STEGOSAFE: USER-FRIENDLY TOOL FOR HIDDEN IMAGE COMMUNICATION

Presented By:
NAME - Aamir Kamal
COLLEGE NAME - Vellore Institute of Technology
DEPARTMENT - B.Tech Computer Science and Engineering

edunet foundation

OUTLINE

- Problem Statement
- System Development Approach
- Algorithm & Deployment (Step by Step Procedure)
- Result
- Conclusion
- Future Scope
- References

PROBLEM STATEMENT

- In today's digital age, traditional password-based authentication methods are increasingly vulnerable to phishing, data breaches, and brute-force attacks. Two-Factor Authentication (2FA) adds a layer of security but often requires third-party apps or SMS codes, which can be intercepted or inconvenient. Many users struggle with complex authentication processes, reducing overall system usability. Current solutions fail to balance security and ease of access effectively. Therefore, there is a growing need for a robust, user-friendly image-based authentication system.

PROPOSED SOLUTION

- The proposed system uses **digital image steganography** to securely hide messages inside images without making them noticeable.
- Unlike encryption, which makes the presence of confidential data obvious, this system ensures that sensitive information remains undetectable.
- A web-based interface makes it easy to use, allowing users to upload an image, enter a message, and encode it seamlessly. The system efficiently retrieves hidden messages without affecting image quality. It improves security, accessibility, and ease of use compared to existing steganography tools.

SYSTEM APPROACH

System Requirements

Software Requirements:

- **Operating System**: Windows 10 or later / Linux
- **Frontend**: HTML, CSS, JavaScript
- **Backend**: Python (3.7 or later), Flask
- **IDE**: PyCharm / VS Code

Hardware Requirements:

- **Processor**: Intel i5 or higher
- **RAM**: Minimum 8 GB (16 GB recommended)
- **Hard Disk**: 512 GB or more

LIBRARIES USED

1. **OpenCV (cv2):**Used for image processing tasks such as reading, writing, and modifying images. Helps manipulate pixel values to embed and extract hidden messages.
2. **NumPy (numpy) :** Handles numerical operations on image data. Converts images into arrays for easy pixel manipulation .Makes bit-level operations efficient while embedding or extracting messages.
3. **PIL (Pillow - PIL.Image):**Provides easy handling of image files in various formats like PNG and JPEG. Supports image loading, saving, and modification.
4. **Flask (flask):** Creates a web-based interface for users to upload and download images. Handles requests for encoding and decoding messages.

ALGORITHM & DEPLOYMENT

- **Requirement Analysis & Research**
  - Understand the limitations of traditional authentication methods.
  - Explore the use of QR codes and steganography for secure communication.
  - Finalize tools, libraries, and frameworks for implementation.
- **Environment Setup**
  - Install Python (v3.7+), Flask, OpenCV, Pillow, qrcode, pyzbar, NumPy.
  - Set up the project folder structure for frontend and backend development.
- **QR Code Generation Module**
  - Create a function to generate a QR code from encrypted user/session data.
  - Save it as an image for display or steganographic embedding.
- **Image Embedding (Optional)**
  - If using steganography, embed the QR code or data inside another image using LSB (Least Significant Bit) or other techniques.

- **Backend Integration with Flask**
  - Create Flask routes for QR code generation and decoding.
  - Handle uploaded images and decode QR or hidden data securely.
- **QR Code Decoding & Validation**
  - Use OpenCV and pyzbar to decode scanned/uploaded QR images.
  - Match the decoded content with session/authentication data.
- **Testing and Validation**
  - Test with multiple image formats, data types, and attack scenarios.
  - Validate speed, accuracy, and data integrity.
- **Security Enhancements**
  - Add optional encryption layers before QR generation.
  - Implement secure session management and token validation.
- **Deployment**
- Host the Flask app on a local or cloud server (e.g., Heroku, Render).
- Deploy the frontend and backend as an integrated solution.
- Monitor performance and prepare documentation.

