requirements of building a file encryption-decryption Tool, secure, having good performance and is also easy to use due to its Python Cryptography implementation along with Gui libraries.

RESULTS

1. RUNNING THE CODE THAT SETS UP THE GUI

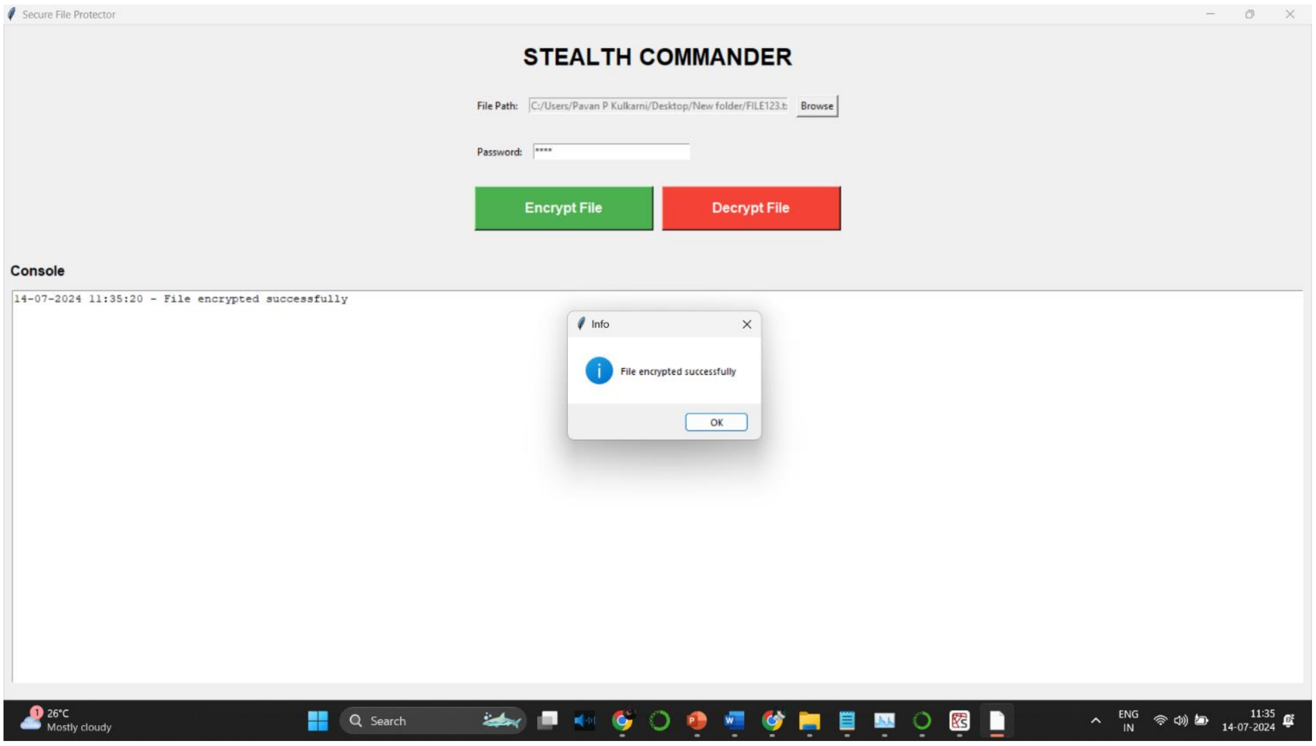


