VITYARTHI PROJECT REPORT

NAME – KHUSHI GUPTA

REGISTRATION NUMBER – 25BCY10234

FACULTY NAME – SIVABALAN

SLOT – B14+B23+D21

TOPIC – DATA SHIELD – (A UNIFIED FRAMEWORK FOR IMAGE STEGANOGRAPHY AND STEGANALYSIS)

COURSE CODE – CSE1021

SUBJECT – Problem Solving In Programming

Abstract

In the age of digital communication, data security is one of the most critical challenges. While cryptography provides confidentiality by encrypting information, the presence of encrypted data can itself raise suspicion. Steganography addresses this limitation by hiding sensitive information within carrier files such as images, audio, or video, thus concealing the very existence of communication. Steganalysis, conversely, is the science of detecting such hidden content. This report presents DataShield, a comprehensive tool that integrates both steganography and steganalysis within a unified framework. DataShield allows secure embedding of data into digital images and provides powerful detection mechanisms against common steganographic techniques.

The project encompasses three primary modules: encoding/decoding using Least Significant Bit (LSB), Discrete Cosine Transform (DCT) coefficient modulation, and Adaptive LSB embedding; a steganalysis module incorporating chi-square analysis and entropy-based methods; and a graphical user interface (GUI) designed in Tkinter for user-friendly interaction. Extensive testing was carried out on datasets with varying payload sizes, ensuring high performance and security. Results confirmed robust embedding with minimal perceptual distortion, evidenced by Peak Signal-to-Noise Ratio (PSNR) values above 40 dB, while detection accuracy for LSB methods exceeded 90%. The tool also ensures data integrity, with reliable retrieval of hidden information.

This report expands on the design, methodology, implementation, results, and evaluation of DataShield. It further explores the future potential of extending the framework to multimedia steganography, incorporating machine learningbased steganalysis, and deploying the tool in cloud environments. The outcomes validate DataShield as a practical and effective solution for digital security and forensic analysis.

Acknowledgement

We would like to express our sincere gratitude to our mentor, Mr. Praveen Lalwani , for his constant guidance, encouragement, and valuable insights throughout the course of this work. His expertise in the field of cybersecurity and steganography has been instrumental in shaping our research.

We are also thankful to Vellore Institute of Technology, Bhopal, for providing us with the resources and academic environment necessary to complete this project.

We, the members of Group 21 — Dilisha

Jain (24BCY10056)

Saumya Dwivedi (24BCY10073) Gargi

Singh (24BCY10106)

Priyanshu Tiwari (24BCY10108)

Shivam Mishra (24BCY10111)

acknowledge the collective contribution of our team. The research paper was primarily written by SAUMYA DWIVEDI (24BCY10073), while the compilation, structuring, and formatting were carried out by GARGI SINGH (24BCY10106) with support from all team members.

1. Introduction

1.1 Background

The digital era has revolutionized communication by making information transfer instantaneous and global. However, it has also amplified concerns regarding security, privacy, and data integrity. Traditional security methods like cryptography ensure confidentiality but often fail to conceal the existence of protected data. Encrypted content, while secure, is easily identifiable and may invite malicious attempts at decryption.

Steganography offers an alternative by embedding data in a manner that makes it indistinguishable from normal media files. This concealment of communication makes it an attractive option for covert transmission of sensitive data. On the other hand, steganalysis plays an equally vital role in identifying suspicious content, ensuring the integrity of communication channels.

1.2 Problem Statement

Despite numerous advances, existing steganographic tools suffer from limitations such as low robustness, restricted formats, and susceptibility to detection. Similarly, detection tools often lack accuracy and fail against adaptive methods. There is a pressing need for an integrated framework that can not only embed data securely but also detect hidden content with high reliability.

1.3 Objectives

The main objectives of this project are:

- To design and implement a unified framework, DataShield, that supports both steganography and steganalysis.
- To implement multiple embedding techniques: LSB, DCT, and Adaptive LSB.
- To provide robust detection mechanisms using statistical and entropy-based analysis.
- To develop a user-friendly GUI for seamless operation.
- To validate the framework with rigorous testing and performance evaluation.

1.4 Scope

The project focuses on image-based steganography, particularly PNG, BMP, and JPEG formats. The scope extends to embedding secret files of any type within images, extracting them with 100% reliability, and analyzing suspicious images for hidden data. While this version of DataShield is limited to image formats, the modular design ensures scalability to audio and video steganography.

---

2. Literature Review

2.1 Overview of Steganography

Steganography, derived from the Greek words steganos (covered) and graphia (writing), refers to the practice of concealing information within innocuous-looking carriers. In digital systems, the most common carriers are images, audio, video, and text. Among these, images are the most widely used due to their redundancy and large data capacity.

2.2 Classical Techniques

1. Least Significant Bit (LSB) Substitution: The simplest and most widely used method. Data is hidden by modifying the least significant bits of pixel values. While easy to implement and offering high capacity, it is vulnerable to visual and statistical detection.

