A
Industry Oriented Mini Project Report
on

RANDOMIZED LEAST SIGNIFICANT BIT (LSB) STEGANOGRAPHY WITH ADVANCED ENCRYPTION STANDARD (AES) ALGORITHM

A Report submitted in partial fulfillment of the
requirements for the award of degree of
Bachelor Of Technology

by

G Mahalakshmi Sai Padmini
(20EG105415)

Bommaku Vaishnavi
(20EG105406)

Chinchalpet Shreya Goud
(20EG105407)

Under the guidance of
Dr. K. Madhuri
Associate Professor
Department of CSE,
Anurag University

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
ANURAG UNIVERSITY
VENKATAPUR-500088
Year 2023-2024

DECLARATION

We hereby declare that the Report entitled **“Randomized Least Significant Bit (LSB) Steganography with Advanced Encryption Standard (AES) Algorithm”** submitted for the award of **Bachelor of Technology** Degree is our Original work and the Report has not formed on the basis for the award of any Degree, Diploma, Associate ship or fellowship of similar other titles. It has not been submitted to any other University or Institution for the award of any degree or diploma.

Place: Anurag University, Hyderabad

G Mahalakshmi Sai Padmini
(20EG105415)

Bommaku Vaishnavi
(20EG105406)

Chinchalpet Shreya Goud
(20EG105407)

ANURAG UNIVERSITY | SCHOOL OF ENGINEERING

CERTIFICATE

This is to certify that the Report entitled **"Randomized Least Significant Bit (LSB) Steganography With Advanced Encryption Standard (AES) Algorithm"** that is being submitted by **Ms. G Mahalakshmi Sai Padmini** bearing the roll number **20EG105415, Ms. Bommaku Vaishnavi** bearing the roll number **20EG105406, Ms. Chinchalpet Shreya Goud** bearing the roll number **20EG105407** in partial fulfillment for the award of B. Tech in **Computer Science and Engineering** to the **Anurag University** is a record of bonafide work carried out by them under my guidance and supervision.

The result embodied in this Report have not been submitted to any other University or Institute for the award of any degree or diploma.

Signature Of Supervisor
Dr. K. Madhuri
Associate Professor
Department of CSE,
Anurag University

Dean, CSE

External Examiner

ACKNOWLEDGMENT

We would like to express our sincere thanks and deep sense of gratitude to our project supervisor **Mrs. DR. K. MADHURI** for her constant encouragement and inspiring guidance without which this project could not have been completed. Her critical reviews and constructive comments improved our grasp of the subject and steered to the fruitful completion of the work. Her patience, guidance and encouragement made this project possible.

We would like to express our special thanks to **DR. V. VIJAYA KUMAR,** Dean School of Engineering, Anurag University, for their encouragement and timely support in our B. Tech program.

We would like to acknowledge our sincere gratitude for the support extended by **DR. G. VISHNU MURTHY**, Dean, Dept. of CSE, Anurag University. We also express our deep sense of gratitude to **DR. V. V. S. S. S. BALARAM**, Academic Co-Ordinator, **DR. PALLAM RAVI**, Project in-charge. Project Co-Ordinator and Project Review committee members, whose research expertise and commitment to the highest standards continuously motivated me during the crucial stages our project work.

G Mahalakshmi Sai Padmini
(20EG105415)

Bommaku Vaishnavi
(20EG105406)

Chinchalpet Shreya Goud
(20EG105407)

ABSTRACT

In our interconnected digital world, secure communication is of paramount importance. Hacking, phishing, and malware attacks continually threaten sensitive data, making data security and privacy preservation top priorities. One approach to secure communication involves combining image steganography, AES encryption, and randomization techniques. Image steganography hides information within digital images, making it difficult to detect. AES encryption, a widely accepted encryption algorithm, renders data indecipherable without the correct decryption key. Randomization techniques add an additional layer of unpredictability, making it even harder for unauthorized parties to breach security. This strategy enables covert data transfer, allowing secure communication without revealing the existence of hidden messages. This capability is invaluable in government, corporate, and personal communications where data security is critical. Secure data transfer is vital in a society where data breaches can have severe consequences. Upholding trust and confidence in digital communication and transactions is essential. Future research opportunities abound in expanding the applications of image steganography and enhancing this approach to stay ahead of emerging threats. As technology evolves and security challenges change, the quest for innovative and effective secure communication methods remains a priority.

In today's hyperconnected digital landscape, ensuring the security of communication is of paramount significance. The relentless onslaught of cyber threats, including hacking, phishing, and malware attacks, poses a constant danger to sensitive information. As a result, safeguarding data security and preserving privacy has become an overarching concern. One compelling approach to achieving secure communication is the fusion of image steganography, AES encryption, and randomization techniques. Image steganography provides a clandestine method for concealing information within digital images, rendering detection remarkably challenging. AES encryption, a globally recognized encryption standard, transforms data into an unintelligible format unless decrypted with the correct key. The integration of randomization techniques further enhances security by introducing an additional layer of unpredictability, thereby thwarting unauthorized access and fortifying data protection. This strategy empowers covert data transmission, facilitating secure communication while concealing the existence of concealed messages. Its applicability spans across diverse domains, including government, corporate, and personal communications, where data security is non-negotiable. In a society where data breaches can result in dire consequences, secure data transfer plays a pivotal role. Upholding trust and confidence in digital communication and transactions is indispensable, and this integrated approach serves as a robust bastion of defence.

Looking forward, the horizons of research opportunities in this domain are expansive. Future investigations can explore the broader applications of image steganography and seek to refine this approach to proactively address emerging security threats. As technology evolves and security challenges continually morph, the pursuit of innovative and effective methods for secure communication remains at the forefront of our endeavours. The preservation of data integrity and privacy is not just a priority; it is a commitment to a safer and more trustworthy digital world.

KEYWORDS

1) Least Significant Bit (LSB) Steganography
2) Portable Network Graphics (PNG) images
3) Advanced Encryption Standard (AES)-256 Encryption
4) Advanced Encryption Standard (AES)-256 Decryption
5) Image Encoding
6) Image Decoding
7) SHA-256 Hash Function
8) Base-64 Encoding
9) Bit Shuffling
10) Header Text

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ABBREVIATION	FULL FORM
LSB	Least Significant Bit
MSE	Mean Squared Error
PSNR	Peak Signal to Noise Ratio
AES	Advanced Encryption Standard
PNG	Portable Network Graphics
RGB	Red, Green, Blue
CMYK	Cyan, Magenta, Yellow, Key (or Black)
RGBA	Red, Green, Blue, Alpha
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
XOR	Exclusive OR
IV	Initialization Vector
SHA	Secure Hash Algorithm
CBC	Cipher Block Chaining
ASCII	American Standard Code for Information Interchange
IDLE	Integrated Development and Learning Environment

LIST OF CONTENTS

1. **INTRODUCTION**

The concept of steganography involves the idea of hiding information in a file[1]. While cryptography is the act of encrypting the data using accepted techniques to prevent others from reading the original data, this data can be audio, video, or text. Steganography has an extra benefit over cryptography which is the secret message can be concealed behind a picture which is difficult to spot. Steganography also has the added advantage that an un-suspecting user will not be aware of the existence of hidden data, whereas this is not the case with data encrypted using cryptography techniques.[2] Steganography has a higher likelihood of being undetected than encryption, which offers more security. This method proposes to use both security techniques to achieve a better system. This method challenges the attackers to decode and decrypt the message as it is very difficult to analyse it. This is because we are introducing one more extra security layer which is "Randomization."

There are many methods to perform cryptography such as Blowfish, Data Encryption Standard (DES), Advanced Encryption Standard (AES) and so on[2]. Also, for steganography there are many methods available such as LSB method, DCT (Discrete Cosine Transform), DWT and FFT. In this work, Least Significant Bit (LSB) technique for steganography and Advanced Encryption Standard – 256 (AES-256) for cryptography and Bit Randomization for the Randomization method are employed in consideration of compatibility, security and efficiency. The shuffled bits then are encrypted using AES-256 where we use BASE-64 Encoding and then the encoded text is encoded into the image by using LSB Steganography. Also we will insert some Header Text to the data and then Encoding takes place. We also employ the usage of the Portable Network Graphics (PNG) images only and we tend to convert the other modes of the images like Greyscale, CMYK, RGBA images to PNG images and henceforth are used for the Steganography.

In this work, a sophisticated hybrid steganography and encryption system is used that makes use of the current technology is introduced. The fusion of tried-and-true technologies not only adheres to the traditional principles of text encoding and encryption but also offers efficient, cutting-edge workarounds for their drawbacks. This new system offers a safe and adaptable architecture that improves security against current threats, strengthens each encrypted and encoded text while decreasing its vulnerabilities, and enables the secrecy and integrity of data interactions. The results are compared based on the PSNR, MSE metrics values and those results are tabulated.

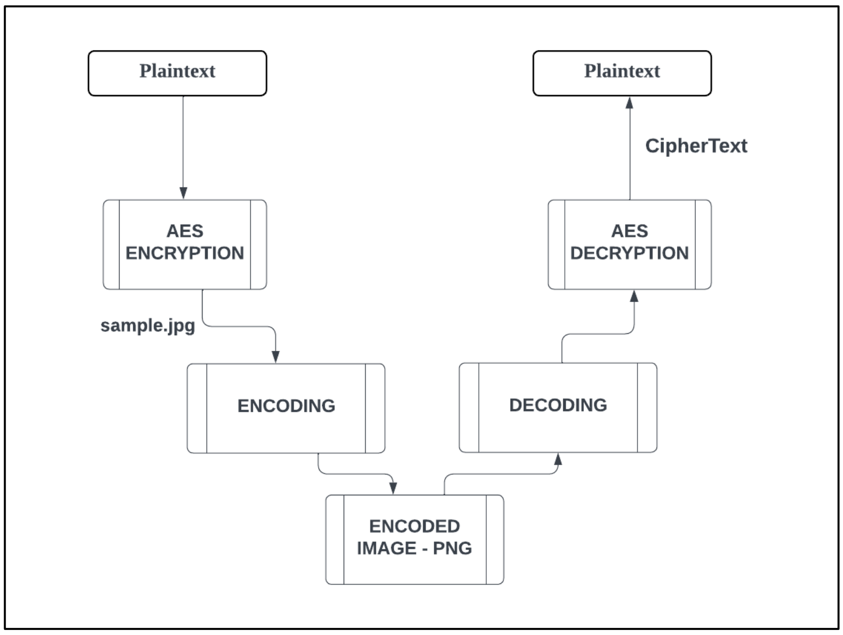


Fig 1.1: LSB Steganography with AES Algorithm

1.1 AES-256 ENCRYPTION AND DECRYPTION

The symmetric encryption technique known as AES (Advanced Encryption Standard) is well known for its security. AES-256, a variation of AES, is very resistant to brute-force assaults since it encrypts data using a 256-bit key. This Key is generated using SHA-256 Hash Function where the message digest of size 256 Bits are used as Key.

The plaintext is broken up into blocks in this AES-256 based encryption, each of which has the same size as the encryption block size (128 bits). AES-256 employs many rounds of substitution, permutation, and XOR operations for both encryption and decryption on 128-bit data blocks. For each encryption round, the key scheduling algorithm creates a round key. AES-256 is widely used in applications like HTTPS for protecting data in transit and disk encryption for securing data at rest since it is computationally efficient. For data integrity and authenticity, it's critical to combine AES-256 with techniques like hashing or digital signatures. Being symmetrical in nature, it emphasizes the significance of safe key management and appropriate implementation by using the same key for both encryption and decryption. Here the additional terms used are:

Initialization Vector (IV): For AES, the IV is a random number with a block size of 128 bits. The IV is used to make sure that using the same key to encrypt the same plaintext numerous times produces distinct ciphertexts.

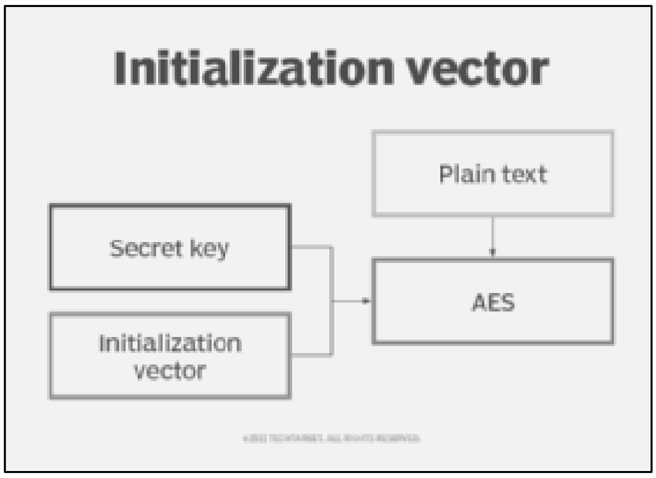


Fig 1.2: Initialization Vector

SHA-256:

A cryptographic hash algorithm called SHA-256 is employed in a variety of security applications. It is intended to accept as input a message or other piece of data and output a fixed-length, 256-bit hash value. As a member of the SHA-2 family, SHA-256 is renowned for its dependability and resistance to collisions, making it ideal for secure password storing and data integrity verification. Some common use cases of SHA-256 include:

1. **Password Storage**: SHA-256 is used to securely store passwords. Instead of storing the actual password, systems store the hash of the password. When a user logs in, the system hashes the entered password and compares it to the stored hash. This way, the actual password remains confidential.
2. **Data Integrity**: SHA-256 is used to verify the integrity of data during transmission. By comparing the hash of the received data to the hash of the original data, one can ensure that the data hasn't been tampered with during transit.

3. **Blockchain Technology**: Many cryptocurrencies, like Bitcoin, use SHA-256 for creating digital signatures and verifying transactions on the blockchain. It plays a crucial role in the security and immutability of blockchain records.

XOR (exclusive OR): It is used to work with binary data. It generates a binary output from two binary inputs. If the number of true (1) bits is odd, the XOR operation yields true (1); if it is even, it returns false (0). It compares the equivalent bits of two binary integers and returns 1 if they differ and 0 if they are identical.

Cipher Block Chaining (CBC) Method:

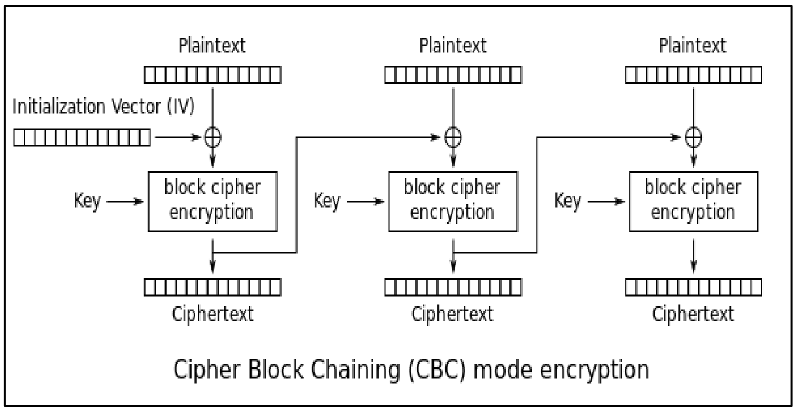


Fig 1.3: Cipher Block Chaining (CBC) Mode

Cipher Block Chaining (CBC) is a widely used method for encrypting data, especially in block cipher modes of operation like AES (Advanced Encryption Standard). In this process, an Initialization Vector (IV) is generated randomly. The IV is crucial for introducing an element of randomness and unpredictability into the encryption process. This prevents patterns in the plaintext from showing up as patterns in the ciphertext, enhancing security.

The first plaintext block is combined with the IV using the XOR (exclusive OR) operation. This XORed value becomes the input for the AES encryption process. AES, in this case, is typically configured with a 256-bit encryption key. The AES encryption operation transforms this XORed value into ciphertext, which becomes the first ciphertext block. What makes CBC particularly secure is that each subsequent plaintext block is XORed with the preceding ciphertext block before encryption. This ensures that identical plaintext blocks don't produce the same ciphertext, adding an extra layer of security. The resulting ciphertext, produced block by block, is what provides confidentiality and data integrity. This chaining of blocks ensures that any modification in one block will disrupt the entire decryption process, making it a robust choice for secure data encryption, especially when confidentiality and integrity are paramount.

It's important to note that the first block in CBC is a bit special since there's no "previous" ciphertext block to XOR with. This is where the IV comes into play, effectively acting as the "previous" ciphertext block for the first plaintext block. This entire process is repeated for each block of plaintext, creating a chain of ciphertext blocks. This method is not only used in AES but is a fundamental concept in modern symmetric-key cryptography, forming the basis for secure communication and data storage.

