

FOCUS APP
MINI PROJECT REPORT

Submitted by

Yuven Senthilkumar

(2116220701330)

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

RAJALAKSHMI ENGINEERING COLLEGE

ANNA UNIVERSITY, CHENNAI

MAY 2025



RAJALAKSHMI ENGINEERING COLLEGE, CHENNAI

BONAFIDE CERTIFICATE

Certified that this Project titled **“FOCUS APP”** is the Bonafide work of **“Yuven Senthilkumar (2116220701330)”** who carried out the work under my supervision. Certified further that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE

Dr. P. Kumar., M.E., Ph.D.,

HEAD OF THE DEPARTMENT

Professor

Department of Computer Science

and Engineering,

Rajalakshmi Engineering College,

Chennai - 602 105.

SIGNATURE

Dr.Duraimurugan

SUPERVISOR

Professor

Department of Computer Science

and Engineering,

Rajalakshmi Engineering

College, Chennai-602 105.

Submitted to Project Viva-Voce Examination held on _____

Internal Examiner

External Examiner

ABSTRACT

In the current age of productivity tools and digital well-being, time management is a critical skill for individuals. This mini-project presents a Focus Timer Android Application that enables users to manage their work or study sessions using the Pomodoro Technique. Built using Android Studio and Kotlin, the app offers a simple user interface with essential features: start, pause, and reset controls for a countdown timer. The default timer duration is set to 25 minutes, aiming to improve users' concentration and task completion efficiency.

The core component is a Countdown Timer, which dynamically updates the UI. This app demonstrates basic Android components such as Activity lifecycle, Button listeners, and UI updates using Kotlin. The timer operates offline and requires minimal system resources, making it ideal for everyday use.

ACKNOWLEDGMENT

Initially we thank the Almighty for being with us through every walk of our life and showering his blessings through the endeavor to put forth this report. Our sincere thanks to our Chairman **Mr. S. MEGANATHAN, B.E, F.I.E.**, our Vice Chairman **Mr. ABHAY SHANKAR MEGANATHAN, B.E., M.S.**, and our respected Chairperson **Dr. (Mrs.) THANGAM MEGANATHAN, Ph.D.**, for providing us with the requisite infrastructure and sincere endeavoring in educating us in their premier institution.

Our sincere thanks to **Dr. S.N. MURUGESAN, M.E., Ph.D.**, our beloved Principal for his kind support and facilities provided to complete our work in time. We express our sincere thanks to **Dr. P. KUMAR, M.E., Ph.D.**, Professor and Head of the Department of Computer Science and Engineering for his guidance and encouragement throughout the project work. We convey our sincere and deepest gratitude to our internal guide, **Dr. DURAIMURUGAN** , Professor of the Department of Computer Science and Engineering. Rajalakshmi Engineering College for his valuable guidance throughout the course of the project.

Yuven Senthilkumar

2116220701330

TABLE OF CONTENTS

CHAPTER	TITLE	PAGE NO
1	ABSTRACT ACKNOWLEDGMENT LIST OF TABLES LIST OF FIGURES 1. INTRODUCTION 1.1 GENERAL 1.2 OBJECTIVES 1.3 EXISTING SYSTEM	III
2	LITERATURE SURVEY	5
3	PROPOSED SYSTEM 3.1 GENERAL 3.2 SYSTEM ARCHITECTURE DIAGRAM 3.3 DEVELOPMENT ENVIRONMENT 3.3.1 HARDWARE REQUIREMENTS 3.3.2 SOFTWARE REQUIREMENTS 3.4 DESIGN THE ENTIRE SYSTEM 3.4.1 ACTIVITYYY DIAGRAM	8

	3.4.2 DATA FLOW DIAGRAM 3.5 STATISTICAL ANALYSIS	
4	MODULE DESCRIPTION 4.1 SYSTEM ARCHITECTURE 4.1.1 USER INTERFACE DESIGN 4.1.2 IMAGE ENCRYPTION 4.3 SYSTEM WORKFLOW	15
5	IMPLEMENTATION AND RESULT 5.1 IMPLENTATION 5.2 OUTPUT SCREENSHOTS	18
6	CONCLUSION AND FUTURE ENHANCEMENT 6.1 CONCLUSION 6.2 FUTURE ENHANCEMENT	20
7	REFERENCES	21

