CHAPTER – 01
INTRODUCTION

CHAPTER 1 : INTRODUCTION

1.1 : INTRODUCTION

The word steganography literally means "covered writing" as derived from Greek . It includes a vast array of methods of secret communications that conceal the very existence of the message . Among these methods are invisible inks , microdots , character arrangement (other than cryptographic methods of permutations and substitution) , digital signatures , convert channels and spread – spectrum communications .

Steganography is the art of concealing the existence of information within seemingly innocuous carriers . Steganography has been exploited throughout history by individuals , military , secret intelligence , and governments to stealthily communicate and transmit information without drawing any attraction . It has numerous applications which range from secret communication , to digital watermarking , data integrity , copyright protection , and data tampering .

The motivation behind developing Image Steganography methods according to its use in various organizations to communicate between its members as well as , it can be used for communication between members of the military or intelligence operatives or agents of companies to hide secret message or in the field of espionage . The main goal of using Steganography is to avoid drawing attention to the transmission of hidden information .

Steganography is of different types:

1. Text steganography
2. Image steganography
3. Audio steganography
4. Video steganography

In all of these methods, the basic principle of steganography is that a secret message is to be embedded in another cover object which may not be of any significance in such a way that the encrypted data would finally display only the cover data. So it cannot be detected easily to be containing hidden information unless proper decryption is used.

Image steganography is a technique used to hide secret information within an image without arousing suspicion. It is a form of data hiding that focuses on embedding confidential data or messages within the pixels of an image, making it appear unchanged to the human eye. The hidden information is imperceptible and can only be extracted by those who have knowledge of the steganographic technique.

The primary objective of image steganography is to ensure secure communication and confidentiality by concealing sensitive data within an innocuous image. This technique has gained significant attention due to its potential applications in various fields, including information security, covert communication, and digital forensics.

Image steganography operates on the principle of exploiting the limitations of human visual perception. It takes advantage of the fact that the human eye is not sensitive enough to detect slight modifications in an image. By carefully manipulating the color values or the least significant bits of the image pixels, hidden data can be embedded, resulting in a visually identical image to the original.

The process of image steganography involves two essential components: embedding and extraction. In the embedding phase, the secret data is concealed within the image using a steganographic algorithm. In the extraction phase, the hidden data is retrieved from the stego-image by applying the reverse process.

Image steganography offers several advantages, including the ability to hide sensitive information in plain sight, the resistance to detection by unauthorized individuals, and the use of popular image formats as carrier files. However, it also presents challenges such as the need for robust algorithms to ensure data integrity, capacity limitations for embedding data, and vulnerability to attacks aimed at detecting hidden information.

In conclusion, image steganography is a powerful technique for secure communication and information hiding. It enables the covert transmission of confidential data within digital images, providing an additional layer of security. As technology advances, image steganography continues to evolve, offering new possibilities and challenges in the field of data protection and privacy.

1.2 : OBJECTIVE

The objective of image steganography is to hide a secret message or information within an image in such a way that it cannot be detected easily. This process of hiding the secret message is done in such a way that it appears to be a normal image to the naked eye, and only the intended recipient who has the key to decode the hidden message can extract it.

The primary objective of image steganography is to provide a secure and covert way of communicating sensitive information. It is a crucial technique for protecting privacy and confidentiality in the digital world, especially in areas where the transfer of classified or sensitive data is of utmost importance.

Another objective of image steganography is to ensure the authenticity of the data. By embedding the hidden message within an image, it becomes challenging for an attacker to modify or tamper with the data without destroying the hidden message.

Image steganography finds its applications in various fields, including military, healthcare, finance, and law enforcement, among others. It is an essential tool for secret communication and can be used for various purposes, such as in forensics to detect image tampering or to protect sensitive data during transmission.

Overall, the objective of image steganography is to provide a secure, covert, and efficient way of communicating sensitive information while maintaining data integrity and authenticity**.**

The objective of image steganography is to facilitate secure and covert communication by embedding secret information within an image. The primary goals of image steganography can be summarized as follows:

1. **Confidentiality:** The main objective is to ensure the confidentiality of sensitive information during transmission. By hiding the information within the image, steganography aims to prevent unauthorized access and interception by encrypting the data in a manner that is undetectable to the naked eye.

2. **Covert Communication**: Image steganography enables the transmission of secret messages in a covert manner. The goal is to make the hidden information indistinguishable from regular image data, minimizing the chances of detection by unintended recipients or eavesdroppers.

3. **Data Integrity:** Another objective is to maintain the integrity of the image and the hidden message. The process of embedding the secret information should not significantly distort the original image or affect its visual quality. The hidden message should be successfully retrieved without any loss or corruption.

4. **Resistance to Attacks:** Steganography techniques aim to withstand various attacks and attempts to uncover the hidden information. Robust algorithms and encryption methods are employed to ensure the hidden data remains intact and secure against statistical analysis, digital forensics, and other attacks.

5. **Versatility:** Image steganography can be applied to various image formats and can accommodate different types of hidden information, including text, audio, video, or other files. The objective is to provide a flexible and adaptable solution for concealing a wide range of data within images.

   Overall, the objective of image steganography is to provide a reliable and efficient method of secure communication, protecting sensitive information from unauthorized access, interception, and detection while maintaining the integrity and authenticity of the image.

1.3: ADVANTAGES OF IMAGE STEGANGRAPHY

Image steganography offers several advantages in the realm of information hiding and secure communication. Here are some of the key advantages:

1. **Hidden Communication**: Image steganography allows for hidden communication, where secret messages or data can be concealed within innocent-looking images. This covert communication can be useful in scenarios where confidentiality is crucial, such as military, intelligence, or sensitive corporate communications.

2. **Inconspicuous Nature**: Steganographic images appear visually identical to regular images, making it difficult for unauthorized individuals to detect the presence of hidden information. This characteristic enables steganography to maintain a low profile and enhances the security of the hidden data.

3. **Large Payload Capacity**: Images typically have a large storage capacity, allowing for a significant amount of data to be concealed within them. The available space for embedding secret messages is much greater in images compared to other file formats, like text or audio.

4. **Multiple Image Formats**: Steganography can be applied to various image formats, such as JPEG, PNG, BMP, and GIF, providing flexibility and compatibility across different platforms and applications. This wide range of supported formats makes it easier to choose an appropriate cover image for hiding information.

5. Resistance to Interception: Steganography can enhance the confidentiality of information by making it less susceptible to interception during transmission. By embedding messages within images, the hidden data becomes less conspicuous and can avoid detection by eavesdroppers or unauthorized individuals.

6. Integration with Existing Infrastructure: Image steganography can be seamlessly integrated into existing systems and infrastructure, as images are widely used in digital media and communication. This integration simplifies the implementation of steganographic techniques and enables their adoption in various domains.

7. Complementary Security Measures: Steganography can be used in conjunction with encryption and other security measures to provide an additional layer of protection. By combining steganography with encryption, the hidden information is further safeguarded, as even if the encrypted data is somehow compromised, the encrypted message remains concealed within the image.

Despite these advantages, it is important to note that steganography is not foolproof and has its limitations. Various steganalysis techniques and attacks can be employed to detect or extract hidden information. Therefore, careful consideration and proper implementation of steganographic methods are necessary to ensure the security and effectiveness of the hidden data.

1.4: DISADVANTAGES OF IMAGE STEGANOGRAPHY

While image steganography offers several advantages, it is important to be aware of its limitations and potential drawbacks. Here are some of the disadvantages of image steganography:

1. Vulnerability to Detection: Image steganography techniques are not completely immune to detection. Advanced steganalysis techniques can be employed to analyze and detect the presence of hidden information within images. Various statistical analyses, visual inspections, or specialized algorithms can be used to identify anomalies or patterns that indicate the presence of steganographic data.

2. Lossy Compression Impact: Steganography in lossy compressed image formats, such as JPEG, can result in degradation of the image quality. When data is embedded into a compressed image, the compression algorithm may modify the image data, resulting in a loss of image fidelity. This loss of quality can be noticeable, especially if the hidden message occupies a significant portion of the image.

3. Limited Payload Size: Although images have a large storage capacity compared to other file formats, the amount of data that can be embedded within an image is still limited. The available space for hiding the secret message depends on factors like image dimensions, color depth, and desired imperceptibility. Embedding large amounts of data may lead to a significant alteration of the image, raising suspicions.

4. Security Risks in Transmission: While steganography can enhance confidentiality during transmission, it is not a standalone secure communication solution. If the transmission of the steganographic image is not properly secured, it can still be intercepted or manipulated. Encrypting the hidden message and employing secure transmission channels are crucial to maintaining the integrity and confidentiality of the hidden data.

5. Loss of Data Robustness: Embedding data within an image can make it vulnerable to loss or corruption. Any unintended modifications or data loss during image processing, compression, or conversion can potentially render the hidden message irrecoverable. It is important to consider the robustness of the steganographic method used to ensure the hidden data remains intact under various scenarios.

