FortiQode

A project by Dhruv Singh.

This report explains the design and working of a custom QR-like encrypted messaging system. The system allows users to embed secret text messages inside a QR-code styled 100x100 grid image. Unlike normal QR codes, the embedded data is encrypted with a symmetric key before being converted to binary and stored in the image. This ensures that the data cannot be read without the correct key.

# Encryption & Embedding Process

1. A user inputs a message (up to 192 characters).  
2. A random symmetric key is generated using the Fernet implementation of AES provided by the cryptography library.  
3. The input message is appended with a stop marker (\_pineapple) and encrypted using the key.  
4. The ciphertext is converted into a binary stream.  
5. The binary stream is embedded into a 100x100 numpy matrix (image array), skipping reserved orientation zones.  
6. Random binary noise (gibberish) is filled in unused cells, making the image look visually similar to a QR code.  
7. The orientation squares (finder patterns) are drawn in three corners for alignment, styled like real QR codes with alternating black/white borders.

Diagram: The figure below represents the embedding process:

[Message] → [Encrypt with Symmetric Key] → [Binary Data] → [Embed in QR-like Image]

# Decryption & Extraction Process

1. A user loads the QR-like encrypted image.  
2. Binary bits are extracted from the image, ignoring orientation zones.  
3. The extracted binary is reconstructed into the ciphertext string.  
4. Using the symmetric key, the ciphertext is decrypted to recover the original message.  
5. The stop marker (\_pineapple) is removed to return the clean user input.

# Possible Improvements

* The current system requires a Python script to decrypt the QR-like image. In the future, it can be improved to integrate with real QR scanning mobile apps.
* A shared secret key (distributed securely) would allow scanning apps to automatically decrypt upon scanning.
* Error correction methods (similar to QR code Reed-Solomon coding) can be added to improve reliability.
* Dynamic scaling: the image size could adapt based on the amount of data encrypted.

# Libraries Used

* matplotlib: Used for displaying and saving the QR-like image.
* numpy: Provides the matrix operations to store binary pixel values.
* random: Fills unused cells with random binary noise.
* cryptography (Fernet): Provides symmetric AES-based encryption and decryption.

# Unique Use Cases

1. Confidential Information Sharing: Users can share sensitive messages in QR-like images, only decryptable with a secret key.  
2. Offline Secure Messaging: Messages can be shared as printed QR-like codes and later scanned and decrypted offline.  
3. Multi-factor Authentication: The encrypted QR image can serve as one authentication factor, with the symmetric key as another.  
4. Secure Event Tickets: Tickets can be distributed as encrypted QR-like codes, preventing forgery without the decryption key.  
5. Educational Tool: Demonstrates cryptography, encoding, and steganography concepts to students.