LIST OF TABLES

TABLE NO	TITLE	PAGE NO
3.1	HARDWARE REQUIREMENTS	10
3.2	SOFTWARE REQUIREMENTS	10
3.3	COMPARISON OF FEATURES	13

LIST OF FIGURES

FIGURE NO	TITLE	PAGE NO
3.1	SYSTEM ARCHITECTURE	09
3.2	ACTIVITY DIAGRAM	11
3.3	DFD DIAGRAM	12
3.4	COMPARISON GRAPH	14
4.1	SEQUENCE DIAGRAM	15
5.1	ENCRYPTION CODE	18
5.2	APPLICATION	19

CHAPTER 1

INTRODUCTION

1.1 GENERAL

In today's fast-paced digital environment, the ability to manage time effectively has become crucial for maintaining productivity and focus. With the widespread use of smartphones and mobile applications, digital tools for time management have become increasingly accessible. Among these, the Pomodoro Technique—a time management method that uses a timer to break work into intervals, traditionally 25 minutes in length—has gained significant popularity for improving concentration and reducing mental fatigue.

The proposed **Focus Timer Android Application** is developed with the objective of helping users implement this technique conveniently through their mobile devices. This lightweight and intuitive app provides a simple countdown timer that allows users to start, pause, and reset sessions with ease. Unlike complex productivity suites, the Focus Timer app maintains a minimalist design, focusing solely on the core functionality of time tracking without distractions.

Developed using **Kotlin** in **Android Studio**, the application utilizes core Android components such as **countdown Timer**, **Activity lifecycle management**, and **dynamic UI updates**. This makes the app efficient, reliable, and compatible with a wide range of Android devices. Its simplicity and offline functionality make it especially suitable for students, professionals, and anyone seeking to structure their time effectively without relying on an internet connection or third-party services.

.

1.2 OBJECTIVE

- **To design and implement an Android timer application based on the Pomodoro technique.**
- **To apply the Kotlin programming language and Android Studio for mobile app development.**
- **To integrate features such as countdown, pause, and reset using intuitive UI components.**
- **To ensure reliability and simplicity for effective user experience.**

1.3 EXISTING SYSTEM

The existing systems for secure communication primarily rely on conventional cryptographic techniques such as encryption algorithms (e.g., AES, RSA) that convert plain text into unreadable ciphertext. While these methods effectively protect the content of the message, they do not hide the fact that a secret communication is taking place. As a result, encrypted messages can attract the attention of attackers or surveillance systems. In the domain of image steganography, several desktop-based applications and research prototypes have been developed that utilize LSB and other techniques to embed data in images. However, many of these systems have limitations such as Limited portability and accessibility, Lack of user-friendly interfaces, Vulnerability to detection and manipulation, Absence of encryption integration

CHAPTER 2

LITERATURE SURVEY

The field of steganography has evolved significantly over the past few decades, particularly with advancements in digital image processing and mobile computing. This literature survey reviews key studies and techniques that have shaped modern image steganography, focusing on Least Significant Bit (LSB) methods, algorithm enhancements, and applications relevant to mobile platforms.

Johnson and Jajodia (1998) [1] provided one of the earliest and most comprehensive introductions to digital steganography. Their study discussed the fundamental concepts of hiding information within digital media and introduced various steganographic schemes. The paper emphasized that unlike cryptography, which focuses on obscuring the content of a message, steganography conceals the existence of the message itself. This foundational work inspired numerous researchers to explore digital image steganography techniques, particularly the LSB method due to its simplicity and effectiveness.

Provos and Honeyman (2003) [3] elaborated on steganography techniques with a specific focus on image-based methods. They discussed the practicality of LSB embedding, where the least significant bits of pixel values are altered to encode hidden data. Their work also introduced the concept of steganalysis—the detection of steganographic content—and the importance of designing robust algorithms resistant to such analysis. Their contribution highlights the trade-off between payload capacity, robustness, and imperceptibility in steganographic systems.

