An encryption and decryption tool

**Project Objective**

The main goal of this project is to create a tool that makes file encryption and decryption simple and accessible to everyone. We want to provide users with a secure and user-friendly way to protect their sensitive files by encrypting them, and just as easily, allowing them to decrypt their files when needed. Whether you're dealing with confidential documents, personal photos, or any other type of data, this tool ensures that your files remain private and secure.

This project is designed with simplicity in mind, featuring a straightforward interface that guides users through the entire encryption and decryption process. The user selects the file, enters a password, chooses one of the available encryption algorithms (AES, Blowfish, or 3DES), and then encrypts or decrypts the file at the click of a button. It's a secure, no-fuss solution for anyone looking to keep their files safe.

As we look ahead, we’re excited to build on this foundation by adding a **file-sharing feature**. This will allow users to securely share their encrypted files with others, taking the protection of personal data one step further. We believe that everyone should have easy access to strong security for their digital files, and this project is just the beginning of that vision.

The code :-

The code is provided in the directory the document is shared .

The explanation of the code :-

**1. Libraries Used**

* **os**:
  + This module allows you to interact with the operating system, which is essential for things like working with files or generating random values. In your code, os.urandom() is used to create secure random initialization vectors (IVs) for encryption, and os.path.exists() is used to check if the file the user selects actually exists before proceeding.
* **tkinter**:
  + This is Python’s standard library for building graphical user interfaces (GUIs). It allows you to create windows, buttons, text boxes, and other interactive elements. In your code, it’s used to build a simple but effective interface where the user can enter the file path, choose an algorithm, and interact with the encryption/decryption process through buttons and input fields.
* **cryptography.hazmat.primitives.ciphers**:
  + The cryptography library is what makes your encryption and decryption possible. It provides access to various cryptographic functions, including ciphers and encryption modes. The modules you import here (Cipher, algorithms, and modes) are used to implement the actual encryption and decryption logic for different algorithms like AES, Blowfish, and 3DES.
* **cryptography.hazmat.primitives.padding**:
  + Encryption algorithms typically work on blocks of data of a fixed size. This means that when data doesn’t neatly fit into these blocks, it must be padded to meet the required length. The padding.PKCS7 function is used in your code to add extra bytes to data before encryption, and it removes those extra bytes during decryption to restore the original data.
* **cryptography.hazmat.backends**:
  + This is a part of the cryptography library that provides access to the backend (the actual implementation) of the cryptographic operations. In your code, you’re using the default\_backend() to let the library automatically choose the appropriate backend for performing encryption and decryption.

**2. pad\_data(data, algorithm)**

* **Purpose**:
  + Encryption algorithms often require data to be a certain size (called a block size), so when the data doesn’t fit neatly into the expected block size, you need to pad it out. The padding ensures that the encryption algorithm can process the data properly without errors.
* **How It Works**:
  + The function first checks what algorithm you're using (AES, Blowfish, or 3DES). Each of these algorithms has a specific block size, so the function adjusts accordingly:
    - **AES** uses 128-bit blocks (16 bytes), so the data is padded to make sure it fits into multiples of 16 bytes.
    - **Blowfish** and **3DES** use 64-bit blocks (8 bytes), so padding is applied accordingly.
* **Why It's Important**:
  + Padding ensures that the input data can be processed by the encryption algorithm, even if it’s not an exact multiple of the required block size. Without padding, encryption could fail, or the encrypted data could be corrupted.

**3. unpad\_data(padded\_data, algorithm)**

* **Purpose**:
  + After decryption, the data will still have the extra padding bytes. This function removes those padding bytes to get back the original content.
* **How It Works**:
  + It looks at the algorithm you're using and uses the same block size as in the pad\_data function to determine how many bytes to remove.
  + The unpadding is done using PKCS7, the same scheme used for padding. It works by checking the padding pattern and removing it to recover the original data.
* **Why It's Important**:
  + Without unpadding, you’d end up with extra, meaningless bytes in your decrypted data, which would cause issues when trying to use it. This function restores the data to its original, usable form after decryption.

