

---

# **CAPSTONE PROJECT**

## **SECURE DATA HIDING IN IMAGES USING STEGANOGRAPHY**

**Presented By:**

**Student Name : PRATIK DHAGE**

**College Name & Department :**

**D Y Patil College of engineering and technology, Kolhapur**

# OUTLINE

- Problem Statement
- Technology used
- Wow factor
- End users
- Result
- Conclusion
- Git-hub Link
- Future scope

---

# PROBLEM STATEMENT

**Data security is a paramount issue in the digital era, particularly when handling sensitive information. Conventional encryption techniques can be complicated and often require additional storage for encrypted files. This project presents an innovative image-based encryption and decryption system, enabling users to securely embed secret messages within images. Access to the decrypted message is safeguarded by a password, ensuring only authorized individuals can retrieve the information.**

# TECHNOLOGY USED

## **Programming Language:**

- **Python** – Used for building encryption and decryption logic.

## **Libraries & Frameworks:**

- **OpenCV** – Handles image processing and manipulation.
- **Numpy** – Supports array-based operations for embedding messages.
- **PyQt6** – Creates a user-friendly GUI for encryption and decryption.

## **Platforms:**

- Runs on **Windows, Linux and macOS** with python installed.

# WOW FACTORS

## What Makes This Project Unique?

- **Image-based Steganography** – Unlike conventional encryption, the message is concealed within the pixel values of an image.
- **Intuitive GUI** – Users can interact through a graphical interface without relying on command-line operations.
- **Secure Access** – Decryption requires the correct password, ensuring an extra layer of protection.
- **Storage-Free Encryption** – The message remains embedded within the image, removing the need for separate encrypted files.

## END USERS

- **Cybersecurity Enthusiasts** – Dive into encryption and steganography techniques.
- **Journalists & Activists** – Share sensitive information discreetly and securely.
- **Students & Researchers** – Gain insights into cryptography and image processing.
- **Photographers & Designers** – Embed copyright details to safeguard intellectual property.
- **General Users** – Conceal private messages or sensitive data within images effortlessly.