Example:

Plaintext:

U	**n**	**i**	**v**	**e**	**r**	**s**	**i**	**t**	**y**

Padded Plaintext (128 bit):_b'University\x06\x06\x06\x06\x06\x06'

Key:b'\xa8\xf1\x96\x9c\x12\x8c\xbcV\x8cD}\r\xd0~\xde\xae\xd8\xd1\xc9[$\xc1\xef gM\xa3\xadW\xe2U\x8dN'

IV:_b'\x8c\xe0fZH\x9f\x0eN\xafik\xe7RX\xa5r'

Ciphertext:_ b'u\x1e\xfc\xd5\xba\xe2\x05\xafW\xd0\xc1\xaa\x14\xad\x98@'

1.2 LSB STEGANOGRAPHY

LSB (Least Significant Bit) steganography in RGB mode is a clever and straightforward method for concealing data within images. It operates by subtly modifying the least significant bits of the red, green, and blue color channels of an image.

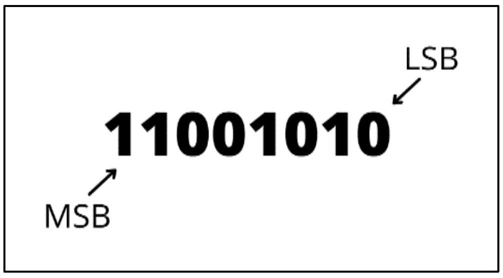


Fig 1.4: LSB Demonstration in 8 bit

By doing so, it embeds hidden information without causing noticeable visual changes in the image, making it a favored choice for those seeking to obscure watermarks, secret messages, or other concealed content within digital photos. This technique leverages the fact that the least significant bits have the least impact on the overall color perception, ensuring that the alterations go largely unnoticed to the human eye.

Despite its popularity and ease of use, RGB LSB steganography has its limitations. It's not suitable for hiding extensive amounts of data due to the limited capacity within the least significant bits. Furthermore, it may not be secure against advanced steganalysis techniques or high-quality image compression. Nonetheless, this method finds applications in both legitimate and illicit contexts, such as digital watermarking for copyright protection or, unfortunately, for covert communication in scenarios where privacy and security are at odds with legal and ethical norms. Its simplicity and effectiveness make it an interesting area of study in the field of information security and digital forensics, as it requires a balance between the imperceptibility of the alterations and the data hiding capacity.

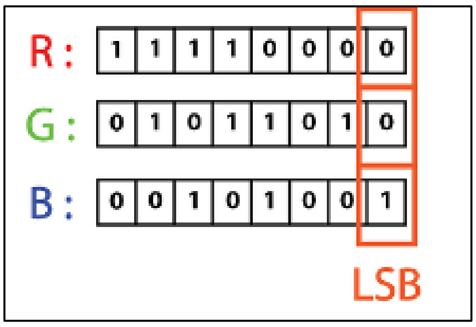


Fig 1.5: RGB Colors in a pixel

Example:

Message: hello

ASCII: 104 101 108 111

Message (Bits): 01101000 01100101 01101100 01101111

<u>Image used to encode is:</u>

Fig 1.6: Original Image

Fig 1.7: Encoded Image

1.3 BIT SHUFFLING

Bit shuffling is a cryptographic method that is essential for boosting encryption process security and randomness. Bit shuffling adds dispersion to the ciphertext by shifting the bits inside the data, making it more difficult for attackers to identify patterns. This procedure strengthens the resilience against predicted assaults by introducing a crucial element of randomization. This strengthens confusion and diffusion—two essential components of robust cryptography. Bit shuffling adds an extra layer of security and is very useful when you need the highest level of security by bringing complexity. Here we use Bit shuffling to shuffle the first 4 bits to the last and the last bits to the first to induce both Confusion and Diffusion.

Example:

Original Bits:

1	**0**	**0**	**1**	**0**	**1**	**1**	**1**

Shuffled Bits:

0	**1**	**1**	**1**	**1**	**0**	**0**	**1**

1.4 BASE-64 ENCODING

Base64 encoding is a fundamental method for representing binary data as text, which facilitates secure transmission and storage within systems primarily designed for text-based data. This encoding scheme is pivotal in overcoming the challenges posed by binary data when used in communication protocols or data formats that are text-oriented. These systems often struggle to handle binary data directly because binary data may contain characters that are either forbidden, misinterpreted, or altered during transmission or storage. To mitigate these issues, Base64 encoding provides a standardized solution.

The core principle of Base64 encoding involves taking groups of 6 binary bits and mapping them to one of 64 predefined text characters. These 64 characters are typically A-Z, a-z, 0-9, and the symbols '+', and '/'. This mapping is based on the Base64 Encoding table, which ensures consistency in the conversion process across different platforms and applications. By converting binary data into this plain text format, the data becomes universally acceptable and resistant to unintended corruption during transfer or storage.

Base64 encoding plays a crucial role in a wide range of applications. It's essential for email attachments, where binary data like images or documents needs to be transmitted as text within the email's body. It's also commonly used in web technologies for embedding binary data, such as images or audio, within HTML, CSS, and JavaScript files. Additionally, Base64 encoding is fundamental in cryptographic systems where securely storing cryptographic keys and certificates in a text format is necessary. Its simplicity, security, and compatibility make it a cornerstone in ensuring data integrity and reliable transmission across various platforms and environments.

Example:

TEXT:

M	**a**	**n**

ASCII:

77	**97**	**110**

BIT PATTERN:

0	**1**	**0**	**0**	**1**	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**1**	**1**	**0**

BASE-64 INDEX:

19	**22**	**5**	**46**

BASE-64 ENCODED TEXT:

T	**W**	**F**	**u**

1.5 PNG IMAGES

Using PNG (Portable Network Graphics) graphics in the project offers a multitude of advantages, particularly when considering data security and concealment through techniques like LSB steganography. One of the most prominent benefits of PNG is its status as a lossless image format. This means that when images are stored in the PNG format, there are no compression artifacts or data loss due to compression. In the context of LSB steganography, this is of paramount importance, as even minor distortions in the image could result in the loss of concealed data. The absence of artifacts in PNG images ensures that the hidden information remains intact and undetectable, making it a reliable choice for secure data embedding. Furthermore, PNG images are known for their support of a 24-bit RGB color palette. This extensive color palette provides a wide array of colors, enabling the concealed data to be seamlessly integrated into the image without raising suspicion from the human eye. This attribute is especially valuable for steganographic applications where the primary goal is to hide information within the image without altering its appearance. This extensive color range ensures that the modifications to the image are subtle and unnoticeable. PNG's widespread support across various systems and software applications ensures compatibility and ease of sharing. This makes it an excellent choice for projects where seamless integration and sharing across different platforms and environments are essential. Additionally, PNG images incorporate zlib compression and adhere to open standards. These features contribute to reducing the storage and transmission requirements, which is a crucial consideration in many projects. In summary, PNG's lossless quality, extensive color palette, compatibility, transparency, and compression standards collectively make it a compelling choice for projects that require secure data embedding and widespread accessibility.

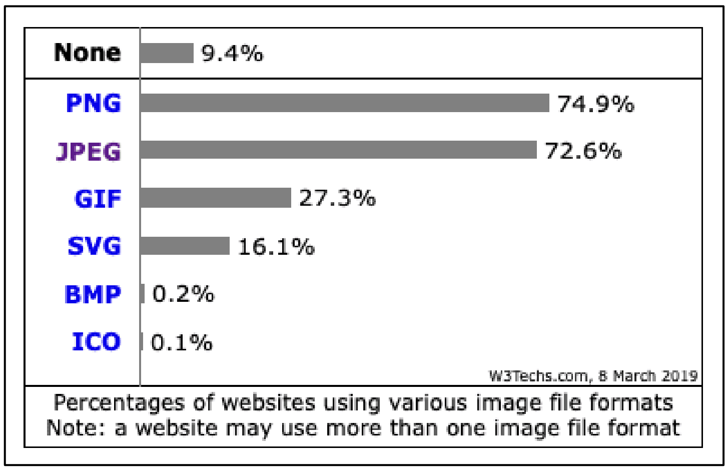


Fig 1.8: Survey on Various Image Formats

1.6 HEADER TEXT

"Headertext" in steganography refers to the process of inserting a message or piece of data at the start of a file before encoding it to be hidden. This method is helpful since it protects the message's integrity and reduces the likelihood that it will be corrupted during encoding or compression. Additionally, the headertext acts as a quick reference for message extraction, making it simple for receivers to find the concealed data. It may also include metadata or details about the hidden content, making data administration and retrieval easier. The header is significant because it facilitates synchronization during decoding and can be helpful in error recovery if decoding problems occur or the file is corrupted. Advantages are:

1. **Data Integrity:** Placing the message or data at the start of the file helps protect its integrity. When files undergo encoding or compression, alterations to the data could occur. By adding the message at the outset, it's less likely to be affected during these processes.
2. **Quick Reference:** The headertext serves as a quick reference point for message extraction. When a recipient wants to retrieve the concealed data, they can quickly locate the headertext, simplifying the extraction process.

3. **Metadata Inclusion:** The headertext may also contain metadata or information about the hidden content. This additional information can aid in data management and retrieval. For example, it could specify the type of data hidden, its source, or any relevant details.

1.7 MOTIVATION OF THE PROJECT

This project, was primarily motivated by the pressing need to bolster data security and privacy in the digital realm. In today's interconnected world, the constant threats of hacking and unauthorized data access have made safeguarding sensitive information an urgent priority. This motivated the pursuit of innovative techniques to enhance secure communication.

1.8 PROBLEM DEFINITION

This project aims to enhance data security and confidentiality in digital communication. The motivation arises from the increasing cybersecurity threats, highlighting the need for robust solutions. The problem revolves around securing data during transmission and storage. By combining image steganography, AES encryption, and randomization, we strive to create a comprehensive solution. Our objectives include covert data transfer, encryption fortification, and defense against known-plaintext, chosen-ciphertext, and ciphertext-only attacks.

1.9 OBJECTIVES OF THE PROJECT

The project's objectives are to enhance data security through a combination of steganography, AES encryption, and randomization. It aims to achieve covert data transfer, strengthen encryption.

2. LITERATURE SURVEY

[1] **Adit Pabbi, 2021 et.al:** In this suggested technique, which combines encryption and steganography, the arrangement is there in place where a system is used to send a message from to the recipient and improve the security of the transmission while simultaneously maintaining privacy. The recipient sends both the key and the encoded picture, explaining why the Using an encrypted approach is possible. This method makes use of the AES with the least significant bit (LSB) steganography technique. To do this, we tend to use encryption.

[2] **Utsav Sheth, 2016 et.al:** This method explains the steganography technique, which involves hiding text within a picture. Here, each image byte is altered such that the bottom nibble now includes each nibble of the input text. This implementation's steganography technique maximizes data capacity while still guaranteeing security. As Java is a popular programming language because of how simple it is to use and how many libraries it has. We use the AWT and SWING libraries in Java, and a straightforward GUI has been created.

[3] **C. Lalengmawia, 2016 et.al:** In this method, a sophisticated picture steganographic method is used that can convey vast amounts of data. The Advanced Encryption Standard (AES) method, which essentially produces random pixel positions, is utilized. Because this method chooses the place using AES at random and determines the amount of Least Significant Bits for embedding information dynamically, making the information is more secure. Results are compared with four state-of-the-art methods.

[4] **Masumeh Damrudi, 2019 et.al:** Data security is improved by using a mix of encryption and steganography. This method uses the LSB (Least Significant Bit) steganography technique using the cryptographic algorithms AES, RSA, DES, 3DES, and Blowfish to encrypt a message that should be concealed in a cover picture. Execution duration, PSNR (Peak Signal to Noise Ratio), MSE (Mean Square Error), and the histogram of the main and covered picture are used to display the findings.

[5] **Inas Jawad Kadhim, 2019 et.al:** This study comprises of picture steganography that are now used. This also gives a thorough review of picture steganography, including its fundamental principles, prerequisites, many facets, varieties, and performance assessments. Here, many performance study metrics for assessing

steganographic systems are also included. Additionally, we go through how to choose various cover media for various purposes as well as a few cutting-edge steganalysis systems.

[6] **Abbas Cheddad, 2019 et.al:** This technique offers a state-of-the-art evaluation and analysis of the many steganography techniques now in use, along with some widely accepted standards and recommendations from the literature. This paper ends with a few suggestions and support for the object-oriented embedding technique. This paper does not focus on steganography analysis, although it will still be briefly covered.

[7] **Mark Rennel D. Molato, 2018 et.al:** This paper presents a less complex and straight-forward image steganography technique that has an acceptable embedding capacity, an appropriate level of security and a significantly low probability of detection of either visual assessment or statistical analysis techniques. In this technique, we use YCbCr color space rather than the cover image's RGB original color space. Additionally, it was demonstrated by the Chi-square and RS analysis simulation findings that there is a minimal likelihood that the suggested technique's stego-images will be discovered.

[8] **Mansi S. Subhedar, 2014 et.al:** The fundamental components of an effective steganographic algorithm are security, high embedding capacity, and imperceptibility. This technique offers a novel method based on the discrete wavelet transform (DWT) and singular value decomposition (SVD) for embedding hidden information in cover images. This method replaces the secret picture's single values in the HH band with those of the cover image. Additionally, the findings of the experiments demonstrate resilience against various geometric and image processing challenges.

[9] **Dipti, K. S, 2010 et.al:** In this method, One conceals the communication's existence, while the other modifies the content of the message. We can conceal the message using a variety of approaches in the spatial, frequency, and other domains of steganography. Since it is exceedingly challenging to discover concealed messages in the frequency domain, this paper apply a variety of transformations, including DCT, FFT, and Wavelets, among others. In this project, along with a newly created improved security module, there is a creation of a system using a novel approach that integrates both cryptography and steganography.

[10] **Raphael, A. J., 2011 et.al:** As a result, the perspective from the point of

security of such data has a difficult job in the current situation. The final user must demand that his private information be shielded from illegal access.

Sl.no	Author (s)	Method	Advantages	Disadvantages
1	Adit Pabbi, Rakshit Malhotra, Manikandan K	LSB Steganography with AES Algorithm Using GUI - 2021	Enhanced Security, Covert Communication	Impact on Image Quality, Limited Transmission Channels
2	Utsav Sheth, Shiva Saxena	Image Steganography Using AES Encryption, Least Significant Nibble and Java - 2016	Graphical user interface, Comprehensive libraries	Limited to lower nibble, Lossy compression limitations
3	C. Lalengmawia, A. Bhattacharya	Image Steganography using Advanced Encryption Standard for implantation of Audio/Video Data by embedding the position of pixels randomly - 2016	Increased security, dynamic embedding, comparative analysis	Lack of specific details, Potential impact on image quality
4	Masumeh Damrudi, Kamal Jadidy Aval	Image Steganography using LSB and encrypted message with AES, and other hybrid algorithms by considering the factors PSNR, MSE - 2019	Concealment of the message, Increased resistance to the attack	Potential degradation of the image quality, Increased complexity
5	Inas Jawad Kadhim, Prashan Premaratne, Peter James Vial and Brendan Halloran	Survey of image steganography: Techniques, Evaluations, and trends in future research - 2019	Secure image transmission, availability of digital images	Limited Embedding capacity, Impact on image quality

Table 2.1: Literature Survey – Comparison of the Methods

Accordingly we have taken a project which will improve the image quality by using **“RANDOMIZED STEGANOGRAPHY WITH AES ALGORITHM”**

3. RANDOMIZED LEAST SIGNIFICANT BIT (LSB) STEGANOGRAPHY WITH ADVANCED ENCRYPTION STANDARD (AES) ALGORITHM

The proposed methodology consists of steps to be performed:

i. Selecting the option either to Encrypt and Encode the Text or to Decode and Decrypt the Text.

ii. Then provide the Image path and Checks if the image is in PNG Format of RGB mode.

iii. If other formats are provided, then we are not processing it. If other modes of the image like Greyscale, CMYK, RGBA then they are converted to RGB mode.

iv. Enter the Plaintext that needs to be sent from the Sender to Receiver.

v. Enter the Password that is present with the both Sender and Receiver.

vi. The display the IV, Key which is generated based on SHA-256 Value generated from the Password and the Padded Plaintext based on AES Block Size.

vii. Shuffling of the Bits takes place and the corresponding Cipher Text is generated.

viii. The above generated Cipher Text is encoded using Base-64 and the resulting Base-64 Text is appended with the Header Text.

ix. This text is encoded into the Image using LSB Steganography

x. Then, the Stego Image is displayed.

xi. While in the Decoding and Decryption Stage, We will select the next option and proceed to enter the path of the Stego Image.

xii. Then we need to check whether the Stego Image is present or not and then the receiver must also provide the password.

xiii. Then the Decoding happens where the text is decoding from the Image and then Header Text is validated.

xiv. The Decoded Text from the Image is again Decoded using BASE-64 Decoding.

xv. Then the Bits are Reshuffled and then AES-256 Decryption is performed which is followed by removing the padded bits.

xvi. The Final Plain Text that the Sender Sent is Shown to the Receiver.