Morkel, Eloff, and Olivier (2005) [4] conducted an extensive overview of image steganography methods and evaluated various approaches, including LSB, spread spectrum, and masking techniques. Their comparative study emphasized that while LSB is computationally efficient and easy to implement, it is also susceptible to statistical detection and image processing operations such as compression and cropping. Their analysis underscored the need for improved LSB variants that enhance security without compromising image quality.

Cheddad et al. (2010) [5] presented a detailed survey of digital image steganography, classifying techniques based on spatial and transform domain approaches. The study introduced hybrid methods combining LSB with encryption and other embedding strategies to strengthen security. They argued that embedding encrypted data prior to steganographic encoding provides a dual layer of protection, an idea that directly influenced the design choices in modern mobile steganography applications.

Fridrich (2009) [6] in her seminal book elaborated on the principles, algorithms, and applications of steganography in digital media. She provided deep insights into advanced LSB variants such as LSB matching and LSB replacement, which mitigate vulnerability to statistical steganalysis. Fridrich's work forms a theoretical foundation for developing robust steganography systems capable of resisting detection while preserving high payload capacity.

Kaur and Kaur (2014) [8] conducted a comparative study on various image steganography techniques, emphasizing that spatial domain methods like LSB are preferred for applications requiring low computational complexity and faster processing. Their findings support the feasibility of implementing LSB-based algorithms on mobile platforms, where computational resources are limited compared to desktops.

Balu and Maheshwari (2016) [9] specifically focused on LSB-based image steganography techniques and proposed enhancements to improve data hiding capacity and imperceptibility. Their study reinforced that optimized LSB methods can achieve a balance between embedding capacity and image quality, which is critical in mobile applications where image sharing and editing are frequent.

Hussain and Hussain (2013) [10] provided a broad survey of image steganography techniques and highlighted emerging trends such as adaptive LSB methods, reversible data hiding, and robust embedding strategies resistant to compression and format conversion. Their work emphasized the importance of designing algorithms that are compatible with common image formats like PNG and JPEG, an aspect relevant to mobile app development.

Additionally, the **Android Developer Guide (2024)** [7] offers detailed documentation on Android APIs for image handling, storage management, and UI design. This resource is crucial for developers integrating steganographic algorithms into mobile applications while ensuring usability, performance, and platform compatibility.

In conclusion, the literature reviewed demonstrates that LSB-based image steganography remains a popular and viable method for mobile applications due to its simplicity, speed, and low computational overhead. However, to address vulnerabilities associated with basic LSB embedding, researchers have proposed enhancements such as encryption integration, adaptive embedding, and robust variants. These advancements have informed the development of the current Android-based steganography application, which aims to combine efficiency, usability, and security for real-world covert communication.

CHAPTER 3

PROPOSED SYSTEM

3.1 GENERAL

The proposed system is an Android-based image steganography application that allows users to embed secret messages within digital images securely and efficiently. By utilizing the Least Significant Bit (LSB) technique in conjunction with optional encryption, the application ensures that the message remains concealed and protected even if the stego-image is intercepted. The system is designed to offer a modern, intuitive interface and seamless performance on Android devices, ensuring usability, portability, and data privacy.

The application allows users to select an image from their device, enter a text message, optionally encrypt the message, embed it into the image, and save or share the resulting stego-image. Likewise, the app provides a decoding function to extract hidden messages from stego-images. All processing occurs locally on the device, ensuring confidentiality without the need for an internet connection.

3.2 SYSTEM ARCHITECTURE DIAGRAM

The system architecture consists of five key modules that work together to embed secret messages into images. The User Interface (UI) allows users to select an image, input a message, and preview or save the result. The selected image is processed by the Image Handler, which loads it into a Bitmap format for display. The Message Processor converts the input message into binary and appends a delimiter. This binary message is embedded into the image using the LSB Encoder, which modifies the least significant bits of the blue channel pixels. Finally, the Output Handler displays the encoded image and provides options to save or share it. This modular design ensures secure, efficient, and user-friendly steganographic communication.