6. Computational Overhead: Some steganography techniques can be computationally intensive, especially when dealing with high-resolution images or large amounts of data. Embedding and extracting hidden information may require significant processing power and time, making it less suitable for real-time or resource-constrained applications.

7. Ethical and Legal Considerations: The use of image steganography raises ethical and legal concerns, as it can be employed for illicit purposes, such as unauthorized communication, copyright infringement, or malicious activities. It is important to ensure that steganography is used responsibly, within legal boundaries, and in compliance with applicable regulations.

CHAPTER 02

SURVEY OF TECHNOLOGIES

CHAPTER 02 : SURVEY OF TECHNOLOGIES

2.1 : SURVEY OF TECHNOLOGIES

The survey of technologies used in my image steganography project : -

1. Python: Python is a versatile and widely used programming language that offers a range of features and libraries suitable for image steganography. Python's simplicity and readability make it an ideal choice for implementing steganography algorithms. It provides built-in data structures and functions, as well as extensive support for mathematical operations and string manipulation, which are essential for working with image data.

Python's extensive collection of libraries is a key advantage for image steganography. Libraries such as Pillow (PIL), OpenCV, and scikit-image offer comprehensive functionality for image processing tasks like loading, resizing, and manipulating images at the pixel level. These libraries provide a solid foundation for implementing steganography algorithms that involve embedding and extracting hidden data from images.

2. Spyder IDE: Spyder is a powerful integrated development environment (IDE) designed specifically for scientific computing with Python. It provides a feature-rich environment for writing, testing, and debugging Python code, making it well-suited for image steganography projects.

Spyder offers a user-friendly interface with a code editor, variable explorer, and integrated IPython console. These features facilitate efficient coding, experimentation, and data analysis during the development of image steganography algorithms. Spyder's debugging capabilities help identify and fix issues in the code, ensuring the reliability and accuracy of the steganography implementation.

Additionally, Spyder's data visualization tools allow for the exploration and analysis of image data, aiding in the evaluation of steganography techniques and their performance.

3. Tkinter Library: Tkinter is the standard GUI toolkit for Python, providing a set of tools for creating graphical user interfaces. It enables the development of interactive and visually appealing applications, including image steganography tools.

Tkinter offers a wide range of widgets, such as buttons, labels, entry fields, and file dialogs, which are essential for creating user-friendly interfaces for image steganography applications. These widgets allow users to interact with the application, select images, enter secret messages, and trigger the steganography process.

By utilizing Tkinter, developers can create intuitive and visually pleasing interfaces for their image steganography projects, enhancing the overall user experience.

4. Image Processing Libraries: Python provides several powerful image processing libraries, including Pillow (PIL), OpenCV, and scikit-image. These libraries offer extensive functionality for working with images, making them invaluable for image steganography projects.

Pillow (PIL) is a widely used library for image manipulation tasks, providing capabilities such as image loading, resizing, cropping, and pixel-level manipulation. OpenCV is a highly popular library for computer vision and image processing, offering advanced features like image filtering, edge detection, and feature extraction. scikit-image focuses on image processing algorithms and provides tools for image enhancement, segmentation, and morphological operations.

These image processing libraries enable developers to preprocess images, extract relevant features, and manipulate pixels for steganography purposes. They provide the necessary tools to implement steganography algorithms that embed secret information within images without significantly altering their visual appearance.

5. Cryptography Libraries: Python offers various cryptography libraries, such as cryptography and PyCrypto, which provide encryption and decryption algorithms. These libraries are essential for ensuring the security and confidentiality of the hidden information in steganography applications.

The cryptography libraries in Python offer a wide range of encryption techniques, including symmetric key algorithms like Advanced Encryption Standard (AES) and asymmetric key algorithms like RSA. These algorithms allow for secure encryption of the hidden data before embedding it into the image. The encryption keys play a vital role in protecting the confidentiality and integrity of the hidden information.

By incorporating cryptography libraries into image steganography projects, developers can ensure that the hidden data remains secure and accessible only to authorized parties.

6. Steganography Libraries: Python also has specific steganography libraries, such as Stegano and Steganography, which provide pre-built functions and utilities for hiding data within images. These libraries simplify the process of embedding and extracting secret information, reducing the complexity of implementation.

Steganography libraries in Python offer various steganography techniques, including Least Significant Bit (LSB) insertion, phase encoding, and frequency domain methods. They provide convenient functions to embed secret messages or files into the image's pixel values, as well as extract the hidden information from the steganographic image.

These libraries save developers from having to implement steganography algorithms from scratch, as they provide ready-to-use functions and methods for seamless integration into image steganography projects.

By utilizing Python, Spyder IDE, and the Tkinter library, developers can leverage the extensive capabilities of these technologies to implement efficient and user-friendly image steganography applications. The combination of these tools simplifies the development process, enhances the functionality, and improves the overall performance of image steganography systems.

2.2 : LANGUAGE USED IN THE PROJECT

Python is a widely-used programming language known for its versatility, readability, and ease of use. It was created by Guido van Rossum and first released in 1991. Since then, Python has gained immense popularity and has become one of the most widely adopted programming languages across various domains, from web development and data science to artificial intelligence and automation.

Key Features and Benefits of Python:

1. **Simple and Readable Syntax**: Python's syntax is designed to be clean, clear, and easy to understand. Its minimalistic and readable code structure makes it an excellent choice for beginners and experienced programmers alike. Python's emphasis on code readability enhances collaboration and maintainability, as it allows developers to express complex ideas in a concise and straightforward manner.
2. **Large Standard Library**: Python comes with a comprehensive standard library that provides a wide range of modules and functions for various tasks. This extensive library simplifies development by offering pre-built solutions for common programming tasks, such as file handling, network communication, data manipulation, and more. The standard library's rich collection of modules eliminates the need for extensive external dependencies, making Python self-sufficient for many applications.
3. **Cross-Platform Compatibility**: Python is a platform-independent language, meaning code written in Python can run on multiple operating systems, including Windows, macOS, and Linux, with little or no modifications. This cross-platform compatibility ensures that Python applications can be easily

deployed and executed on different environments, making it highly versatile and flexible.

4. **Extensive Third-Party Libraries**: In addition to its standard library, Python boasts a vast ecosystem of third-party libraries and packages. These libraries, such as NumPy, Pandas, TensorFlow, Django, and Flask, provide specialized functionalities and tools for specific domains. These powerful libraries enable developers to leverage existing solutions, accelerate development timelines, and create complex applications with ease.

5. **Strong Community Support**: Python has a vibrant and supportive community of developers and enthusiasts. This active community contributes to the continuous growth and improvement of the language by creating open-source libraries, sharing knowledge, and providing assistance through forums and online communities. The wealth of resources available, including documentation, tutorials, and code examples, makes learning Python and solving programming challenges more accessible.

6. **Versatility and Application Range**: Python's versatility enables it to be used in various domains and applications. From web development and data analysis to machine learning and scientific computing, Python has a wide range of applications. Its flexibility and compatibility with other languages allow for seamless integration with existing systems and technologies.

In conclusion, Python's simplicity, readability, extensive library support, cross-platform compatibility, strong community, and versatility make it an exceptional programming language for both beginners and experienced developers. Its popularity continues to grow, driven by its power, flexibility, and the vast ecosystem of libraries and tools available. Whether you're a beginner looking to learn programming or a seasoned developer tackling complex projects, Python provides a solid foundation and empowers you to build a wide range of applications efficiently and effectively.

Use of Python in image steganography:

1. **Image Processing Libraries:** Python provides powerful image processing libraries like PIL (Python Imaging Library) and Pillow. These libraries offer a wide range of functions and algorithms for working with images. Developers can load and save images in various formats, resize and crop images, apply filters for enhancing or modifying images, and extract pixel-level data for manipulation. These capabilities are crucial for performing steganographic operations, such as embedding and extracting hidden data in the pixel values of images.

2. **Cryptography Libraries:** Python has robust cryptography libraries, including Cryptography and PyCrypto. These libraries offer a comprehensive set of cryptographic functions and algorithms for secure data encryption and decryption. In the context of image steganography, these libraries are used to encrypt the hidden data before embedding it into the image and decrypt it during the extraction process. Encryption ensures the confidentiality and integrity of the hidden information, preventing unauthorized access or tampering.

3. **Ease of Use:** Python's simplicity and readability make it an ideal programming language for image steganography. Its clean and intuitive syntax allows developers to write clear and concise code, making it easier to implement steganographic algorithms. Python's syntax is closer to human language, enabling developers to express their ideas and logic in a more natural way. Additionally, Python's extensive documentation and large community support provide valuable resources and assistance to developers working on image steganography projects.

4. **Availability of Steganography Libraries:** Python has dedicated steganography libraries, such as Stegano, that provide higher-level functionalities for hiding and extracting data from images. These libraries abstract the low-level implementation details of steganographic algorithms, making it easier for developers to integrate steganography techniques into their projects without having to understand the complex image processing and encryption algorithms involved. The availability of these libraries saves

development time and effort and allows developers to focus on the higher-level aspects of their steganography applications.