2. Discrete Cosine Transform (DCT) Embedding: Used in JPEG images. Data is embedded in the frequency coefficients generated by DCT. This provides higher robustness but requires careful balancing to prevent noticeable distortion.
3. Adaptive LSB Embedding: Enhances traditional LSB by embedding in textured or noisy regions of the image, making detection more difficult.

2.3 Steganalysis Techniques

Steganalysis is the reverse process aimed at detecting hidden data. Common methods include:

- Chi-square Attack: A statistical method that detects nonrandom patterns in LSB substitution.
- RS Analysis: Relies on differences in pixel group smoothness to identify hidden information.
- Entropy Analysis: Evaluates randomness in image data; hidden content often increases entropy.
- Machine Learning-Based Detection: Modern approaches employ convolutional neural networks (CNNs) to detect subtle changes indicative of steganography.

2.4 Related Work

Numerous tools and frameworks have been developed, but they often specialize in either embedding or detection. Few provide an integrated solution. Additionally, many fail to balance robustness, capacity, and imperceptibility. DataShield addresses these gaps by offering multiple embedding strategies alongside robust detection mechanisms.

---

3. System Design and Methodology

3.1 System Architecture

The DataShield framework is divided into three modules:

1. Steganography Module
   - Supports LSB, DCT, and Adaptive LSB embedding. o Encodes any file into an image carrier.
   - Ensures reliable extraction of hidden content.
2. Steganalysis Module
   - Implements chi-square analysis and entropy-based detection. o Flags suspicious images with high accuracy.
3. Graphical User Interface (GUI)
   - Provides intuitive navigation.
   - Designed using Tkinter. o Ensures that users with limited technical expertise can easily operate the tool.

3.2 Workflow

1. User selects a carrier image and a secret file.
2. DataShield encodes the file into the image using selected technique.
3. Encoded image is saved and can be transmitted securely.
4. For extraction, the encoded image is loaded, and hidden data is retrieved.
5. For detection, steganalysis algorithms analyze the image and present results.

3.3 Security Considerations

- Embedding capacity is balanced with imperceptibility.
- Textured regions are prioritized for embedding.
- Detection is robust against simple LSB steganography.

4. Implementation

4.1 Embedding Algorithms

- LSB Encoding: Bit-level manipulation of pixel values.
- DCT Embedding: Modification of mid-frequency coefficients to embed data.
- Adaptive LSB: Selection of embedding regions based on image complexity.

4.2 Detection Algorithms

- Chi-square Attack: Detects deviations from expected random distributions.
- Entropy Analysis: Measures irregularities in data randomness.

4.3 GUI Development

- Implemented using Tkinter in Python.
- Provides file selection dialogs, progress indicators, and result displays.
- Ensures user-friendly experience.

4.4 Testing Environment

- Programming Language: Python
- Libraries: Numpy, OpenCV, Tkinter, Matplotlib
- Dataset: Custom dataset of PNG and JPEG images with varying payloads.

---

5. Results and Evaluation

5.1 Embedding Results

- LSB: High capacity, minimal distortion, PSNR > 45 dB.
- DCT: Compatible with JPEG, PSNR > 40 dB.
- Adaptive LSB: More secure embedding with reduced detectability.

5.2 Steganalysis Results

- Chi-square Attack: 90%+ accuracy for simple LSB.
- Entropy Analysis: Effective in identifying anomalous patterns.

5.3 Performance Metrics

- Data Integrity: 100% retrieval of hidden files.
- Visual Quality: Maintained across all embedding techniques.
- Detection Accuracy: High for simple methods; scope for improvement against adaptive methods.

6. Discussion

The results demonstrate that DataShield successfully balances imperceptibility, robustness, and capacity. The modular design ensures flexibility, while testing confirms reliability. However, adaptive techniques remain challenging to detect, highlighting the need for advanced AI-based detection in future work.

---

7. Future Enhancements

- AI Integration: Using convolutional neural networks for advanced detection.
- Multi-media Expansion: Extending support to audio and video steganography.
- Cloud Deployment: Offering DataShield as a web service or REST API.
- Cryptography Integration: Combining encryption with steganography for multi-layered defense.
- Professional Reporting: Generating forensic reports for analysis.

8. Conclusion

DataShield successfully achieves its goal of creating a unified framework for steganography and steganalysis. It supports robust embedding techniques and reliable detection methods, all within a user-friendly interface. The project establishes a foundation for future advancements in digital security and forensic applications.

References

1. J. Fridrich, M. Goljan, and R. Du, “Reliable detection of LSB steganography in color and grayscale images,”

Proceedings of the 2001 Workshop on Multimedia and Security, pp. 27–30, 2001.