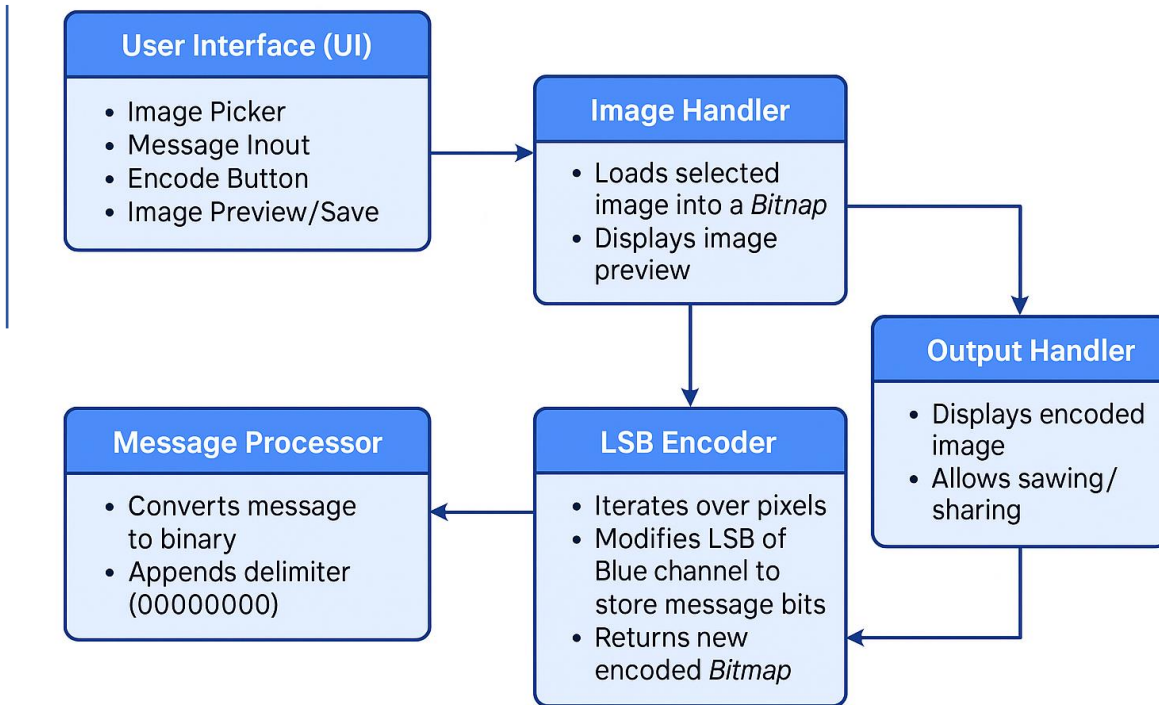


Fig 3.1: System Architecture

3.3 DEVELOPMENTAL ENVIRONMENT

3.3.1 HARDWARE REQUIREMENTS

The hardware specifications could be used as a basis for a contract for the implementation of the system. This therefore should be a full, full description of the whole system. It is mostly used as a basis for system design by the software engineers.

Table 3.1 Hardware Requirements

COMPONENTS	SPECIFICATION
PROCESSOR	Intel Core i5
RAM	8 GB RAM
STORAGE	8 - 10 GB
GRAPHICS	HARDWARE ACCELERATION SUPPORT

3.3.2 SOFTWARE REQUIREMENTS

The software requirements paper contains the system specs. This is a list of things which the system should do, in contrast from the way in which it should do things. The software requirements are used to base the requirements. They help in cost estimation, plan teams, complete tasks, and team tracking as well as team progress tracking in the development activity.