5. **Cross-platform Compatibility:** Python is a cross-platform programming language, meaning steganography applications developed in Python can run on different operating systems, including Windows, macOS, and Linux. This cross-platform compatibility ensures that steganography tools built with Python can reach a wider audience and be used on various devices and platforms. It eliminates the need for developing separate versions of the tool for different operating systems, making deployment and distribution more convenient.
6. **GUI Development:** Python offers several GUI frameworks, such as Tkinter, PyQt, and wxPython, which allow developers to create user-friendly graphical user interfaces for image steganography applications. These frameworks provide a set of tools and components for building interactive interfaces where users can select images, input data or messages to be hidden, and view the results. GUI development in Python enables a more intuitive and user-friendly experience, making it easier for non-technical users to utilize the steganography tool effectively.
7. **Steganography Libraries:** Python has dedicated steganography libraries like Stegano that simplify the implementation of steganographic techniques. These libraries provide higher-level functions and algorithms for hiding and extracting data from images. Developers can leverage these libraries to avoid low-level implementation details and focus on the core steganography logic. It saves time and effort, making the development process more efficient.

In summary, Python's extensive image processing libraries, cryptography support, ease of use, availability of dedicated steganography libraries, cross-platform compatibility, and GUI development capabilities make it an excellent choice for implementing image steganography algorithms and developing user-friendly steganography applications. Python's combination of features and resources empowers developers to create efficient and secure image steganography tools with a smooth user experience.

2.3 : IDE used in this Project

Spyder is an open-source integrated development environment (IDE) designed specifically for scientific computing and data analysis using the Python programming language. It provides a comprehensive set of tools and features that make it well-suited for working with data and conducting scientific research.

Here are some key features and characteristics of Spyder IDE:

1. Editor: Spyder offers a powerful code editor with features like syntax highlighting, code completion, code linting, and intelligent code introspection. It provides a user-friendly interface for writing, editing, and managing Python scripts.

2. Interactive Console: Spyder includes an interactive Python console that allows you to execute code snippets, experiment with algorithms, and quickly evaluate expressions or variables. The console supports features like command history, code profiling, debugging, and variable inspection.

3. Variable Explorer: Spyder provides a dedicated Variable Explorer window that displays the current state of variables, arrays, and data structures. It allows you to explore and manipulate variables during runtime, making it easy to inspect and analyze data.

4. Data Visualization: Spyder integrates with popular data visualization libraries like Matplotlib and NumPy, allowing you to create interactive plots, charts, and graphs directly from your code. It provides a flexible plotting pane that enables interactive exploration and customization of visualizations.

5. Integrated Documentation: Spyder offers built-in access to Python's documentation, making it convenient to explore the details and usage of Python modules, functions, and classes. The documentation is accessible through a help pane within the IDE.

6. Project Management: Spyder supports the organization of your code into projects, allowing you to group related scripts, data files, and resources together. It provides a project explorer that helps you navigate and manage your project's structure.

7. Plugin System: Spyder is highly extensible through its plugin system. It allows you to enhance the functionality of the IDE by installing additional plugins for specific tasks, such as version control integration, data analysis packages, and specialized tools.

8. Integrated Debugger: Spyder includes a built-in debugger that enables you to step through your code, set breakpoints, inspect variables, and analyze the flow of execution. It helps you identify and resolve errors or issues in your Python programs.

9. Cross-Platform Compatibility: Spyder is available for Windows, macOS, and Linux, ensuring compatibility across different operating systems. It provides a consistent development environment regardless of the platform you are working on.

CHAPTER 03

REQUIREMENTS AND ANALYSIS

CHAPTER 03 : REQUIREMENTS AND ANALYSIS

3.1 :REQUIREMENT AND ANALYSIS

1. Introduction:

- Provide a detailed explanation of image steganography and the LSB technique.

- Discuss the significance and applications of image steganography in various domains, such as data security, privacy protection, and covert communication.

- Explain the advantages and challenges associated with the LSB technique, including its simplicity, capacity, and vulnerability to steganalysis.

- Highlight the motivation behind implementing image steganography using the LSB technique.

2. Functional Requirements:

- **Image Selection**: Specify the methods for image selection, such as allowing users to browse and choose images from their local storage or providing predefined image options.

- **Data Embedding**: Describe the process of embedding secret data into the selected image using the LSB technique, including how the bits are replaced and the order in which the pixels are modified.

- **Data Extraction**: Explain the steps involved in extracting hidden data from steganographic images using the LSB technique, ensuring the integrity and accuracy of the extracted data.

- **Image Visualization**: Discuss how the system should present the original image and the steganographic image side by side for comparison, emphasizing the importance of maintaining visual quality.

- **Encryption**: If applicable, elaborate on the encryption algorithms that can be integrated into the system to enhance the security of the hidden data, such as symmetric or asymmetric encryption methods.

3. Non-Functional Requirements:

- **Performance:** Specify the desired performance metrics, such as embedding and extraction speed, to ensure the system can handle real-time scenarios without significant delays.

- **Capacity**: Determine the maximum amount of data that can be embedded without causing noticeable degradation in image quality, considering factors like image resolution, color depth, and compression.

- **Security**: Discuss the measures to be taken to ensure the confidentiality and integrity of the hidden data, such as applying encryption to the data before embedding and implementing anti-steganalysis techniques.

- **User Interface:** Describe the desired characteristics of the user interface, including its layout, ease of navigation, and informative feedback to guide users during the embedding and extraction processes.

- **Compatibility:** Identify the image file formats that the system should support, such as JPEG and PNG, and ensure compatibility with various operating systems and platforms.

- **Robustness**: Address the system's ability to handle different image characteristics, such as images with varying color depths, resolutions, and compression types, while maintaining accurate data embedding and extraction.

4. Analysis:

- **Image Formats**: Conduct a detailed analysis of popular image formats, considering factors such as image quality, compression techniques, and compatibility with the LSB technique. Evaluate the suitability of each format for steganographic purposes.

- **Capacity Analysis**: Determine the maximum amount of data that can be embedded in an image while ensuring minimal visual distortion. Consider factors like the number of LSBs used for embedding, pixel value ranges, and human visual perception.

- **Security Analysis:** Evaluate the vulnerabilities of the LSB technique to steganalysis attacks, such as statistical analysis and visual detection methods. Explore countermeasures to mitigate these vulnerabilities, such as adding noise or applying encryption to the embedded data.

- **Performance Analysis:** Measure the performance of the embedding and extraction processes in terms of speed and resource utilization. Analyze the computational complexity of the algorithms used and identify any potential bottlenecks or optimizations.

- **User Experience Analysis**: Conduct usability tests and gather feedback from users to evaluate the effectiveness of the system's interface. Identify areas for improvement, such as clarity of instructions, visual cues, and error handling, to enhance the overall user experience.

5. Conclusion:

- Summarize the key findings and insights from the requirements and analysis phase.

- Emphasize the importance of addressing both functional and non-functional requirements to ensure the successful implementation of the image steganography system.

- Highlight the significance of the

analysis, such as understanding image formats, capacity limitations, security vulnerabilities, performance considerations, and user experience.

- Provide a brief outlook on how the requirements and analysis will inform the subsequent stages of system design, development, and testing, ensuring a robust and effective image steganography solution.

3.2 : SYSTEM REQUIREMENTS

1. **Hardware Requirements:**

- **Processor:** The steganography system requires a capable processor to handle the computational tasks involved in embedding and extracting data from images. A higher-performance processor will result in faster processing times.
- **Memory:** Adequate RAM is essential for efficient image processing and encryption/decryption operations. Sufficient memory ensures that the system can handle large image files and perform complex algorithms without performance degradation.
- **Storage:** Sufficient storage space is needed to store the steganography application, images, and any associated data. This includes both the capacity to store the application itself and the available space for storing the resulting steganographic images.
- **Display:** A high-resolution display is desirable to accurately visualize images and user interfaces. It allows users to view the images clearly and examine the embedded data effectively.
- **Input Devices:** Standard input devices such as a keyboard, mouse, or touch-enabled devices are necessary for user input. They enable users to interact with the steganography system, input data, and initiate actions.

2. **Software Requirements:**

- **Operating System:** The steganography application should be compatible with the targeted operating system(s), such as Windows, macOS, or Linux. This ensures that users can run the application smoothly on their preferred platforms.
- **Programming Languages:** The choice of programming language(s) for developing the steganography system will depend on factors such as developer expertise, performance requirements, and availability of libraries and frameworks.