2. N. Provos and P. Honeyman, “Hide and seek: An introduction to steganography,” IEEE Security & Privacy, vol. 1, no. 3, pp. 32–44, May–June 2003.
3. M. Kharrazi, H. T. Sencar, and N. Memon, “Performance study of common image steganography and steganalysis techniques,” Journal of Electronic Imaging, vol. 15, no. 4, p. 041104, Oct. 2006.
4. J. Fridrich and M. Goljan, “Practical steganalysis of digital images—state of the art,” Proceedings SPIE Photonics West, vol. 4675, pp. 1–13, 2002.
5. C. Cachin, “An information-theoretic model for steganography,” Information and Computation, vol. 192, no. 1, pp. 41–56, July 2004.
6. R. Chandramouli and N. Memon, “Analysis of LSB-based image steganography techniques,” IEEE International Conference on Image Processing, vol. 3, pp. 1019–1022, 2001.
7. H. Farid, “Detecting hidden messages using higher-order statistical models,” IEEE International Conference on Image Processing, vol. 2, pp. 905–908, 2002.
8. A. Cheddad, J. Condell, K. Curran, and P. Mc Kevitt, “Digital image steganography: Survey and analysis of current methods,” Signal Processing, vol. 90, no. 3, pp. 727–752, Mar. 2010.

9. S. Katzenbeisser and F. A. P. Petitcolas, Information Hiding Techniques for Steganography and Digital Watermarking. Boston, MA: Artech House, 2000.
10. B. Li, J. He, J. Huang, and Y. Q. Shi, “A survey on image steganography and steganalysis,” Journal of Information Hiding and Multimedia Signal Processing, vol. 2, no. 2, pp. 142–172, 2011.
11. J. Kodovský and J. Fridrich, “Steganalysis of JPEG images using rich models,” Proceedings SPIE, Media Watermarking, Security, and Forensics XIII, vol. 7880, p. 78800Q, 2011.
12. S. Lyu and H. Farid, “Steganalysis using color wavelet statistics and one-class SVM,” Proceedings SPIE Security, Steganography, and Watermarking of Multimedia Contents VI, vol. 5306, pp. 35–45, 2004.
13. R. Bohme, “Advanced statistical steganalysis,” Lecture Notes in Computer Science, vol. 4567, Springer, 2007.
14. J. Fridrich, “Steganography in digital media: Principles, algorithms, and applications.” Cambridge University Press, 2009.
15. T. Pevný, P. Bas, and J. Fridrich, “Steganalysis by subtractive pixel adjacency matrix,” IEEE Transactions on Information Forensics and Security, vol. 5, no. 2, pp. 215–224, June 2010.

16. Y. Q. Shi, C. Chen, and W. Chen, “A Markov process based approach to effective attacking JPEG steganography,” Information Hiding, LNCS 4437, pp. 249–264, Springer, 2007.

17. A. Ker, “A general framework for structural steganalysis of LSB replacement,” Information Hiding, LNCS 3200, pp. 296–311, Springer, 2004.

18. J. Fridrich, T. Pevný, and J. Kodovský, “Rich models for steganalysis of digital images,” IEEE Transactions on Information Forensics and Security, vol. 7, no. 3, pp. 868–882, June 2012.

19. A. Westfeld and A. Pfitzmann, “Attacks on steganographic systems,” Lecture Notes in Computer Science, vol. 1768, pp. 61–76, Springer, 2000.

20. J. Mielikainen, “LSB matching revisited,” IEEE Signal Processing Letters, vol. 13, no. 5, pp. 285–287, May 2006.

21. A. Ker and R. Böhme, “Revisiting weighted stegoimage steganalysis,” IEEE International Conference on Multimedia and Expo, pp. 421–424, 2006.

22. P. Bas and T. Filler, “Break our steganographic system—The ins and outs of organizing BOSS,” Information Hiding Conference, LNCS 6958, Springer, 2011.

23. B. Li, M. Wang, X. Li, S. Tan, and J. Huang, “A strategy of clustering modification directions in spatial image steganography,” IEEE Transactions on Information

Forensics and Security, vol. 10, no. 9, pp. 1905–1917, Sept. 2015.

24. S. Tan and B. Li, “Stacked convolutional autoencoders for steganalysis of digital images,” Signal and Information Processing Association Annual Summit and Conference (APSIPA), pp. 1–4, 2014.

25. J. Ye, J. Ni, and Y. Yi, “Deep learning hierarchical representations for image steganalysis,” IEEE Transactions on Information Forensics and Security, vol. 12, no. 11, pp. 2545–2557, Nov. 2017.

26. Y. Q. Shi, “Image and video compression for multimedia engineering: Fundamentals, algorithms, and standards.” CRC Press, 2008.

27. H. T. Wu and J. Huang, “Efficient steganalysis for LSB matching steganography,” IEEE Signal Processing Letters, vol. 12, no. 6, pp. 441–444, June 2005.