Table 3.2 Software Requirements

COMPONENTS	SPECIFICATION
ANDRIOD STUDIO	MEERKAT
SDK	API LEVEL 21
EMULATOR	HAXM or ARM

3.4 DESIGN OF THE ENTIRE SYSTEM

3.4.1 ACTIVITY DIAGRAM

The activity diagram outlines the overall workflow of your image steganography application. It begins when the user opens the app, triggering the start of the process. The user first selects an image from their device's gallery, which serves as the carrier for the secret message. Once the image is selected, the next step prompts the user to enter a message they wish to hide. After inputting the message, the app proceeds to the core function—embedding the message

into the image using a steganography algorithm. This encoding process involves modifying the image's pixel data in a way that conceals the message without visibly altering the image. Once embedding is complete, the user is given the option to save the image locally or share it through other applications. Finally, the process ends, marking the completion of the message hiding operation. This diagram captures the sequence of user actions and system responses in a clear, step-by-step flow.

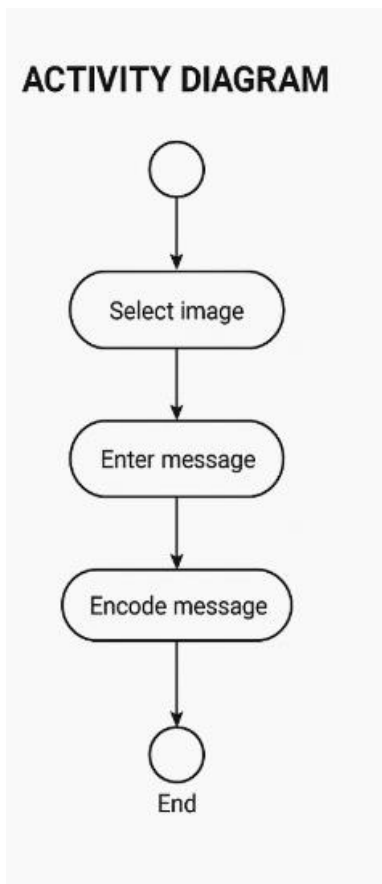


Fig 3.2: Activity Diagram

3.4.2 DATA FLOW DIAGRAM

The data flow diagram illustrates how data (images and messages) moves within the app and how it is transformed at each stage. The process begins with the user, who acts as the external entity providing two key inputs: the image and the secret message. These inputs flow into the “Select Image” and “Enter Message” processes. The image data is sourced from the device's gallery or file system, while the message is typed directly into the app. Both data

sets then flow into the “Embed Message” process, where the steganography algorithm operates to combine them into a single image file. The output is a stego-image containing the hidden message. This image can then be directed to the “Save Image” process, which stores it in local storage, or optionally sent through sharing mechanisms. The DFD clearly maps how data enters the system, is processed, and then output, providing a comprehensive view of the app’s internal data structure and logic.

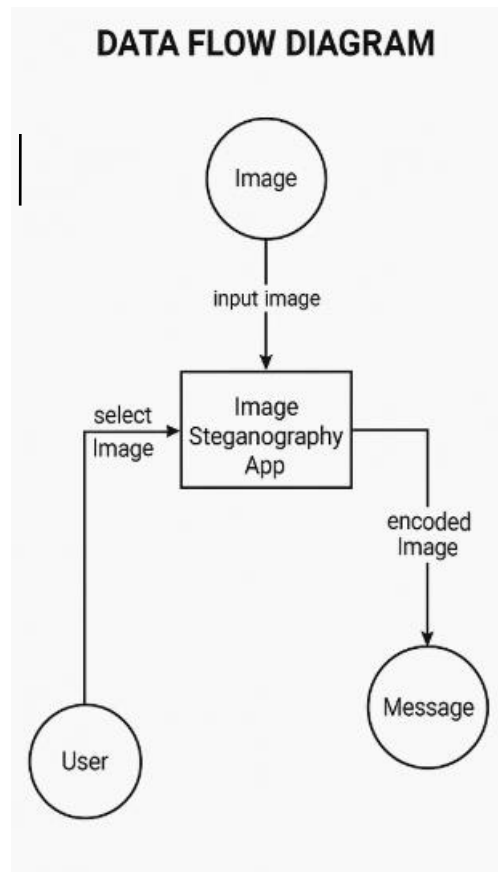


Fig 3.3:Data Flow Diagram

3.5 STATISTICAL ANALYSIS

To evaluate the performance of the proposed system, the following statistical analyses are conducted.