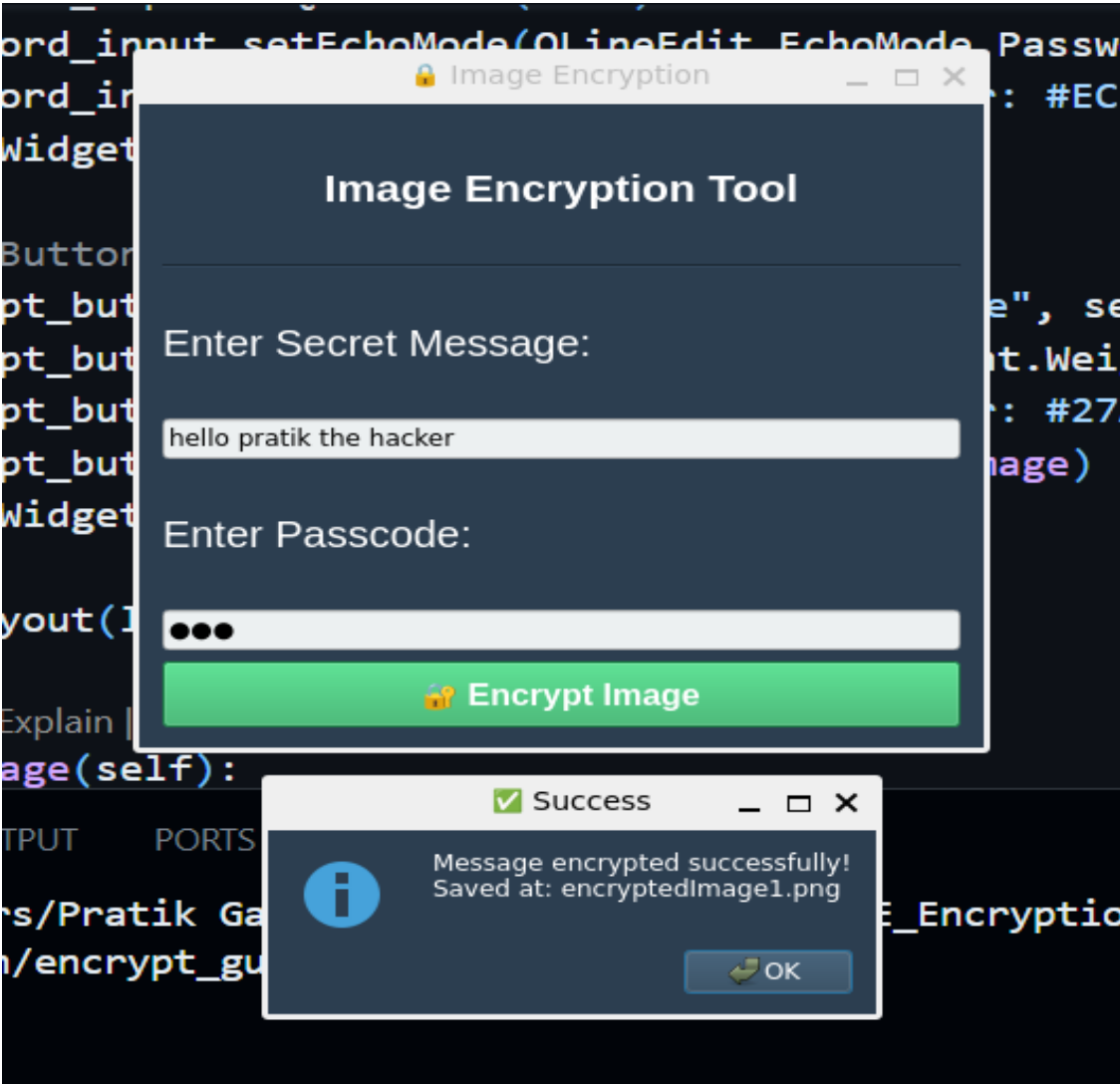
# RESULTS

The screenshot displays a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project named 'AICTE\_EN...' with subfolders 'assets' and 'v'. The 'assets' folder contains 'Decryption\_Output.png' and 'Encryption\_Output.png'. The 'v' folder contains 'bin', 'include', 'lib', and 'lib64'. The 'bin' folder contains 'pyenv.cfg', 'decrypt\_gui.py', and 'encrypt\_gui.py'. The 'lib' folder contains 'encryptedImage1.png', 'README.md', 'rose.png', 'Secure Data hiding in I...', and 'tempCodeRunnerFile.py'. The 'lib64' folder is empty. The code editor shows the following code:

```
1 import sys
2 import cv2
3 import numpy as np
4 from PyQt6.QtWidgets import (
5     QApplication, QWidget, QVBoxLayout, QPushButton,
6     QLabel, QLineEdit, QMessageBox, QFrame
7 )
8 from PyQt6.QtGui import QFont, QPalette, QColor
9 from PyQt6.QtCore import Qt
10
11 # Default input and output paths
12 image_path = r"rose.png"
13 output_path = r"encryptedImage1.png"
14
15 class EncryptGUI(QWidget):
16     def __init__(self):
17         super().__init__()
18
19         self.setWindowTitle("🔒 Image Encryption")
20         self.setGeometry(100, 100, 400, 350)
21         self.setStyleSheet("background-color: #2C3E5")
22
23         layout = QVBoxLayout()
24
25         # Title Label
26         self.title_label = QLabel("Image Encryption")
27         self.title_label.setFont(QFont("Arial", 16,
28         self.title_label.setAlignment(Qt.AlignmentF1
29
30         # Divider Line
31         divider = QFrame()
32         divider.setFrameShape(QFrame.Shape.HLine)
33         divider.setFrameShadow(QFrame.Shadow.Sunken)
34         layout.addWidget(divider)
35
36         # Secret Message Input
37         self.label = QLabel("Enter Secret Message:")
38         self.label.setFont(QFont("Arial", 15))
39         layout.addWidget(self.label)
40         self.message_input = QLineEdit(self)
41         self.message_input.setStyleSheet("background-color: #ECF0F1; color: black;")
42         layout.addWidget(self.message_input)
43
44         # Password Input
45         self.label2 = QLabel("Enter Passcode:")
46         self.label2.setFont(QFont("Arial", 15))
47         layout.addWidget(self.label2)
48         self.password_input = QLineEdit(self)
49         self.password_input.setEchoMode(QLineEdit.EchoMode.Password)
50         self.password_input.setStyleSheet("background-color: #ECF0F1; color: black;")
51         layout.addWidget(self.password_input)
52
53         # Encrypt Button
54         self.encrypt_button = QPushButton("🔒 Encrypt Image", self)
55         self.encrypt_button.setFont(QFont("Arial", 12, QFont.Weight.Bold))
56         self.encrypt_button.setStyleSheet("background-color: #27AE60; color: white;")
57         self.encrypt_button.clicked.connect(self.encrypt_image)
58         layout.addWidget(self.encrypt_button)
59
60         self.setLayout(layout)
61
62     def encrypt_image(self):
63         message = self.message_input.text()
64         password = self.password_input.text()
65         if not message or not password:
66             QMessageBox.warning(self, "⚠ Error", "Message and password cannot be e
67             return
68
69         img = cv2.imread(image_path)
70         if img is None:
71             QMessageBox.warning(self, "⚠ Error", "Image not found or cannot be rea
72             return
73
74         n, m, z = 0, 0, 0
75
76         # Store password and message length in the image
77         img[n, m, z] = np.uint8(len(password))
78         img[n + 1, m + 1, (z + 1) % 3] = np.uint8(len(message))
79         n += 2
80         m += 2
81         z = (z + 2) % 3
82
83         # Encrypt password and message into the image
84         for char in password + message:
85             img[n, m, z] = np.uint8(ord(char))
86             n += 1
87             m += 1
88             z = (z + 1) % 3
89
90         cv2.imwrite(output_path, img) # Save as PNG to avoid compression
91         QMessageBox.information(self, "✅ Success", f"Message encrypted successful
92
93 if __name__ == "__main__":
94     app = QApplication(sys.argv)
95     window = EncryptGUI()
96     window.show()
97     sys.exit(app.exec())
```