- **Libraries and Frameworks:** Depending on the selected programming language, specific libraries and frameworks may be required for image processing, encryption/decryption, and graphical user interface development. For example, OpenCV or PIL/Pillow can be used for image manipulation tasks.
- **Image Processing Libraries:** Image processing libraries are essential for loading, saving, and modifying pixel values of images. These libraries provide functions for image manipulation, such as resizing, cropping, and applying filters.
- **Encryption Algorithms:** The steganography system may require encryption algorithms to secure the hidden data within images. Common encryption algorithms include AES (Advanced Encryption Standard), RSA (Rivest-Shamir-Adleman), or their variants.
- **Graphical User Interface (GUI):** If the steganography system includes a graphical user interface, a GUI framework or toolkit is needed for creating a user-friendly interface. Examples of GUI frameworks are Tkinter for Python, PyQt for Qt-based applications, or JavaFX for Java-based applications.

3. **Security Requirements**:

- **Encryption Techniques:** The steganography system should support strong encryption techniques to ensure the confidentiality of the hidden data. Robust encryption algorithms with appropriate key sizes should be employed to make it computationally difficult for unauthorized users to decipher the hidden information.
- **Key Management:** The system may require secure key management practices. This includes generating cryptographic keys, securely storing and distributing them, and ensuring that only authorized users have access to the keys.
- **Secure Data Handling:** The steganography application should handle sensitive data securely. It should prevent unauthorized access to the embedded data, protect against data leakage, and implement secure protocols for data transmission if applicable.

4. **Performance Requirements:**

- **Processing Speed:** The steganography system should aim to provide efficient and timely embedding and extraction of data. Fast processing times enhance the user experience and reduce the time required to perform steganographic operations.
- **Resource Utilization:** The application should utilize system resources, such as CPU and memory, optimally. Efficient resource utilization ensures that the system operates smoothly without causing excessive resource consumption or slowing down other concurrent processes.

5. **Compatibility Requirements:**

- **Image Formats**: The steganography system should support a variety of popular image formats, including JPEG, PNG to accommodate a wide range of images.
- **Cross-Platform Compatibility:** If the application is intended to run on multiple operating systems, it should be designed and developed to ensure cross-platform compatibility.

6 . User Interface Requirements :

- **Intuitive and User-Friendly:** The user interface should be intuitive, easy to navigate, and provide clear instructions for embedding and extracting data from images.
- **Visual Feedback:** The system should provide visual feedback during the embedding and extraction processes to indicate progress and completion.
- These system requirements serve as a foundation for developing an image steganography system that is efficient, secure, and user-friendly. They ensure that the hardware and software components are appropriately configured to support the desired functionality and performance of the application.

CHAPTER 04

DESIGN AND METHODOLOGY

CHAPTER 04 : DESIGN METHODOLOGY

4.1 DESIGN METHODOLOGY

IMAGE STEGANOGRAPHY

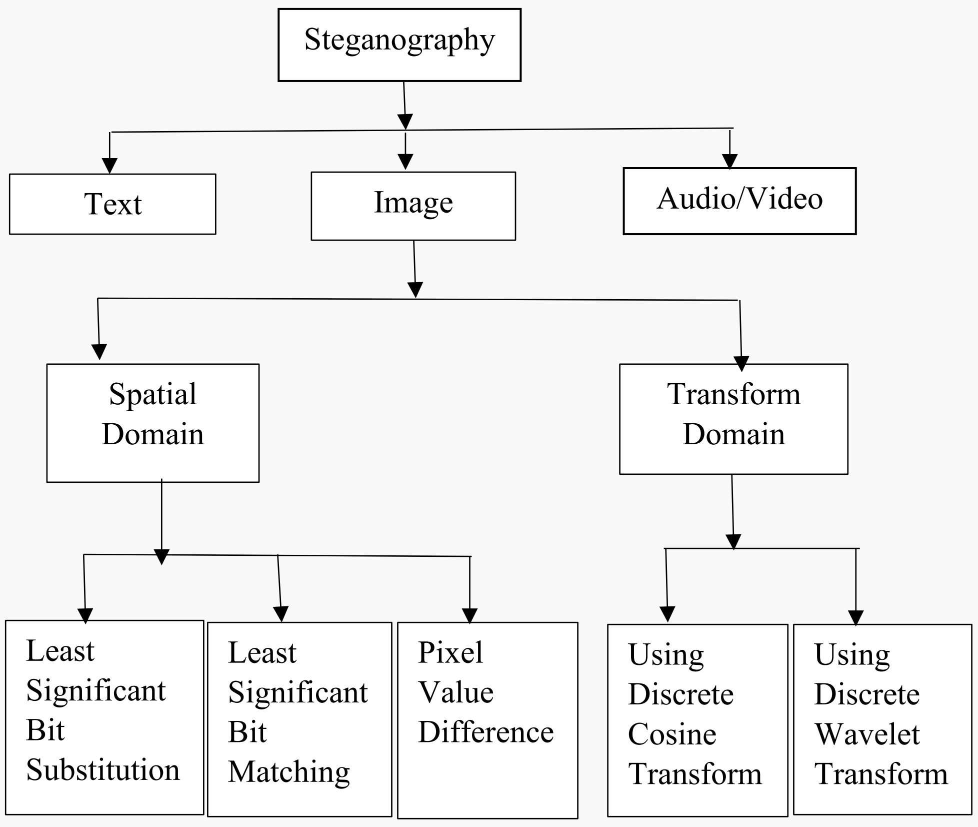

Steganography
Text
Image
Audio/Video
Spatial Domain
Transform Domain
Least Significant Bit Substitution
Least Significant Bit Matching
Pixel Value Difference
Using Discrete Cosine Transform
Using Discrete Wavelet Transform

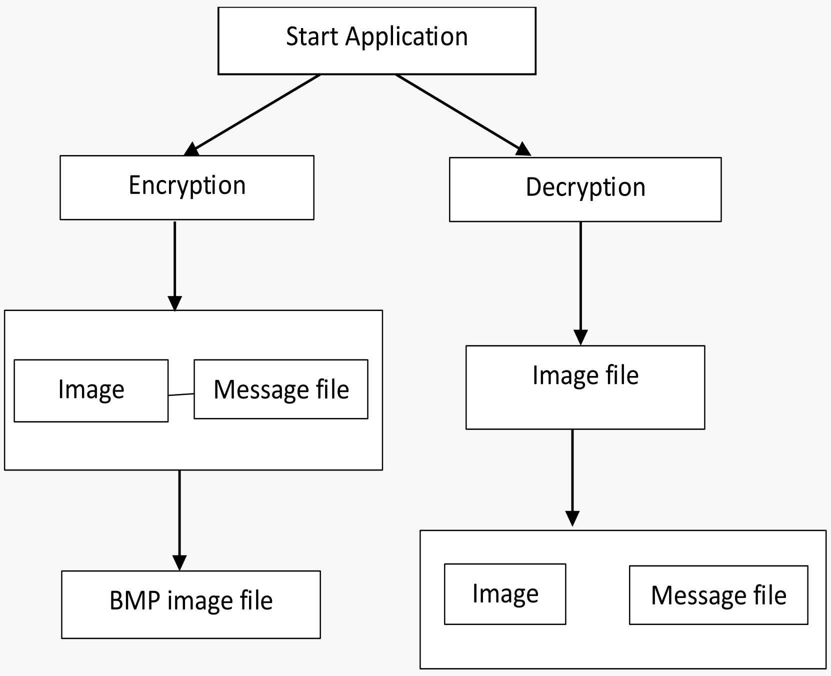
Image steganography is the process of concealing data inside an image in a way that the naked eye cannot see. Secret messages have long been concealed in works of art, such as paintings and sculptures, using this method. Image steganography has grown in popularity as a secure communication and data-hiding method in the digital era.

Image steganography has numerous applications in domains such as military, law enforcement, media, and personal communication. It is used to communicate sensitive information without being detected and can take the shape of email attachments, social media posts, and digital photographs. It can also be used to embed watermarks in digital photos to restrict unauthorised use.

Because of growing concerns about data security and privacy, the use of image steganography has increased significantly in recent years. With the advent of digital communication and storage, the importance of secure data transit has grown. Image steganography is one of the techniques used to create secure communication by concealing messages.

This study attempts to provide an overview of image steganography, its applications, and the various image steganography approaches. The needs and analysis, design technique, and conclusion of the image steganography process will also be included in the article.