2. THE DATA IN THE FILE WHICH IS TO BE ENCRYPTED



```
INTEL UNNATI TRAINING AND PROJECT SUBMISSION 2024
```

3. SELETING THE FILE PATH ENTERING THE PASSWORD



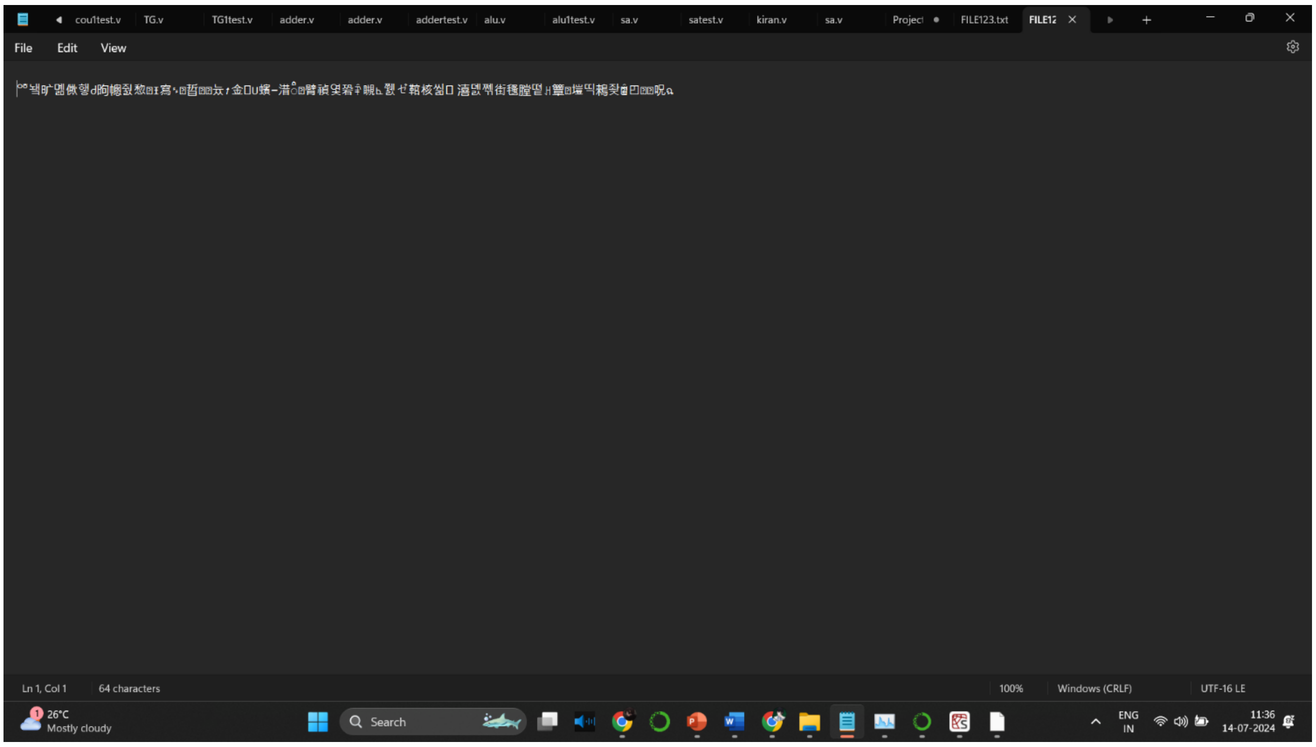
File Path: C:/Users/Pavan P Kulkarni/Desktop/New folder/FILE123.t