**4. encrypt\_aes(file\_path, key)**

* **Purpose**:
  + This function handles the process of encrypting a file using the AES (Advanced Encryption Standard) algorithm, one of the most widely used and secure encryption methods.
* **How It Works**:
  + First, the function reads the content of the file that you want to encrypt. It then pads the data to fit the AES block size.
  + Next, it generates a random 16-byte initialization vector (IV) using os.urandom(). The IV is used to add randomness to the encryption process, ensuring that even if the same file is encrypted multiple times with the same key, the output will be different each time.
  + The cipher is then created using the AES algorithm with the provided key and the generated IV, and the encryption happens in **CFB mode** (Cipher Feedback mode). This mode is a stream cipher, meaning it encrypts the data in small pieces, making it more efficient for encrypting files of varying sizes.
  + The result is the encrypted data (ciphertext), which is then saved to a new file with the .enc extension, and the IV is prepended to the ciphertext for later decryption.
* **Why It's Important**:
  + AES is widely regarded as one of the most secure encryption algorithms available, and it’s important to use a secure method for both generating the key and ensuring the encryption process is robust.
  + The random IV ensures that the same file will result in different ciphertext each time it’s encrypted, even with the same password/key.

**5. encrypt\_blowfish(file\_path, key)**

* **Purpose**:
  + This function works similarly to the AES encryption function but uses the **Blowfish** algorithm, which is another symmetric encryption algorithm.
* **How It Works**:
  + Just like in the AES encryption function, it reads the file, pads the data, generates a random IV, and encrypts the data using the Blowfish algorithm in **CFB mode**.
  + The key length for Blowfish can vary, but in your case, it's using the provided key.
* **Why It's Important**:
  + Blowfish is still a valid encryption algorithm, though it’s less commonly used today due to some vulnerabilities in older implementations. However, it’s still a fast and effective option in many use cases.

**6. encrypt\_3des(file\_path, key)**

* **Purpose**:
  + This function handles Triple DES (3DES) encryption, another strong symmetric encryption algorithm, which applies the DES algorithm three times to each block of data.
* **How It Works**:
  + Like the AES and Blowfish encryption functions, it reads the file, pads the data, generates a random IV, and encrypts the data using the **3DES algorithm** in **CFB mode**.
* **Why It's Important**:
  + 3DES is older and slower compared to AES, but it can still be effective in environments where AES isn't available. It’s good to offer a variety of encryption options to users, and 3DES remains a reliable fallback.

**7. handle\_encrypt(algorithm)**

* **Purpose**:
  + This is the function that ties everything together when the user clicks the "Encrypt" button. It validates the input and calls the appropriate encryption function based on the user’s choice (AES, Blowfish, or 3DES).
* **How It Works**:
  + The function first checks that the password is exactly 16 characters long (this is a requirement for AES, Blowfish, and 3DES).
  + Then, it checks if the selected file exists. If the file exists, it uses the appropriate algorithm function (like encrypt\_aes, encrypt\_blowfish, or encrypt\_3des) to perform the encryption.
  + After encryption, the function displays a success message with the path to the encrypted file.

**8. decrypt\_file(file\_path, key, algorithm)**

* **Purpose**:
  + This function is responsible for decrypting an encrypted file and returning the original data.
* **How It Works**:
  + The function first reads the encrypted file, extracting the IV (the first 16 bytes for AES or 8 bytes for other algorithms).
  + It then selects the appropriate decryption algorithm (AES, Blowfish, or 3DES) and decrypts the data in **CFB mode**.
  + The decrypted data is unpadded, and the original file content is written to a new file with a \_decrypted suffix.
* **Why It's Important**:
  + This function ensures that encrypted files can be restored to their original state, and it handles different algorithms properly by adjusting for the IV size and decryption method.

