Capstone Project Presentation:Image Steganography Using AES Encryption

A Capstone Project by **Lavesh Agarwal**

Collage:- **MBM University, Jodhpur**

Organization:- **Edunet Foundation.**

Problem Statement :- Hiding Information in Images

This project addresses critical data security needs by hiding secret messages within images using steganography and AES encryption. The output image appears identical to the original, ensuring confidentiality and stealth.

Technology & Tools Used

System Requirements:

- Python 3.10
- VS Code or Jupyter Notebook
- PNG image input/output

Libraries Used:

- opencv-python (cv2)
- pycryptodome
- hashlib, string, os, numpy

Step-by-Step Workflow

1. Encrypt Message

Using AES with a secret key (CBC mode).

2. Convert to Binary & Add Header

Encrypted bytes converted to binary with a 32-bit data length header.

3. Embed in LSBs

Binary data embedded into the least significant bits of image pixels.

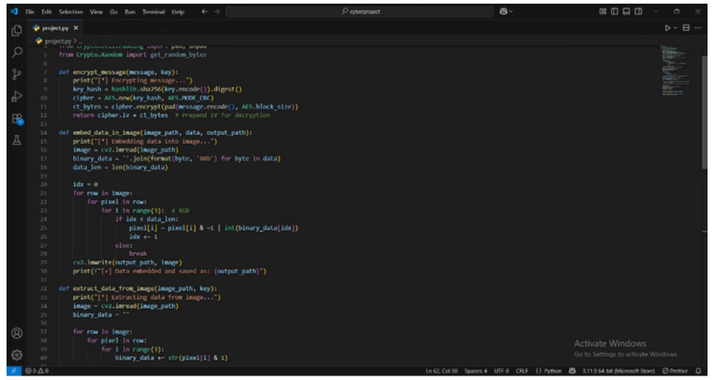
4. Save Stego-Image

The image with the hidden data is saved.

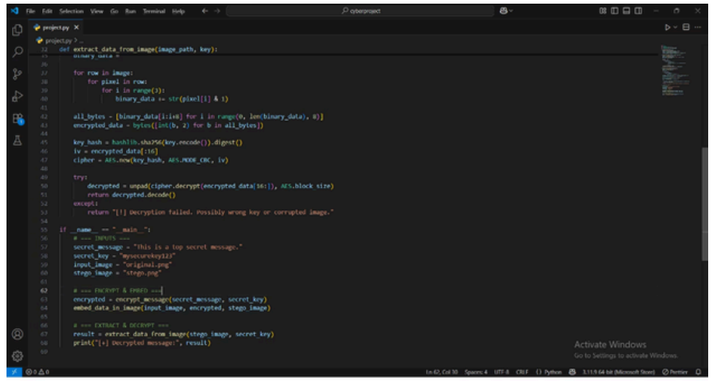
5. Extract & Decrypt

LSBs extracted, header read, data reconstructed, and decrypted with the same key.