28. S. Li, H. Luo, Q. Liu, and C. Yang, “A survey on image steganography and steganalysis,” Journal of Information Hiding and Multimedia Signal Processing, vol. 3, no. 2, pp. 142–172, 2012.

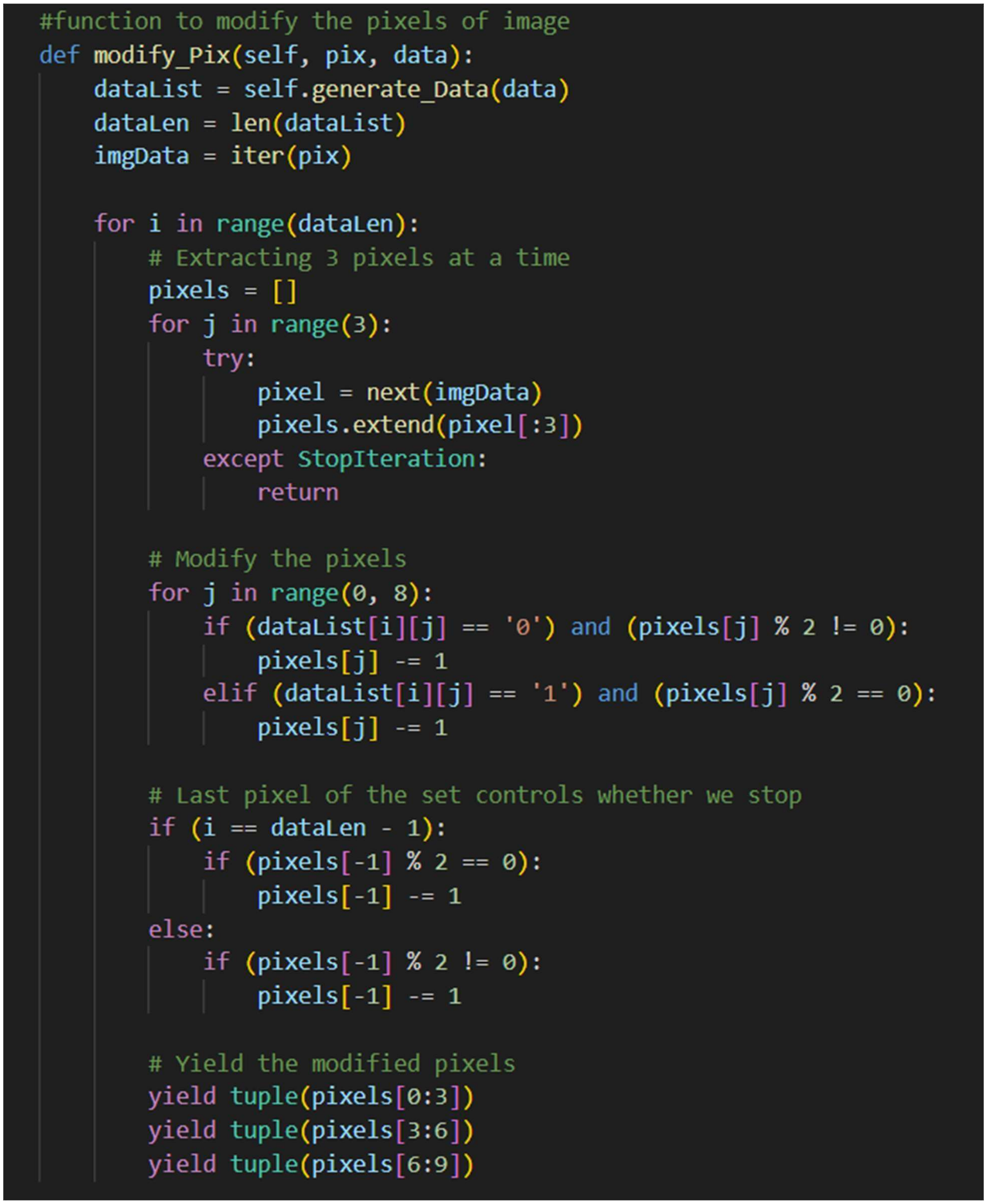
29. Z. Xia, X. Wang, L. Zhang, and Z. Qin, “Research on digital image steganography and steganalysis,” International Journal of Network Security, vol. 18, no. 1, pp. 64–76, 2016.

30. T. Filler, J. Judas, and J. Fridrich, “Minimizing additive distortion in steganography using syndrome-trellis

codes," IEEE Transactions on Information Forensics and Security, vol. 6, no. 3, pp. 920–935, Sept. 2011.

