Kingdom of Saudi Arabia
Ministry of Higher Education
Qassim University
College of computer

المملكة العربية السعودية
وزارة التعليم العالي
جامعة القصيم
كلية الحاسب

Project Report

Coding and Cryptography Course

MATH319 | 471

Supervisor: Dr. Al Anoud Al-Salman

Submission Date: 6 December 2025

Group17

Students & IDs:

Lubna Aleid	451203449
Dona Alfarraj	451203450
Hams Alshatawa	451203403
Adhwaa Alhudaithy	451203518
Sandra Alomar	451203405

Table of Contents

Secure file storage:

The program provides a secure and user-friendly method for storing files by relying on the AES encryption algorithm, which is one of the strongest and most widely used symmetric encryption methods in data protection worldwide. The program offers users an integrated environment that includes login authentication, file uploading, file encryption, and file retrieval when needed.

The Program Approach:

First, we study and understand the main function and the main purpose of the program, which is to encrypt/decrypt files and protect the user's data. Also, the program must be easy to use. The main goal is to protect the files and user data in a safe and easy way.

- **Authentication:**

  We decided to start with user authentication because it's necessary to identify the identity of the user to protect the data.

- **Registration and Login:**
  - Registration: allows the user to enter the username and password. Then, the program ensures both fields are not empty and the username is not taken before, the program stores the password safely.
  - Login: request from the user to enter the username and password, check if the username is in the USER_FILE, and compare the entered password with the password stored.

- **User Interface:**

  We use Tkinter because it's simple and appropriate for simple programs. We include 1. Login. 2. Register. 3. Dashboard.

- **Encryption and Decryption:**

  We used AES algorithm because it's strong and fast. We use AES-CBC mode.

Project Function:

- **User Interface:**

  User interface (UI) is how users interact with computer systems. It contains username and password, and if the user wants to login or register, it gives the user the choice to encrypt or decrypt a file.
  The UI includes:
  - Login Window: allows the user to enter a username and password, login to their account or register.
  - Register Window: allows new users to create an account.
  - Dashboard Window: allows the logged-in user to encrypt or decrypt files.

- **User Authentication:**
  - The user clicks on registration: when the user creates an account, they should enter the username and password. The program checks that both fields are not empty, and the username does not already exist in the file USER_FILE. Then the password is converted into a hash using SHA-256.
  - The user login: the user should first enter their username and password. The program makes sure that the username already exists in the USER_FILE; if not the user can't enter, and the program sends an error message. Then the program compares the password entered and the password in the file. If both are correct, the user can enter; if they are not, an error message appears, and the program prevents access.
  - Encrypt or decrypt the files only if the user successfully logged in.
  - In the encryption file, the program prompts the user to enter the password for the encryption. In decryption, the programs ask the user to enter the same password as in the encryption process.

- **File Upload:**

  The user selects a file from their device, and the program then prompts them to enter a password for encryption. Then, the program starts to encrypt the file. After the program encrypts the file, the program sends a message that the program finished with the encrypted file.
  If the user doesn’t select a file, the program sends a message that says no file selected for encryption.

- **AES Encryption:**

  The program encrypts the file using the AES algorithm in CBC (Cipher Block Chaining) mode. This provides secure encryption by linking each block of data to the previous one.

- **Retrieval and Decryption:**

  If the user chooses to encrypt a file. The program prompts the user to enter the password that is used for encryption. If the password is correct, the program starts to decrypt the file. Then, the user has the original file with a message that says the program decrypted the file successfully.

Program code functions:

1- Hams Alshatawa.

- **Dashboard window:**

```
# DASHBOARD WINDOW
def dashboard_window(username, parent_root):

    dash = tkinter.Toplevel(parent_root)
    dash.title(f"Secure File Storage - {username}")
    dash.geometry("350x250")
    dash.resizable(False, False)
    dash.configure(bg="#FFF8EF")

    tkinter.Label(dash, text=f"Welcome, {username}", font=("Arial", 14, "bold"), bg="#FFF8EF").pack(pady=15)

    button_frame = tkinter.Frame(dash, bg="#FFF8EF")
    button_frame.pack(pady=20)

    tkinter.Button(button_frame, text="Encrypt File 🔒", width=20,
                   command=encrypt_file, bg="#F4165C", fg="white",
                   font=("Arial", 12)).grid(row=0, column=0, padx=10, pady=10)

    tkinter.Button(button_frame, text="Decrypt File 🔒", width=20,
                   command=decrypt_file, bg="#F4165C", fg="white",
                   font=("Arial", 12)).grid(row=1, column=0, padx=10, pady=10)

    def logout_action():
        parent_root.deiconify()

    tkinter.Button(dash, text="Logout", width=12, command=logout_action,
                   bg="#96d4af", fg="white", font=("Arial", 11)).pack(pady=5)

    def on_close():
        dash.destroy()
        parent_root.deiconify()

    dash.protocol("WM_DELETE_WINDOW", on_close)
```