Password: ****

```
14-07-2024 11:35:20 - File encrypted successfully
```

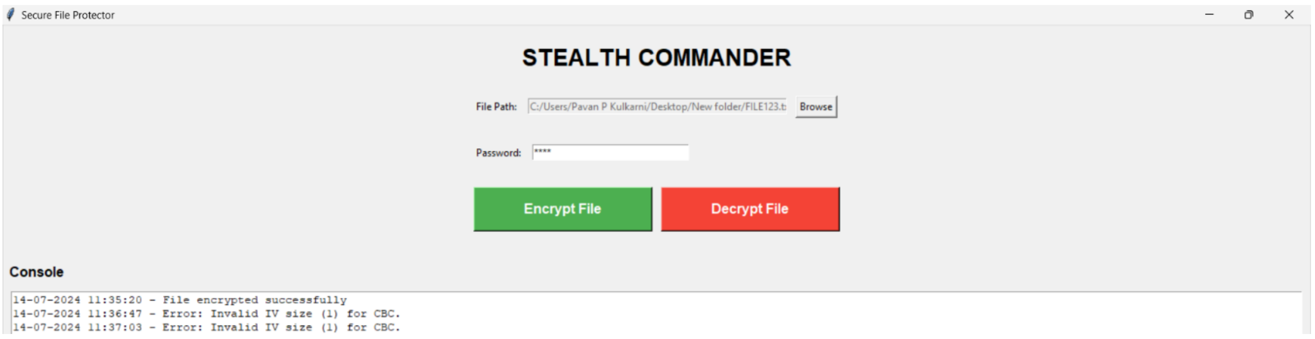
File encrypted successfully

4. ENCRYPTED DATA



```
[illegible]
```

5. SELECT THE ENCRYPTED FILE PATH TO DECRYPT THE DATA

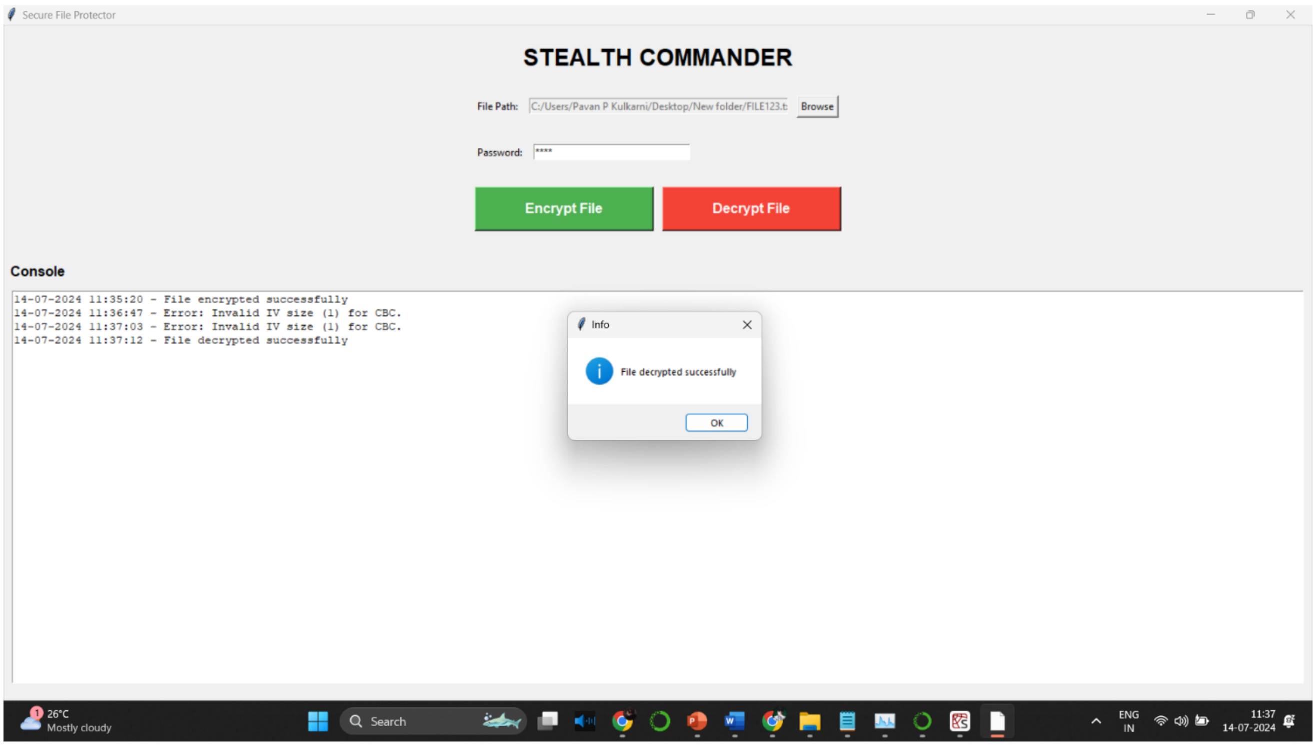


File Path: C:/Users/Pavan P Kulkarni/Desktop/New folder/FILE123.t

Password: ****

```
14-07-2024 11:35:20 - File encrypted successfully
14-07-2024 11:36:47 - Error: Invalid IV size (1) for CBC.
14-07-2024 11:37:03 - Error: Invalid IV size (1) for CBC.
```