**9. handle\_decrypt()**

* **Purpose**:
  + This function is triggered when the user clicks the "Decrypt" button. It validates the user’s input and calls the decrypt\_file function to perform the decryption process.
* **How It Works**:
  + Like the encryption handler, this function checks if the password is exactly 16 characters and if the file exists.
  + It then calls decrypt\_file with the selected algorithm and performs the decryption, displaying a success or error message depending on the outcome.

**10. set\_algorithm(algorithm)**

* **Purpose**:
  + This function updates the selected\_algorithm variable, which tracks the algorithm chosen by the user in the GUI.
* **How It Works**:
  + It takes the chosen algorithm (AES, Blowfish, or 3DES) and stores it in the selected\_algorithm list.

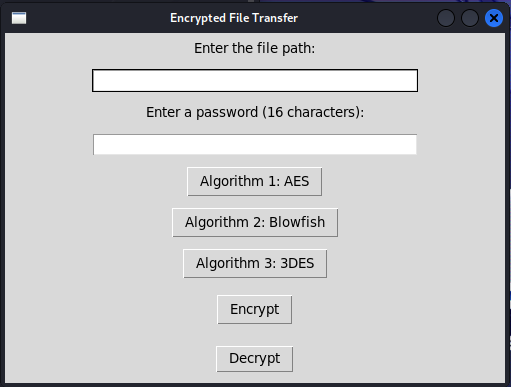
**11. GUI Setup**

* **Purpose**:
  + This is where the interface for interacting with the encryption/decryption tool is created.
* **How It Works**:
  + The Tk() object creates the main window, and labels, text fields, and buttons are added to allow the user to input the file path, password, and choose the encryption algorithm.
  + Buttons are created to trigger the encryption and decryption processes when clicked.
  + The mainloop() keeps the window open and listens for user actions.

**Summary:**

This tool allows users to encrypt and decrypt files using three strong encryption algorithms: AES, Blowfish, and 3DES. It uses a graphical user interface (GUI) to make it easy to use and integrates padding, IV handling, and error checking to ensure that the encryption process is secure and smooth. The overall design of the tool makes it flexible, with the option to choose between different encryption algorithms and interact with the program in a user-friendly way.

Output:-



Output explanation:-

Thanks for the clarification! Here's how the output of your code would look from a user interaction standpoint in a detailed, humanized way:

**GUI Layout Overview**

1. **File Path Input Field**:
   * The first input space at the top is where the user will enter the path of the file they want to encrypt or decrypt.
   * It allows the user to manually type in the path or select the file by browsing (though browsing isn't directly enabled in this version, it could be added if needed).
2. **Password Input Field**:
   * The next field is for entering a **16-character password** that will be used for encryption and decryption.
   * This password is crucial because it's the key used in the encryption algorithms. If the password is not exactly 16 characters long, an error message will be shown.
3. **Encryption Algorithm Buttons**:
   * Below the password input, there are **three buttons**:
     + **Algorithm 1: AES**
     + **Algorithm 2: Blowfish**
     + **Algorithm 3: 3DES**
   * These buttons allow the user to select the encryption algorithm they wish to use. Clicking one of them will set the chosen algorithm (AES, Blowfish, or 3DES) as the active option for the encryption or decryption process.
4. **Encrypt and Decrypt Buttons**:
   * At the bottom of the interface, there are **two action buttons**:
     + **Encrypt**: Clicking this button will trigger the encryption process, using the selected algorithm and the password provided. The tool will read the file, encrypt it, and save the encrypted version with the .enc extension.
     + **Decrypt**: This button triggers the decryption process. It uses the selected algorithm to decrypt the encrypted file (with the .enc extension) and restores the original content to a new file with the \_decrypted suffix.
5. **Error and Success Messages**:
   * If the user enters a password that isn't exactly 16 characters long, or if the file path doesn't exist, the program will show an **error message** explaining the issue (like "Password must be exactly 16 characters long" or "File not found").
   * If the encryption or decryption is successful, the user will see a **success message** with the name of the newly created file (either encrypted or decrypted).