RESULT

app.py | stegosafe.py | frontend.html | result.html | styles.css | script.js

```python
from PIL import Image
def encode_message(image_path, message, output_path):
    img = Image.open(image_path)
    if img.mode != 'RGBA':
        img = img.convert('RGBA')  # Ensure the image is in RGBA format
    encoded = img.copy()
    width, height = img.size
    message += "###"
    binary_message = ''.join([format(ord(char), '08b') for char in message
    data_index = 0
    for row in range(height):
        for col in range(width):
            if data_index < len(binary_message):
                r, g, b, a = img.getpixel((col, row))
                new_r = (r & ~1) | int(binary_message[data_index])
                encoded.putpixel((col, row), (new_r, g, b, a))
                data_index += 1
            else:
                break
        if data_index >= len(binary_message):
            break
    encoded.save(output_path, "PNG")  # Save as PNG to support RGBA
    print(f"Message encoded and saved as {output_path}")
def decode_message(image_path):
    img = Image.open(image_path)
    width, height = img.size
    binary_message = ""
    for row in range(height):
        for col in range(width):
            r, g, b, a = img.getpixel((col, row))
            binary_message += str(r & 1)
    message = ''.join([chr(int(binary_message[i:i+8], 2)) for i in range(
```

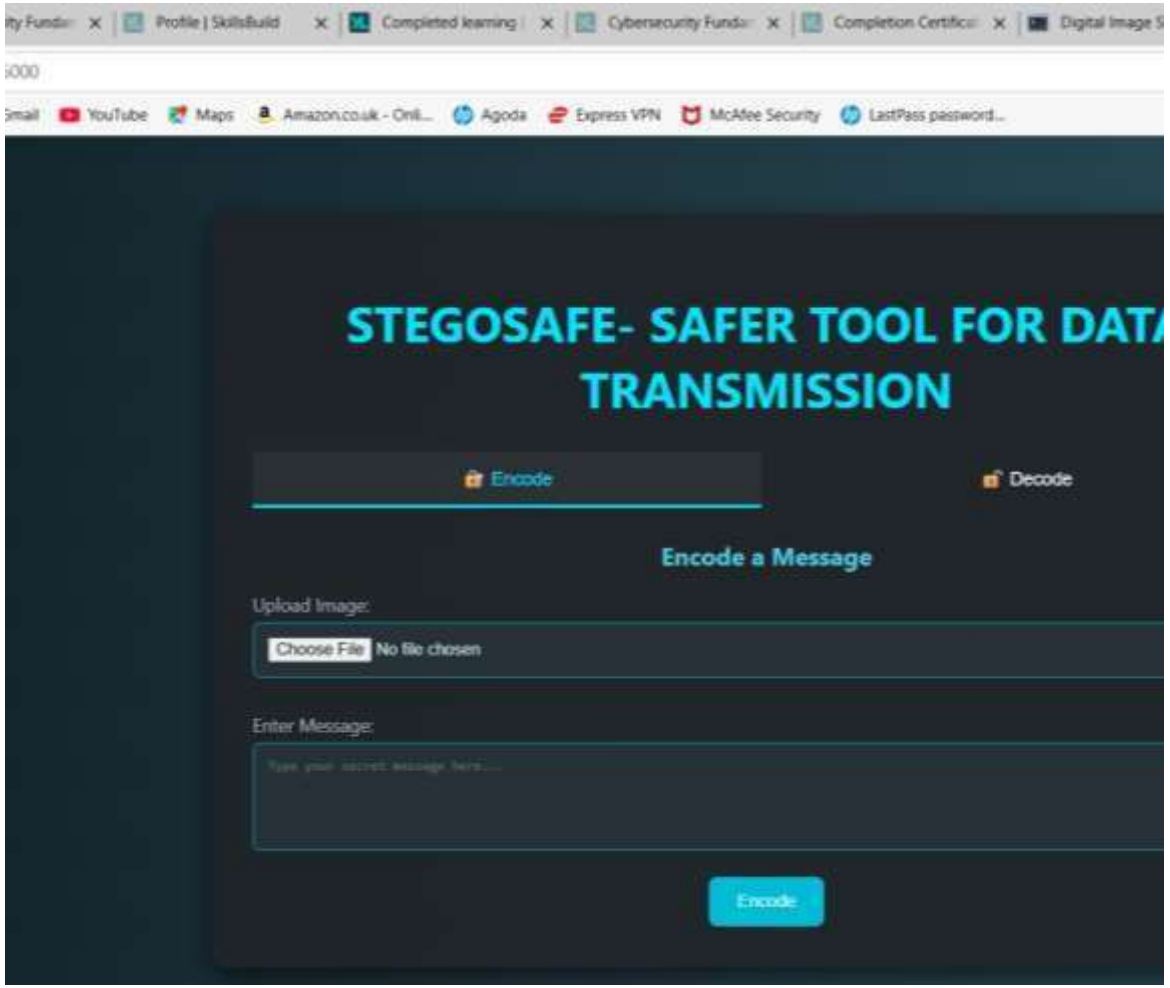
CODE

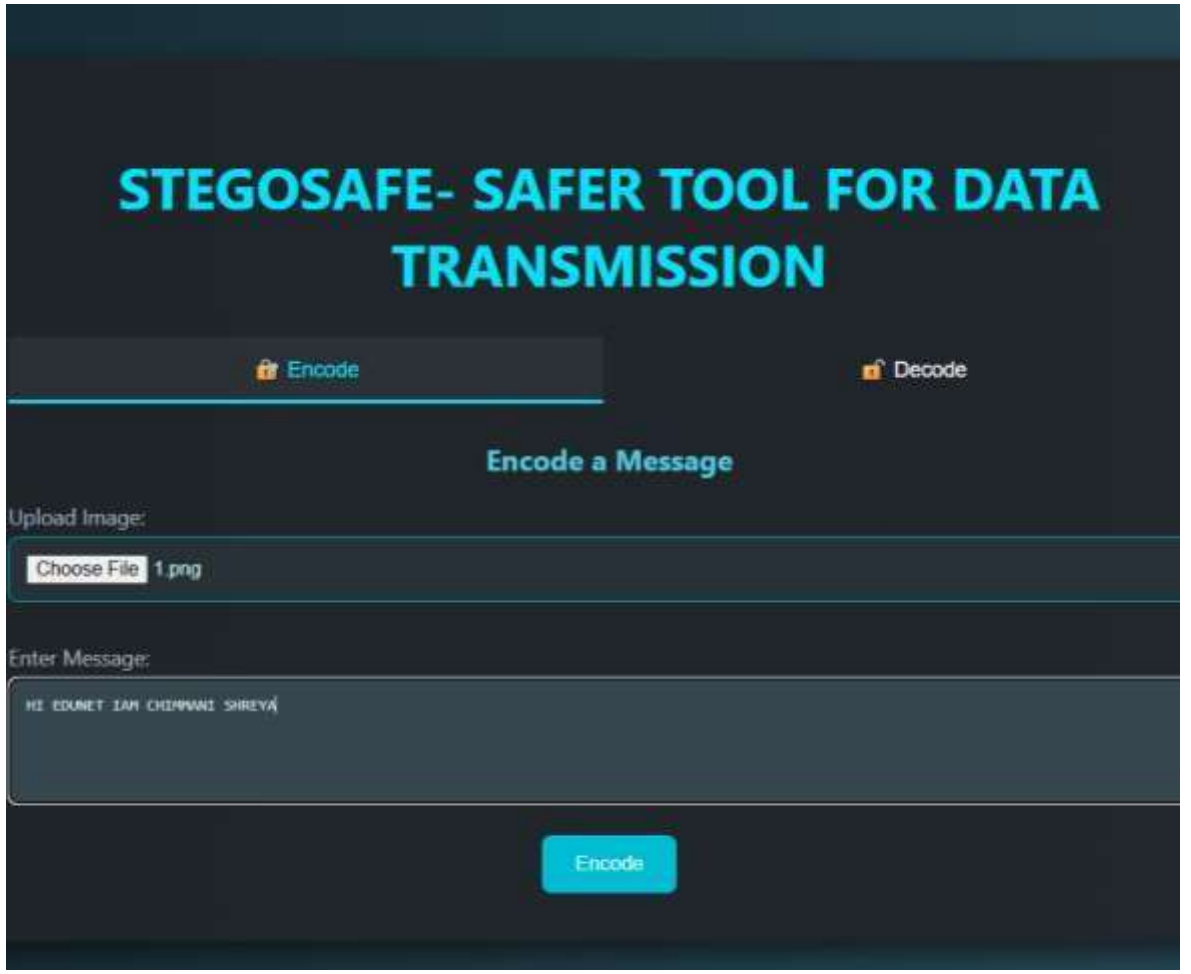
app.py | steg.py | frontend.html | result.html | styles.css | script.js

```python
from flask import Flask, request, render_template, send_file
import os
from steg import encode_message, decode_message
app = Flask(__name__)
UPLOAD_FOLDER = "uploads"
OUTPUT_FOLDER = "output"
os.makedirs(UPLOAD_FOLDER, exist_ok=True)
os.makedirs(OUTPUT_FOLDER, exist_ok=True)
@app.route('/')
def home():
    return render_template('frontend.html')
@app.route('/process', methods=['POST'])
def process():
    action = request.form.get("action")
    uploaded_file = request.files.get("image")
    if not uploaded_file:
        return "No file uploaded", 400
    file_path = os.path.join(UPLOAD_FOLDER, uploaded_file.filename)
    uploaded_file.save(file_path)
    if action == "encode":
        message = request.form.get("message", "")
        if not message:
            return "No message provided for encoding", 400
        output_path = os.path.join(OUTPUT_FOLDER, "encoded_" + uploaded_file.filenam
        encode_message(file_path, message, output_path)
        return send_file(output_path, as_attachment=True)
    elif action == "decode":
        decoded_message = decode_message(file_path)
        # Pass the decoded message to the template
        return render_template('result.html', decoded_message=decoded_message)
    else:
        return "Invalid action", 400
if __name__ == '__main__':
    app.run(debug=True)
```