- **FLOWCHART OF IMAGE STEGANOGRAPHY**



In this project I have used one of the Spatial Domain Methods i.e. Least Significant Bit (LSB) among : (i) Pixel Value Differencing (PVD)

(ii) Least Significant Bit (LSB)

(iii) Edges Based data embedding method (EBE)

(iv) Random Pixel embedding method (RPE)

➢ LEAST SIGNIFICANT BIT (LSB)

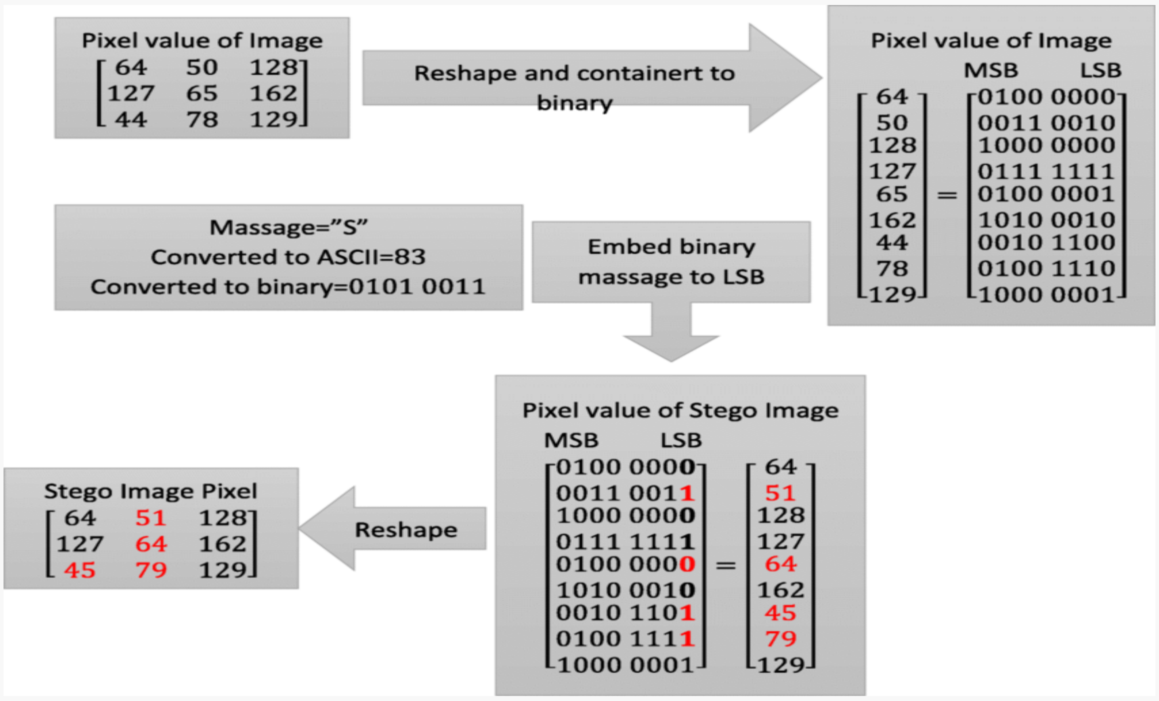
Least significant bit (LSB) method is a common, simple approach to embedding information in a cover file.

In steganography, LSB substitution method is used. I.e. since every image has three components (RGB). This pixel information is stored in encoded format in one byte. The first bits containing this information for every pixel can be modified to store the hidden text. For this, the preliminary condition is that the text to be stored has to be smaller or of equal size to the image used to hide the text.

LSB based method is a spatial domain method. But this is vulnerable to cropping and noise. In this method, the MSB (most significant bits) of the message image to be hidden are stored in the LSB (least significant bits) of the image used as the cover image.

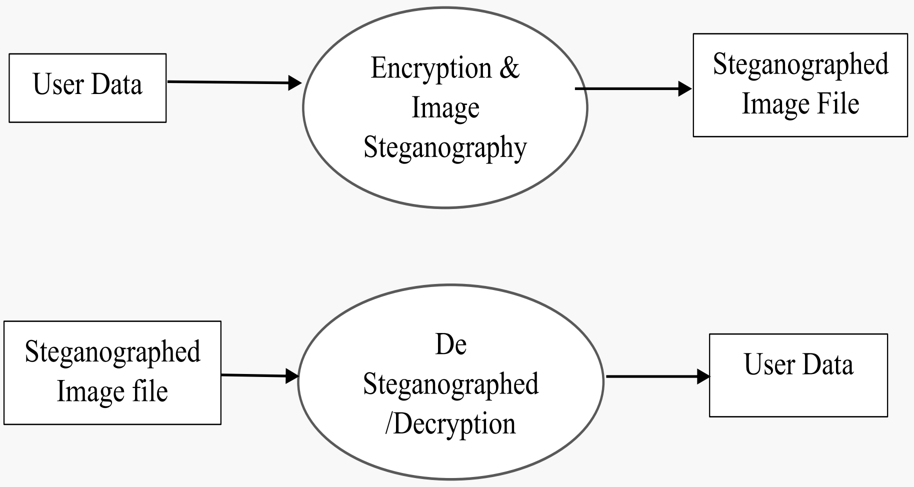
It is known that the pixels in an image are stored in the form of bits. In a grayscale image, the intensity of each pixel is stored in 8 bits (1byte). Similarly for a colour (RGB-red, green, blue) image, each pixel requires 24 bits (8bits for each layer).

The Human visual system (HVS) cannot detect changes in the colour or intensity of a color when the LSB bit is modified. This is psycho-visual redundancy since this can be used as an advantage to store information in these bits and yet notice no major difference in the image.



- **Data Flow Diagram**

LEVEL 0



User Data
Encryption & Image Steganography
Steganographed Image File
Steganographed Image file
De Steganographed /Decryption
User Data

Fig . Level 0 Data Flow Diagram

The level 0 of DFD specifies the user data that is encrypted and then the cipher text is hiding into the image file using steganography. The output obtained is steganographed image file. Using steganographic algorithm the encrypted text is separated from the image file. Using decryption the cipher text is converted to user data.

LEVEL 1

The level 0 of DFD specifies the user data that is encrypted and then the cipher text is hiding into the image file using steganography. The output obtained is steganographed image file. Using steganographic algorithm the encrypted text is separated from the image file. Using decryption the cipher text is converted to user data.

The level 1 of DFD specifies the user data which is encrypted using secret key and then the cipher text is hiding into the image file using steganography with the help of secret key.. Thus the image file with encrypted data is stored in a file. The image file is steganographed with the help of secret key. Thus the encrypted text is separated from image file. Using decryption cipher text is converted to plain text using secret key.