This function creates a dashboard window that appears after login using Toplevel to open a separate window with a fixed size It shows a welcome label with the username then builds a frame that holds two main buttons one for encrypting files and one for decrypting files each connected to their functions and styled with colors, fonts and spacing Below that it adds a logout button that brings back the parent window by calling deiconify() The code also defines a function that if the user closes the dashboard window it gets destroyed and the parent login window becomes visible again Overall it's a simple Tkinter interface that lets the user encrypt, decrypt, or log out .

2-Lubna Aleid

- **Login window**:

```
# LOGIN WINDOW
def login_window():
    def login_action():
        username = entry_user.get()
        password = entry_pass.get()
        if authentication(username, password):
            messagebox.showinfo("Login Successful", f"Welcome, {username}!")
            login.withdraw()
            dashboard_window(username, login)
        else:
            messagebox.showwarning("Login Failed", "Incorrect username or password.")
            return open_register()

    def open_register():
        register_window(login)

    login = tkinter.Tk()
    login.title("Secure File Storage – Login")
    login.geometry("350x250")
    login.configure(bg="#FFF8EF")

    tkinter.Label(login, text="Login", font=("Arial", 16, "bold"), bg="#FFF8EF").pack(pady=10)

    form_frame = tkinter.Frame(login, bg="#FFF8EF")
    form_frame.pack(pady=10)

    tkinter.Label(form_frame, text="Username:", font=("Arial", 12), bg="#FFF8EF").grid(row=0, column=0,
    padx=5, pady=5, sticky="e")
    entry_user = tkinter.Entry(form_frame, font=("Arial", 12))
    entry_user.grid(row=0, column=1, padx=5, pady=5)

    tkinter.Label(form_frame, text="Password:", font=("Arial", 12), bg="#FFF8EF").grid(row=1, column=0,
    padx=5, pady=5, sticky="e")
    entry_pass = tkinter.Entry(form_frame, show="*", font=("Arial", 12))
    entry_pass.grid(row=1, column=1, padx=5, pady=5)

    button_frame = tkinter.Frame(login, bg="#FFF8EF")
    button_frame.pack(pady=15)

    tkinter.Button(button_frame, text="Login", width=12, command=login_action,
                   bg="#F4165C", fg="white", font=("Arial", 12)).grid(row=0, column=0, padx=10)

    tkinter.Button(button_frame, text="Register", width=12, command=open_register,
                   bg="#F4165C", fg="white", font=("Arial", 12)).grid(row=0, column=1, padx=10)

    def on_root_close():
        if messagebox.askokcancel("Quit", "Do you want to quit?"):
            login.destroy()

    login.protocol("WM_DELETE_WINDOW", on_root_close)
    login.mainloop()
```

This function is responsible for displaying the login window. We defined the login interface using Tk() from the tkinter library, choosing the title, size, and background color. The interface has two labels for the username and password, which are assigned to the variables entry_user , entry_pass. It also has two buttons. The first button, titled login, calls the login_action() function, which in turn calls the authentication function and sends the variables after redefining them. If authentication is successful, the window is hidden, and the dashboard window function is called, along with a successful login message via messagebox.showinfo(). If authentication is unsuccessful, an incorrect username or password message is displayed via messagebox.showerror(), and then the open_resister() function is called.