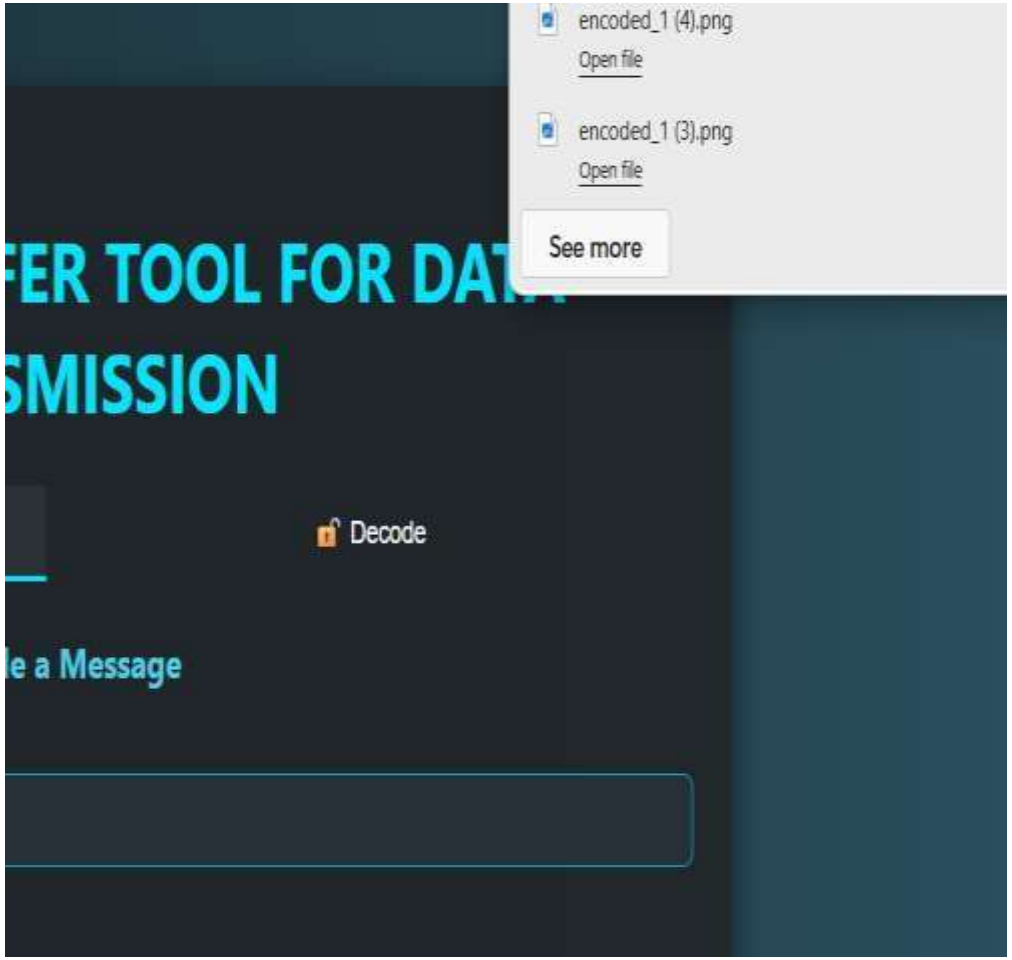
edunet foundation

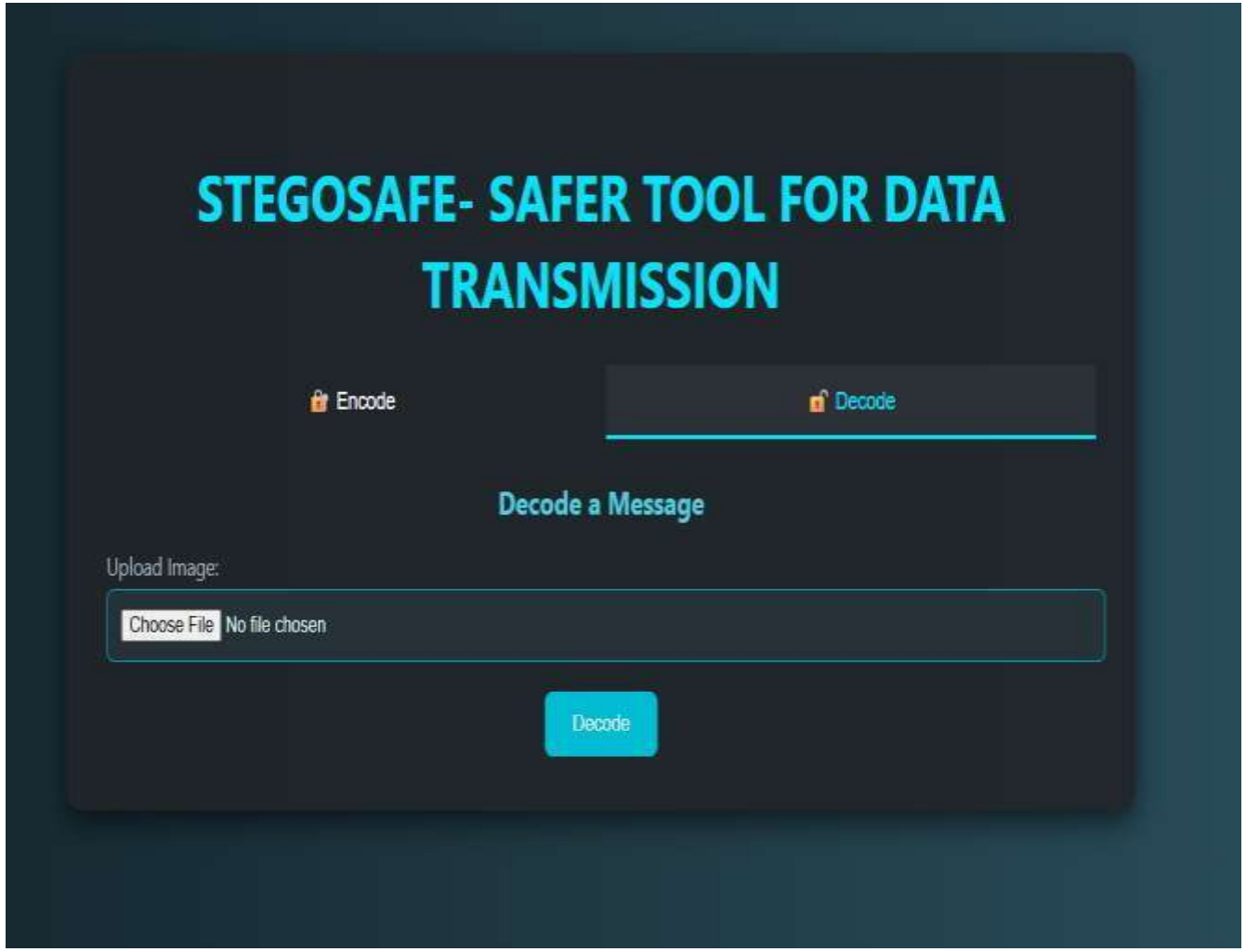
OUTPUT SCREENS



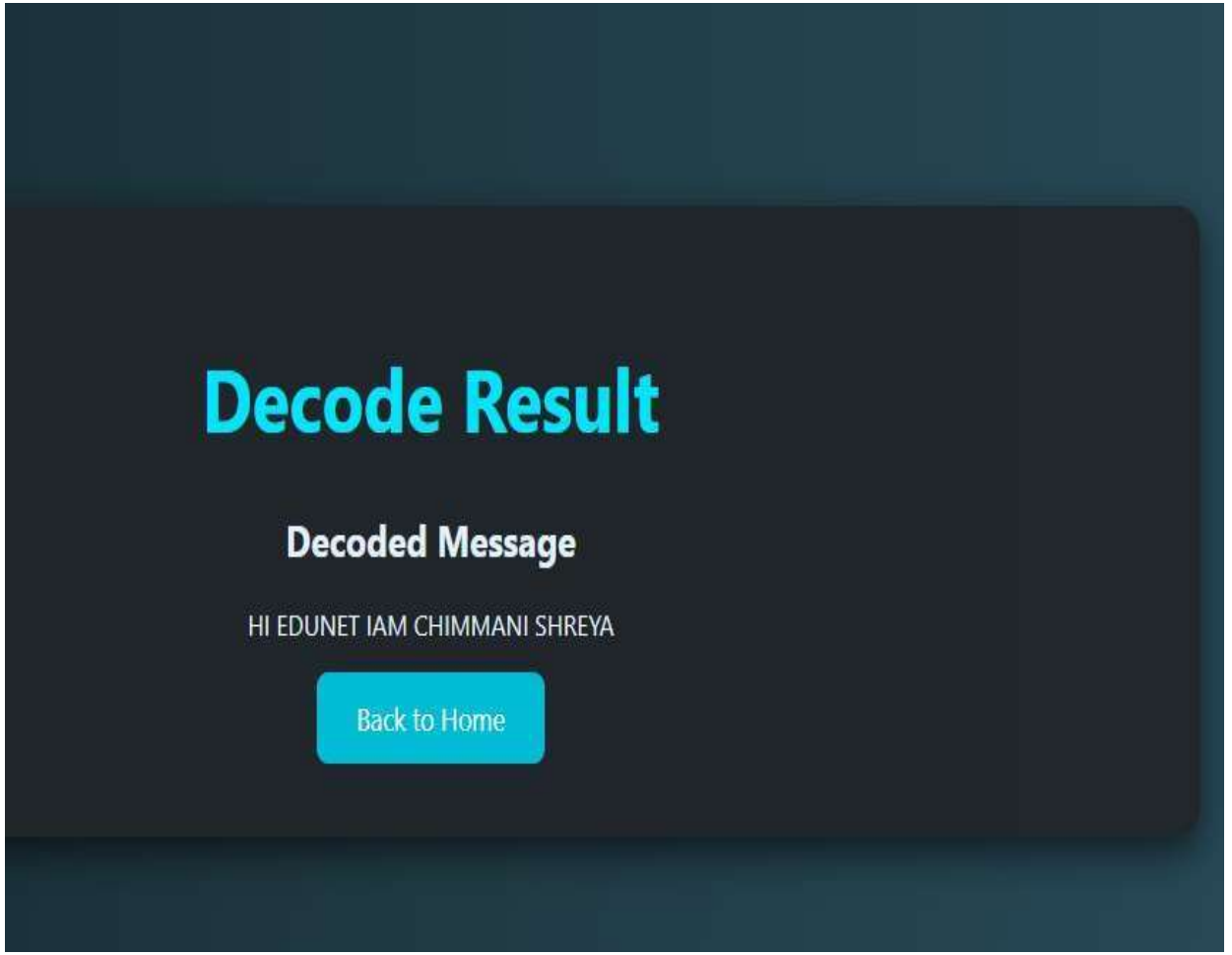


edunet foundation


encoded_1 (4).png
Open file
encoded_1 (3).png
Open file
See more
FER TOOL FOR DAT
SMISSION
Decode
le a Message


STEGOSAFE- SAFER TOOL FOR DATA TRANSMISSION
Encode
Decode
Decode a Message
Upload Image:
Choose File
No file chosen
Decode


STEGOSAFE- SAFER TOOL FOR DATA TRANSMISSION
Encode
Decode
Decode a Message
Upload Image:
Choose File
encoded_1 (4).png
encoded_1 (4).png
Decode


Decode Result
Decoded Message
HI EDUNET IAM CHIMMANI SHREYA
Back to Home

GITHUB REPOSITORY LINK

- https://github.com/Lustrouscause07/Steganography-Project.git

CONCLUSION