3.1 ENCODING AND ENCRYPTION STAGE

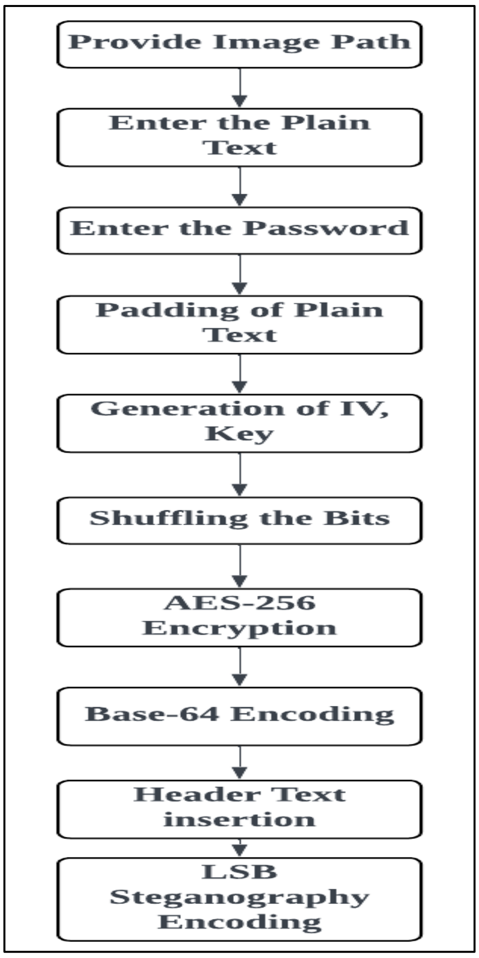


Fig 3.1: Encoding and Encrypting Stage Steps

RGB Mode - PNG Format:

We are just utilizing RGB Mode in PNG Format for this project. The procedure is terminated if the specified Image path is not in PNG Format. If the image is not in RGB mode, we will verify the image's mode. If the image is in greyscale, CMYK, or RGBA, we may be able to convert it to RGB.

If the RGB Mode In PNG conversion and verification described above are successful, we usually display the number of pixels in the Picture.

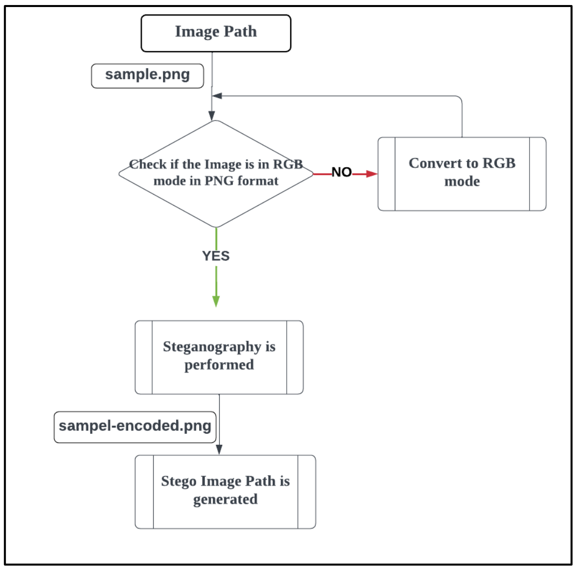


Image Path
sample.png
Check if the Image is in RGB mode in PNG format
NO
Convert to RGB mode
YES
Steganography is performed
sampel-encoded.png
Stego Image Path is generated

Fig 3.2: Checking for RGB Mode in PNG Format

Plain Text:

The plaintext that is given must be that which should be encoded and encrypted. This can be any length as long as it doesn't go above the size indicated by the number of pixels in the image.

Password:

The Sender shares the password with the Receiver as well. The password is sent to the SHA-256 algorithm, which uses it to generate a message digest hash value from which the 256 bit key for AES encryption may be generated.

Padding the Plaintext:

The given plaintext needs to be padded in multiples of the AES-Block Size or according to the size of the AES Block Size, which is 128 bits.

Generation of IV, Key:

In the Cipher Block Chaining (CBC) mode, AES-256 encryption is carried out using block size IV, or 128 bits, which is the size of the AES-Block Size. This is constructed using a combination of randomness and the previously mentioned password, which is then sent to the SHA-256 function, that employs the 256-bit message digest as its key.

Bit Shuffling:

When the Padded Plaintext is translated to ASCII and the 8 Bits are created for each byte, bit shuffle is utilized to encourage confusion and dispersion. The first four bits are shuffled to the positions of the final four bits, and the last four bits are shuffled to the positions of the first four bits.

AES-256 Encryption:

Here, AES encryption is performed using the IV, Key, and the resulting shuffled Plaintext. The encrypted Text is then produced.

Appending the IV: The ciphertext is inserted together with IV. Because the IV will be with both the Sender and the Receiver and may be validated by the Receiver, appending the IV will aid in improving data integrity.

Base-64 Encoding:

In order to protect against the Cipher Text- Only and Plain Text- Only attacks, the IV added Cipher Text is subsequently encoded using Base-64.

Header Text Insertion:

Header Text is the specified text that is used to be attached to Base-64 Encoded Text before LSB Steganography is performed.

LSB Steganography Encoding:

LSB Steganography is performed where the following operations takes place:

i. Take a picture that you wish to use to conceal a message in order to load it.
ii. Convert your secret message to binary code, which consists of a string of 0s and 1s. Each character in the message is represented by an array of 8 bits, such as '01001000' for the letter 'H'.
iii. Dissect the image into individual, color-specific pixels, which are very little dots. Starting at the top-left pixel, this script runs.
iv. Start with your secret message's initial binary number and pixel.

v. Consider the pixel's color values, such as red, green, and blue. Each color's final digit, or least significant bit, should be changed to correspond to the binary digit in your message.

vi. If the message bit is ‘0’ then value of LSB of the colour of pixel must be ‘0’

vii. If the message bit is ‘1’ then value of LSB of the colour of pixel must be ‘1’.

viii. 9th bit is called as **“STATUS BIT”.** (3rd pixel’s blue LSB)

ix. 9th bit should be ‘0’ if the message continues or make it as ‘1’ if the message ends with that character.

x. Save the Encoding image with the secret message as the new file.

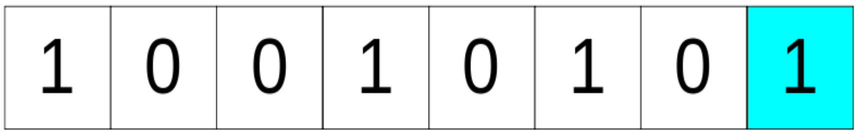


Fig 3.3: Message Bits

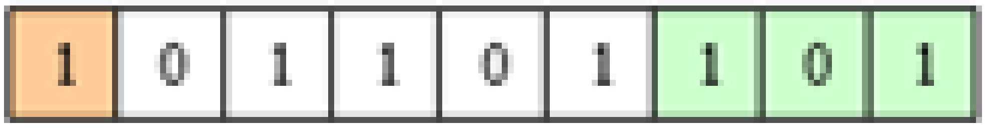


Fig 3.4: 3 Pixels RGB Values LSB

3.2 DECODING AND DECRYPTION STAGE

LSB Steganography Decoding:

The LSB Decoding of the data from the Saved Encoded Image. The steps are as follows:

i. The first step is to provide the decoded picture.

ii. The instruction starts by obtaining the image's pixel information. It begins at the top-left corner of the first pixel.

iii. Set up variables for the current pixel and the decoded message.

iv. Begin a loop that keeps going until a certain condition is satisfied. The least significant bit of the final color channel (often blue) in the current set of three pixels is being checked by this condition. This bit serves as a signal that the message has finished being encoded.

v. Take the least significant bit from each of the three color channels—red, green, and blue—for each group of three pixels (representing a letter). This series of bits creates a binary value.

vi. Produce an ASCII character from the binary value by converting it to ASCII. We start by removing any leading or following whitespace from the binary value. Then, it is transformed from base 2 (binary) to base 10 (decimal), resulting in an integer that is equivalent to an ASCII value. The decoded message is added with this ASCII character.

vii. Continue the procedure by moving on to the following character's next group of three pixels.

viii. Keep repeating this loop up until the signal indicating the end of the message—a collection of pixels where the final color channel's least important bit is odd (not even)—is located. The encoded message has now reached its conclusion.

ix. The decoded message is returned at the conclusion of the loop. In essence, it is the covert message that was encoded within the images.

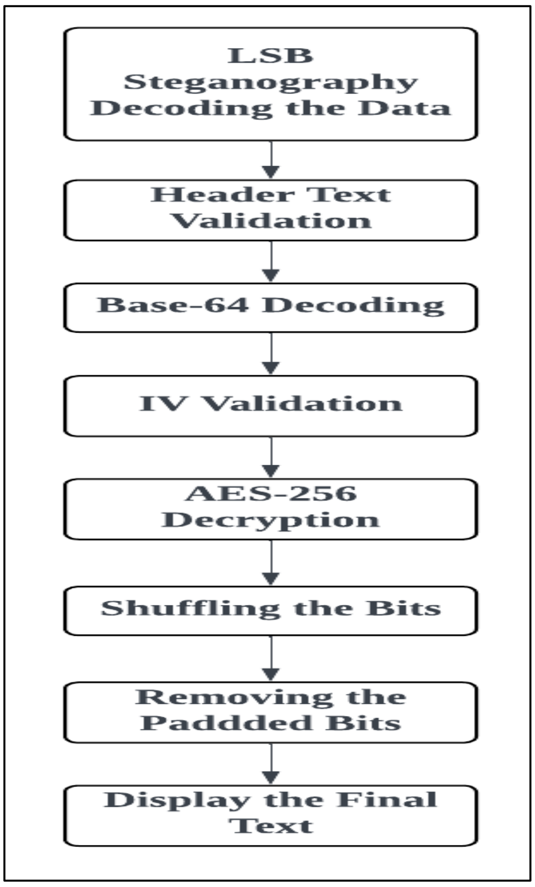


Fig 3.5: Decoding and Decrypting Stage Steps

Header Text Validation:

If the header Text retrieved from the decoded data matches the one shared by the sender, it is considered valid. The data's integrity will then be confirmed, allowing us to proceed.

Base-64 Decoding:

Following then, BASE-64 is used to decode the data. Every byte in the Base-64 text is transformed to its corresponding 6 bits throughout this decoding process.

Validating the IV:

To ensure that the right data format is received, the IV retrieved from the resultant Data is then compared to the IV provided by the Sender. If it is solely verified, AES-256 decryption occurs.

AES-256 Decryption:

Here, AES Decryption is performed using the IV, Key, and the resulting Decoded Text. The Plain Text is then produced.

Bit shuffling:

When the Padded Plaintext is translated to ASCII and the 8 Bits are created for each byte, bit shuffle is utilized to encourage confusion and dispersion. The first four bits are shuffled to the positions of the final four bits, and the last four bits are shuffled to the positions of the first four bits.

Removing the Padded Bits:

Then, the padding based on AES – Block Size is removed and the original plaintext is obtained and is then Displayed.

ENCRYPTION AND ENCODING PHASE WITH ILLUSTRATION:

S.NO.	LEVEL OF PROCESSING	RESULTS
1.	Image Path	hello.png
2.	Plaintext	Anurag University
3.	Password	123456789789
4.	Padded Plaintext	AnuragUniversity\x0f\x0f\x0f\x0f\x0f\x0f\x0f\x0f\x0f\x0f\x0f\x0f\x0f\x0f\x0f\x0f
5.	Key	b\xa8\x61\x96\x9c\x12\x8c\xbc\\x8c0}\r\xd0\xde\xae\xd8\xd1\xc9[$\xc1\xefgM\x3\xadrezu\x8d
6.	IV	b'\x17\x1f\xa5\xfa\x8a\xbf\xbf\xedT\x1c\xb7:W\xc5\x99
7.	Shuffled Text	b'\xbafFYsdk\x9e\x95\x05\xb1v\x9b66\x16\xc8/\x15\xf0\xacmd\x13\x73\xdc\x8ds\xee
8.	Cipher Text appended with IV	b\x17\x2F\xa5\xfa\xBa\xbf\xbf\xedT\x1c\xb7:WW\xc5\x98\xbafFYsdk\x9e\x95[\xd5\xb1v\x96&6\x16\xc8/\x15\x70\xcd\x13\3\cdsvee
9.	Base-64 Encoded Text	Fx+1+og/+1UHLc6V07FkLpmRlLzZGuelVvvsXabJjYwyCBWBKxtZBPz311z7m41
10.	Header Text	HeaderText
11.	Encoded Text	HeaderTextFx+L+oq/v+1UHLc6V87FkLpmRl1z2GuelVvVsXabJjYyc8V8XxtZBPz311z7n4i
12.	Encoded Image Path	hello-encoded.png

Table 3.1: Encryption & Encoding Results

DECODING AND DECRYPTION PHASE WITH ILLUSTRATION:

S.NO.	LEVEL OF PROCESSING	RESULTS
1.	Image Path	hello-encoded.png
2.	Password	123456789789
3.	Decoded Text	HeaderTextFx+L+oq/v+1UHLc6V87FkLpmRl1z2GuelVvVsXabJjYyc8V8XxtZBPz311z7n4i
4.	Header Text Validation	HeaderText
5.	Decoded Text after Removing Header Text	Fx+1+og/+1UHLc6V07FkLpmRlLzZGuelVvvsXabJjYwyCBWBKxtZBPz311z7m41
6.	Base-64 Decoded Text	b\x17\x2F\xa5\xfa\xBa\xbf\xbf\xedT\x1c\xb7:WW\xc5\x98\xbafFYsdk\x9e\x95[\xd5\xb1v\x96&6\x16\xc8/\x15\x70\xcd\x13\3\cdsvee
7.	Removing the IV	b'\xbafFYsdk\x9e\x95\x05\xb1v\x9b66\x16\xc8/\x15\xf0\xacmd\x13\x73\xdc\x8ds\xee
8.	Shuffled Text	AnuragUniversity\x0f\x0f\x0f\x0f\x0f\x0f\x0f\x0f\x0f\x0f\x0f\x0f\x0f\x0f\x0f\x0f
9.	Plain Text	Anurag University

Table 3.2: Decoding & Decryption Results

4. IMPLEMENTATION

We have two Program Files that are used in our Project which are **project.py** and **metrics.py**.

Program File: **project.py**

Input: Image Path, Plain Text, Password.

Output: Encoded Image Path

List of what all are sent from sender to receiver: Key generated from the Password , IV, Encoded Image Path.

4.1 FUNCTIONALITIES

1) **convertToRGBAndSave:** In this function, we take the Image Path as input and we will check if it RGB Mode, If it is not we will try to convert it if the image is GreyScale, CMYK, RGBA and we will display the Path.

2) **getPixelCount:** If the image is of RGB Mode, then height and width of the image is displayed. Along with the number of pixels present in the given Image.

3) **is_png_image:** This functionality is used to check whether the given image is PNG or not. If not, error is displayed. If yes, the Steganography process continues.

4) **show_image:** The image is displayed using the img.show() where both the Original Image as well as Encoded Image are displayed using this function.

5) **encrypt:** AES-256 Encryption takes place in this function. Along with, the Key generation from the Password, Padding the PlainText, Shuffling the bits, Encryption based on AES, Appending the IV to the Cipher Text,

Base-64 Encoding, Header Text Insertion are also performed.

6) **text_header_bits:** The Header Text must be converted to the binary bits.

7) **shuffle_bits:** Here, in this function Shuffling the bits is performed where the first 4 bits are shuffled to the last 4 bits and also vice versa.

8) **deshuffle_bits:** Here, in this function De-shuffling of the bits is performed where the same logic as that of shuffle_bits is performed where the first 4 bits are shuffled to that of the last 4 bits.

9) **decrypt:** Here, AES-256 Decryption function takes place where Header Text validation, Base-64 Decoding, Removing the IV and validating it, Decrypting it with AES, Deshuffling the bits is performed.

10) **encodeImage:** In this function, Least Significant Bit (LSB) Encoding takes place where the data is encoded into the Image and the Encoded Image Path is saved.

11) **decodeImage:** In this function, Least Significant Bit (LSB) Decoding takes place where the data is decoded from the Image and the Decryption takes place.

4.2 ATTRIBUTES

1) **DEBUG:** A boolean indicating whether to display debug information.

2) **console:** An instance of the Rich library's Console for advanced text output.

3) **headertext:** A string containing header text added before encoding the image.