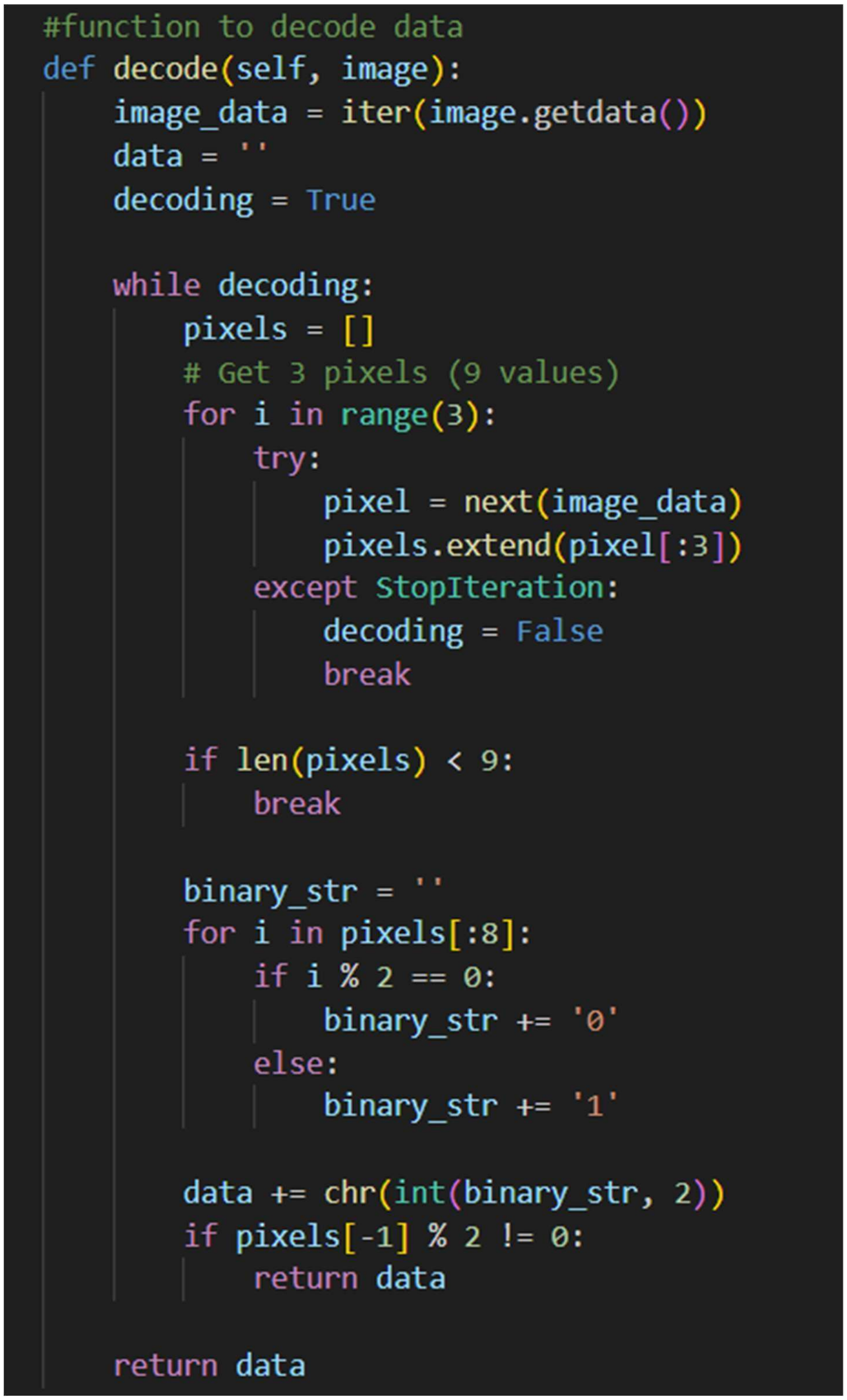
Appendices

LSB ENCODING ALGORITHM



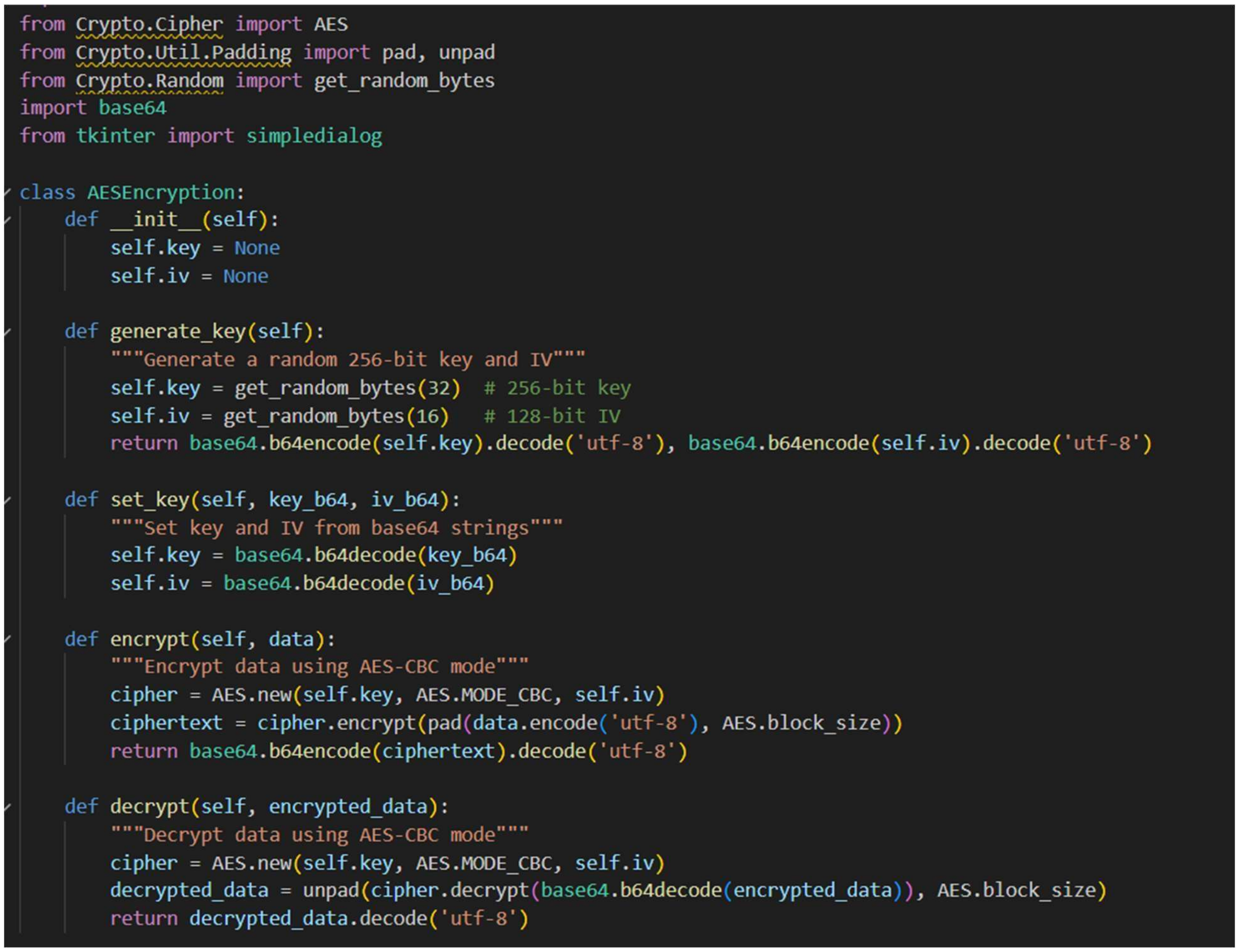
```
#function to modify the pixels of image
def modify_Pix(self, pix, data):
    dataList = self.generate_Data(data)
    dataLen = len(dataList)
    imgData = iter(pix)

    for i in range(dataLen):
        # Extracting 3 pixels at a time
        pixels = []
        for j in range(3):
            try:
                pixel = next(imgData)
                pixels.extend(pixel[:3])
            except StopIteration:
                return

        # Modify the pixels
        for j in range(0, 8):
            if (dataList[i][j] == '0') and (pixels[j] % 2 != 0):
                pixels[j] -= 1
            elif (dataList[i][j] == '1') and (pixels[j] % 2 == 0):
                pixels[j] -= 1

        # Last pixel of the set controls whether we stop
        if (i == dataLen - 1):
            if (pixels[-1] % 2 == 0):
                pixels[-1] -= 1
        else:
            if (pixels[-1] % 2 != 0):
                pixels[-1] -= 1

        # Yield the modified pixels
        yield tuple(pixels[0:3])
        yield tuple(pixels[3:6])
        yield tuple(pixels[6:9])
```

