Task 4: ADVANCED ENCRYPTION TOOL

Instruction:

BUILD A TOOL TO ENCRYPT AND DECRYPT FILES USING ADVANCED ALGORITHMS LIKE AES-256.

DELIVERABLE:

A ROBUST ENCRYPTION APPLICATION WITH A USER-FRIENDLY INTERFACE.

Script:

```
import tkinter as tk
from tkinter import filedialog, messagebox
from Cryptodome.Cipher import AES
from Cryptodome.Random import get_random_bytes
import hashlib
import os

class EncryptionApp:
    def __init__(self, master):
        self.master = master
        master.title("AES-256 File Encryption/Decryption")
        master.geometry("400x300")

        self.file_path = tk.StringVar()
        self.password = tk.StringVar()

        tk.Label(master, text="File:").grid(row=0, column=0, pad
        tk.Entry(master, textvariable=self.file_path, width=30)
        tk.Button(master, text="Browse", command=self.browse_fil

        tk.Label(master, text="Password:").grid(row=1, column=0,
        tk.Entry(master, textvariable=self.password, show="*", w

        tk.Button(master, text="Encrypt", command=self.encrypt_f
        tk.Button(master, text="Decrypt", command=self.decrypt_f

    def browse_file(self):
        filename = filedialog.askopenfilename()
        self.file_path.set(filename)

    def encrypt_file(self):
        file_path = self.file_path.get()
        password = self.password.get()

        if not file_path or not password:
            messagebox.showerror("Error", "Please select a file
            return

        try:
            key = hashlib.sha256(password.encode()).digest()
            with open(file_path, 'rb') as file:
                data = file.read()

            cipher = AES.new(key, AES.MODE_GCM)
            ciphertext, tag = cipher.encrypt_and_digest(data)

            encrypted_file_path = file_path + ".encrypted"
            with open(encrypted_file_path, 'wb') as file:
                [file.write(x) for x in (cipher.nonce, tag, ciph

            messagebox.showinfo("Success", f"File encrypted and
        except Exception as e:
            messagebox.showerror("Error", f"Encryption failed:
```

```
    def decrypt_file(self):
        file_path = self.file_path.get()
        password = self.password.get()

        if not file_path or not password:
            messagebox.showerror("Error", "Please select a file
            return

        try:
            key = hashlib.sha256(password.encode()).digest()
            with open(file_path, 'rb') as file:
                nonce = file.read(16)
                tag = file.read(16)
                ciphertext = file.read()

            cipher = AES.new(key, AES.MODE_GCM, nonce=nonce)
            data = cipher.decrypt_and_verify(ciphertext, tag)

            decrypted_file_path = os.path.splitext(file_path)[0
            with open(decrypted_file_path, 'wb') as file:
                file.write(data)

            messagebox.showinfo("Success", f"File decrypted and
        except Exception as e:
            messagebox.showerror("Error", f"Decryption failed:

root = tk.Tk()
app = EncryptionApp(root)
root.mainloop()
```

This application provides a robust encryption solution using AES-256 with a user-friendly graphical interface. Here's a breakdown of its key features:

User Interface

The application uses `Tkinter` to create a simple and intuitive GUI. It includes:

- A file selection field with a browse button
- A password entry field
- Encrypt and Decrypt buttons

Encryption Process

1. The user selects a file and enters a password.
2. The password is hashed using SHA-256 to create a 32-byte key.
3. AES-256 in GCM (Galois/Counter Mode) is used for encryption, providing both confidentiality and authenticity.
4. The encrypted file is saved with a ".encrypted" extension.

Decryption Process

1. The user selects the encrypted file and enters the password.
2. The application reads the nonce and tag from the file.
3. It then decrypts the file using the provided password and verifies its integrity.
4. The decrypted file is saved without the ".encrypted" extension.

Security Features

- Uses AES-256, a highly secure encryption algorithm.
- Employs GCM mode, which provides authentication along with encryption.
- Generates a unique nonce (number used once) for each encryption operation.
- Uses SHA-256 for key derivation from the password.

Error Handling

The application includes error handling to manage issues such as incorrect passwords, file access problems, or corruption of encrypted files.

To use this application:

1. Run the script to launch the GUI.
2. Select a file using the "Browse" button.
3. Enter a strong password.
4. Click "Encrypt" to encrypt the file or "Decrypt" to decrypt an encrypted file.