CSE3501

Information Security Analysis and Audit

A Project Report on the Title:

Secure Messaging Application using RSA Cryptography and LSB Steganography

Submitted in partial fulfilment of the requirements for the degree of

Bachelor of Technology

in

Computer Science with specialization in Information Security

Ishaan Lonial 20BCI0240

Yash Bobde 20BCI0176

Khushi Sharma 20BCI0243

Jitesh Aggarwal 20BCI0185

Submitted to:

Dr. Raja S.P.

1. Abstract

As the amount of digital information that is shared between computers and other kinds of networks continues to rise, people all over the world are becoming increasingly concerned about the integrity of the data that they store. Due to the fact that the data can be attacked and manipulated by individuals who are not authorised to do so, the dissemination of digital data has given rise to a significant cause for concern. This is because the data can be accessed by more people. As a consequence of this, it is of the utmost importance to communicate while concealing the content of one's messages in some way, shape, or form. Because the use of cryptography would result in the message that is received being incomprehensible, which would raise suspicions, we have decided to use steganography instead of cryptography. Steganography allows us to conceal information without raising suspicions. Steganography gives us the ability to hide our identities while keeping everything else about us transparent.

2. Introduction

The objective of this project is to create a secure messaging application, and to do so, we will be utilising the RSA Cryptography and LSB Steganography algorithms. In order to achieve this goal, an application will need to be developed that is capable of encrypting a message and embedding the resultant encrypted message within an image. The receiver is responsible for obtaining the image, after which the app will decrypt it with the key that was provided. It is a secure method of communication because even if an unscrupulous third party is involved, all that they will see is an image being sent from one location to another. This prevents the disclosure of sensitive information.

3. Literature Survey

S.No.	Title	Authors	Descriptive Summary
1	Steganography Method Using Effective Combination of RSA Cryptography and Data Compression	A Bose, Malaya Kumar Hota, A Kumar, S Sherki - 2019	In this paper, the authors use a combination of RSA cryptography, data compression, and Hash- LSB steganography technique to make the data secure. RSA cryptography algorithm is implemented as the receiver will have to use the private key to decrypt the message because the confidential message has been encrypted using the public key. The secret message is compressed using the Huffman coding algorithm and the cover image is compressed using Discrete wavelet transform
2	A Robust and Secured Image Steganography using LSB and Random Bit Substitution	U Ali, Md. Ehsan Ali, Md. Palash Uddin – 2019	In this paper, a new approach to image stenography with the least significant bit substitution is proposed where the information is embedded in the random bit position of the pixel.
3	Secure and Robust Digital Image Watermarking using Coefficient Differencing and Chaotic Encryption	Nazir A. Loan, Nasir N. Hurrah, Shabir A. Parah – 2018	This paper presents the chaotic encryption based blind digital image watermarking technique applicable to both grayscale and colour images
4	Adaptive Digital Watermarking for Copyright Protection of Digital Images in Wavelet Domain	SP Vaidya, PC Mouli – 2015	In this paper an adaptive invisible watermarking scheme is proposed in the wavelet domain. The proposed is adaptive in the sense that the scaling and the embedded

			factors are calculated using the Bhattacharya distance.
5	Comparison of different techniques for Steganography in images	FM Shelke, AA Dongre, PD Soni – 2014	This paper is classified into three categories which are; pure, secret key, and public key steganography. According to the type of the cover object, there are different types of steganography which are; image, audio, video, network, and text.
	Information hiding using least significant bit steganography and cryptography		To overcome the disadvantage of the LSB method by appending encrypted data in an image in place of plain textual data. To encrypt the data RSA and Diffie Hellman algorithms were used.
6	Enhancing the security and quality of LSB-based image steganography	N Akhtar, P Johri, S Khan - 2013	Rather than storing the bits sequentially, they are stored in the random order generated by the RC4 algorithm which uses the stego key shared by both sender and receiver