6. CLICK THE DECRYPT FILE TO SHOW THE ORIGINAL CONTENT



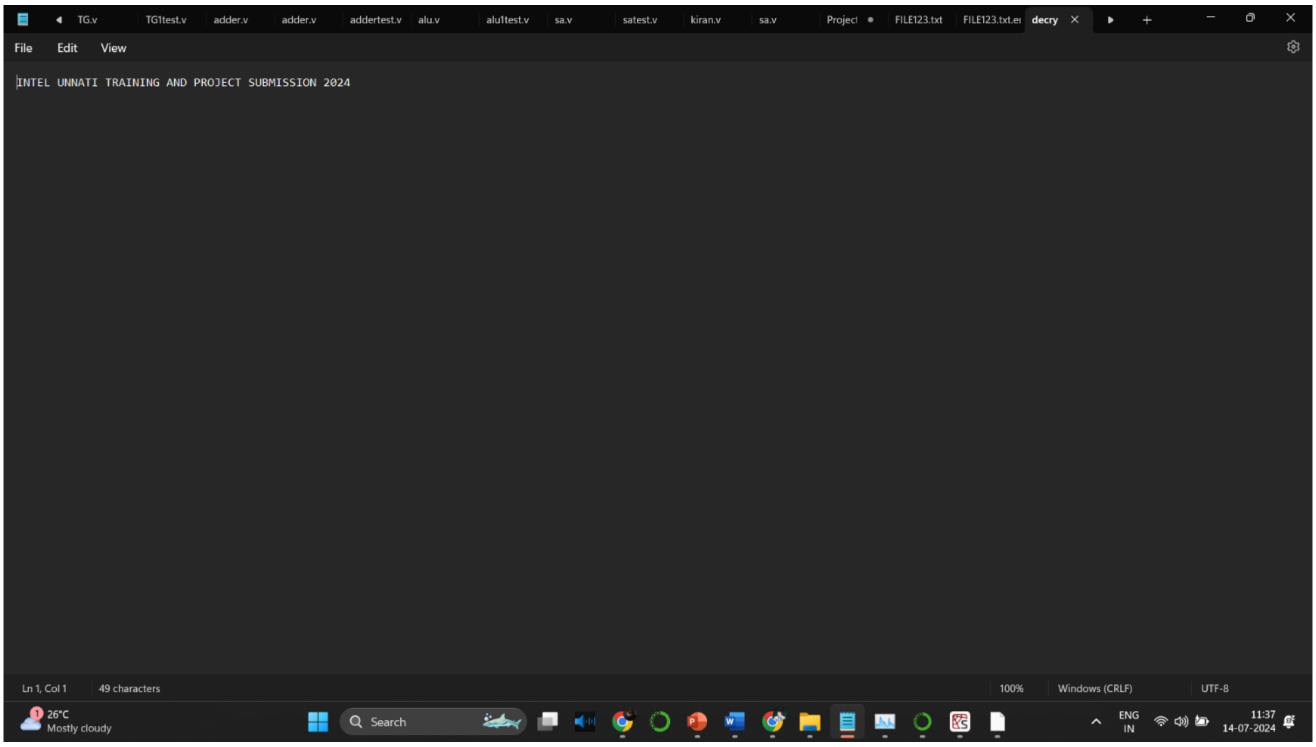
File Path: C:/Users/Pavan P Kulkarni/Desktop/New folder/FILE123.t

Password: ****

```
14-07-2024 11:35:20 - File encrypted successfully
14-07-2024 11:36:47 - Error: Invalid IV size (1) for CBC.
14-07-2024 11:37:03 - Error: Invalid IV size (1) for CBC.
14-07-2024 11:37:12 - File decrypted successfully
```