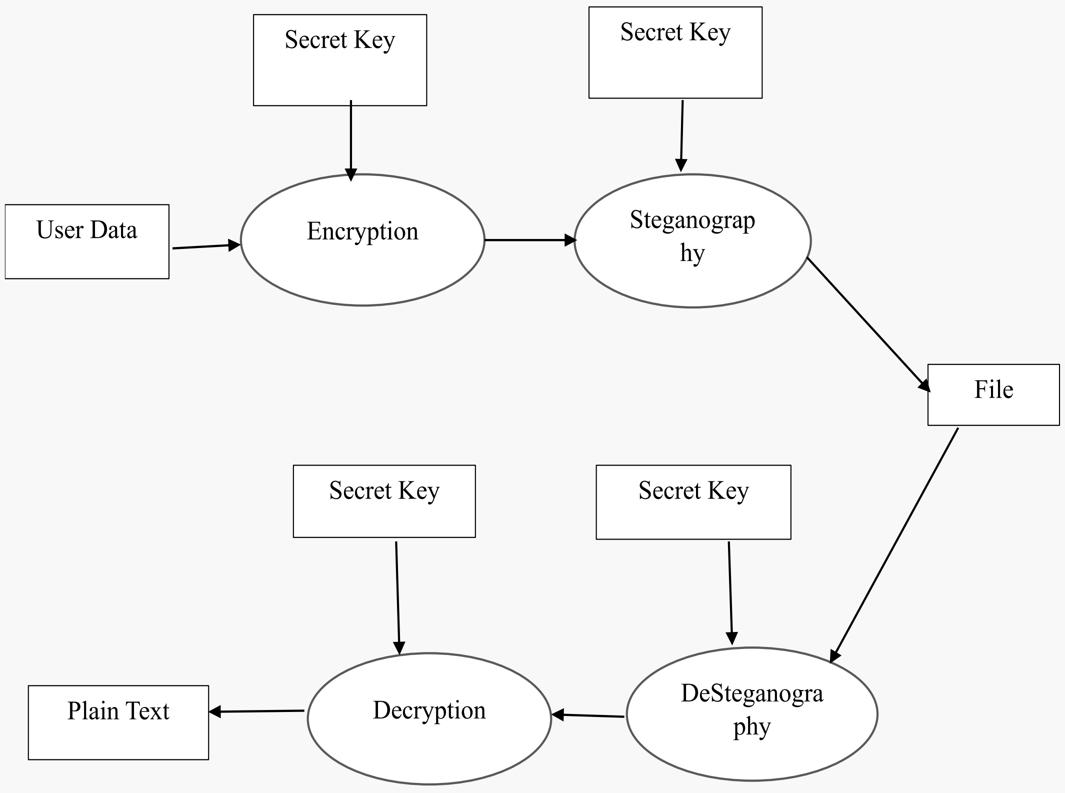


Fig . Level 1 Data Flow Diagram

CHAPTER 05

CODING AND SCREENSHOT

5.1 CODING AND SCREENSHOTS

CODING PERFORMED USING PYTHON IN SPYDER IDE

```
from tkinter import *

from tkinter import messagebox

from argparse import FileType

#from stegano import lsb

from tkinter.filedialog import *

from PIL import ImageTk,Image

#from  stegano.lsbset import generators

from stegano import lsb

from tkinter import font as tkFont

from stegano import exifHeader as aaa

import os

from subprocess import Popen


def encode():

    main.destroy()

    enc=Tk()

    #encode.geometry('600x220')

    #enc.attributes("-fullscreen", True)
```

```
enc.wm_attributes('-transparentcolor')

img=ImageTk.PhotoImage(Image.open("Encode.png"))

fontl = tkFont.Font(family='Algerian', size=32)

label1=Label(enc,image=img)

label1.pack()



LabelTitle=Label(text="ENCODE",bg="Blue",fg="white",width=15)

LabelTitle['font']=fontl

LabelTitle.place(relx=0.6, rely=0.1)



def openfile():

    global fileopen

    global imagee



    fileopen=StringVar()

    fileopen=askopenfilename(initialdir="/Desktop",title="select
file",filetypes=(("jpeg,png files","*jpg *png"),("all files","*.*")))

    imagee=ImageTk.PhotoImage(Image.open(fileopen))



    Labelpath=Label(text=fileopen)

    Labelpath.place(relx=0.6, rely=0.25, height=21, width=450)
    Labelimg=Label(image=imagee)

    Labelimg.place(relx=0.7, rely=0.3, height=200, width=200)


Button2 = Button(text="Openfile",command=openfile)

Button2.place(relx=0.7, rely=0.2, height=31, width=94)


secimg=StringVar()

radio1=Radiobutton(text='jpeg',value='jpeg',variable=secimg)

radio1.place(relx=0.7, rely=0.57)


radio2=Radiobutton(text='png',value='png',variable=secimg)

radio2.place(relx=0.8, rely=0.57)


Label1 =Label(text="Enter message")

Label1.place(relx=0.6, rely=0.6, height=21, width=104)

entrysecmes=Entry()

entrysecmes.place(relx=0.7, rely=0.6, relheight=0.05, relwidth=0.200)


Label2 =Label(text="File Name")

Label2.place(relx=0.6, rely=0.70, height=21, width=104)


entrysave=Entry()
```

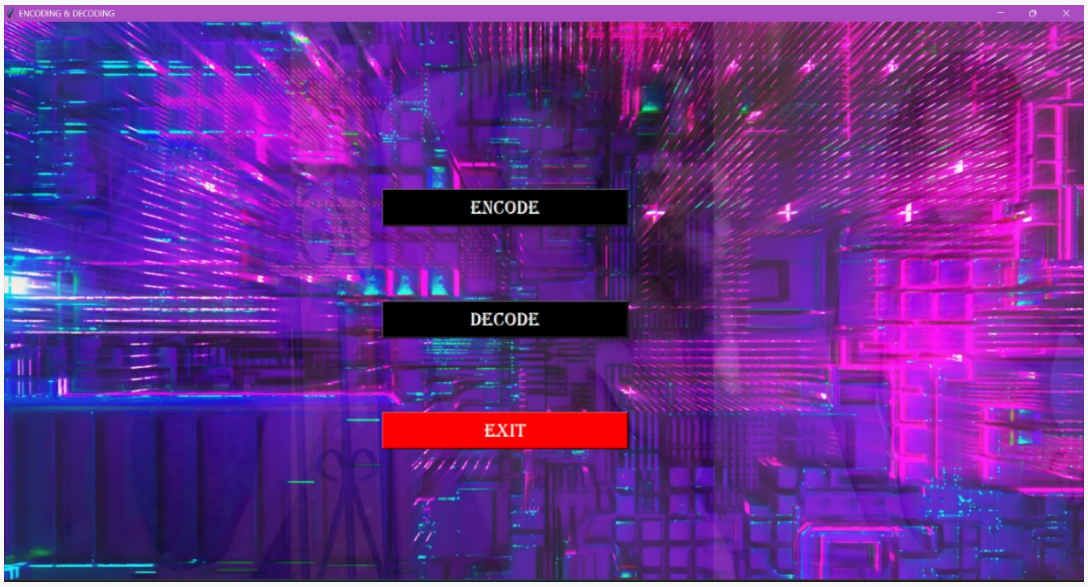
```
    entrysave.place(relx=0.7, rely=0.70, relheight=0.05, relwidth=0.200)

    def encode():

        if secimg.get()=="jpeg":

            inimage=fileopen

            response=messagebox.askyesno("popup","do you want to encode")

            if response==1:

                aaa.hide(inimage,entrysave.get()+'.jpg',entrysecmes.get())

                messagebox.showinfo("popup","successfully
encode"+entrysave.get()+".jpeg")


            else:

                messagebox.showwarning("popup","Unsuccessful")



        if secimg.get()=="png":

            inimage=fileopen

            response=messagebox.askyesno("popup","Do you want to encode ?")

            if response==1:

                lsb.hide(inimage,message=entrysecmes.get()).save(entrysave.get()+'
.png')

                messagebox.showinfo("popup","successfully encode to
"+entrysave.get()+".png")


            else:
```

```
            messagebox.showwarning("popup","Unsuccessful")


    def back():
        enc.destroy()
        #execfile('image steganography using lsb.py')
        #os.system('python imagesteganographyusinglsb.py')
        Popen('python steganography.py')


    Button2 = Button(text="ENCODE",command=encode)
    Button2.place(relx=0.7, rely=0.8, height=31, width=94)


    Buttonback = Button(text="Back",command=back)
    Buttonback.place(relx=0.7, rely=0.85, height=31, width=94)


    enc.mainloop()



def decode():
    main.destroy()
    dec=Tk()
    #dec.attributes("-fullscreen", True)
    dec.wm_attributes('-transparentcolor')
```

```
img=ImageTk.PhotoImage(Image.open("Decode.jpg"))

fontl = tkFont.Font(family='Algerian', size=32)

label1=Label(dec,image=img)

label1.pack()



LabelTitle=Label(text="DECODE",bg="blue",fg="white",width=15)

LabelTitle['font']=fontl

LabelTitle.place(relx=0.6, rely=0.1)



secimg=StringVar()

radio1=Radiobutton(text='jpeg',value='jpeg',variable=secimg)

radio1.place(relx=0.7, rely=0.57)




radio2=Radiobutton(text='png',value='png',variable=secimg)

radio2.place(relx=0.8, rely=0.57)




def openfile():

    global fileopen

    global imagee
```

```
        fileopen=StringVar()

        fileopen=askopenfilename(initialdir="/Desktop",title="select file",filetypes=(("jpeg files, png file","*jpg *png"),("all files","*.*")))


        imagee=ImageTk.PhotoImage(Image.open(fileopen))

        Labelpath=Label(text=fileopen)

        Labelpath.place(relx=0.6, rely=0.25, height=21, width=450)


        Labelimg=Label(image=imagee)

        Labelimg.place(relx=0.7, rely=0.3, height=200, width=200)




    def deimg():

        if secimg.get()=="png":

            messag=lsb.reveal(fileopen)


        if secimg.get()=="jpeg":

            messag=aaa.reveal(fileopen)


        Label2=Label(text=messag)

        Label2.place(relx=0.7, rely=0.7, height=21, width=204)
    Button2 = Button(text="Openfile",command=openfile)

    Button2.place(relx=0.7, rely=0.2, height=31, width=94)


    Button2 = Button(text="DECODE",command=deimg)

    Button2.place(relx=0.7, rely=0.8, height=31, width=94)


    def back():

        dec.destroy()

        #execfile('image steganography using lsb.py')

        #os.system('python imagesteganographyusinglsb.py')

        Popen('python steganography.py')


    Buttonback = Button(text="Back",command=back)

    Buttonback.place(relx=0.7, rely=0.85, height=31, width=94)


    dec.mainloop()

#main program

main=Tk()

main.title('ENCODING & DECODING ')

main.geometry("1000x750")
```

```
#main.attributes("-fullscreen", True)

fontl = tkFont.Font(family='Algerian', size=20)


global image1

image1=ImageTk.PhotoImage(Image.open("home.jpg"))

label=Label(main,text="lalal",image=image1)

label.pack()


encbutton=Button(text='Encode',fg="white",bg="black",width=20,command=
encode)

encbutton['font'] =fontl

encbutton.place(relx=0.35,rely=0.3)


decbutton=Button(text='Decode',fg="white",bg="black",width=20,command=
decode)

decbutton['font'] =fontl

decbutton.place(relx=0.35,rely=0.5)

def exit():

    main.destroy()

closebutton=Button(text='EXIT',fg="white",bg="red",width=20,command=exi
t)

closebutton['font'] =fontl

closebutton.place(relx=0.35,rely=0.7)
```



main.mainloop()

The provided code initializes the main window of the application and sets up the user interface for the encoding and decoding functionality. Let's go through each section:

1. **main=Tk()**: Creates the main window of the application using the Tkinter library.
2. **main.title('ENCODING & DECODING ')**: Sets the title of the main window.
3. **main.geometry("1000x750")**: Sets the size of the main window to 1000 pixels wide and 750 pixels high.
4. **fontl = tkFont.Font(family='Algerian', size=20)**: Defines a font object to be used for the buttons.
5. **image1=ImageTk.PhotoImage(Image.open("home.jpg"))**: Loads an image named "home.jpg" and creates an image object using the **ImageTk.PhotoImage()** function from the PIL library.
6. **label=Label(main,text="lalal",image=image1)**: Creates a label widget that displays the image as the background of the main window.
7. **label.pack()**: Places the label widget in the main window.