- **Register window:**

```
# REGISTRATION WINDOW
def register_window(parent_root):
    def register_action():
        username = entry_user.get()
        password = entry_pass.get()
        registration(username, password)
        if username and password:
            register_win.destroy()
            parent_root.withdraw()
          #____________________________

            dashboard_window(username, parent_root)

    register_win = tkinter.Toplevel()
    register_win.title("Secure File Storage - Register")
    register_win.geometry("350x250")
    register_win.resizable(False, False)
    register_win.configure(bg="#FFF8EF")

    tkinter.Label(register_win, text="Register", font=("Arial", 16, "bold"), bg="#FFF8EF").pack(pady=10)

    form_frame = tkinter.Frame(register_win, bg="#FFF8EF")
    form_frame.pack(pady=10)

    tkinter.Label(form_frame, text="Username:", font=("Arial", 12), bg="#FFF8EF").grid(row=0,
    column=0, padx=5, pady=5, sticky="e")
    entry_user = tkinter.Entry(form_frame, font=("Arial", 12))
    entry_user.grid(row=0, column=1, padx=5, pady=5)

    tkinter.Label(form_frame, text="Password:", font=("Arial", 12), bg="#FFF8EF").grid(row=1,
    column=0, padx=5, pady=5, sticky="e")
    entry_pass = tkinter.Entry(form_frame, show="*", font=("Arial", 12))
    entry_pass.grid(row=1, column=1, padx=5, pady=5)

    tkinter.Button(register_win, text="Register", width=15, command=register_action,
                   bg="#F4165C", fg="white", font=("Arial", 12)).pack(pady=15)
```

If the user clicks the register button, the open_register() function will be called immediately. This internally defined function defines a new user for whom we have already configured the interface. It also take Variables from the user, entry_user and entry_pass, and has a single button named register. This button calls the register_action() function, which is also an internal function. registerion function provides it with the username and password. it crashes the register interface, hides the login window, and finally calls the dashboard function, providing it with the welcome username and a login window to keep the windows sequential.

3- Adhwaa Alhudaithy

- **Registration**:

```
#  USER REGISTRATION
def registration(username, password):
    if not username or not password:
        messagebox.showerror("Error", "You must complete all required information!")
        return
    hashed_pass = hashlib.sha256(password.encode()).hexdigest()
    if os.path.exists(USERS_FILE):
        with open(USERS_FILE, "r") as user_file:
            for line in user_file: #edit here
                stored_username, stored_pass = line.strip().split(",")

                if username == stored_username and hashed_pass == stored_pass:
                    messagebox.showerror("Error", "Username already exists!")
                    return

    hashed_pass = hashlib.sha256(password.encode()).hexdigest()
    with open(USERS_FILE, "a") as user:
        user.write(username + "," + hashed_pass + "\n")
    messagebox.showinfo("Success", "Registration Successful!")

#  USER AUTHENTICATION
```

The function registration () takes two inputs: one foe the username, and one for the password. Registers a username and password, encrypted using sha256, which makes it strong. First, the function check if the username and password field are filled, if not, it sends a message that says, “you must complete all required information!” Then the password is encrypted using sha256, and this represents the password in hexadecimal format. Then, the function opened the file USERS_FILE and checked if the new username was in the file or not. If the username was in the file, the program sends a message that says, “Error, username already exists!” and return. If not the function complete and stored the new username and password in the USERS_FILE, then send a message “Success, Registration Successful”

- **Authentication**:

```
#  USER AUTHENTICATION
def authentication(username, password):
    if not os.path.exists(USERS_FILE):
        messagebox.showerror("Error", "No users registered yet!")
        return False

    hashed_pass = hashlib.sha256(password.encode()).hexdigest()
    with open(USERS_FILE, "r") as user:
        for line in user:
            stored_username, stored_hashed_pass = line.strip().split(",")
            if username == stored_username and hashed_pass == stored_hashed_pass:
                return True
    return False
```

The Authentication function return either true or false and takes two inputs one for the username and the other for the password. This function checks if the username and password are registered or not. First, the function checks if the file USERS_FILE exists or not, if it's not, the program display a message "Error, no users registered yet!" Then return false, the function encrypted the entered password using sha256. Then the function opened the file USERS_FILE and started to compare if the username entered same as stored username, and if the encrypted password was the same as the stored encrypted password. If it is same, return true and if it is false, return false.