Figure 1: Encryption Code

Figure 2: Encryption Output





```

1  import sys
2  import cv2
3  import os
4  from PyQt6.QtWidgets import (
5      QApplication, QWidget, QVBoxLayout, QPushButton,
6      QLabel, QLineEdit, QMessageBox, QFrame
7  )
8  from PyQt6.QtGui import QFont
9  from PyQt6.QtCore import Qt
10
11  # Encrypted image path
12  image_path = r"encryptedImage1.png"
13
14  class DecryptGUI(QWidget):
15      def __init__(self):
16          super().__init__()
17
18          self.setWindowTitle("🔓 Image Decryption")
19          self.setGeometry(100, 100, 400, 250)
20          self.setStyleSheet("background-color: #2C3E50; color: white;")
21
22          layout = QVBoxLayout()
23
24          # Title Label
25          self.title_label = QLabel("Image Decryption Tool", self)
26          self.title_label.setFont(QFont("Arial", 18, QFont.Weight.Bold))
27          self.title_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
28          layout.addWidget(self.title_label)
29
30          # Divider Line
31          divider = QFrame()
32          divider.setFrameShape(QFrame.Shape.HLine)
33          divider.setFrameShadow(QFrame.Shadow.Sunken)
34          layout.addWidget(divider)
35
36          # Password Input
37          self.label = QLabel("Enter Passcode for Decryption:")
38          self.label.setFont(QFont("Arial", 14))
39          layout.addWidget(self.label)
40          self.password_input = QLineEdit(self)
41          self.password_input.setEchoMode(QLineEdit.EchoMode.Password)
42          self.password_input.setStyleSheet("background-color: #ECF0F1; color: black;")
43          layout.addWidget(self.password_input)
44
45          # Decrypt Button
46          self.decrypt_button = QPushButton("🔓 Decrypt Image", self)
47          self.decrypt_button.setFont(QFont("Arial", 12, QFont.Weight.Bold))
48          self.decrypt_button.setStyleSheet("background-color: #E74C3C; color: white;")
49          self.decrypt_button.clicked.connect(self.decrypt_image)
50          layout.addWidget(self.decrypt_button)
51
52          self.setLayout(layout)
53
54  # Encrypted image path
55  image_path = r"encryptedImage1.png"
56
57  class DecryptGUI(QWidget):
58      def decrypt_image(self):
59          return
60
61          n, m, z = 0, 0, 0
62
63          # Retrieve stored password length and message length
64          password_length = int(img[n, m, z])
65          message_length = int(img[n + 1, m + 1, (z + 1) % 3])
66          n += 2
67          m += 2
68          z = (z + 2) % 3
69          extracted_password = ""
70
71          # Extract stored password
72          for _ in range(password_length):
73              extracted_password += chr(int(img[n, m, z]))
74              n += 1
75              m += 1
76              z = (z + 1) % 3
77
78          # Debugging: Print extracted password
79          print(f"Extracted Password: {extracted_password}")
80
81          # Check if password is correct
82          if extracted_password != password_attempt:
83              QMessageBox.warning(self, "❌ Error", "Incorrect password! Access denied")
84              return
85
86          message = ""
87          # Extract hidden message
88          for _ in range(message_length):
89              message += chr(int(img[n, m, z]))
90              n += 1
91              m += 1
92              z = (z + 1) % 3
93
94          QMessageBox.information(self, "✅ Decryption Successful", f"Secret Message: {message}")
95
96  if __name__ == "__main__":
97      app = QApplication(sys.argv)
98      window = DecryptGUI()
99      window.show()
100      sys.exit(app.exec())