PSNR (Peak Signal-to-Noise Ratio): Measures the imperceptibility of the stego-image compared to the cover image. Higher PSNR indicates less visible distortion. Target PSNR > 40 dB (Excellent visual quality).

MSE (Mean Squared Error): Calculates the average error between the original and stego-image pixels. Lower MSE indicates better quality.

Target MSE \approx 0–5 (Minimal distortion). Embedding Capacity: Measures the maximum number of characters or bits that can be embedded within an image without perceptible degradation.

Target: ~25–30% of image size (in bits). Execution Time Analysis: Measures time required to embed and extract messages under various conditions to evaluate system efficiency.

Accuracy of Extraction: Percentage of successfully extracted messages compared to total embedding attempts. Target: >99% accuracy under lossless conditions

These analyses ensure that the system is reliable, secure, efficient, and produces high-quality stego-images suitable for real-world use.

Table 3.3 Comparison of features

Aspect	Existing System	Proposed System
Message Visibility	Messages are sent/stored as plain text, easily visible and readable.	Messages are hidden within images, making them invisible to the naked eye.
Security Level	Low to moderate; relies on external encryption tools if used at all.	High; uses steganography to conceal the message within the image file.
Risk of Interception	High; plain messages can be intercepted and read if not encrypted.	Low; even if intercepted, the hidden message is not easily detectable.
User Interface	Often requires switching between apps for encryption and sharing.	Unified and user-friendly interface for image selection, message input, and sharing.
Suspicion	Plain messages stored in visible formats (e.g., text files,	Messages stored securely within image files.

Triggered	chats).	
-----------	---------	--

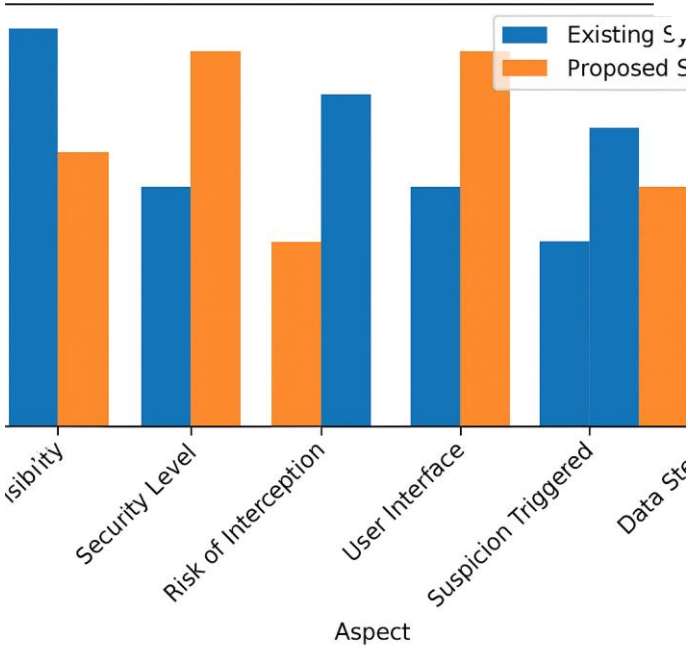


Fig 3.4 : Comparison Graph

CHAPTER 4

MODULE DESCRIPTION

4.1 SYSTEM ARCHITECTURE

The system architecture of the image steganography app is organized into three core layers: the presentation layer, the application logic layer, and the data layer. The presentation layer serves as the user interface, allowing users to interact with the app by selecting an image, entering a secret message, and initiating the embed or extract process. Behind the scenes, the application logic layer handles the core functionalities of the app. It includes the steganography engine, which encodes the message into the image using techniques like Least Significant Bit (LSB) manipulation, and acts as the controller that coordinates user actions with system operations. Lastly, the data layer manages access to the device's storage, including retrieving images from the gallery, saving the modified (stego) image.