4- Dona Alfarraj

- **Files encrypt & Fles decrypt:**

```
# File Upload

# ENCRYPT FILE BUTTUON
def encrypt_file():
    input_File = filedialog.askopenfilename(title="Select File to Encrypt")
    if input_File:
        password = simpledialog.askstring("Password", "Enter password for encryption:", show='*')
        if password:
            output_File = input_File + ".enc"
            encryption(input_File, output_File, password)
            messagebox.showinfo("Success", f"File encrypted successfully!\n{output_File}")
        else:
            messagebox.showerror("Error", "Password is required for encryption!")
    else:
        messagebox.showerror("Error", "No file selected for encryption!")

# DECRYPT FILE
def decrypt_file():
    input_File = filedialog.askopenfilename(title="Select File to Decrypt")
    if input_File:
        password = simpledialog.askstring("Password", "Enter password for decryption:", show='*')
        if password:
            if input_File.endswith(".enc"):
                output_File = input_File[:-4] + ".dec"
            else:
                output_File = input_File + ".dec"
            decryption(input_File, output_File, password)
        else:
            messagebox.showerror("Error", "Password is required for decryption!")
    else:
        messagebox.showerror("Error", "No file selected for decryption!")
```

Files encrypt:

When this function is called, the program shows a message saying “Select file to encrypt” and opens a window for the user to choose a file.If the user picks a file, another message appears saying “Password” and “Enter password for encryption:”, asking the user to type a password. The text is hidden while typing , after the user enters the password, the code creates the output filename by adding “.enc” to the original name, once the encryption is done successfully, the program shows a message saying “File encrypted successfully!” and “Success”.If the user does not type a password, it shows an error message: “Error” and “Password is required for encryption!” ,and if the user doesn’t select a file, it shows another error: “Error” and “No file selected for encryption!”.

Fles decrypt:

When this function is called, the program shows a message saying “Select file to decrypt”, after the user chooses a file, another message appears saying “Password” and “Enter password for decryption:”, and the password is hidden while typing , after the user enters the password, the code checks if the filename ends with “.enc”.If it does, the program removes “.enc” and adds “.dec” as the new extension.If the file does not end with “.enc”, the program simply adds “.dec” to the filename.If the user does not enter a password, an error message appears saying “Error” and “Password is required for decryption!”, and if the user does not select a file, another error message appears saying “Error” and “No file selected for decryption!”.

5- Sandra Alomar

- **Encryption:**

```
# FILE ENCRYPTION
def encryption(input_file, output_file, password):
    salt = get_random_bytes(16)
    key = PBKDF2(password.encode('utf-8'), salt, dkLen=32, count=100000)
    iv = os.urandom(16)
    cipher = AES.new(key, AES.MODE_CBC, iv)

    try:
        with open(input_file, 'rb') as inputFile, open(output_file, 'wb') as outputFile:
            outputFile.write(salt)
            outputFile.write(iv)

            while True:
                chunk = inputFile.read(4096)
                if len(chunk) == 0:
                    break
                if len(chunk) % 16 != 0:
                    chunk = pad(chunk, 16)
                cipherText = cipher.encrypt(chunk)
                outputFile.write(cipherText)
    except FileNotFoundError:
        messagebox.showerror("Error", "File not found!")
```

This function takes three parameters: input_file, output_file, and password. The main purpose is to convert an open-access file into an encrypted format that any interrupters can't read without the correct password.

Salt generation: we start by generating a 16-byte random salt using the get random bytes function to ensure that even if same password is used for different files the encryption key will change with each one so attackers won't recognise a pattern

Initialisation vector: we also generate a 16-byte random IV using the os.random() function. This vector is important to ensure that identical plain-texts produce different cipher-texts to add an extra layer of safety

Key Derivation: we use the PBKDF2 function to derive a strong 32-byte AES key using the user's password and the salt. This function uses thousands of iterations which slows down any brute-force attack making truly difficult for an attacker to guess it.

AES creation: we now create the cipher object in CBC mode, which encrypts the data in chunks of 16-bytes blocks. Then the CBC mode links the current block to the previous one to prevent patterns from appearing in the encrypted file.

File encryption loop: the function will now read the input file in chunks of 4096 bytes, that prevents memory overflow when handling large files. Each chunk will be checked for:

-its size, if not a multiple of 16 it's padded using the pad() function to be compatible with AES block size.

-if true, it'll be encrypted using cipher.encrypt() function and written into the output file and before adding it the salt and IV will be added into the file first.