4) **original_image_path**: Stores the path to the original image.
5) **message:** Stores the message to be hidden in the image.
6) **plaintext_bits:** Stores the binary representation of the message.
7) **password:** Stores the encryption/decryption password.
8) **cipher:** Stores the encrypted or encoded message.
9) **image:** Represents the image being processed.
10) **decrypted:** Stores the decrypted message (if applicable).
11) **total_pixels:** Stores the total number of pixels in the image.
12) **total_cipher_bits:** Stores the binary representation of the encoded message.
13) **pixels_used:** Indicates the number of pixels used to encode the message.
14) **newimg:** Represents a copy of the image used for encoding.

Program File: **metrics.py**

Input: Original Image, Encoded Image

Output: PSNR, MSE and Histogram Visualization

4.3 FUNCTIONALITIES

1) **resize_encoded_image:** This function resizes the encoded image to match the dimensions of the original image. It is essential to ensure both images have the same dimensions for accurate comparisons.

2) **calculate_psnr_mse:** These functions calculate the Peak Signal-to-Noise

Ratio (PSNR) and Mean Squared Error (MSE) between the original and encoded images. The first function is used to calculate PSNR and MSE, while the second function returns these metrics along with information about changed pixels.

3) **format_file_size:** This function calculates and formats the file size (in MB or KB) of the specified image file. It is used to provide information about the sizes of the original and encoded images.

4.4 ATTRIBUTES

1) **original_image_path:** A variable storing the file path to the original image.

2) **encoded_image_path:** A variable storing the file path to the encoded image.

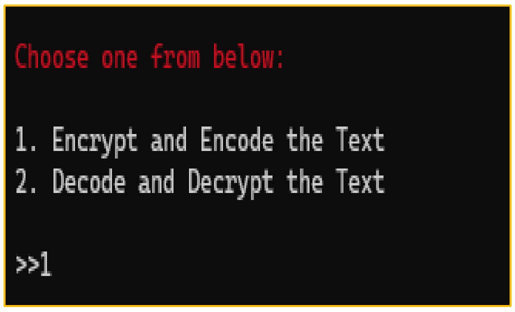
3) **text_size_bytes:** A variable representing the size of text data in bytes.

4) **Various variables** (psnr, mse, original_image_size, encoded_image_size, and histograms) are used to store calculated metrics and image data for further processing and visualization.

4.5 SAMPLE CODE

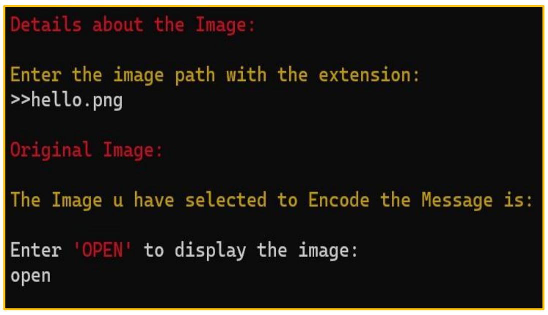
```
#AES-256 Encryption
def encrypt(key, source, encode=True,print_output=True):
    start_time = time.time()        #To start the encryption
    key = SHA256.new(key).digest()
    IV = Random.new().read(AES.block_size)
    encryptor = AES.new(key, AES.MODE_CBC, IV)
    # Calculating padding length
    padding_length = AES.block_size - (len(source) % AES.block_size)
    # Pad the source data
    source += bytes([padding_length]) * padding_length
        shuffle = shuffle_bits(source)    #Calling the shuffling function
        encrypttext=encryptor.encrypt(shuffle)
        print(f"\nAfter Shuffling the bits, the shuffled text is:{shuffle}")
        print(f"\n[red]Details about encrypting the shuffled text using AES-256
encryption :[/red]")
        print(f"\nThe shuffled bits are encrypted and the ciphertext is:
[green]{encrypttext} [/green]")
        print(f"\nLength of CipherText in bytes: [blue]{len(encrypttext)}[/blue]")  #
Print the encrypted text
        print(f"Length of CipherText in bits: [blue]{len(encrypttext)*8}[/blue]")
        print("\n[yellow]Cipher Text bits:[/yellow]")
        hex_string = binascii.hexlify(encrypttext)      #Converting to binary bits
        binary_string = bin(int(hex_string, 16))[2:].zfill(len(hex_string) * 4)
        print("\n[")
        for i in range(0, len(binary_string), 8):
            row = binary_string[i:i + 8]
            print("    " + ", ".join(map(str, row)) + ",")
        print("]")
        print(f"\n[yellow]The above shuffled CipherText is appended to the
IV.[/yellow]")
        data= IV + encrypttext
        print(f"\n The resultant Cipher Text after it is appended to IV is : ")
        print(f"[green]{IV}[/green] + [red]{encrypttext}[/red]")
        print(f"\n[yellow]{data}[/yellow]")
    if encode:
        print("\nThe obtained ciphertext is encoded using BASE-64 encoding: \n")
        final_ciphertext= base64.b64encode(data).decode()     #Base-64 Encoding
        print(f"[green]{base64.b64encode(data).decode()}[/green]")
        ascii_values = [ord(char) for char in final_ciphertext]
        binary_values = [format(value, '08b') for value in ascii_values]
        bit_string = ''.join(binary_values)
        print("[yellow]\nThe BASE-64 encoded ciphertext bits: [/yellow] ")
        print("\n[")
        for i in range(0, len(bit_string), 8):
            row = bit_string[i:i + 8]
            print("[green]    " + ", ".join(map(str, row)) + ",")
        print("]")
        print(f"[yellow]\nLength of the BASE-64 encoded text in bits[/yellow] :
{len(bit_string)}")
        encryption_time = time.time() - start_time  # Calculating the encryption time
        print(f"\n[red]Encryption Time: {encryption_time:.5f} seconds[/red]")
        return base64.b64encode(data).decode()
    else:
        return data

#Function used to shuffle the bits
def shuffle_bits(data, print_output=True):
    num_bytes = len(data)
    num_bits = num_bytes * 8
    shuffled_data = bytearray(num_bytes)
    for i in range(num_bytes):
        original_byte = data[i]
        shuffled_byte = 0
        # Shuffle the bits within the byte
        for j in range(4):
            shuffled_byte |= ((original_byte >> (4 + j)) & 1) << j     #To shuffle to
rightmost place
            shuffled_byte |= ((original_byte >> j) & 1) << (j + 4)  #To shuffle to left
most place
        shuffled_data[i] = shuffled_byte
    if print_output:
        print("\n[yellow]Original Bits:[/yellow]")
        original_bits = ''.join(format(byte, '08b') for byte in data)
        print("\n[")
        for i in range(0, num_bits, 8):
            row = original_bits[i:i + 8]
            print("    " + ", ".join(map(str, row)) + ",")
        print("]")
        print("\n[yellow]\nShuffled Bits:[/yellow]")
        shuffled_bits = ''.join(format(byte, '08b') for byte in shuffled_data)  # Convert
to binary string
        print("[")
        for i in range(0, num_bits, 8):
            row = shuffled_bits[i:i + 8]
            print("   " + ", ".join(map(str, row)) + ",")
        print("]")
    return bytes(shuffled_data)

#Code used to deshuffle the bits
def deshuffle_bits(data, print_output=True):
    num_bytes = len(data)
    num_bits = num_bytes * 8
    original_data = bytearray(num_bytes)
    for i in range(num_bytes):
        shuffled_byte = data[i]
        original_byte = 0
        for j in range(4):
            original_byte |= ((shuffled_byte >> (4 + j)) & 1) << j
            original_byte |= ((shuffled_byte >> j) & 1) << (j + 4)
        original_data[i] = original_byte
    return bytes(original_data)

#Code for AES-256 Decryption function
def decrypt(key, source, decode=True):
    start_time = time.time()  #To start the decryption time
    if decode:
        source = base64.b64decode(source.encode())
    key = SHA256.new(key).digest()
    IV = source[:AES.block_size]
    decryptor = AES.new(key, AES.MODE_CBC, IV)
    data = decryptor.decrypt(source[AES.block_size:])
    print(f"[red]\nDetails about decrypting the message: [/red]")
    print(f"\nThe text is in BASE-64 Encoding format. So after Decoding it, The
Text is: [green]{source}[/green]")
    print(f"\nKey is [green]{key}[/green]")
    print(f"\nIV is [green]{IV}[/green]")
    print(f"\nThe Decrypted text is : {data} ")
    # Calling the Deshuffling function
    data = deshuffle_bits(data)
    # Removing the padding bits
    padding_length = data[-1]
    data = data[:-padding_length]
    decryption_time = time.time() - start_time  # Calculating decryption time
    print(f"\n[red]Decryption Time: {decryption_time:.5f} seconds[/red]")
    return data

#Function used for LSB EMBEDDING STEGANOGRAPHY FOR ENCODING
THE IMAGE
def encodeImage(image, message, filename):
    with console.status("[green]Encoding image......") as status:
        try:
            width, height = image.size
            pix = image.getdata()
            current_pixel = 0
            tmp = 0
            x = 0
            y = 0
            original_pixels = list(image.getdata())
            for ch in message:
                binary_value = format(ord(ch), '08b')
                p1 = pix[current_pixel]
                p2 = pix[current_pixel + 1]
                p3 = pix[current_pixel + 2]
                three_pixels = [val for val in p1 + p2 + p3]
                for i in range(0, 8):
                    current_bit = binary_value[i]
                    if current_bit == '0':
                        if three_pixels[i] % 2 != 0:
                            three_pixels[i] = three_pixels[i] - 1 if three_pixels[i] == 255 else
three_pixels[i] + 1
                    elif current_bit == '1':
                        if three_pixels[i] % 2 == 0:
                            three_pixels[i] = three_pixels[i] - 1 if three_pixels[i] == 255 else
three_pixels[i] + 1
                current_pixel += 3
                tmp += 1
                if tmp == len(message):
                    if three_pixels[-1] % 2 == 0:
                        three_pixels[-1] = three_pixels[-1] - 1 if three_pixels[-1] == 255
else three_pixels[-1] + 1
                else:
                    if three_pixels[-1] % 2 != 0:
                        three_pixels[-1] = three_pixels[-1] - 1 if three_pixels[-1] == 255
else three_pixels[-1] + 1
                three_pixels = tuple(three_pixels)
                st = 0
                end = 3
                for i in range(0, 3):
                    image.putpixel((x, y), three_pixels[st:end])
                    st += 3
                    end += 3
                    if x == width - 1:
                        x = 0
                        y += 1
                    else:
                        x += 1
                print("\n\n")
                print(f"[red]Details while Encoding the message to the image: [/red]")
                print("[yellow]Original File: [u]%s[/u][/yellow]" % filename)
                encoded_filename = filename.split('.')[0] + "-encoded.png"
                image.save(encoded_filename)
                print("[green]Image encoded and saved as [u][bold]%s[/green][/u][/bold]"
% encoded_filename)
                # Compare original and encoded images
                encoded_image = Image.open(encoded_filename)
                encoded_pixels = list(encoded_image.getdata())
                num_pixels_changed = sum([1 for orig, enc in zip(original_pixels,
encoded_pixels) if orig != enc])
                total_pixels = len(original_pixels)
                percent_change = (num_pixels_changed / total_pixels) * 100
                print(f"\n[red]Comparison between Original and Encoded Image:[/red]")
                print(f"\n[yellow]Total Pixels in the Image:[/yellow] {total_pixels}")
                print(f"[yellow]\nNumber of Pixels Changed:[/yellow]
{num_pixels_changed}")
                print(f"[yellow]\nPercentage Change:[/yellow] {percent_change:.10f}%")
                if num_pixels_changed > 0:
                    draw = ImageDraw.Draw(encoded_image)
                    line_color = (255, 0, 0)  # Red color
                    line_thickness = 20
                    draw.line((0, 0, num_pixels_changed, 0), fill=line_color,
width=line_thickness)
                    encoded_image_with_line_filename = filename.split('.')[0] + "-encoded-
with-line.png"
                    encoded_image.save(encoded_image_with_line_filename)
                    encoded_image.show()
                encoded_filename = filename.split('.')[0] + "-encoded.png"
                image.save(encoded_filename)
                print("[green]Image encoded and saved as [u][bold]%s[/green][/u][/bold]"
% encoded_filename)
        except Exception as e:
                print("[red]An error occurred - [/red]%s" % e)
                sys.exit(0)

#Function for LSB EMBEDDING STEGANOGRAPHY DECODING THE
IMAGE
def decodeImage(image):
    with console.status("[green]Decoding image.....") as status:
        try:
                pix = image.getdata()
                current_pixel = 0
                decoded = ""
                while True:
                    binary_value = ""
                    p1 = pix[current_pixel]
                    p2 = pix[current_pixel + 1]
                    p3 = pix[current_pixel + 2]
                    three_pixels = [val for val in p1 + p2 + p3]

                    for i in range(0, 8):
                    if three_pixels[i] % 2 == 0:
                        binary_value += "0"
                    elif three_pixels[i] % 2 != 0:
                        binary_value += "1"

                    binary_value.strip()
                    ascii_value = int(binary_value, 2)
                    decoded += chr(ascii_value)
                    current_pixel += 3
                    if three_pixels[-1] % 2 != 0:
                    break
                return decoded
        except Exception as e:
                print("[red]An error occurred - [/red]%s" % e)
                sys.exit(0)
```

4.6 EXPERIMENT SCREENSHOTS: project.py



```
Choose one from below:

1. Encrypt and Encode the Text
2. Decode and Decrypt the Text

>>1
```

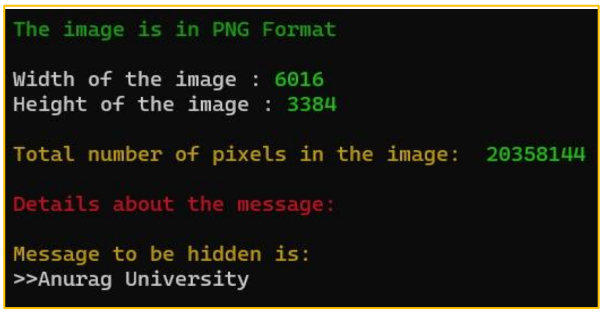
Fig 4.1: Development Environment 1



```
Details about the Image:

Enter the image path with the extension:
>>hello.png

Original Image:

The Image u have selected to Encode the Message is:

Enter 'OPEN' to display the image:
open
```

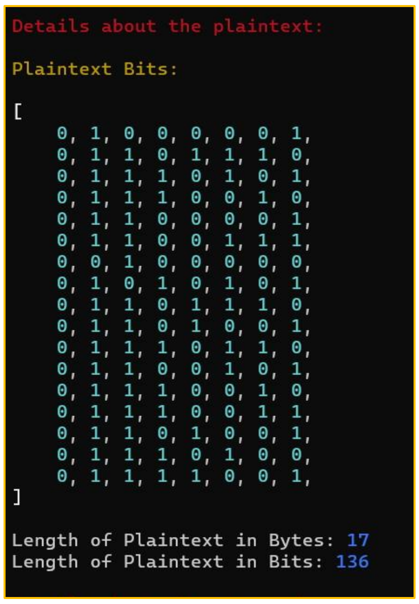
Fig 4.2: Development Environment 2

Fig 4.3: Image that is Selected



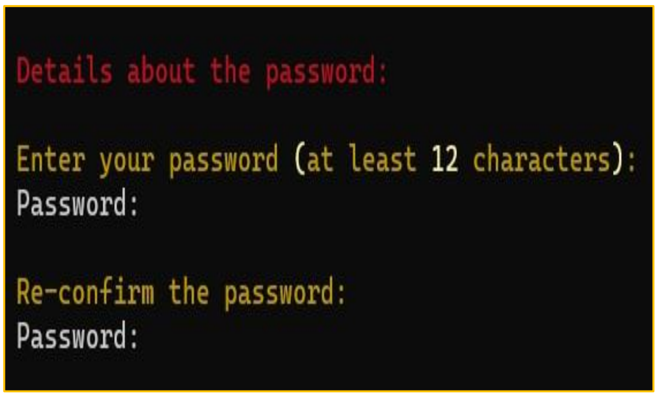
```
The image is in PNG Format

Width of the image : 6016
Height of the image : 3384

Total number of pixels in the image:  20358144

Details about the message:

Message to be hidden is:
>>Anurag University
```