8. **encbutton=Button(text='Encode',fg="white",bg="black",width=20,command=encode)**: Creates a button labeled "Encode" with white text on a black background. The button is assigned the **encode()** function as the command to be executed when clicked.

9. **encbutton['font'] =fontl**: Sets the font of the "Encode" button using the previously defined **fontl** font object.

10. **encbutton.place(relx=0.35,rely=0.3)**: Places the "Encode" button in the main window at the relative coordinates (0.35, 0.3).

11. **decbutton=Button(text='Decode',fg="white",bg="black",width=20,command=decode)**: Creates a button labeled "Decode" with white text on a black background. The button is assigned the **decode()** function as the command to be executed when clicked.

12. **decbutton['font'] =fontl**: Sets the font of the "Decode" button using the previously defined **fontl** font object.

13. **decbutton.place(relx=0.35,rely=0.5)**: Places the "Decode" button in the main window at the relative coordinates (0.35, 0.5).

14. **closebutton=Button(text='EXIT',fg="white",bg="red",width=20,command=exit)**: Creates a button labeled "EXIT" with white text on a red background. The button is assigned the **exit()** function as the command to be executed when clicked.

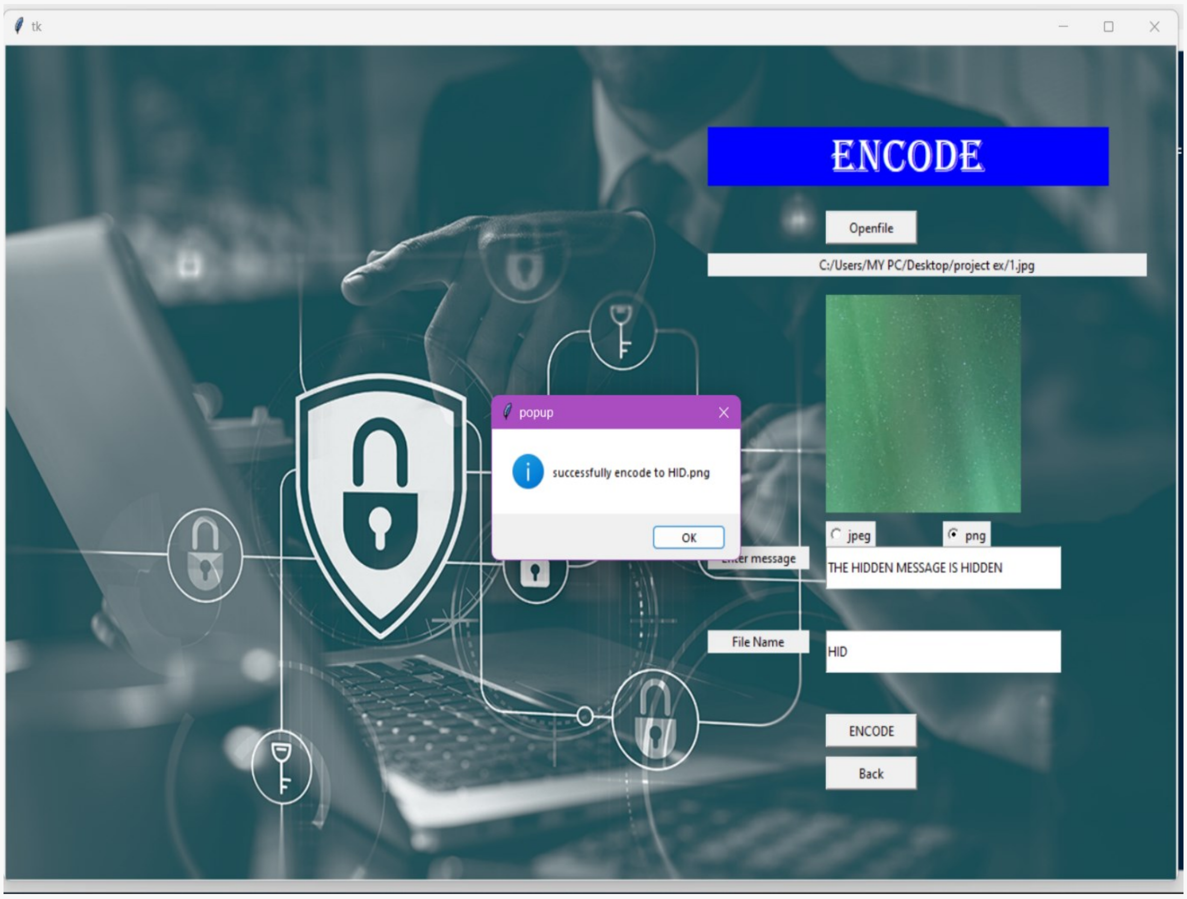
15. **closebutton['font'] =fontl**: Sets the font of the "EXIT" button using the previously defined **fontl** font object.

16. **closebutton.place(relx=0.35,rely=0.7)**: Places the "EXIT" button in the main window at the relative coordinates (0.35, 0.7).

17. **main.mainloop()**: Starts the event loop of the main window, which keeps the application running until it is closed by the user.

In summary, the code sets up a graphical user interface with a main window displaying an image as the background. It provides three buttons: "Encode" and

"Decode" for executing the corresponding functions, and an "EXIT" button to close the application.



C:/Users/MY PC/Desktop/project ex/1.jpg

successfully encode to HID.png

jpeg [✓ png]

THE HIDDEN MESSAGE IS HIDDEN

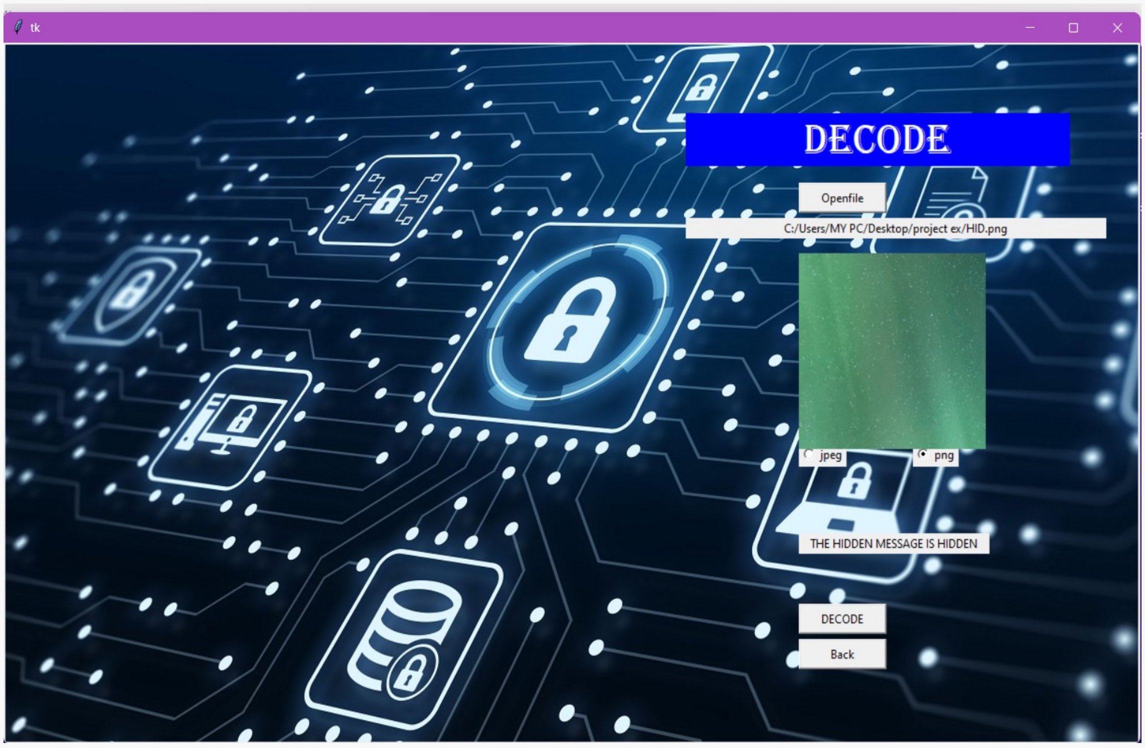
HID

The encode() function in the provided code performs the following actions :

1. It destroys the main window (**main.destroy()**) to switch to a new window for the encoding functionality.
2. It creates a new Tkinter window (**enc=Tk()**) to display the encoding interface.
3. It sets the window attributes (**enc.wm_attributes('-transparentcolor')**) to make the window transparent.
4. It loads an image (**img=ImageTk.PhotoImage(Image.open("Encode.png"))**) and creates a label to display the image (**label1=Label(enc,image=img)**).
5. It creates a label for the title of the encoding interface (**LabelTitle=Label(text="ENCODE",bg="Blue",fg="white",width=15)**).