- **Decryption:**

```
# Retrieval and Decryption

# FILE DECRYPTION  (FIXED)
def decryption(input_file, output_file, password):
    try:
        with open(input_file, 'rb') as inputFile:

            salt = inputFile.read(16)
            iv = inputFile.read(16)

            if len(salt) != 16 or len(iv) != 16:
                raise ValueError("Invalid encrypted file format!")

            key = PBKDF2(password.encode('utf-8'), salt, dkLen=32, count=100000)
            cipher = AES.new(key, AES.MODE_CBC, iv)

            total_size = os.path.getsize(input_file)
            payload_size = total_size - 32
            processed_bytes = 0

            with open(output_file, 'wb') as outputFile:
                while True:
                    chunk = inputFile.read(4096)
                    if not chunk:
                        break

                    processed_bytes += len(chunk)
                    decrypted_chunk = cipher.decrypt(chunk)

                    if processed_bytes == payload_size:
                        decrypted_chunk = unpad(decrypted_chunk, 16)

                    outputFile.write(decrypted_chunk)

        messagebox.showinfo("Success", f"File decrypted successfully!\n{output_file}")

    except ValueError as e:
        messagebox.showerror("Error", f"Decryption failed: {e}")
    except FileNotFoundError:
        messagebox.showerror("Error", "File not found!")
```

The decryption function takes the same previous parameters and works as an inverse operation to convert encrypted files back to their original form.

Reading salt and IV: the function will start by reading the first 16-bytes as the salt and the following 16-bytes as the IV. This step is important because the key derivation requires the exact same salt and IV used when encrypting.

Key derivation: the AES key is now reconstructed from the password the user provides and the retrieved salt using the PBKDF2 function to ensure that only the correct password will generate the right key.

AES cipher initialisation: We now create a new AES cipher object in CBC mode to start decrypting the cipher-text.

File decryption loop: the function now will start reading the remaining of the file chunk by chunk and each one is decrypted using cipher.decrypt() function with paying closer attention to the last one which may contain added padding during encryption, so it's processed using the unpad to remove the extra bytes and get original file size.

Error handling: the function handles two errors:

if the file format isn't correct it raises: ValueError.

If the file name entered doesn't exist it raises: FileNotFoundError.

Lastly, thees functions apply modern cryptography methods using python's (pycryptodome) library to provide fast and easy encryption & decryption methods.

Integration:

The project works by connecting the encryption and decryption functions. First, the user picks an option, and the program runs the right function.When encrypting, the function makes the key and IV, then encrypts the file and puts this information at the start of the new file.When decrypting, the function uses the same information from the encrypted file to recreate the key and unlock the file.This way, both functions work together, making it easy and safe to store and get back files.

Privacy:

- **Why Use AES?**

1- very strong encryption key: AES supports keys of 128, 192, or 256 bits.

2- complex algorithm reviewed by a large community.

3- Globally accepted and approved: The algorithm is approved by official security and scientific bodies and is used in government, banking, network, storage systems, and security protocol applications.

5- **Efficient and fast execution:** Despite its strength, AES is fast (both in programming and hardware) and works effectively even with large files.

- **How can date leakage be prevented?**

The program keeps your data safe by doing everything on your own device. Nothing goes online, so no one else can see your files or passwords.It locks your files with strong encryption, and your password is never saved directly only a secure code of it so even if someone gets access, they can’t know your password.Because everything happens on your device, the chance of your data leaking is really, really low.

Challenged Faced:

In this project, we should have a little knowledge of python and user interface. Also, we have some challenges with encrypted or decrypted files, which are:

- **Secure Authentication:**

  We should ensure that usernames and passwords are handled securely. It was a little challenging.

- **Key Derivation and Encryption Logic:**

  We must understand how to generate an encryption key from a user's password. Also, make sure the AES encryption process works correctly and doesn't have any errors.

- **User Interface:**

  Creating a good user interface in a simple way for the user to understand the program was challenging.

- **Error Handling and User Message:**

  We had to make the program user-friendly; we had to make it clear, and we had to send an error message to the user to understand the issue.

- **Testing With Different File Types and Sizes:**

  We tried to make the program know how to encrypt or decrypt with different file formats and different sizes.

Individual contribution:

Report: Adhwaa Alhudaithy & Dona Alfarraj

Code: Lubna Aleid & Sandra Alomar

Presentation: Hams Alshatawa