Fig 4.4: Development Environment 3



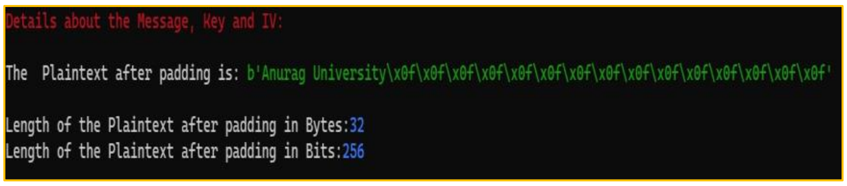
```
Details about the plaintext:

Plaintext Bits:

[
    0, 1, 0, 0, 0, 0, 0, 1,
    0, 1, 1, 0, 1, 1, 1, 0,
    0, 1, 1, 1, 0, 1, 0, 1,
    0, 1, 1, 1, 0, 0, 1, 0,
    0, 1, 1, 0, 0, 0, 0, 1,
    0, 1, 1, 0, 0, 1, 1, 1,
    0, 0, 1, 0, 0, 0, 0, 0,
    0, 1, 0, 1, 0, 1, 0, 1,
    0, 1, 1, 0, 1, 1, 1, 0,
    0, 1, 1, 0, 1, 0, 0, 1,
    0, 1, 1, 1, 0, 1, 1, 0,
    0, 1, 1, 0, 0, 1, 0, 1,
    0, 1, 1, 1, 0, 0, 1, 0,
    0, 1, 1, 1, 0, 0, 1, 1,
    0, 1, 1, 0, 1, 0, 0, 1,
    0, 1, 1, 1, 0, 1, 0, 0,
    0, 1, 1, 1, 1, 0, 0, 1,
]

Length of Plaintext in Bytes: 17
Length of Plaintext in Bits: 136
```

Fig 4.5: Development Environment 4



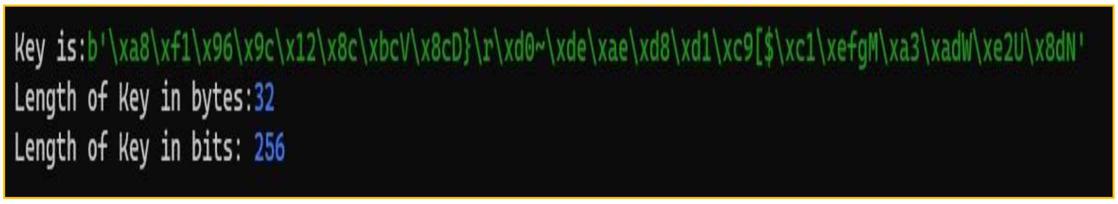
```
Details about the password:

Enter your password (at least 12 characters):
Password:

Re-confirm the password:
Password:
```

Fig 4.6: Development Environment 5



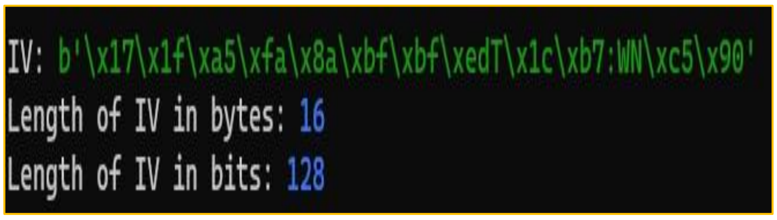
```
Details about the Message, Key and IV:

The  Plaintext after padding is: b'Anurag University\x0f\x0f\x0f\x0f\x0f\x0f\x0f\x0f\x0f\x0f\x0f\x0f\x0f\x0f\x0f'

Length of the Plaintext after padding in Bytes:32
Length of the Plaintext after padding in Bits:256
```

Fig 4.7: Development Environment 6



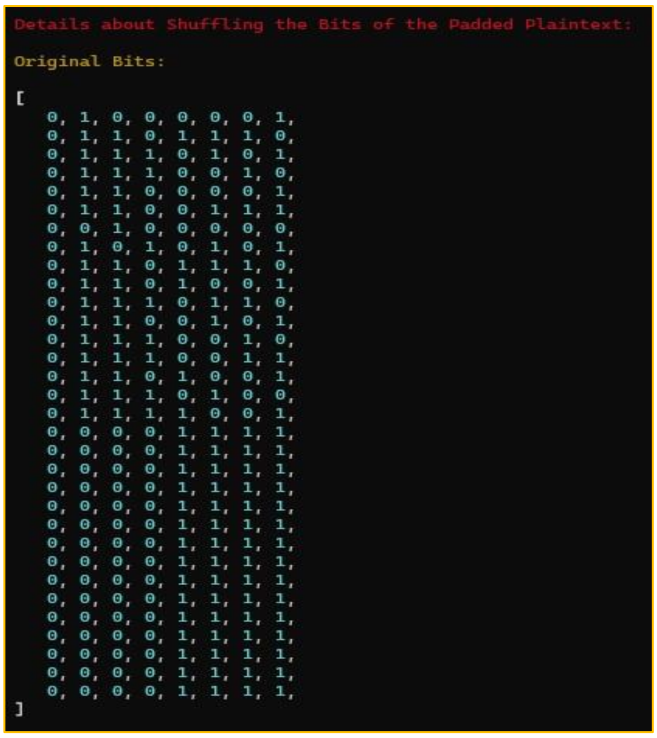
```
Key is:b'\xa8\xf1\x96\x9c\x12\x8c\xbcV\x8cD}\r\xd0~\xde\xae\xd8\xd1\xc9[$\xc1\xefgM\xa3\xadW\xe2U\x8dN'
Length of Key in bytes:32
Length of Key in bits: 256
```

Fig 4.8: Development Environment 7



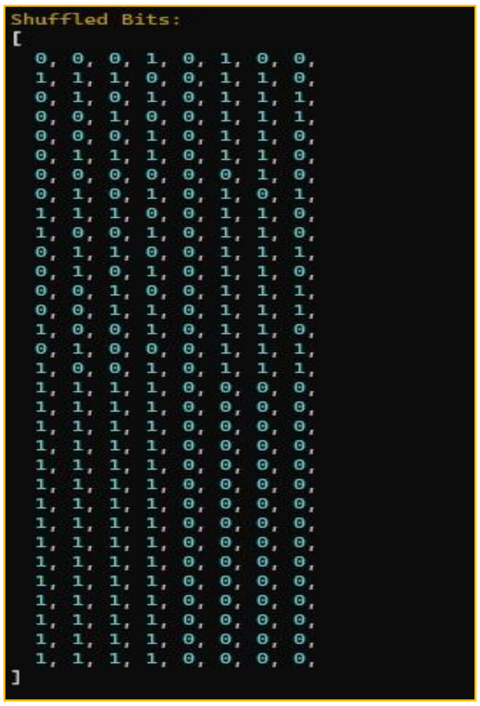
```
IV: b'\x17\x1f\xa5\xfa\x8a\xbf\xbf\xedT\x1c\xb7:WN\xc5\x90'
Length of IV in bytes: 16
Length of IV in bits: 128
```

Fig 4.9: Development Environment 8



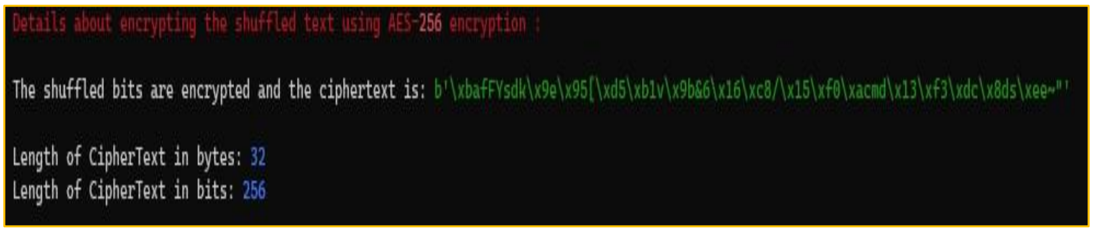
```
Details about Shuffling the Bits of the Padded Plaintext:

Original Bits:

[
    0, 1, 0, 0, 0, 0, 0, 1,
    0, 1, 1, 0, 1, 1, 1, 0,
    0, 1, 1, 1, 0, 1, 0, 1,
    0, 1, 1, 1, 0, 0, 1, 0,
    0, 1, 1, 0, 0, 0, 0, 1,
    0, 1, 1, 0, 0, 1, 1, 1,
    0, 0, 1, 0, 0, 0, 0, 0,
    0, 1, 0, 1, 0, 1, 0, 1,
    0, 1, 1, 0, 1, 1, 1, 0,
    0, 1, 1, 0, 1, 0, 0, 1,
    0, 1, 1, 1, 0, 1, 1, 0,
    0, 1, 1, 0, 0, 1, 0, 1,
    0, 1, 1, 1, 0, 0, 1, 0,
    0, 1, 1, 1, 0, 0, 1, 1,
    0, 1, 1, 0, 1, 0, 0, 1,
    0, 1, 1, 1, 0, 1, 0, 0,
    0, 1, 1, 1, 1, 0, 0, 1,
    0, 0, 0, 0, 1, 1, 1, 1,
    0, 0, 0, 0, 1, 1, 1, 1,
    0, 0, 0, 0, 1, 1, 1, 1,
    0, 0, 0, 0, 1, 1, 1, 1,
    0, 0, 0, 0, 1, 1, 1, 1,
    0, 0, 0, 0, 1, 1, 1, 1,
    0, 0, 0, 0, 1, 1, 1, 1,
    0, 0, 0, 0, 1, 1, 1, 1,
    0, 0, 0, 0, 1, 1, 1, 1,
    0, 0, 0, 0, 1, 1, 1, 1,
    0, 0, 0, 0, 1, 1, 1, 1,
    0, 0, 0, 0, 1, 1, 1, 1,
    0, 0, 0, 0, 1, 1, 1, 1,
    0, 0, 0, 0, 1, 1, 1, 1,
    0, 0, 0, 0, 1, 1, 1, 1,
]
```

Fig 4.10: Development Environment 9



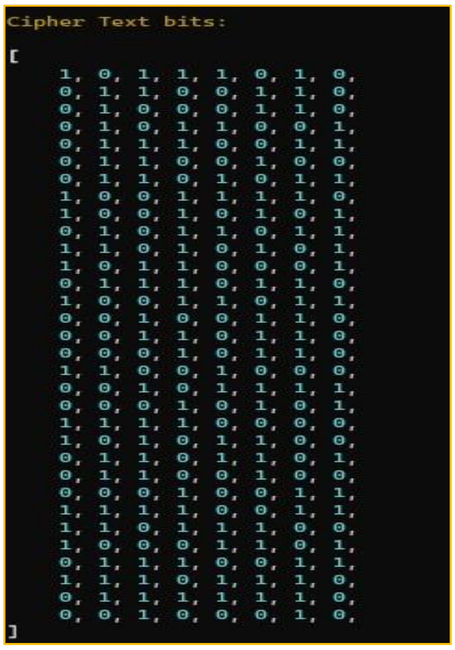
```
Shuffled Bits:
[
   0, 0, 0, 1, 0, 1, 0, 0,
   1, 1, 1, 0, 0, 1, 1, 0,
   0, 1, 0, 1, 0, 1, 1, 1,
   0, 0, 1, 0, 0, 1, 1, 1,
   0, 0, 0, 1, 0, 1, 1, 0,
   0, 1, 1, 1, 0, 1, 1, 0,
   0, 0, 0, 0, 0, 0, 1, 0,
   0, 1, 0, 1, 0, 1, 0, 1,
   1, 1, 1, 0, 0, 1, 1, 0,
   1, 0, 0, 1, 0, 1, 1, 0,
   0, 1, 1, 0, 0, 1, 1, 1,
   0, 1, 0, 1, 0, 1, 1, 0,
   0, 0, 1, 0, 0, 1, 1, 1,
   0, 0, 1, 1, 0, 1, 1, 1,
   1, 0, 0, 1, 0, 1, 1, 0,
   0, 1, 0, 0, 0, 1, 1, 1,
   1, 0, 0, 1, 0, 1, 1, 1,
   1, 1, 1, 1, 0, 0, 0, 0,
   1, 1, 1, 1, 0, 0, 0, 0,
   1, 1, 1, 1, 0, 0, 0, 0,
   1, 1, 1, 1, 0, 0, 0, 0,
   1, 1, 1, 1, 0, 0, 0, 0,
   1, 1, 1, 1, 0, 0, 0, 0,
   1, 1, 1, 1, 0, 0, 0, 0,
   1, 1, 1, 1, 0, 0, 0, 0,
   1, 1, 1, 1, 0, 0, 0, 0,
   1, 1, 1, 1, 0, 0, 0, 0,
   1, 1, 1, 1, 0, 0, 0, 0,
   1, 1, 1, 1, 0, 0, 0, 0,
   1, 1, 1, 1, 0, 0, 0, 0,
   1, 1, 1, 1, 0, 0, 0, 0,
   1, 1, 1, 1, 0, 0, 0, 0,
]
```

Fig 4.11: Development Environment 10



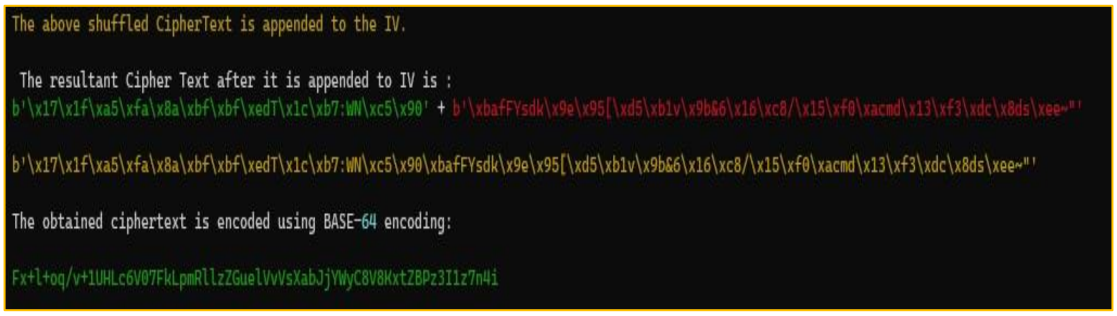
```
Details about encrypting the shuffled text using AES-256 encryption :

The shuffled bits are encrypted and the ciphertext is: b'\xbafFYsdk\x9e\x95[\xd5\xb1v\x9b&6\x16\xc8/\x15\xf0\xacmd\x13\xf3\xdc\x8ds\xee~"'

Length of CipherText in bytes: 32
Length of CipherText in bits: 256
```

Fig 4.12: Development Environment 11



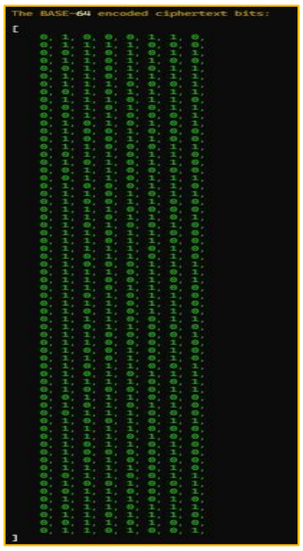
```
Cipher Text bits:

[
    1, 0, 1, 1, 1, 0, 1, 0,
    0, 1, 1, 0, 0, 1, 1, 0,
    0, 1, 0, 0, 0, 1, 1, 0,
    0, 1, 0, 1, 1, 0, 0, 1,
    0, 1, 1, 1, 0, 0, 1, 1,
    0, 1, 1, 0, 0, 1, 0, 0,
    0, 1, 1, 0, 1, 0, 1, 1,
    1, 0, 0, 1, 1, 1, 1, 0,
    1, 0, 0, 1, 0, 1, 0, 1,
    0, 1, 0, 1, 1, 0, 1, 1,
    1, 1, 0, 1, 0, 1, 0, 1,
    1, 0, 1, 1, 0, 0, 0, 1,
    0, 1, 1, 1, 0, 1, 1, 0,
    1, 0, 0, 1, 1, 0, 1, 1,
    0, 0, 1, 0, 0, 1, 1, 0,
    0, 0, 1, 1, 0, 1, 1, 0,
    0, 0, 0, 1, 0, 1, 1, 0,
    1, 1, 0, 0, 1, 0, 0, 0,
    0, 0, 1, 0, 1, 1, 1, 1,
    0, 0, 0, 1, 0, 1, 0, 1,
    1, 1, 1, 1, 0, 0, 0, 0,
    1, 0, 1, 0, 1, 1, 0, 0,
    0, 1, 1, 0, 1, 1, 0, 1,
    0, 1, 1, 0, 0, 1, 0, 0,
    0, 0, 0, 1, 0, 0, 1, 1,
    1, 1, 1, 1, 0, 0, 1, 1,
    1, 1, 0, 1, 1, 1, 0, 0,
    1, 0, 0, 0, 1, 1, 0, 1,
    0, 1, 1, 1, 0, 0, 1, 1,
    1, 1, 1, 0, 1, 1, 1, 0,
    0, 1, 1, 1, 1, 1, 1, 0,
    0, 0, 1, 0, 0, 0, 1, 0,
]
```