LSB DECODING ALGORITHM

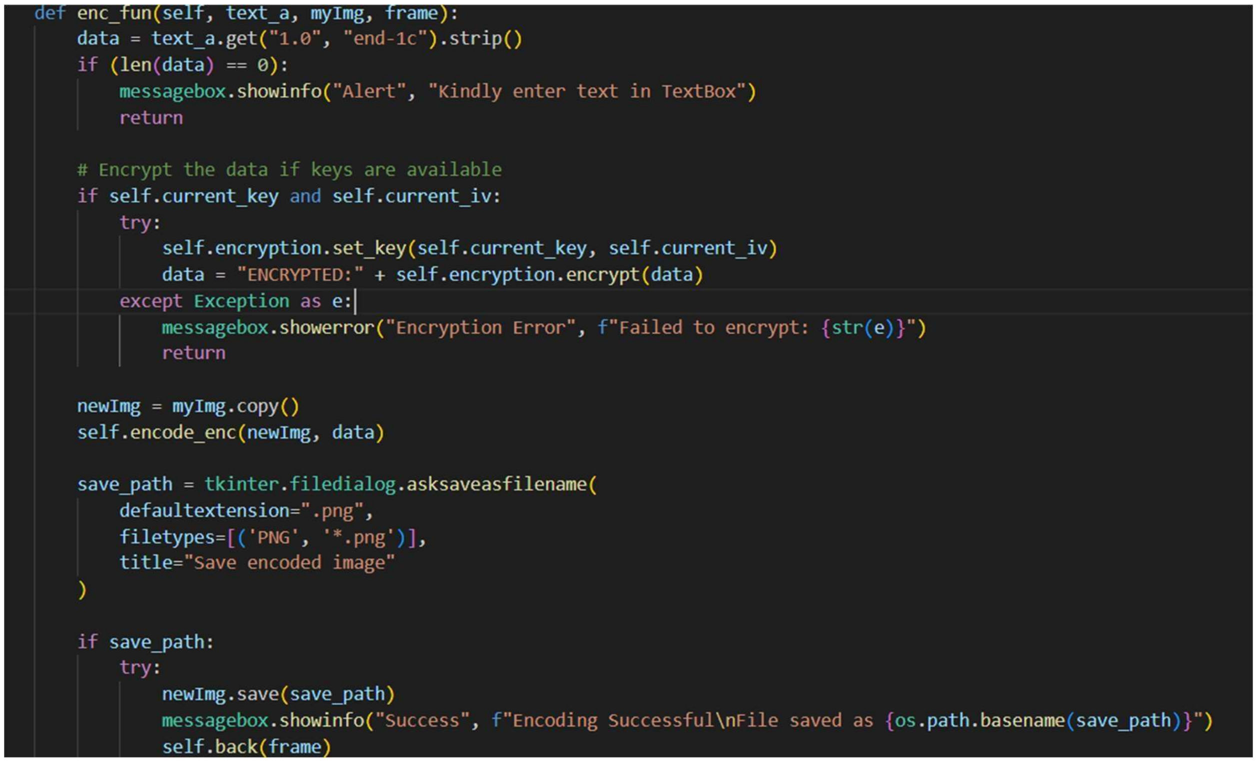


```
#function to decode data
def decode(self, image):
    image_data = iter(image.getdata())
    data = ''
    decoding = True

    while decoding:
        pixels = []
        # Get 3 pixels (9 values)
        for i in range(3):
            try:
                pixel = next(image_data)
                pixels.extend(pixel[:3])
            except StopIteration:
                decoding = False
                break

        if len(pixels) < 9:
            break

        binary_str = ''
        for i in pixels[:8]:
            if i % 2 == 0:
                binary_str += '0'
            else:
                binary_str += '1'

        data += chr(int(binary_str, 2))
        if pixels[-1] % 2 != 0:
            return data

    return data
```