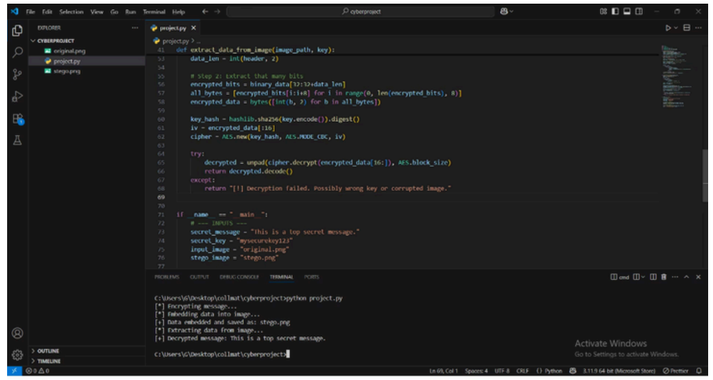
Code snippet



```
from crypto.Random import get_random_bytes

def encrypt_message(message, key):
    print("[*] Encrypting message...")
    key_hash = hashlib.sha256(key.encode()).digest()
    cipher = AES.new(key_hash, AES.MODE_CBC)
    ct_bytes = cipher.encrypt(pad(message.encode(), AES.block_size))
    return cipher.iv + ct_bytes  # Prepend IV for decryption

def embed_data_in_image(image_path, data, output_path):
    print("[*] Embedding data into image...")
    image = cv2.imread(image_path)
    binary_data = ''.join(format(byte, '08b') for byte in data)
    data_len = len(binary_data)

    idx = 0
    for row in image:
        for pixel in row:
            for i in range(3):  # RGB
                if idx < data_len:
                    pixel[i] = pixel[i] & ~1 | int(binary_data[idx])
                    idx += 1
                else:
                    break
    cv2.imwrite(output_path, image)
    print(f"[+] Data embedded and saved as: {output_path}")

def extract_data_from_image(image_path, key):
    print("[*] Extracting data from image...")
    image = cv2.imread(image_path)
    binary_data = ""

    for row in image:
        for pixel in row:
            for i in range(3):
                binary_data += str(pixel[i] & 1)
```



```
def extract_data_from_image(image_path, key):
    binary_data = ""

    for row in image:
        for pixel in row:
            for i in range(3):
                binary_data += str(pixel[i] & 1)

    all_bytes = [binary_data[i:i+8] for i in range(0, len(binary_data), 8)]
    encrypted_data = bytes([int(b, 2) for b in all_bytes])

    key_hash = hashlib.sha256(key.encode()).digest()
    iv = encrypted_data[:16]
    cipher = AES.new(key_hash, AES.MODE_CBC, iv)

    try:
        decrypted = unpad(cipher.decrypt(encrypted_data[16:]), AES.block_size)
        return decrypted.decode()
    except:
        return "[!] Decryption failed. Possibly wrong key or corrupted image."

if __name__ == "__main__":
    # === INPUTS ===
    secret_message = "This is a top secret message."
    secret_key = "mysecurekey123"
    input_image = "original.png"
    stego_image = "stego.png"

    # === ENCRYPT & EMBED ===
    encrypted = encrypt_message(secret_message, secret_key)
    embed_data_in_image(input_image, encrypted, stego_image)

    # === EXTRACT & DECRYPT ===
    result = extract_data_from_image(stego_image, secret_key)
    print("[+] Decrypted message:", result)
```



```
def extract_data_from_image(image_path, key):
    data_len = int(header, 2)

    # Step 2: Extract that many bits
    encrypted_bits = binary_data[32:32+data_len]
    all_bytes = [encrypted_bits[i:i+8] for i in range(0, len(encrypted_bits), 8)]
    encrypted_data = bytes([int(b, 2) for b in all_bytes])

    key_hash = hashlib.sha256(key.encode()).digest()
    iv = encrypted_data[:16]
    cipher = AES.new(key_hash, AES.MODE_CBC, iv)

    try:
        decrypted = unpad(cipher.decrypt(encrypted_data[16:]), AES.block_size)
        return decrypted.decode()
    except:
        return "[!] Decryption failed. Possibly wrong key or corrupted image."

if __name__ == "__main__":
    # === INPUTS ===
    secret_message = "This is a top secret message."
    secret_key = "mysecurekey123"
    input_image = "original.png"
    stego_image = "stego.png"

C:\Users\G\Desktop\colimat\cyberproject>python project.py
[*] Encrypting message...
[*] Embedding data into image...
[+] Data embedded and saved as: stego.png
[*] Extracting data from image...
[+] Decrypted message: This is a top secret message.

C:\Users\G\Desktop\colimat\cyberproject>
```

Screenshots demonstrating the successful execution of the steganography code, showing the original image and the output image with hidden data.

Original vs. Output Image

Original Photo

Output Image

Visual comparison of the original and stego-image, highlighting their identical appearance despite hidden data.

GitHub Repository

Explore the complete project source code and comprehensive documentation on our dedicated GitHub repository:

hhttps://github.com/laveshagawal/Steganography-project-

Full Source Code

Access all project files for review and future enhancements.

Execution Screenshots

Visual evidence of the successful steganography process.

Dependency List

Simple setup using the provided requirements.txt file.

Detailed Documentation

A comprehensive README.md for quick setup and usage instructions.

Conclusion & Learnings

Key Achievements:

- Successfully implemented steganography using AES in Python.
- Achieved stealth and confidentiality.
- Effective for PNG images; avoids distortion.

Challenges & Improvements:

- Bit overflow errors, message extraction issues.
- Future: Add JPEG/DCT support, GUI/web interface.

Thank You!

Access the full source code and documentation on our GitHub repository.

Please feel free to ask any questions.

Contact: laveshagarwal27mbm@gmail.com