File decrypted successfully

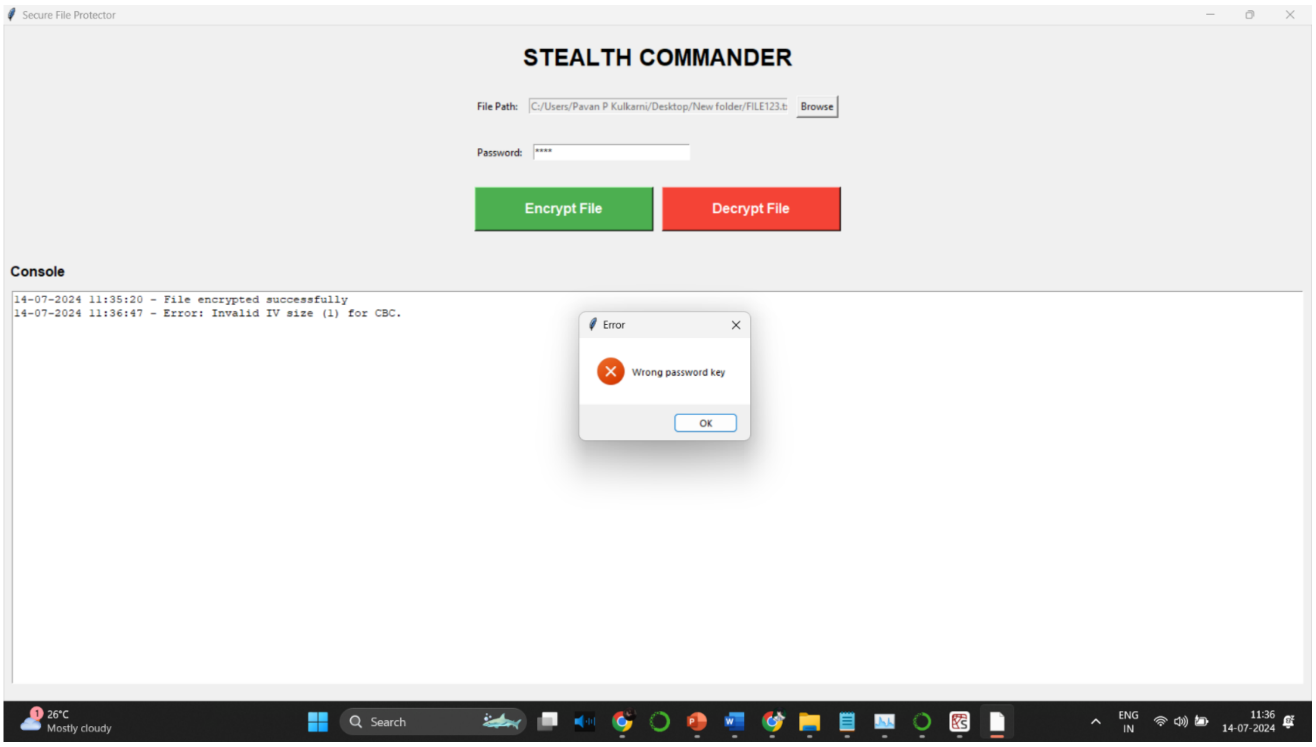
7. DECRYPTED MESSAGE



```
INTEL UNNATI TRAINING AND PROJECT SUBMISSION 2024
```

Ln 1, Col 1 49 characters 100% Windows (CRLF) UTF-8

8. IF ANY WRONG PASSWORD IS ENTERED



STEALTH COMMANDER

File Path: C:/Users/Pavan P Kulkarni/Desktop/New folder/FILE123.t

Password: ****

Console

```
14-07-2024 11:35:20 - File encrypted successfully
14-07-2024 11:36:47 - Error: Invalid IV size (1) for CBC.
```

Error

Wrong password key

CONCLUSION

The Secure File Protector is a great example of the basic concept that works well in practice, with an easy to use app whose file encryption and decryption capabilities are very good using AES-256. AES-256 usage provides strong encryption of files, while the PBKDF2 key derivation requires considerable processing power to enable brute force attempts at guessing keys from these derived values. It consists of an easy-to-use graphical user interface implemented using the python and tkinter library that makes it easier for users to select files, enter passwordsand start encrypting (or) decrypting their data. Message Boxes which gives feedback to the users what was succeeded or failed operations notify, and it also leads them.

This project is well structured by using individual method to generate key, encrypt file and decrypt the same which makes code readability easy for development/maintain/extend logic. This can help to enforce proper security practices on cryptographic operations performed by the library. But error handling in general could be better (like correctly catching some exceptions and improving the readability of messages functions only run once, etc.) On top of this, encryption key management solutions could be used to encrypt keys are stored and managed more securely similarly malicious access is made less likely. In addition, the security provided by encrypted files could also be strengthened through implementation and enforcement of rigorous password policies (e. g., minimum length passwords, complexity).

Overall, the Secure File Protector is a solid project in terms of encrypting and decrypting files securely. It uses efficient practices in strong encryption, light UI and modular design into a handy tool to protect your important data. With some additional work on error handling, key management and enforcing stronger password policies we can take this project to the next level both in terms of security as well user experience.