Fig 4.13: Development Environment 12

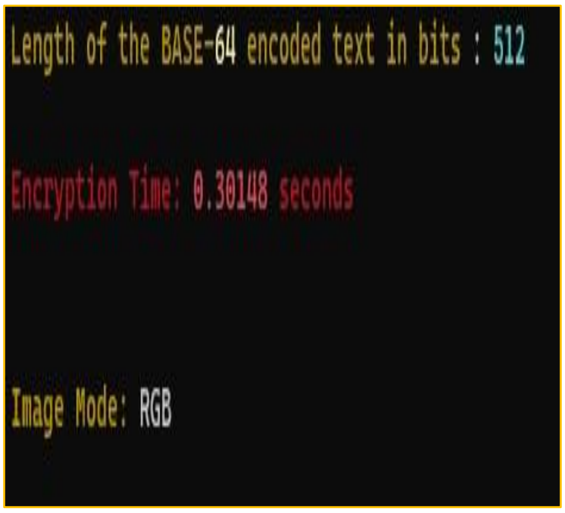


```
The above shuffled CipherText is appended to the IV.

 The resultant Cipher Text after it is appended to IV is :
b'\x17\x1f\xa5\xfa\x8a\xbf\xbf\xedT\x1c\xb7:WN\xc5\x90' + b'\xbafFYsdk\x9e\x95[\xd5\xb1v\x9b&6\x16\xc8/\x15\xf0\xacmd\x13\xf3\xdc\x8ds\xee~"'

b'\x17\x1f\xa5\xfa\x8a\xbf\xbf\xedT\x1c\xb7:WN\xc5\x90\xbafFYsdk\x9e\x95[\xd5\xb1v\x9b&6\x16\xc8/\x15\xf0\xacmd\x13\xf3\xdc\x8ds\xee~"'

The obtained ciphertext is encoded using BASE-64 encoding:

Fx+l+oq/v+1UHLc6V07FkLpmRllzZGuelVvVsXabJjYWyC8V8KxtZBPz3I1z7n4i
```

Fig 4.14: Development Environment 13

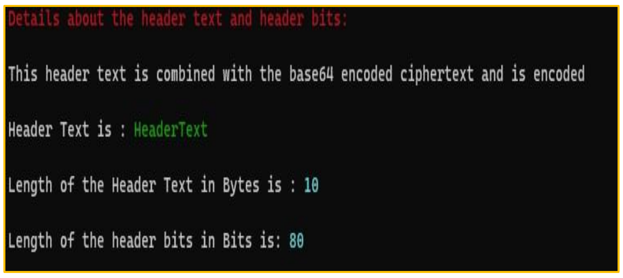


```
The BASE-64 encoded ciphertext bits:
[
    0, 1, 0, 0, 0, 1, 1, 0,
    0, 1, 1, 1, 1, 0, 0, 0,
    0, 0, 1, 0, 1, 0, 1, 1,
    0, 1, 1, 0, 1, 1, 0, 0,
    0, 0, 1, 0, 1, 0, 1, 1,
    0, 1, 1, 0, 1, 1, 1, 1,
    0, 1, 1, 1, 0, 0, 0, 1,
    0, 0, 1, 0, 1, 1, 1, 1,
    0, 1, 1, 1, 0, 1, 1, 0,
    0, 0, 1, 0, 1, 0, 1, 1,
    0, 0, 1, 1, 0, 0, 0, 1,
    0, 1, 0, 1, 0, 1, 0, 1,
    0, 1, 0, 0, 1, 0, 0, 0,
    0, 1, 0, 0, 1, 1, 0, 0,
    0, 1, 1, 0, 0, 0, 1, 1,
    0, 0, 1, 1, 0, 1, 1, 0,
    0, 1, 0, 1, 0, 1, 1, 0,
    0, 0, 1, 1, 0, 0, 0, 0,
    0, 0, 1, 1, 0, 1, 1, 1,
    0, 1, 0, 0, 0, 1, 1, 0,
    0, 1, 1, 0, 1, 0, 1, 1,
    0, 1, 0, 0, 1, 1, 0, 0,
    0, 1, 1, 1, 0, 0, 0, 0,
    0, 1, 1, 0, 1, 1, 0, 1,
    0, 1, 0, 1, 0, 0, 1, 0,
    0, 1, 1, 0, 1, 1, 0, 0,
    0, 1, 1, 0, 1, 1, 0, 0,
    0, 1, 1, 1, 1, 0, 1, 0,
    0, 1, 0, 1, 1, 0, 1, 0,
    0, 1, 0, 0, 0, 1, 1, 1,
    0, 1, 1, 1, 0, 1, 0, 1,
    0, 1, 1, 0, 0, 1, 0, 1,
    0, 1, 1, 0, 1, 1, 0, 0,
    0, 1, 0, 1, 0, 1, 1, 0,
    0, 1, 1, 1, 0, 1, 1, 0,
    0, 1, 0, 1, 0, 1, 1, 0,
    0, 1, 1, 1, 0, 0, 1, 1,
    0, 1, 0, 1, 1, 0, 0, 0,
    0, 1, 1, 0, 0, 0, 0, 1,
    0, 1, 1, 0, 0, 0, 1, 0,
    0, 1, 0, 0, 1, 0, 1, 0,
    0, 1, 1, 0, 1, 0, 1, 0,
    0, 1, 0, 1, 1, 0, 0, 1,
    0, 1, 0, 1, 0, 1, 1, 1,
    0, 1, 1, 1, 1, 0, 0, 1,
    0, 1, 0, 0, 0, 0, 1, 1,
    0, 0, 1, 1, 1, 0, 0, 0,
    0, 1, 0, 1, 0, 1, 1, 0,
    0, 0, 1, 1, 1, 0, 0, 0,
    0, 1, 0, 0, 1, 0, 1, 1,
    0, 1, 1, 1, 1, 0, 0, 0,
    0, 1, 1, 1, 0, 1, 0, 0,
    0, 1, 0, 1, 1, 0, 1, 0,
    0, 1, 0, 0, 0, 0, 1, 0,
    0, 1, 0, 1, 0, 0, 0, 0,
    0, 1, 1, 1, 1, 0, 1, 0,
    0, 0, 1, 1, 0, 0, 1, 1,
    0, 1, 0, 0, 1, 0, 0, 1,
    0, 0, 1, 1, 0, 0, 0, 1,
    0, 1, 1, 1, 1, 0, 1, 0,
    0, 0, 1, 1, 0, 1, 1, 1,
    0, 1, 1, 0, 1, 1, 1, 0,
    0, 0, 1, 1, 0, 1, 0, 0,
    0, 1, 1, 0, 1, 0, 0, 1,
]
```



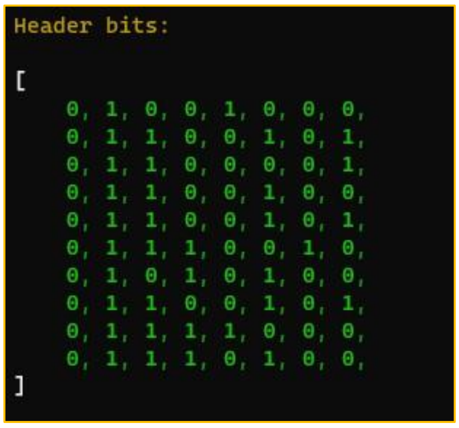
```
Length of the BASE-64 encoded text in bits : 512

Encryption Time: 0.30148 seconds

Image Mode: RGB
```

Fig 4.15: Development Environment 14 Fig 4.16: Development Environment 15



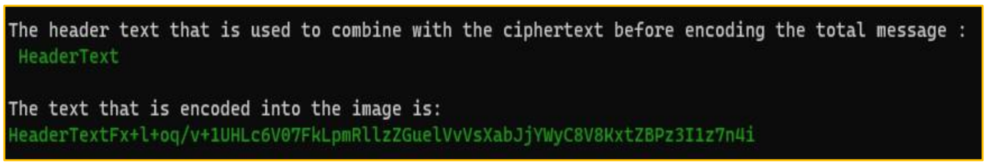
```
Details about the header text and header bits:

This header text is combined with the base64 encoded ciphertext and is encoded

Header Text is : HeaderText

Length of the Header Text in Bytes is : 10

Length of the header bits in Bits is: 80
```

Fig 4.17: Development Environment 16



```
Header bits:

[
    0, 1, 0, 0, 1, 0, 0, 0,
    0, 1, 1, 0, 0, 1, 0, 1,
    0, 1, 1, 0, 0, 0, 0, 1,
    0, 1, 1, 0, 0, 1, 0, 0,
    0, 1, 1, 0, 0, 1, 0, 1,
    0, 1, 1, 1, 0, 0, 1, 0,
    0, 1, 0, 1, 0, 1, 0, 0,
    0, 1, 1, 0, 0, 1, 0, 1,
    0, 1, 1, 1, 1, 0, 0, 0,
    0, 1, 1, 1, 0, 1, 0, 0,
]
```

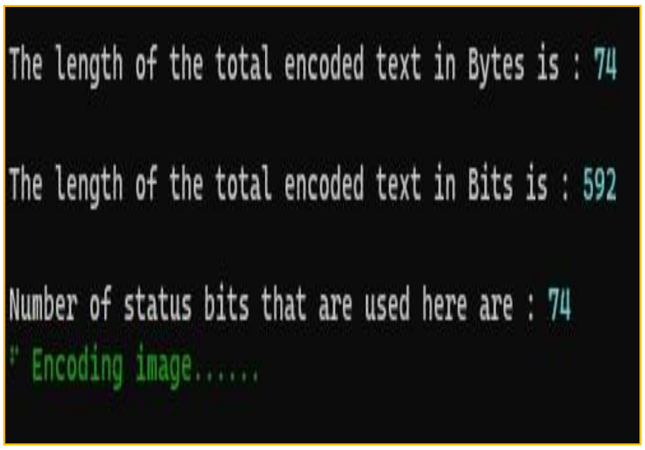
Fig 4.18: Development Environment 17



```
The header text that is used to combine with the ciphertext before encoding the total message :
 HeaderText

The text that is encoded into the image is:
HeaderTextFx+l+oq/v+1UHLc6V07FkLpmRllzZGuelVvVsXabJjYWyC8V8KxtZBPz3I1z7n4i
```

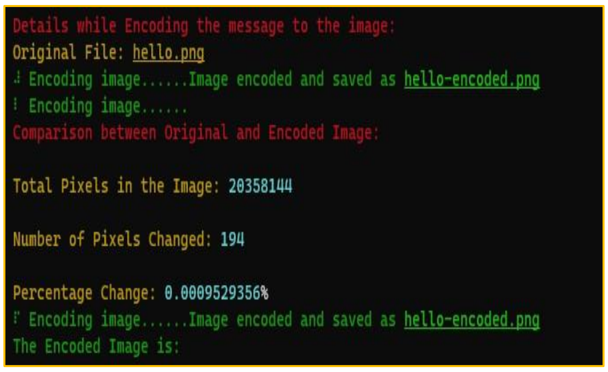
Fig 4.19: Development Environment 18

```
The encoded text bits are:
[
    [illegible]
]
```



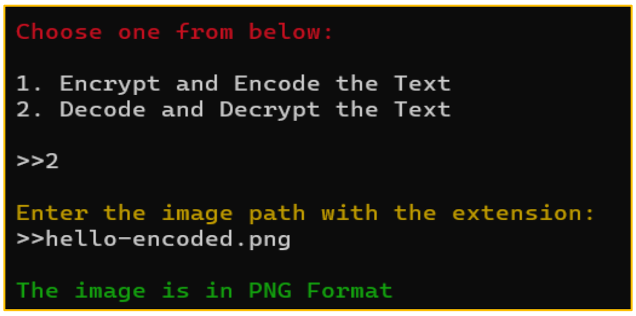
```
The length of the total encoded text in Bytes is : 74

The length of the total encoded text in Bits is : 592

Number of status bits that are used here are : 74
" Encoding image.....
```

Fig 4.20: Development Environment 19　Fig 4.21: Development Environment 20



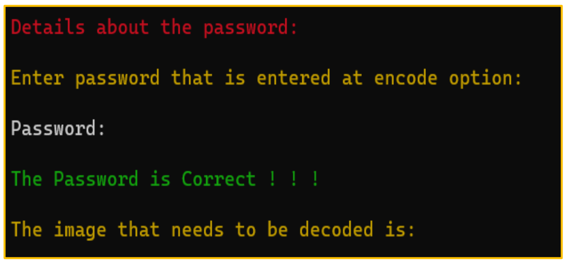
```
Details while Encoding the message to the image:
Original File: hello.png
⠧ Encoding image......Image encoded and saved as hello-encoded.png
⠇ Encoding image......
Comparison between Original and Encoded Image:

Total Pixels in the Image: 20358144

Number of Pixels Changed: 194

Percentage Change: 0.0009529356%
⠏ Encoding image......Image encoded and saved as hello-encoded.png
The Encoded Image is:
```

Fig 4.22: Development Environment 21



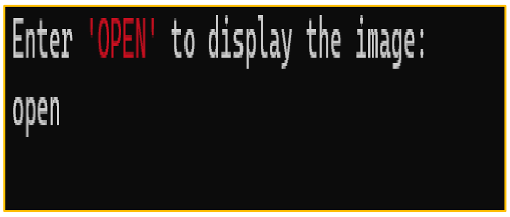
```
Choose one from below:

1. Encrypt and Encode the Text
2. Decode and Decrypt the Text

>>2

Enter the image path with the extension:
>>hello-encoded.png

The image is in PNG Format
```

Fig 4.23: Development Environment 22



```
Details about the password:

Enter password that is entered at encode option:

Password:

The Password is Correct ! ! !

The image that needs to be decoded is:
```

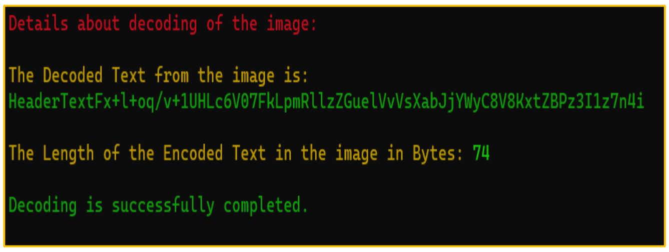
Fig 4.24: Development Environment 23



```
Enter 'OPEN' to display the image:
open
```

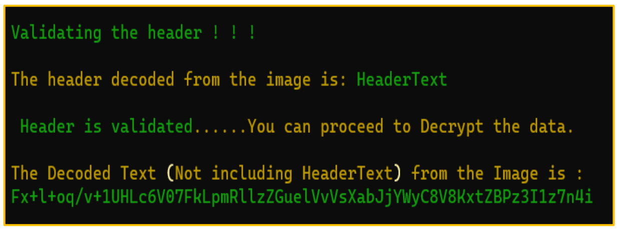
Fig 4.24: Development Environment 24

Fig 4.25: Encoded Image



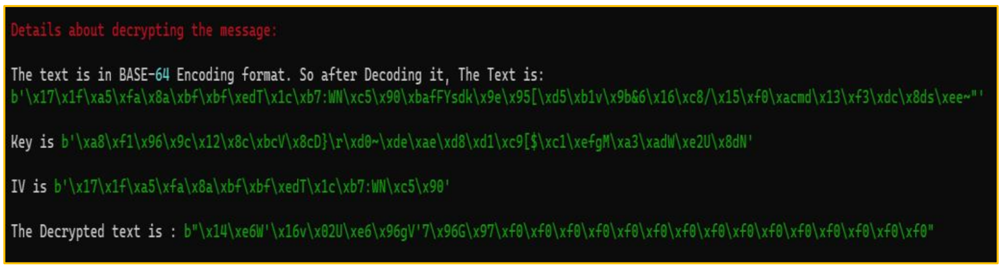
```
Details about decoding of the image:

The Decoded Text from the image is:
HeaderTextFx+l+oq/v+1UHLc6V07FkLpmRllzZGuelVvVsXabJjYWyC8V8KxtZBPz3I1z7n4i

The Length of the Encoded Text in the image in Bytes: 74

Decoding is successfully completed.
```