**Flow of Interaction**

1. **Select the file**: The user will first enter the path of the file they want to encrypt or decrypt in the **"Enter the file path:"** box.
2. **Enter the password**: The user then enters a **16-character password** in the **"Enter a password (16 characters):"** field. This password is used as the key for encryption or decryption.
3. **Choose the encryption algorithm**:
   * After entering the file path and password, the user can choose one of the encryption algorithms by clicking one of the three buttons:
     + AES (Algorithm 1)
     + Blowfish (Algorithm 2)
     + 3DES (Algorithm 3)
   * The algorithm they choose will determine how the file gets encrypted or decrypted.
4. **Encryption**:
   * After choosing the algorithm, the user clicks the **"Encrypt"** button to start encrypting the file. If the file exists and the password is correct, the tool will pad the file, encrypt it using the selected algorithm, and save it as a new file with the .enc extension.
5. **Decryption**:
   * For decryption, the user clicks the **"Decrypt"** button, provided they have already selected a file that was previously encrypted. The tool will then decrypt the .enc file using the password and algorithm they selected, saving the decrypted file with a \_decrypted suffix.

**Example Scenario in this project (files included in the directory)**

* **Step 1**: The user enters a file path l "Enter the file path:" box.( /home/ashish/Desktop/prj\_ash/prj\_2/test\_doc)
* **Step 2**: The user enters a 16-character password, like testingdoc\_12345 in the password field.
* **Step 3**: The user clicks the **"Algorithm 1: AES"** button to select AES encryption.
* **Step 4**: The user clicks the **"Encrypt"** button. If everything goes correctly, the file document.txt is encrypted using AES and saved as **test\_doc.enc**.
* **Step 5**: Later, the user decides to decrypt the file. They select the path to the encrypted file (**test\_doc.enc**) and click **"Decrypt"**. The tool will decrypt it and save the original content as **test\_doc\_decrypted.**

**UI Elements**

* **Labels**: The labels are used to tell the user what to do at each step (like "Enter the file path:", "Enter a password", etc.).
* **Buttons**: These let the user choose between encryption algorithms, and to either encrypt or decrypt the file.
* **Message Boxes**: Whenever the user encounters an issue, a message box appears telling them what went wrong (e.g., invalid password length, missing file) or confirming a successful operation (e.g., "File encrypted successfully").

In essence, this GUI provides a simple and effective way for the user to interact with your encryption tool. It guides them step-by-step through the process of choosing a file, selecting an encryption method, and either encrypting or decrypting the file with ease.

**Project Summary**

This project revolves around creating a simple but effective encryption and decryption tool with a clean and easy-to-use graphical interface. The main goal is to give users an intuitive way to protect their files. Here’s how it works:

1. **Choosing the File**: The user simply enters the path to the file they want to protect (or unlock). No complicated setups, just direct file access.
2. **Setting a Password**: To encrypt or decrypt, a password is needed. This password, exactly 16 characters long, serves as the key for the encryption process.
3. **Picking an Encryption Method**: There are three encryption options to choose from—AES, Blowfish, and 3DES. Each offers a different level of security and is suited to various use cases.
4. **Encrypting or Decrypting**: Once the file, password, and algorithm are chosen, the user clicks a button to either encrypt or decrypt the file. The encrypted file gets a .enc extension, and the decrypted file is saved with a \_decrypted tag.

The tool is designed to be as user-friendly as possible, with helpful prompts to guide users through any errors, like an incorrect password or a file that’s missing. When everything goes right, the tool also shows a message confirming the success of the encryption or decryption process.

**Looking Ahead: Future Enhancements**

We're not stopping here! In the future, we plan to add a **file-sharing feature**, making it even easier to send encrypted files safely. This feature will allow users to securely share their files with others, ensuring the data stays protected during the transfer process. Whether it's integrating cloud storage or exploring secure peer-to-peer options, we’ll ensure that your files are kept safe every step of the way. Stay tuned for more updates as we continue to build on this project!