6. It defines the font for the title label (**fontl = tkFont.Font(family='Algerian', size=32)**).
7. It packs the title label and image label into the window using the **pack()** method.
8. It defines a function **openfile()** that allows the user to select an image file to be encoded.
9. It creates a button (**Button2 = Button(text="Openfile",command=openfile)**) that triggers the **openfile()** function.
10. It creates two radio buttons (**radio1** and **radio2**) for selecting the image format (JPEG or PNG).
11. It creates labels and entry fields for entering the secret message and the desired file name for the encoded image.
12. It defines another function **encode()** that is triggered when the "ENCODE" button is clicked.
13. Inside the **encode()** function, it checks the selected image format and performs the encoding operation accordingly using the **lsb.hide()** or **aaa.hide()** functions from the **stegano** module.
14. It displays success or failure messages using **messagebox.showinfo()** and **messagebox.showwarning()** functions.
15. It defines a function **back()** that closes the encoding window and returns to the main program.
16. It creates "ENCODE" and "Back" buttons that trigger the **encode()** and **back()** functions, respectively.
17. It starts the event loop (**enc.mainloop()**) to run the encoding window.

Overall, the **encode()** function sets up a user interface for image encoding, allows the user to select an image file, choose the encoding format, enter a secret message, and specify the output file name. It then performs the encoding operation and displays the result to the user.


tk
DECODE
Openfile
C:/Users/MY PC/Desktop/project ex/HID.png
jpeg
png
THE HIDDEN MESSAGE IS HIDDEN
DECODE
Back

The **decode()** function in the provided code performs the following actions:

1. It destroys the main window (**main.destroy()**) to switch to a new window for the decoding functionality.

2. It creates a new Tkinter window (**dec=Tk()**) to display the decoding interface.

3. It sets the window attributes (**dec.wm_attributes('-transparentcolor')**) to make the window transparent.

4. It loads an image (**img=ImageTk.PhotoImage(Image.open("Decode.jpg"))**) and creates a label to display the image (**label1=Label(dec,image=img)**).

5. It creates a label for the title of the decoding interface (**LabelTitle=Label(text="DECODE",bg="blue",fg="white",width=15)**).

6. It defines the font for the title label (**fontl = tkFont.Font(family='Algerian', size=32)**).

7. It packs the title label and image label into the window using the **pack()** method.

8. It creates two radio buttons (**radio1** and **radio2**) for selecting the image format (JPEG or PNG) to be decoded.

9. It defines a function **openfile()** that allows the user to select an image file to be decoded.

10. It creates a button (**Button2 = Button(text="Openfile",command=openfile)**) that triggers the **openfile()** function.

11. It creates a label and an image label to display the selected image file.

12. It defines a function **deimg()** that is triggered when the "DECODE" button is clicked.

13. Inside the **deimg()** function, it checks the selected image format and performs the decoding operation accordingly using the **lsb.reveal()** or **aaa.reveal()** functions from the **stegano** module.

14. It creates a label (**Label2**) to display the decoded message.

15. It defines a function **back()** that closes the decoding window and returns to the main program.

16. It creates "DECODE" and "Back" buttons that trigger the **deimg()** and **back()** functions, respectively.

17. It starts the event loop (**dec.mainloop()**) to run the decoding window.

    Overall, the **decode()** function sets up a user interface for image decoding, allows the user to select an encoded image file, choose the image format, and performs the decoding operation. It then displays the decoded message to the user.

CHAPTER 06

CONCLUSION

CHAPTER 06 : CONCLUSION

6.1 CONCLUSION

It is observed that through LSB Substitution Steganographic method, the results obtained in data hiding are pretty impressive as it utilizes the simple fact that any image could be broken up to individual bit-planes each consisting of different levels of information. It is to be noted that as discussed earlier, this method is only effective for bitmap images as these involve lossless compression techniques. Also, in this project grey-scale images have been used for demonstration. But this process can also be extended to be used for color images where, bit-plane slicing is to be done individually for the top four bit-planes for each of R, G, B of the message image, which are again to be placed in the R , G , B planes of the cover image , and extraction is done similarly .

It is also important to discuss that though steganography was once undetected, with the various methods currently used, it is not only easy to detect the presence but also retrieving them is easier. For instance, without having to use a software or complex tools for detection, simple methods to observe if an image file has been manipulated are:

1. Size of the image: A Steganographic image has a huge storage size when compared to a regular image of the same dimensions. I.e. if the original image storage size would be few KBs, the Steganographic image could be several MBs in size. This again varies with the resolution and type of image used.
2. Noise in image: A Steganographic image has noise when compared to a regular image .This is the reason why initially little noise is added to the cover image, so that the Steganographic image doesn't appear very noisy when compared to the original cover image. Though this project focusses on LSB and spatial domain steganography, few details about transform domain methods have also been researched, basics of which have been discussed. So through the various articles and theory available, it is observed that transform domain methods perform better in comparison with spatial domain methods.

6.2 FUTURE SCOPE

The field of image steganography holds significant potential for future development and offers a wide scope of applications. Here are some future possibilities and the scope of image steganography:

1. **Advanced Steganographic Techniques**: As computing power increases and new algorithms emerge, there is a scope for developing more sophisticated steganographic techniques. These techniques can involve innovative methods for embedding data within images, such as adaptive embedding strategies, multiple embedding algorithms, and advanced data hiding schemes. The future holds opportunities to explore more robust and secure steganographic algorithms that can withstand advanced attacks.
2. **High-Capacity Steganography:** Currently, image steganography techniques focus on hiding relatively small amounts of data within images. However, future possibilities lie in developing high-capacity steganography methods that can embed large volumes of data without significantly degrading the image quality. This can involve exploring new compression algorithms, adaptive embedding techniques, and optimization strategies to maximize the data capacity while maintaining visual fidelity.
3. **Multimedia Steganography:** Image steganography can be extended to include other multimedia formats such as audio and video. The scope lies in developing techniques that can hide data within these formats, enabling secure and covert communication through various media channels. Multimedia steganography opens up new avenues for applications in areas like digital entertainment, multimedia forensics, and secure multimedia sharing platforms.
4. **Deep Learning and Artificial Intelligence**: The integration of deep learning and artificial intelligence techniques with image steganography holds great potential. Future developments may involve training neural networks to generate steganographic images, improving the efficiency and effectiveness of hiding and extraction processes. This can lead to intelligent steganography systems that can automatically adapt to different image characteristics, optimize embedding strategies, and enhance security against detection.

5. **Countermeasures and Anti-Steganalysis Techniques:** As steganalysis techniques advance, there is a need for robust countermeasures to resist detection. Future possibilities include the development of advanced anti-steganalysis techniques that can evade detection by exploiting vulnerabilities in steganalysis algorithms or employing sophisticated encryption and data hiding methods. This ongoing cat-and-mouse game between steganography and steganalysis will continue to drive innovation and improvements in both fields.

6. **Practical Applications**: Image steganography has a wide range of practical applications across various domains. It can be utilized in areas such as confidential communication, secure data storage, digital forensics, watermarking, copyright protection, and covert messaging. Future developments may involve tailoring steganography techniques to meet specific industry requirements and addressing emerging challenges in sectors like cybersecurity, defense, journalism, and healthcare.

In conclusion, the future of image steganography is filled with exciting possibilities. Advancements in technology, algorithms, and computing power will drive the development of more robust and secure steganographic techniques. The scope of image steganography extends to various domains, offering solutions for secure communication, data protection, multimedia applications, and countermeasures against steganalysis. As digital communication continues to evolve, image steganography will play a crucial role in ensuring privacy, security, and covert data transmission.

❖ BIBLIOGRAPHY

- S. M. Masud Karim , Md. Saifur Rahman , Md. Ismail Hossain “ A New Approach for LSB Based Image Steganography using Secret Key” , International Conference on Computer and Information Technology (ICCIT) – Dec . 2011

- H.C. Wu , N.I Wu , C.S. Tsai and M.S. Hwang , “Image steganographic scheme based on pixel value differencing and LSB replacement method”, IEEE Proceedings on Vision, Image and Signal processing - 2005

- Mamta Juneja 1, Parvinder Singh Sandhu “Designing of Robust Image Steganography Technique Based on LSB Insertion and Encryption” 2009 International Conference on Advances in Recent Technologies in Communication and Computing – Oct - 2009

- Fan L, Gao T and Yang Q, Cao Y , ― “An extended matrix Encoding algorithm for Steganography of high embedding efficiency”, Computer Electrical Engineering,vol.37, No. 6, 2011

- Bassam Jamil Mohd, Saed Abed, Thaier Al- Hayajneh and Sahel Alouneh, ―FPGA Hardware of the LSB Steganography Method,‖ IEEE Transaction on consumer Electronics - 2012.