```

Figure 3: Decryption code

Figure 4: Decryption Output

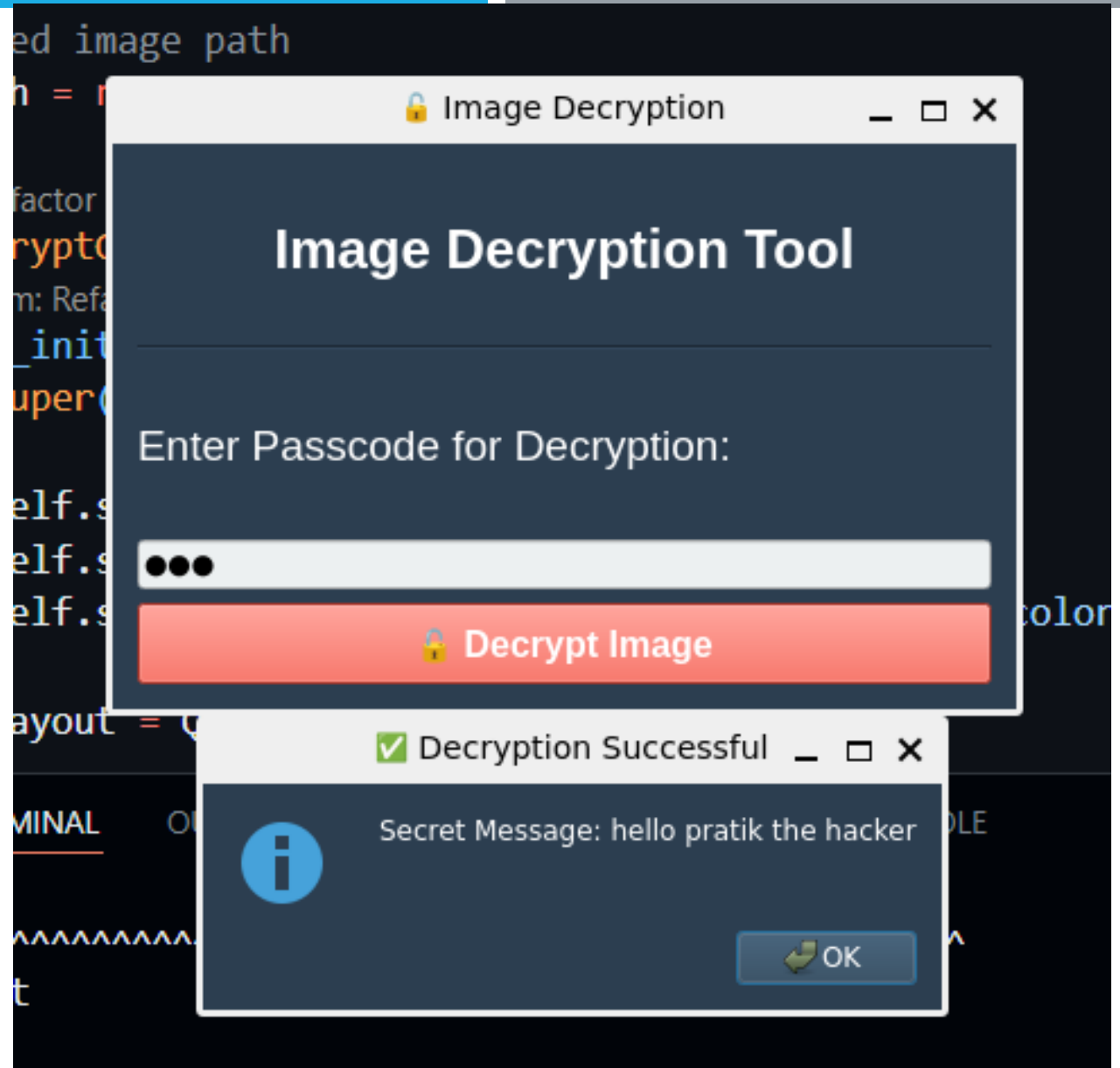


Figure 5 : Encrypted Image



## CONCLUSION

### Project Summary:

- This project offers a straightforward yet powerful method for encrypting messages within images using steganography and password protection.
- With an intuitive graphical user interface (GUI), it ensures accessibility for both technical and non-technical users.
- Leveraging OpenCV and NumPy, the system enables fast and efficient message embedding and extraction while preserving image integrity.
- Ultimately, this project showcases a practical implementation of image-based cryptography, enhancing secure communication.

## GITHUB LINK

- [https://github.com/CodeSmithPratik/IBM\\_AICTE\\_CYBERSECURITY\\_INTERNSHIP\\_PROJECT.git](https://github.com/CodeSmithPratik/IBM_AICTE_CYBERSECURITY_INTERNSHIP_PROJECT.git)

## FUTURE SCOPE(OPTIONAL)

- Stronger Security Measures** – Integrate AES or RSA encryption with steganography for a dual-layer protection system.
- Expanded Format Support** – Enable compatibility with multiple image formats, including JPG, BMP, and TIFF.
- Cross-Platform Availability** – Develop mobile and web-based versions for broader accessibility.
- AI-Powered Security** – Utilize deep learning to detect tampered images and enhance security mechanisms.

---

**THANK YOU** 🌸 🌸