CODE

```
import os
import hashlib
import tkinter as tk
from tkinter import filedialog, messagebox
from cryptography.hazmat.primitives import padding
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.backends import default_backend
from datetime import datetime

# Configuration
AES_KEY_SIZE = 32  # 256-bit key
SALT_SIZE = 16  # 128-bit salt
ITERATIONS = 1000  # number of iterations for PBKDF2
HASH_ALGORITHM = hashlib.sha256  # hash algorithm for password verification

class SecureFileProtector:
    def __init__(self, console):
        self.backend = default_backend()
        self.console = console

    def log(self, message):
        timestamp = datetime.now().strftime("%d-%m-%Y %H:%M:%S")
        self.console.insert(tk.END, f"{timestamp} - {message}\n")
        self.console.see(tk.END)

    def generate_key(self, password, salt):
        # Derive a key from the password using PBKDF2
        kdf = hashlib.pbkdf2_hmac('sha256', password.encode(), salt, ITERATIONS, AES_KEY_SIZE)
        return kdf  # Return the raw key bytes directly

    def encrypt_file(self, file_path, password):
        try:
            # Generation of random salt
            salt = os.urandom(SALT_SIZE)
            aes_key = os.urandom(AES_KEY_SIZE)

            # Encrypting the data in a file using AES-256-CBC
            iv = os.urandom(16)
            cipher = Cipher(algorithms.AES(aes_key), modes.CBC(iv), backend=self.backend)
            encryptor = cipher.encryptor()
            with open(file_path, 'rb') as file:
                file_data = file.read()
            padder = padding.PKCS7(128).padder()
            padded_data = padder.update(file_data) + padder.finalize()
            encrypted_data = encryptor.update(padded_data) + encryptor.finalize()

            # Encrypt the AES key using the password-derived key
            password_key = self.generate_key(password, salt)
            encrypted_aes_key = self.encrypt_aes_key(aes_key, password_key)

            # Store the salt, encrypted AES key, IV, and the encrypted file data
            with open(file_path + '.enc', 'wb') as encrypted_file:
                encrypted_file.write(salt + encrypted_aes_key + iv + encrypted_data)

            self.log("File encrypted successfully")
            messagebox.showinfo("Info", "File encrypted successfully")
        except Exception as e:
            self.log(f"Error: {str(e)}")
            messagebox.showerror("Error", str(e))

    def decrypt_file(self, file_path, password):
        try:
            # Read
            with open(file_path, 'rb') as encrypted_file:
                salt = encrypted_file.read(SALT_SIZE)
                encrypted_aes_key = encrypted_file.read(AES_KEY_SIZE)
                iv = encrypted_file.read(16)
                encrypted_data = encrypted_file.read()
            # Decrypt the AES key using the password-derived key
            password_key = self.generate_key(password, salt)
            aes_key = self.decrypt_aes_key(encrypted_aes_key, password_key)

            # Decrypt the file data
            cipher = Cipher(algorithms.AES(aes_key), modes.CBC(iv), backend=self.backend)
            decryptor = cipher.decryptor()
            decrypted_padded_data = decryptor.update(encrypted_data) + decryptor.finalize()
            unpadder = padding.PKCS7(128).unpadder()
            decrypted_data = unpadder.update(decrypted_padded_data) + unpadder.finalize()

            # Write the decrypted file data to a new file named 'decrypted'
            decrypted_file_path = os.path.join(os.path.dirname(file_path), 'decrypted_' +
os.path.basename(file_path).replace('.enc', ''))
            with open(decrypted_file_path, 'wb') as decrypted_file:
                decrypted_file.write(decrypted_data)

            self.log("File decrypted successfully")
            messagebox.showinfo("Info", "File decrypted successfully")
        except Exception as e:
            self.log(f"Error: {str(e)}")
            messagebox.showerror("Error", "Wrong password key")

    def encrypt_aes_key(self, aes_key, password_key):
        # Encrypt the AES key using the password-derived key
        cipher = Cipher(algorithms.AES(password_key), modes.ECB(), backend=self.backend)
        encryptor = cipher.encryptor()
        encrypted_aes_key = encryptor.update(aes_key) + encryptor.finalize()