- The **STEGOSAFE** project successfully demonstrates a practical, secure, and user-friendly solution for hiding and retrieving sensitive information within digital images using steganography. By integrating modern web technologies and image processing libraries, the system allows users to seamlessly encode and decode messages through a clean and intuitive interface. Unlike traditional encryption methods that draw attention, STEGOSAFE ensures that confidential data remains hidden in plain sight. This project highlights the potential of combining security with usability, offering a valuable tool for secure communication in both personal and professional environments.

FUTURE SCOPE(OPTIONAL)

- **Support for Multiple File Types**

Extend support to embed and extract messages from audio and video files in addition to images.

- **Mobile-Friendly Interface**

Develop a responsive version or a dedicated mobile app for on-the-go encoding and decoding.

- **Advanced Encryption Integration**

Combine steganography with AES or RSA encryption to further strengthen message security before embedding.

- **Drag-and-Drop File Upload**

Improve user experience by allowing drag-and-drop image uploads with live previews.

REFERENCES

- Provos & Honeyman – “Hide and Seek: Intro to Steganography,” IEEE, 2003.
- Darabseh et al. – “Image Steganography Using LSB & Edge Detection,” Springer, 2020.
- Morkel et al. – “Overview of Image Steganography,” Univ. of Pretoria, 2005.
- OpenCV Docs – QR Code Detection and Decoding.
- https://docs.opencv.org/
- Flask Docs – Python Web Framework Guide.

THANK YOU
- AAMIR KAMAL

edunet foundation