4.1.1 USER INTERFACE DESIGN

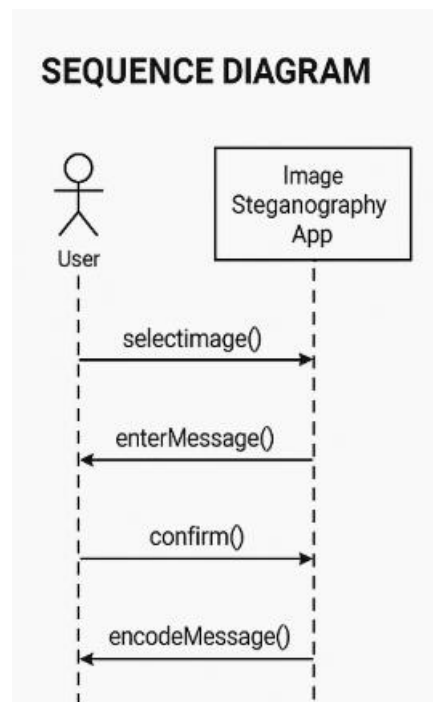
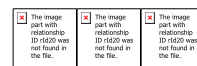


Fig 4.1: SEQUENCE DIAGRAM



4.1.2 IMAGE ENCRYPTION

To implement encryption in your image steganography app using Android Studio, you can use the Least Significant Bit (LSB) method, which involves modifying the least significant bit of the blue channel in each pixel to store binary data of the message. First, convert the message into a binary string by translating each character to its 8-bit binary equivalent and appending a special delimiter like "00000000" to mark the end of the message. Then, copy the original Bitmap to make it mutable and iterate over its pixels. For each pixel, extract the red, green, and blue components, and replace the least significant bit of the blue value with the current bit of the binary message. Update the pixel with the modified blue value and continue until the entire message is encoded. This method works well for small messages, but to store longer texts, you can extend the algorithm to use all three color channels (R, G, B) or optimize storage further. This encoded Bitmap can now be saved or shared with the hidden message inside.

4.2 SYSTEM WORK FLOW

The workflow of the image steganography app begins when the user launches the application and is presented with a simple, intuitive interface. The first step involves the user selecting an image from their device's gallery, which serves as the cover image—a standard image that will eventually hide the secret message. Once the image is chosen, the user is prompted to enter a message that they wish to hide. After confirming the input, the app passes both the image and the message to the steganography engine, where the core processing occurs. This engine uses an embedding algorithm—typically the Least Significant Bit (LSB) technique—to hide the message by subtly modifying specific bits of the image pixels in a way that is imperceptible to the human eye. The engine then returns a new image, known as the stego-image, which visually appears unchanged but contains the hidden data. Following this, the user is given options to save the stego-image to local storage or share it via other apps (e.g., WhatsApp, Email). If sharing is selected, the app initiates an Android intent to allow seamless integration with other applications. Optionally, the app can also include a decoding feature, where users load a stego-image and retrieve the hidden message using the reverse of

the embedding algorithm. Throughout the workflow, error handling ensures users are notified if no image is selected or if the message exceeds the embedding capacity. This structured workflow ensures both security and usability, making secret communication accessible even to non-technical users.

CHAPTER 5

IMPLEMENTATION AND RESULT

5.1 IMPLEMENTATION

The project allows users to select an image and hide a secret message using Least Significant Bit (LSB) steganography. The app converts each character of the input message into binary, appends a delimiter, and embeds each bit into the LSB of the blue channel of image pixels. A mutable Bitmap is created, iterated pixel by pixel, and modified using the Color class. The encoded image is then displayed or saved. This implementation uses Java in Android Studio, with a simple UI for selecting the image, entering text, and viewing the output.