Fig 4.26: Development Environment 25



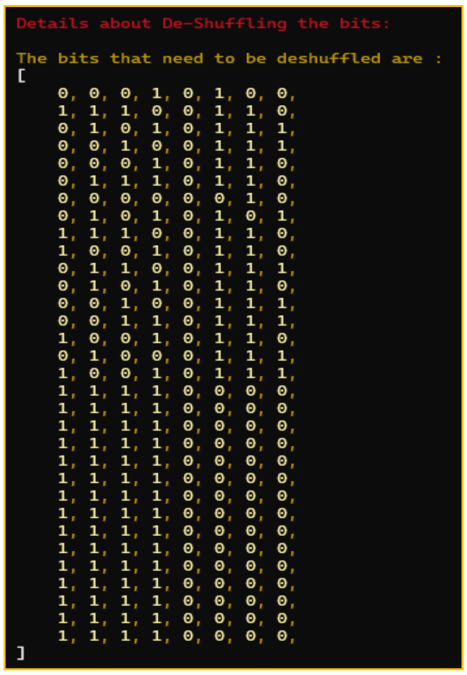
```
Validating the header ! ! !

The header decoded from the image is: HeaderText

 Header is validated......You can proceed to Decrypt the data.

The Decoded Text (Not including HeaderText) from the Image is :
Fx+l+oq/v+1UHLc6V07FkLpmRllzZGuelVvVsXabJjYWyC8V8KxtZBPz3I1z7n4i
```

Fig 4.27: Development Environment 26



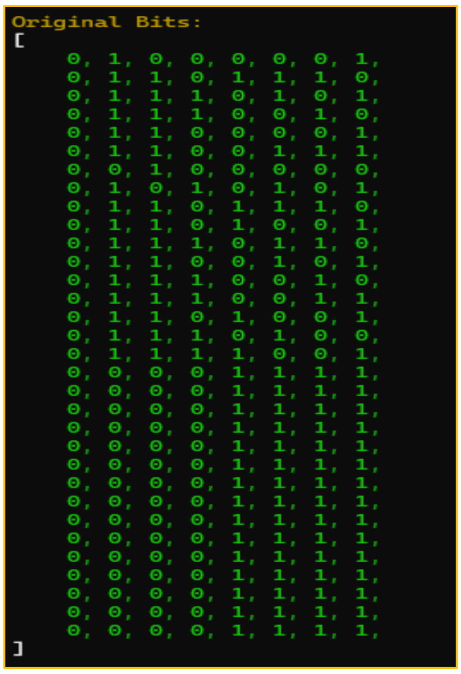
```
Details about decrypting the message:

The text is in BASE-64 Encoding format. So after Decoding it, The Text is:
b'\x17\x1f\xa5\xfa\x8a\xbf\xbf\xedT\x1c\xb7:WN\xc5\x90\xbafFYsdk\x9e\x95[\xd5\xb1v\x9b&6\x16\xc8/\x15\xf0\xacmd\x13\xf3\xdc\x8ds\xee~"'

Key is b'\xa8\xf1\x96\x9c\x12\x8c\xbcV\x8cD}\r\xd0~\xde\xae\xd8\xd1\xc9[$\xc1\xefgM\xa3\xadW\xe2U\x8dN'

IV is b'\x17\x1f\xa5\xfa\x8a\xbf\xbf\xedT\x1c\xb7:WN\xc5\x90'

The Decrypted text is : b"\x14\xe6W'\x16v\x02U\xe6\x96gV'7\x96G\x97\xf0\xf0\xf0\xf0\xf0\xf0\xf0\xf0\xf0\xf0\xf0\xf0\xf0\xf0\xf0"
```

Fig 4.28: Development Environment 27



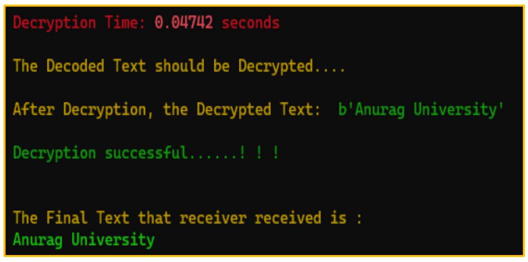
```
Details about De-Shuffling the bits:

The bits that need to be deshuffled are :
[
     0, 0, 0, 1, 0, 1, 0, 0,
     1, 1, 1, 0, 0, 1, 1, 0,
     0, 1, 0, 1, 0, 1, 1, 1,
     0, 0, 1, 0, 0, 1, 1, 1,
     0, 0, 0, 1, 0, 1, 1, 0,
     0, 1, 1, 1, 0, 1, 1, 0,
     0, 0, 0, 0, 0, 0, 1, 0,
     0, 1, 0, 1, 0, 1, 0, 1,
     1, 1, 1, 0, 0, 1, 1, 0,
     1, 0, 0, 1, 0, 1, 1, 0,
     0, 1, 1, 0, 0, 1, 1, 1,
     0, 1, 0, 1, 0, 1, 1, 0,
     0, 0, 1, 0, 0, 1, 1, 1,
     0, 0, 1, 1, 0, 1, 1, 1,
     1, 0, 0, 1, 0, 1, 1, 0,
     0, 1, 0, 0, 0, 1, 1, 1,
     1, 0, 0, 1, 0, 1, 1, 1,
     1, 1, 1, 1, 0, 0, 0, 0,
     1, 1, 1, 1, 0, 0, 0, 0,
     1, 1, 1, 1, 0, 0, 0, 0,
     1, 1, 1, 1, 0, 0, 0, 0,
     1, 1, 1, 1, 0, 0, 0, 0,
     1, 1, 1, 1, 0, 0, 0, 0,
     1, 1, 1, 1, 0, 0, 0, 0,
     1, 1, 1, 1, 0, 0, 0, 0,
     1, 1, 1, 1, 0, 0, 0, 0,
     1, 1, 1, 1, 0, 0, 0, 0,
     1, 1, 1, 1, 0, 0, 0, 0,
     1, 1, 1, 1, 0, 0, 0, 0,
     1, 1, 1, 1, 0, 0, 0, 0,
     1, 1, 1, 1, 0, 0, 0, 0,
     1, 1, 1, 1, 0, 0, 0, 0,
]
```

Fig 4.29: Development Environment 28



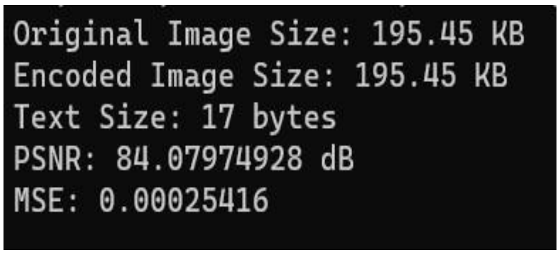
```
Original Bits:
[
     0, 1, 0, 0, 0, 0, 0, 1,
     0, 1, 1, 0, 1, 1, 1, 0,
     0, 1, 1, 1, 0, 1, 0, 1,
     0, 1, 1, 1, 0, 0, 1, 0,
     0, 1, 1, 0, 0, 0, 0, 1,
     0, 1, 1, 0, 0, 1, 1, 1,
     0, 0, 1, 0, 0, 0, 0, 0,
     0, 1, 0, 1, 0, 1, 0, 1,
     0, 1, 1, 0, 1, 1, 1, 0,
     0, 1, 1, 0, 1, 0, 0, 1,
     0, 1, 1, 1, 0, 1, 1, 0,
     0, 1, 1, 0, 0, 1, 0, 1,
     0, 1, 1, 1, 0, 0, 1, 0,
     0, 1, 1, 1, 0, 0, 1, 1,
     0, 1, 1, 0, 1, 0, 0, 1,
     0, 1, 1, 1, 0, 1, 0, 0,
     0, 1, 1, 1, 1, 0, 0, 1,
     0, 0, 0, 0, 1, 1, 1, 1,
     0, 0, 0, 0, 1, 1, 1, 1,
     0, 0, 0, 0, 1, 1, 1, 1,
     0, 0, 0, 0, 1, 1, 1, 1,
     0, 0, 0, 0, 1, 1, 1, 1,
     0, 0, 0, 0, 1, 1, 1, 1,
     0, 0, 0, 0, 1, 1, 1, 1,
     0, 0, 0, 0, 1, 1, 1, 1,
     0, 0, 0, 0, 1, 1, 1, 1,
     0, 0, 0, 0, 1, 1, 1, 1,
     0, 0, 0, 0, 1, 1, 1, 1,
     0, 0, 0, 0, 1, 1, 1, 1,
     0, 0, 0, 0, 1, 1, 1, 1,
     0, 0, 0, 0, 1, 1, 1, 1,
     0, 0, 0, 0, 1, 1, 1, 1,
]
```

Fig 4.30: Development Environment 29



```
Decryption Time: 0.04742 seconds

The Decoded Text should be Decrypted....

After Decryption, the Decrypted Text:  b'Anurag University'

Decryption successful......! ! !


The Final Text that receiver received is :
Anurag University
```

Fig 4.31: Development Environment 30



```
Original Image Size: 195.45 KB
Encoded Image Size: 195.45 KB
Text Size: 17 bytes
PSNR: 84.07974928 dB
MSE: 0.00025416
```

Fig 4.32: metrics.py experiment

5. EXPERIMENTAL SETUP

Used **Python IDLE & Command Prompt** to develop this Project – “Randomized LSB Steganography with AES Algorithm”.

5.1. STEPS TO INSTALL PYTHON IDLE

1. **Download Python:**
   - Visit the official Python website: [https://www.python.org/].
   - Navigate to the "Downloads" section.

2. **Choose Python Version:**
   - Select the version of Python you want to install. It's recommended to choose the latest stable version.
   - Click on the download link for the installer that corresponds to your operating system (Windows, macOS, or Linux).

3. **Run the Installer:**
   - For Windows: Double-click the downloaded executable file (usually ending with `.exe`) to run the installer.
   - For macOS: Open the downloaded `.pkg` file and follow the installation instructions.
   - For Linux: Run the appropriate commands in the terminal as per your distribution's package management system.

4. **Customize Installation (Optional):**
   - During the installation process, you may be given the option to customize the installation. Ensure that the checkbox for "IDLE" or "Tcl/Tk and IDLE" is selected.

5. **Complete the Installation:**
   - Follow the on-screen instructions to complete the installation. Ensure that you check the box that says "Add Python to PATH" during the installation on Windows for easier command-line access.

6. **Verify Installation:**

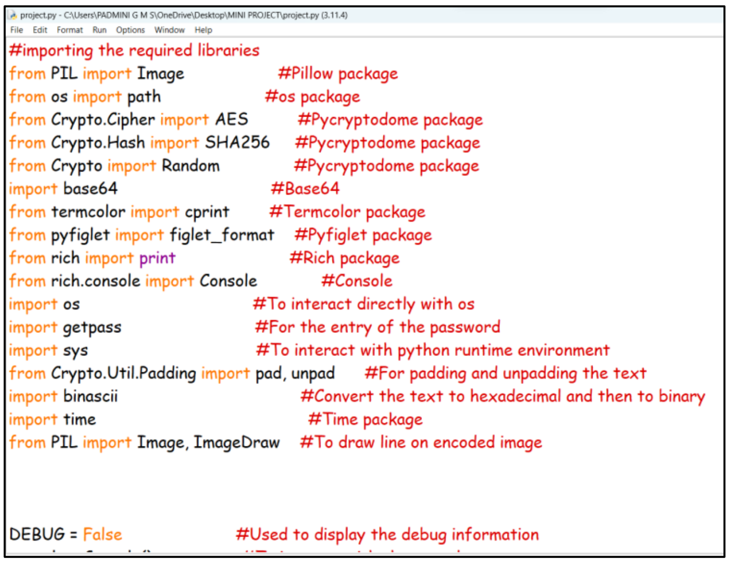
- After the installation is complete, you can verify it by opening a command prompt or terminal and typing `python --version` or `python3 --version` to check that Python is installed.

7. **Launch IDLE:**

- IDLE is Python's integrated development environment, and it is typically installed along with Python. To launch IDLE:
    - For Windows: Look for "IDLE" in the Start menu.
    - For macOS: Open a terminal and type `idle` or look for IDLE in the Applications.
    - For Linux: Open a terminal and type `idle` or `idle3` depending on your Python version.

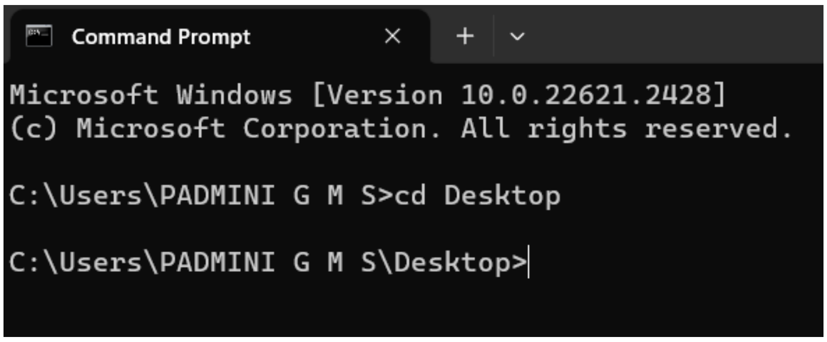
8. **Start Coding:**

- Once IDLE is launched, you can start writing and executing Python code.


project.py - C:\Users\PADMINI G M S\OneDrive\Desktop\MINI PROJECT\project.py (3.11.4)
File Edit Format Run Options Window Help

```
#importing the required libraries
from PIL import Image                   #Pillow package
from os import path                     #os package
from Crypto.Cipher import AES           #Pycryptodome package
from Crypto.Hash import SHA256          #Pycryptodome package
from Crypto import Random               #Pycryptodome package
import base64                           #Base64
from termcolor import cprint            #Termcolor package
from pyfiglet import figlet_format      #Pyfiglet package
from rich import print                  #Rich package
from rich.console import Console        #Console
import os                               #To interact directly with os
import getpass                          #For the entry of the password
import sys                              #To interact with python runtime environment
from Crypto.Util.Padding import pad, unpad      #For padding and unpadding the text
import binascii                         #Convert the text to hexadecimal and then to binary
import time                             #Time package
from PIL import Image, ImageDraw        #To draw line on encoded image



DEBUG = False                   #Used to display the debug information
```

Fig 5.1. Python IDLE



```
Microsoft Windows [Version 10.0.22621.2428]
(c) Microsoft Corporation. All rights reserved.

C:\Users\PADMINI G M S>cd Desktop

C:\Users\PADMINI G M S\Desktop>
```

Fig 5.2 Command Prompt

5.2. PACKAGES USED ARE:

1) **PILLOW PACKAGE:** Used for image processing, to open, manipulate and save various image formats.
2) **OS PACKAGE:** To interact with the operating system.
3) **PYCRYPTODOME PACKAGE:** Used in the code for AES, SHA-256, RANDOM.
4) **BASE-64 PACKAGE**: Used for particularly in image steganography without causing issues when we use character encoding or special characters.
5) **TERMCOLOR PACKAGE:** Used to add the color text, various background colors to output console. Here, we use cprint() function which is used to print the text with different styles and colours.
6) **PYFIGLET PACKAGE:** To create ASCII text and attractive headers using font and width
7) **GETPASS MODULE:** To read passwords and ensure the passwords and sensitive data is not exposed.
8) **SYS PACKAGE:** To work with Python runtime Environment
9) **RANDOM PACKAGE:** For the generation of IV
10) **TIME MODULE:** To calculate the encryption and decryption time
11) **BINASCII MODULE:** To work with binary and ASCII encoded data
12) **NUMPY PACKAGE:** To work with arrays or matrices
13) **SCIKIT-IMAGE PACKAGE:** For calculating image quality metrics, specifically PSNR (Peak Signal-to-Noise Ratio) and MSE (Mean Squared Error).

14) **OPENCV PACKAGE**: For computer vision tasks and image processing.
15) **MATPLOTLIB.PYTPLOT**: To create plots and visualizations.

5.3. PARAMETERS

5.3.1. MEAN SQUARE ERROR (MSE):

MSE = (1 / (m * n)) * Σ(Σ(I(x, y) - K(x, y))^2)

- **m** and **n** are the dimensions of the image (e.g., width and height).
- **I(x, y)** represents the pixel value of the original image at position (x, y).
- **K(x, y)** represents the pixel value of the reconstructed (or compressed) image at the same position.
- MSE measures the average squared difference between the pixel values of the original image and encoded image.
- MSE values close to zero suggest minimal distortion and high image fidelity.
- Mathematical Calculation: It computes the average of the squared pixel-wise differences between the corresponding pixels of the two images.
- Minimal Distortion: An MSE value close to zero indicates that there is minimal distortion or difference between the original and processed image.
- Image Fidelity: In the context of image processing, low MSE values suggest high image fidelity, meaning the processed image closely resembles the original, maintaining image quality and detail.
- Quality Assessment: MSE is a crucial tool for evaluating the effectiveness of image processing techniques. A lower MSE signifies that the process or encoding has preserved the original image content well.