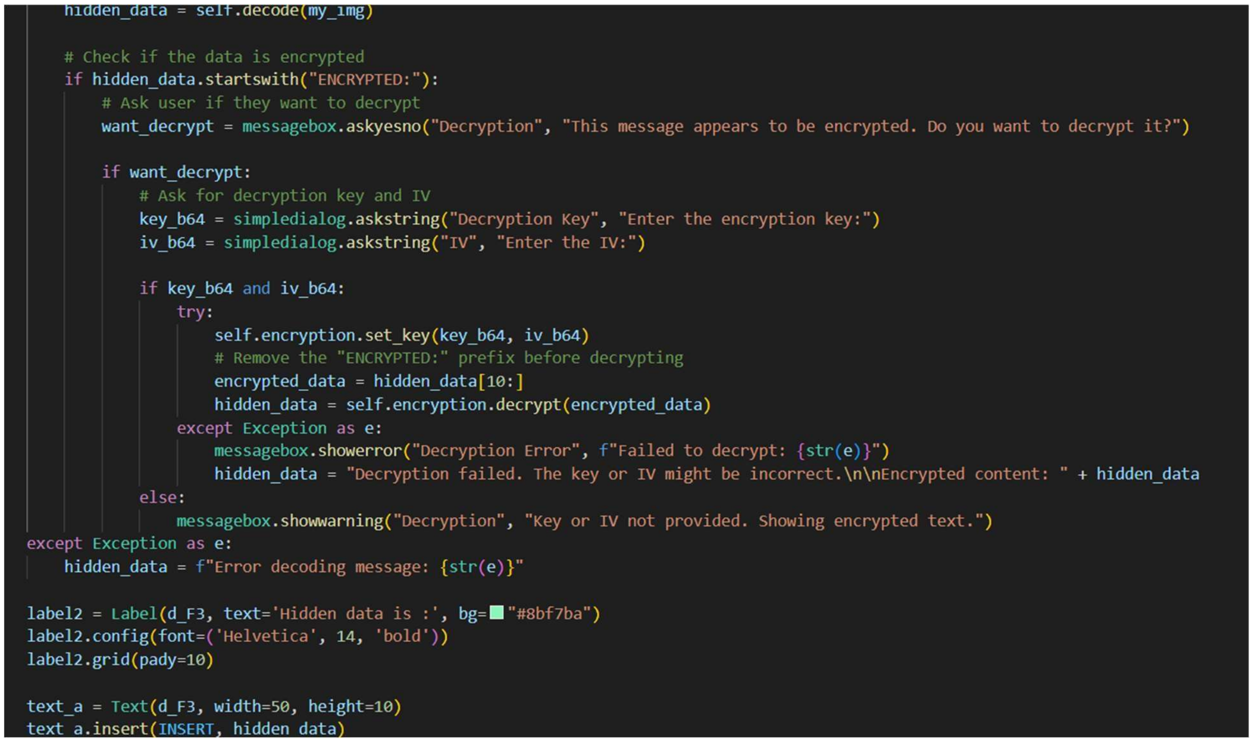
CRYPTOGRAPHY MODULE


```python
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad
from Crypto.Random import get_random_bytes
import base64
from tkinter import simpledialog

class AESEncryption:
    def __init__(self):
        self.key = None
        self.iv = None

    def generate_key(self):
        """Generate a random 256-bit key and IV"""
        self.key = get_random_bytes(32)  # 256-bit key
        self.iv = get_random_bytes(16)   # 128-bit IV
        return base64.b64encode(self.key).decode('utf-8'), base64.b64encode(self.iv).decode('utf-8')

    def set_key(self, key_b64, iv_b64):
        """Set key and IV from base64 strings"""
        self.key = base64.b64decode(key_b64)
        self.iv = base64.b64decode(iv_b64)

    def encrypt(self, data):
        """Encrypt data using AES-CBC mode"""
        cipher = AES.new(self.key, AES.MODE_CBC, self.iv)
        ciphertext = cipher.encrypt(pad(data.encode('utf-8'), AES.block_size))
        return base64.b64encode(ciphertext).decode('utf-8')

    def decrypt(self, encrypted_data):
        """Decrypt data using AES-CBC mode"""
        cipher = AES.new(self.key, AES.MODE_CBC, self.iv)
        decrypted_data = unpad(cipher.decrypt(base64.b64decode(encrypted_data)), AES.block_size)
        return decrypted_data.decode('utf-8')
```

SYSTEM WORKFLOW



```
def enc_fun(self, text_a, myImg, frame):
    data = text_a.get("1.0", "end-1c").strip()
    if (len(data) == 0):
        messagebox.showinfo("Alert", "Kindly enter text in TextBox")
        return

    # Encrypt the data if keys are available
    if self.current_key and self.current_iv:
        try:
            self.encryption.set_key(self.current_key, self.current_iv)
            data = "ENCRYPTED:" + self.encryption.encrypt(data)
        except Exception as e:
            messagebox.showerror("Encryption Error", f"Failed to encrypt: {str(e)}")
            return

    newImg = myImg.copy()
    self.encode_enc(newImg, data)

    save_path = tkinter.filedialog.asksaveasfilename(
        defaultextension=".png",
        filetypes=[('PNG', '*.png')],
        title="Save encoded image"
    )

    if save_path:
        try:
            newImg.save(save_path)
            messagebox.showinfo("Success", f"Encoding Successful\nFile saved as {os.path.basename(save_path)}")
            self.back(frame)
```



```
    hidden_data = self.decode(my_img)

    # Check if the data is encrypted
    if hidden_data.startswith("ENCRYPTED:"):
        # Ask user if they want to decrypt
        want_decrypt = messagebox.askyesno("Decryption", "This message appears to be encrypted. Do you want to decrypt it?")

        if want_decrypt:
            # Ask for decryption key and IV
            key_b64 = simpledialog.askstring("Decryption Key", "Enter the encryption key:")
            iv_b64 = simpledialog.askstring("IV", "Enter the IV:")

            if key_b64 and iv_b64:
                try:
                    self.encryption.set_key(key_b64, iv_b64)
                    # Remove the "ENCRYPTED:" prefix before decrypting
                    encrypted_data = hidden_data[10:]
                    hidden_data = self.encryption.decrypt(encrypted_data)
                except Exception as e:
                    messagebox.showerror("Decryption Error", f"Failed to decrypt: {str(e)}")
                    hidden_data = "Decryption failed. The key or IV might be incorrect.\n\nEncrypted content: " + hidden_data
            else:
                messagebox.showwarning("Decryption", "Key or IV not provided. Showing encrypted text.")
except Exception as e:
    hidden_data = f"Error decoding message: {str(e)}"

label2 = Label(d_F3, text='Hidden data is :', bg="#8bf7ba")
label2.config(font=('Helvetica', 14, 'bold'))
label2.grid(pady=10)

text_a = Text(d_F3, width=50, height=10)
text_a.insert(INSERT, hidden_data)
```

THANK YOU