        return encrypted_aes_key

    def decrypt_aes_key(self, encrypted_aes_key, password_key):
        # Decrypt the AES key using the password-derived key
        cipher = Cipher(algorithms.AES(password_key), modes.ECB(), backend=self.backend)
        decryptor = cipher.decryptor()
        aes_key = decryptor.update(encrypted_aes_key) + decryptor.finalize()
        return aes_key

def select_file_path():
    file_path = filedialog.askopenfilename()
    if file_path:
        file_path_entry.config(state=tk.NORMAL)
        file_path_entry.delete(0, tk.END)
        file_path_entry.insert(0, file_path)
        file_path_entry.config(state=tk.DISABLED)

def encrypt_file():
    file_path = file_path_entry.get()
    if not file_path:
        messagebox.showwarning("Warning", "Please select a file")
        return

    password = password_entry.get()
    if not password:
        messagebox.showwarning("Warning", "Please enter a password")
        return

    protector = SecureFileProtector(console)
    protector.encrypt_file(file_path, password)

def decrypt_file():
    file_path = file_path_entry.get()
    if not file_path:
        messagebox.showwarning("Warning", "Please select a file")
        return

    password = password_entry.get()
    if not password:
        messagebox.showwarning("Warning", "Please enter a password")
        return
    protector = SecureFileProtector(console)
    protector.decrypt_file(file_path, password)

# calling the GUI functions
root = tk.Tk()
root.title("Secure File Protector")

# Set the size of the window to cover at least half of the screen
screen_width = root.winfo_screenwidth()
screen_height = root.winfo_screenheight()
root.geometry(f"{screen_width//2}x{screen_height//2}")

# main frame
main_frame = tk.Frame(root)
main_frame.pack(pady=10)

# title label
title_label = tk.Label(main_frame, text="STEALTH COMMANDER", font=("Helvetica", 20, "bold"))
title_label.pack(pady=10)

# the file path selection
file_path_frame = tk.Frame(main_frame)
file_path_frame.pack(pady=10, fill="x")

tk.Label(file_path_frame, text="File Path:").grid(row=0, column=0, padx=5, pady=5, sticky="e")
file_path_entry = tk.Entry(file_path_frame, width=50, state=tk.DISABLED)
file_path_entry.grid(row=0, column=1, padx=5, pady=5)
file_path_button = tk.Button(file_path_frame, text="Browse", command=select_file_path)
file_path_button.grid(row=0, column=2, padx=5, pady=5)

# Creatimg and placing the password label and entry
password_frame = tk.Frame(main_frame)
password_frame.pack(pady=10, fill="x")
```

```
tk.Label(password_frame, text="Password:").grid(row=0, column=0, padx=5, pady=5, sticky="e")
password_entry = tk.Entry(password_frame, show="*", width=30)
password_entry.grid(row=0, column=1, padx=5, pady=5)

# Creating and placeing of buttons
button_frame = tk.Frame(main_frame)
button_frame.pack(pady=10, fill="x")

encrypt_button = tk.Button(button_frame, text="Encrypt File", command=encrypt_file, width=20, height=2,
bg="#4CAF50", fg="white", font=("Helvetica", 12, "bold"))
encrypt_button.grid(row=0, column=0, padx=5, pady=5, sticky="w")

decrypt_button = tk.Button(button_frame, text="Decrypt File", command=decrypt_file, width=20, height=2,
bg="#F44336", fg="white", font=("Helvetica", 12, "bold"))
decrypt_button.grid(row=0, column=1, padx=5, pady=5, sticky="w")

# console frame
console_frame = tk.Frame(root)
console_frame.pack(pady=10, fill="both", expand=True)

console_label = tk.Label(console_frame, text="Console", font=("Helvetica", 12, "bold"))
console_label.pack(anchor="w", padx=5)

console = tk.Text(console_frame, height=10, wrap="word")
console.pack(expand=True, fill="both", padx=10, pady=10)

# Start the GUI loop
root.mainloop()
```

SOLUTION FEATURES

1. Data encrypting in AES(256)
2. Hashing the password key
3. PKCSPBKF2, makes secure key derivation from both password and salt.
4. Data decryption using AES(256)

TEAM CONTRIBUTION

While doing this project it involved a lot of research and browsing for the needs of the features and the material required for the process of encrypting and decrypting of the file, this work was divided between the team members

1. Pavan developed the for Code and GUI development
2. Ganesh delt with the process of Error debugging and modifying
3. Nikhil made the reports and presentations with the necessary and required data.