Project Report: Secure QR Messenger

Project Title: QR Code Secret Messages with Encryption **Language:** Python

---

1. Introduction

This project is a desktop application that hides secret messages inside QR codes. It is not just a normal QR generator; it uses **AES Encryption** to lock the message. This means even if someone scans the QR code, they cannot read the message without the correct password.

We upgraded the project from a simple text tool to a full **Graphical User Interface (GUI)** with buttons and windows. We also added security features like password checking and protection against hackers.

---

2. Libraries Used & Their Usage

We used four main external libraries to make this project work. Here is what each one does:

1. `tkinter` (Built-in)

- **What it is:** The standard Python library for creating Graphical User Interfaces (GUIs).
- **Usage in Project:** It draws the windows, buttons, tabs, and input boxes. It allows the user to click and type instead of using a black command screen.

2. `pycryptodome` (External)

- **What it is:** A powerful library for cryptography (security math).
- **Usage in Project:**
    - **AES Cipher:** It scrambles the message so it looks like random noise.
    - **SHA-256:** It turns the user's password into a secure 32-byte key.
    - **Padding:** It adds extra characters to the message to make sure it fits the encryption block size.

3. `qrcode` (External)

- **What it is:** A library dedicated to generating QR (Quick Response) codes.
- **Usage in Project:** It takes the encrypted text string and converts it into a scannable square barcode image.

4. `pillow` (PIL) (External)

- **What it is:** The Python Imaging Library. It handles opening and editing images.
- **Usage in Project:** `tkinter` cannot display images directly. We use `pillow` to load the QR code image we saved and display it inside the application window.

---

3. Key Features Explained

We implemented **6 special features** to get the full grade and bonus points.

Feature 1: Password Strength Check

- **Goal:** Stop users from using weak passwords like "1234".
- **How it works:** The system checks if the password has uppercase letters, numbers, and symbols. It shows a **Green** label for strong passwords and **Red** for weak ones.

Feature 2: Brute-Force Protection

- **Goal:** Stop hackers from guessing the password by trying thousands of times.
- **How it works:** If you enter the wrong password **3 times**, the system **locks you out for 30 seconds**. You cannot try again until the timer finishes.

Feature 3: SHA-256 Key Derivation

- **Goal:** Make the encryption mathematically secure.
- **How it works:** AES encryption needs a key of exactly 32 bytes. Since human passwords vary in length, we use **SHA-256** to convert any password into a perfect 32-byte key.

Feature 4: GUI Interface

- **Goal:** Make the app easy to use.
- **How it works:** We replaced the command line with a modern window containing three tabs:
    1. **Register:** To create a user.
    2. **Encrypt:** To make the QR code.
    3. **Decrypt:** To read the message.

Feature 5: File Encryption Support

- **Goal:** Allow users to encrypt large texts easily.
- **How it works:** Added a button called **"Import .txt File"**. You can select a text file from your computer, and the app will automatically read it and prepare it for encryption.

Feature 6: Two-Step Authentication

- **Goal:** Double the security.
- **How it works:** To decrypt a message, you need two things:
    1. A registered **Username**.
    2. The correct **Password**. The system checks your username in a database before letting you unlock the message.

---

4. How to Use the App

Step 1: Register

- Go to the "Register" tab.
- Create a username and a strong password.

Step 2: Encrypt (Hide Message)

- Go to the "Encrypt" tab.
- Type a secret message OR upload a text file.
- Enter a password and click **Generate QR**.
- Save the image.

Step 3: Decrypt (Read Message)

- Go to the "Decrypt" tab.
- Paste the text from the QR code.
- Enter your Username and Password to prove it is you.
- Click **Decrypt** to reveal the secret.

---

5. Conclusion

This project successfully combines security with ease of use. By using **Python** and **AES Encryption**, we created a tool that keeps data safe. The added features like the **GUI** and **Brute-Force Protection** make the software feel professional and ready for real-world use.

Code explaination

Part 1: Detailed Code Explanation

The code is structured using **Object-Oriented Programming (OOP)**. This means we split the code into three "Classes" (logical blueprints), each with a specific job.

1. Class: `AuthManager` (The Security Guard)

This class handles everything related to users: registration, passwords, and locking out hackers.

- **`__init__`**:
  - Creates a dictionary `self.users_db` to act as a database (storing username: password).
  - Sets up variables for **Feature 2 (Brute-Force Protection)**: `failed_attempts` counts wrong guesses, and `lockout_until` stores the time when a user is allowed to try again.
- **`check_password_strength(password)`** **(Feature 1)**:
  - Uses **Regex** (Regular Expressions) to scan the password string.
  - It checks for 5 rules: Length > 8, Uppercase, Lowercase, Number, Special Character.
  - Returns `True` only if all rules are met.
- **`login(username, password)`** **(Feature 2 & 6)**:
  - **Step 1:** Checks if the current time (`time.time()`) is *less* than the `lockout_until` time. If yes, it blocks the user.
  - **Step 2:** Checks if the username exists and if the password matches.
  - **Step 3:** If the password is wrong, it adds +1 to `failed_attempts`. If failures reach 3, it sets a 30-second lockout timer.

2. Class: `CryptoManager` (The Mathematician)

This class handles the encryption logic using the `pycryptodome` library.

- **`get_aes_key(password)`** **(Feature 3)**:
  - AES encryption requires a specific key size (32 bytes). Humans type passwords of random lengths.
  - This function uses **SHA-256** to "hash" the user's password. This turns *any* password into a perfect, secure 32-byte string.
- **`encrypt(plain_text, password)`**:

  - Generates a random **IV (Initialization Vector)**. This ensures that encrypting "Hello" twice results in different codes each time (prevents pattern analysis).
  - Uses `pad()` to add extra characters to the text so it fits the AES block size.
  - Encrypts the text and combines `IV + Encrypted_Data`, then converts it to text (Base64) so it can be put inside a QR code.
- **`decrypt(encrypted_b64, password)`**:
  - Reverses the process. It extracts the IV, regenerates the Key from the password, and uses them to unlock the message.

3. Class: `SecureQRApp` (The Interface)

This class uses `tkinter` to draw the window you see.

- **`__init__`**:
  - Sets up the main window and creates the **Tabs** (Notebook) for Register, Encrypt, and Decrypt **(Feature 4)**.
- **`init_register_ui`**:
  - Draws the registration form.
  - Uses `.bind("<KeyRelease>")` to call `update_strength_meter` every time you type a character. This makes the text turn Green/Red instantly.
- **`init_encrypt_ui`**:
  - Contains the logic for **Feature 5 (File Support)**. The `load_text_file` function opens a Windows file dialog, reads the `.txt` file, and dumps the text into the box.
  - Calls `qrcode.make()` to turn the encrypted string into an image.
- **`init_decrypt_ui`**:
  - This is where **Feature 6 (Two-Step Auth)** happens. It forces the user to fill in User/Pass fields.
  - It calls `auth.login()` first. Only if that returns `True` does it proceed to call `crypto.decrypt()`.

Team members

Yusuf Yasser wardany

Mahmoud hossam

Hazem khaled