5.3.2. PEAK SIGNAL-TO-NOISE RATIO (PSNR)

PSNR = 10 * log10((peak^2) / MSE)

- PSNR is a measure of the quality of a encoded image compared to an original reference image.
- PSNR values above 30 dB are generally considered high and indicative of good image quality.
- The peak pixel value in the picture is represented by the variable "peak". It shows the highest possible value that a pixel in the picture may have. The highest pixel value for an 8-bit picture is 255 (28 - 1), hence "peak" would typically be 255 for normal 8-bit images.
- Measurement of Image Quality: PSNR is a widely used metric for assessing the quality of an image after it has undergone compression or other processing. It quantifies the fidelity of the processed image compared to the original, serving as a critical tool in image quality evaluation.
- Decibel Scale: PSNR is expressed in decibels (dB), which is a logarithmic scale. The decibel scale is particularly useful because it provides a convenient way to express large ranges of image quality. Higher PSNR values indicate better image quality.

5.3.3 CHOSEN CIPHER-TEXT ATTACK

A sort of cryptographic assault known as a chosen ciphertext attack, or CCA, allows the attacker to pick a ciphertext (an encrypted message) and provide it to a system for decryption or other actions. The attacker keeps track of how the system behaves and makes use of this knowledge to learn more about the encryption algorithm, perhaps decrypting the plaintext message or taking advantage of system flaws. CCA attacks are a useful tool for evaluating how secure encryption methods and systems are.

5.3.4 KNOWN PLAIN-TEXT ATTACK

An instance of a known plaintext attack is one in which the attacker has access to both the ciphertext (data that has been encrypted) and some associated plaintext (data that has not been encrypted). This attack aims to determine or extract the encryption key's secret code.
When conducting a known plaintext attack, the attacker has knowledge of or

access to particular portions of the plaintext and ciphertext pairings, which they use to:

Find the Key: A known plaintext attack's main goal is to figure out the secret encryption key. Any additional ciphertext encrypted with the same key may be decrypted if the attacker discovers the key.

Additional Ciphertexts to Decrypt: With the key, the attacker may decrypt more ciphertexts without performing any additional steps.

5.3.5 CIPHER-TEXT ONLY ATTACK:

A ciphertext-only attack, in the realm of cryptanalysis, is a scenario where an attacker has access only to encrypted data (ciphertext) without any knowledge of the corresponding plaintext or the encryption key. In this challenging context, the attacker's objective is to gain insights into the plaintext or the encryption method used, often by employing various statistical or analytical techniques. Ciphertext-only attacks are a real-world concern.

6. DISCUSSION OF RESULTS

To examine the distortion and noise in the same image of various sizes, we utilized the Mean Square Error and Peak Signal to Noise Ratio (PSNR). As can be seen, the PSNR decreases as the image size decreases and when the Encoded text size increases. But is still rather substantial, indicating that the cover image and the genuine image cannot be distinguished by the naked eye. The receiver was able to decode the image and obtain the cipher text after we successfully encrypted the message using AES and concealed it in the image using Steganography. The user can obtain the plain text using AES decryption and the ciphertext.

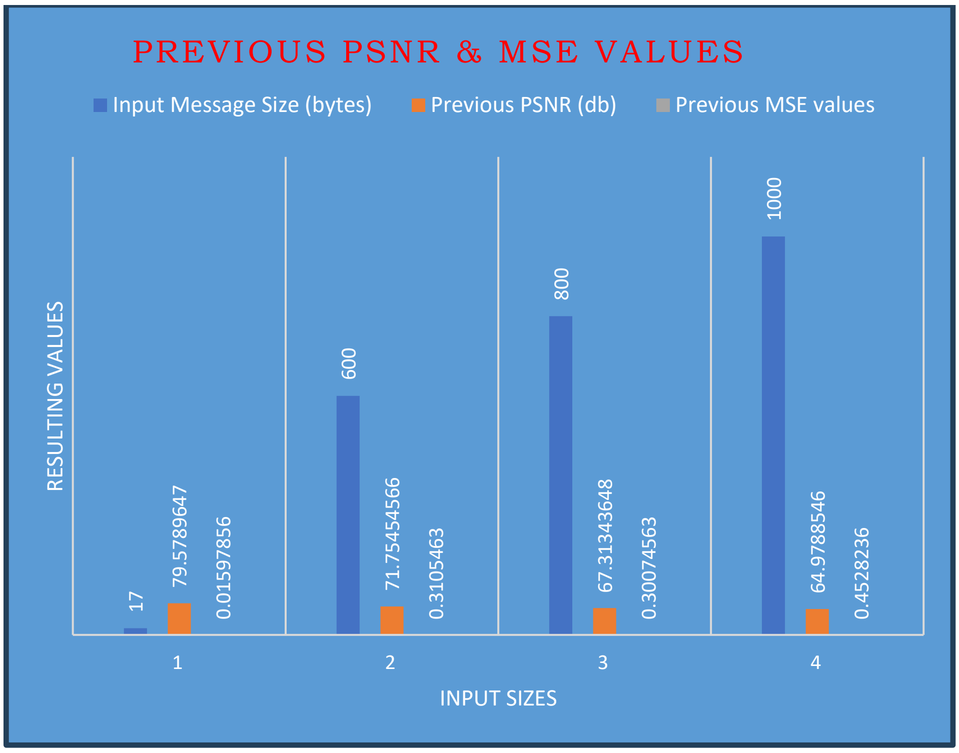


Fig 6.1: Previous PSNR & MSE Values

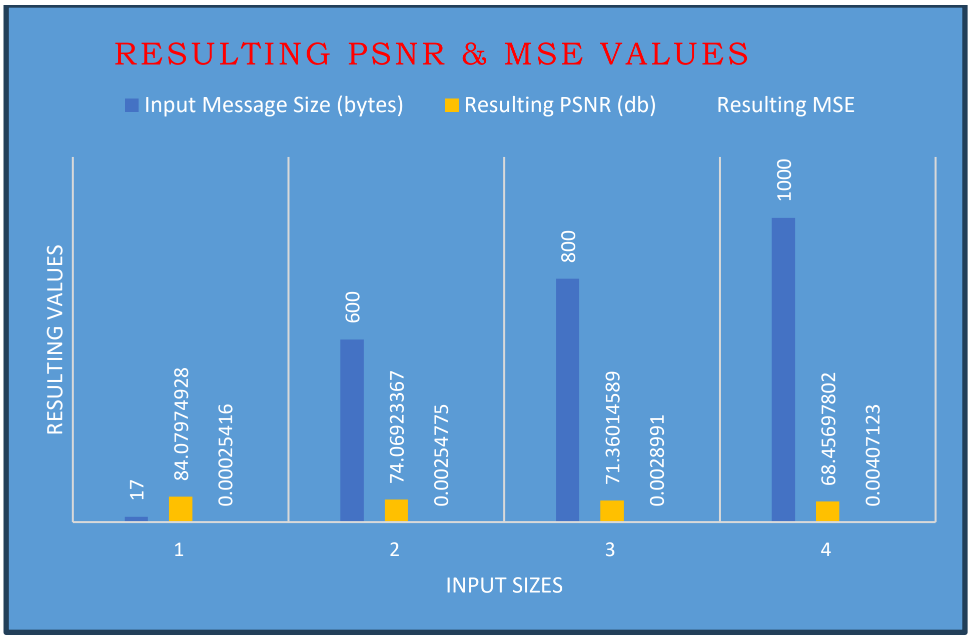
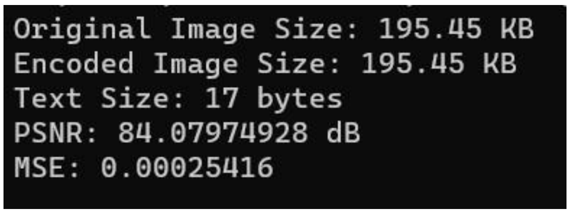


Fig 6.2: Resulting PSNR & MSE Values

Image Size **(KB)**	Input Message Size (bytes)	Resulting PSNR (db)	Resulting MSE
195.45	17	84.07974928	0.00025416
195.45	600	74.06923367	0.00254775
195.45	800	71.36014589	0.00289910
195.45	1000	68.45697802	0.00407123

Table 6.1: Calculation of PSNR & MSE Values



```
Original Image Size: 195.45 KB
Encoded Image Size: 195.45 KB
Text Size: 17 bytes
PSNR: 84.07974928 dB
MSE: 0.00025416
```

Fig 6.3: metrics.py Results

The image labeled as "Fig 6.3" represents the outcome of the steganography process carried out with an initial image size of 195.45 kilobytes (KB). Remarkably, the size of the encoded image remains consistent with the original image at 195.45 KB. This outcome is indicative of the efficiency of the steganographic algorithm and technique employed in this project.

The preservation of image size is significant because it reflects the minimal impact on data size caused by the steganographic process. In many data transmission and storage scenarios, maintaining the original file size is advantageous, as it ensures that no additional storage space or bandwidth is required. This efficient compression and data concealment, achieved through the steganography algorithm, demonstrate the practicality and real-world applicability of this approach. In essence, the provided image size remaining unchanged in the encoded image emphasizes the successful integration of steganography with minimal data size alteration. This is particularly valuable in applications where data integrity, security, and efficient use of resources are paramount, underlining the effectiveness of the utilized algorithm and steganographic methodology.

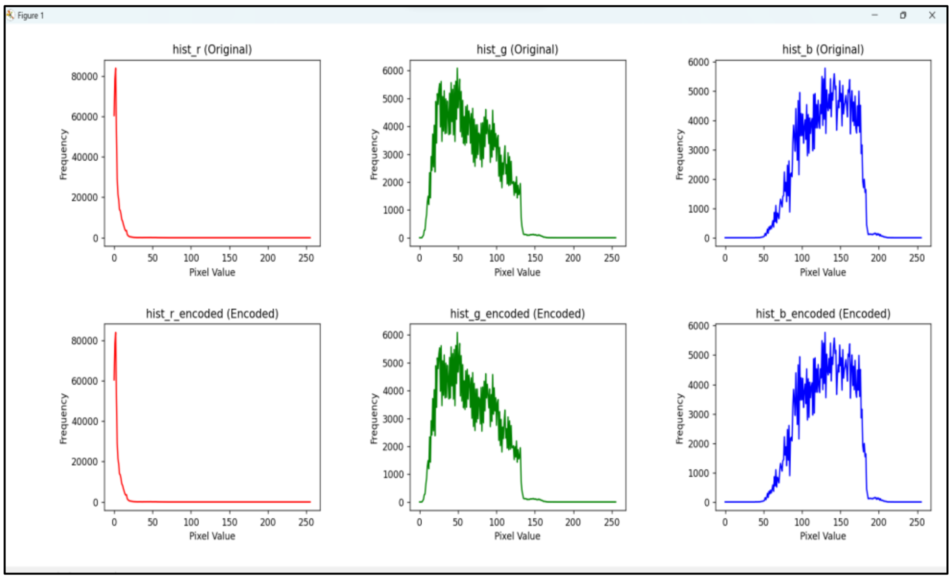


Fig 6.4: Histogram Visualization of the Original and Encoded Images

The histogram visualization presented here serves as a valuable tool for uncovering alterations that might escape the human eye's notice when comparing the original image with its encoded counterpart. What makes this visual representation particularly insightful is that it focuses on the RGB (Red, Green, Blue) color values of both the original and encoded images.

7. SUMMARY, CONCLUSION AND RECOMMENDATION

This study represents a substantial advancement over previous efforts in the domains of steganography and AES encryption, offering a more robust and secure approach. By combining these two techniques, it becomes evident that the synergy between steganography and cryptography significantly enhances security while introducing a layer of complexity that acts as a formidable deterrent against potential system attacks. This approach proves particularly valuable in scenarios where security breaches are intolerable, and revealing any suspicion to potential attackers could have dire consequences.
The specific advantages presented by this work when compared to earlier endeavors are as follows:

I. **Versatile Image Mode Conversion:** Unlike prior works, this study demonstrates the capability to seamlessly convert various image modes, such as Greyscale, CMYK, and RGBA, into the widely compatible RGB mode.

II. **Enhanced Security with AES-256 in CBC Mode:** The adoption of AES-256 encryption in Cipher Block Chaining (CBC) mode stands as a pivotal step towards bolstering security. This approach ensures that data is not only encrypted with a strong algorithm but also with added complexities in the chaining mode, making it exceptionally challenging for unauthorized parties to decrypt the information.

III. **Optimal Base-64 Encoding:** The utilization of Base-64 encoding demonstrates a keen understanding of the right format for encoding binary bits. This choice enhances compatibility and data integrity, making it an optimal choice for securing the hidden information.

IV. **Effective Randomization through Bit Shuffling:** The application of randomization techniques via bit shuffling serves as a robust defense against multiple types of attacks, including Known-Plaintext, Chosen Ciphertext, and Ciphertext-Only attacks.

8. FUTURE ENHANCEMENTS

Future enhancements for this project could include:

Improved Randomization Techniques: Develop more sophisticated randomization methods to further fortify data against known-plaintext attacks.

Enhanced Compatibility: Extend the project's compatibility with various image formats and encryption algorithms, allowing it to be more versatile in different use cases.

User-Friendly Interface: Create a user-friendly graphical interface for easy adoption, even by individuals with limited technical expertise.

Real-Time Data Protection: Implement a real-time data protection feature, ensuring that data is secured as it's generated or received.

Advanced Steganographic Methods: Explore advanced steganographic techniques, such as frequency domain steganography, for even more covert data embedding.

Machine Learning Integration: Employ machine learning for better detection avoidance and encryption key management.

Cross-Platform Compatibility: Make the project compatible with a wider range of operating systems and devices.

9. REFERENCES

[1] Adit Pabbi, Rakshit Malhotra and Manikandan K : " Implementation of Least Significant Bit Image Steganography with Advanced Encryption Standard " – 2021, International Conference on Emerging Smart Computing and Informatics (ESCI)

[2] Utsav Sheth and Shiva Saxena, "Image Steganography Using AES Encryption and Least Significant Nibble", International Conference on Communication and Signal Processing, April 6-8, 2016, India

[3] C. Lalengmawia and A. Bhattacharya, "Image steganography using advanced encryption standard for implantation of audio/video data", 2016 International Conference on Recent Trends in Information Technology (ICRTIT)

[4] Masumeh Damrudi and Kamal Jadidy Aval, Image Steganography using LSB and encrypted message with AES, RSA, DES, 3DES, and Blowfish, 2019 International Journal of Engineering and Advanced Technology (IJEAT)

[5] Inas Jawad Kadhim, Prashan Premaratne, Peter James Vial and Brendan Halloran, Comprehensive survey of image steganography: Techniques, Evaluations, and trends in future research, 2019 Elsevier Journal

[6]Abbas Cheddad, Joan Condell, Kevin Curran, Paul Mc Kevitt, Digital image steganography: Survey and analysis of current methods, 2019 Elsevier Journal

[7] Mark Rennel D. Molato, Bobby D. Gerardo, and Ruji P. Medina. 2018. "Secured Data Hiding and Sharing using Improved LSB-based Image Steganography Technique". In Proceedings of the 4th International Conference on Industrial and Business Engineering (ICIBE' 18). Association for Computing Machinery, New York, NY, USA, 238–243.

[8] Mansi S. Subhedar and Vijay H. Mankar. 2014. “High-Capacity Image Steganography based on Discrete Wavelet Transform and Singular Value Decomposition”. In Proceedings of the 2014 International Conference on Information and Communication Technology for Competitive Strategies (ICTCS ’14). Association for Computing Machinery, New York, NY, USA, Article 63, 1–7.

[9] Dipti, K. S. and Neha, B. 2010. “Proposed System for Data Hiding Using Cryptography and Steganography”. International Journal of Computer Applications. 8(9), pp. 7-10. Retrieved 14th August, 2012

[10] Raphael, A. J., and Sundaram, V. 2011. “Cryptography and Steganography - A Survey”. International Journal of Computer Technology Application, 2(3), ISSN: 2229-6093, pp. 626-630

[11] Nadeem Akhtar, Shahbaaz Khan, and Pragati Johri, “An Improved Inverted LSB Image Steganography”, 2014 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT).

[12] Yang Rener, Zheng Zhiwei, Tao Shun, and Ding Shile, “Image Steganography Combined with DES Encryption Pre-processing”, 2014 Sixth International Conference on Measuring Technology and Mechatronics Automation.

[13] Sahib Khan, Nasir Ahmad, Muhammad Is-mail, Nasru Minallah, Tawab Khan, “A Secure True Edge based 4 Least Significant Bits Steganography”, 2015 International Con-ference on Emerging Technologies (ICET)

[14] K.Thangadurai and G.Sudha Devi, “An analysis of LSB Based Image Steganography Techniques”, 2014 International Conference on Computer Communication and Informatics (ICCCI -2014), Jan. 03 – 05, 2014, Coimbatore, INDIA

[15] Rina Mishra and Praveen Bhanodiya, “A Review on Steganography and Cryptography”,2015 International Conferences on ICACEA.