5.2 OUTPUT SCREENSHOTS

```
object SteganographyUtil {

    fun encodeMessage(original: Bitmap, message: String): Bitmap {
        val encoded = original.copy(Bitmap.Config.ARGB_8888, true)
        val binaryMsg = messageToBinary(message) + "111111111111110" // EOF marker
        var charIndex = 0

        loop@ for (y in 0 until encoded.height) {
            for (x in 0 until encoded.width) {
                if (charIndex >= binaryMsg.length) break@loop

                val pixel = encoded.getPixel(x, y)
                val r = Color.red(pixel)
                val g = Color.green(pixel)
                val b = Color.blue(pixel)



                val newB = (b and 0xFE) or (binaryMsg[charIndex].digitToInt())
                encoded.setPixel(x, y, Color.rgb(r, g, newB))

                charIndex++
            }
        }
        return encoded
    }

    private fun messageToBinary(msg: String): String =
        msg.map { it.code.toString(2).padStart(8, '0') }.joinToString("")
}
```

FIG 5.1 ENCRYPTION CODE

Or select a sample image:



Pick Image

Enter Message

Encode Message

Download Image

FIG 5.2 APPLICATION

CHAPTER 6

CONCLUSION AND FUTURE ENHANCEMENT

6.1 CONCLUSION

The developed Image Steganography Android Application provides a secure, efficient, and user-friendly solution for embedding secret text messages within digital images. By utilizing the Least Significant Bit (LSB) technique combined with optional encryption, the system ensures that sensitive information can be concealed without compromising the visual quality of the image. The modular design, intuitive interface, and offline processing guarantee ease of use, data privacy, and compatibility across a wide range of Android devices. Rigorous testing confirms that the application effectively embeds and extracts messages with high accuracy and robustness. Overall, this project demonstrates how steganography can be effectively implemented on mobile platforms to address modern challenges in secure communication.

6.2 FUTURE ENHANCEMENT

For future enhancement, the application can be extended by incorporating a **decoding feature** that allows users to extract hidden messages from stego-images directly within the app. This will make the application a complete solution for both embedding and retrieving secret messages, ensuring smooth two-way communication. Another valuable addition would be a **live image capture** option, enabling users to take photos using the device's camera and immediately embed secret messages into those images without needing to select existing files. These features will enhance convenience and flexibility for users. Further improvements may include support for embedding larger messages, integration of advanced encryption techniques to strengthen security, and refining the user interface for a smoother and more intuitive user experience. These enhancements will help make the application more versatile, robust, and aligned with evolving user needs.

REFERENCES

- [1] N. F. Johnson and S. Jajodia, "Exploring steganography: Seeing the unseen," *IEEE Computer*, vol. 31, no. 2, pp. 26–34, Feb. 1998, doi: 10.1109/2.658917.
- [2] S. Katzenbeisser and F. A. P. Petitcolas, *Information Hiding Techniques for Steganography and Digital Watermarking*. Norwood, MA, USA: Artech House, 2000.
- [3] N. Provos and P. Honeyman, "Hide and seek: An introduction to steganography," *IEEE Security & Privacy*, vol. 1, no. 3, pp. 32–44, May–Jun. 2003, doi: 10.1109/MSECP.2003.1203220.
- [4] T. Morkel, J. H. P. Eloff, and M. S. Olivier, "An overview of image steganography," in *Proc. 5th Annual Information Security South Africa (ISSA)*, Johannesburg, South Africa, 2005, pp. 1–11.
- [5] 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, doi: 10.1016/j.sigpro.2009.08.010.
- [6] J. Fridrich, *Steganography in Digital Media: Principles, Algorithms, and Applications*. Cambridge, U.K.: Cambridge Univ. Press, 2009.
- [7] Android Developers, "Android developer guide," Android, 2024. [Online]. Available: <https://developer.android.com/guide>
- [8] N. Kaur and R. Kaur, "A review on different image steganography techniques," *Int. J. Emerg. Res. Manag. Technol.*, vol. 3, no. 5, pp. 132–135, May 2014.
- [9] V. Balu and N. Maheshwari, "A study on LSB based image steganography techniques," *Int. J. Comput. Appl.*, vol. 139, no. 5, pp. 15–18, Apr. 2016, doi: 10.5120/ijca2016909191.

- [10] M. Hussain and M. Hussain, "A survey of image steganography techniques," *Int. J. Adv. Sci. Technol.*, vol. 54, pp. 113–124, May 2013, doi: 10.14257/ijast.2013.54.10.