4. Proposed Methodology

"Image steganography" is a term that refers to the process of covertly embedding a message of any kind within the discrete pixels that make up an image. This process has been given the name "image steganography," and the term itself has been given this name. This is done in order to ensure that the information can be sent over a network in a manner that is both covert and secure. The goal of this step is to ensure that the information can be sent. Because it is dependable, has high fidelity, ciphertext, and demonstrates that it has a large payload capacity, this method demonstrates that it is an excellent choice for achieving safety. This is demonstrated by the fact that it demonstrates that it has a large payload capacity. In addition to this, it demonstrates that it can carry a substantial amount of weight. We will be utilising the RSA and LSB Steganography techniques that were described earlier, as well as the methodology that was presented in the form of an architecture diagram, in order to achieve our objective of developing a comprehensive solution. This will allow us to meet our goal of accomplishing our mission.

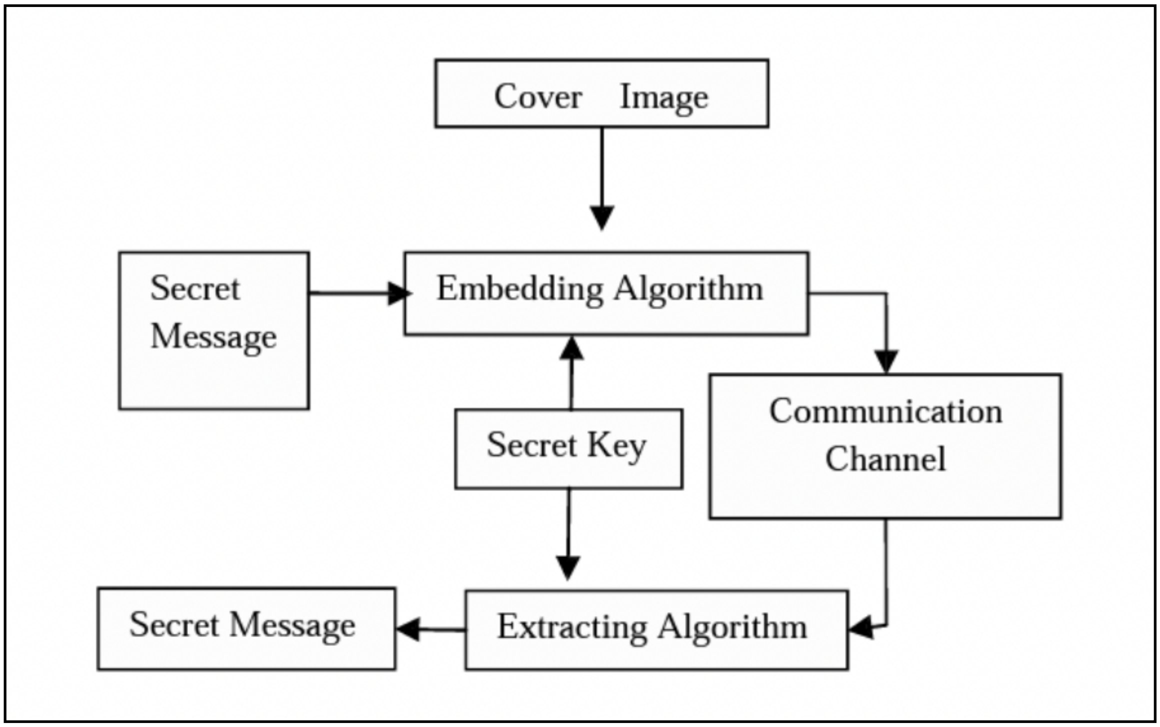


Fig. 1 Architecture Diagram of the Proposed Methodology

4.1 Major Algorithms being Used:

4.1.1 RSA Algorithm:

A message encryption cryptosystem in which two prime numbers are taken initially and then the product of these values is used to create a public and a private key, which is further used in encryption and decryption.

Working of the RSA Algorithm:

a. Choose two large prime no. p & q.
b. Calculate N=p*q
c. Calculate f(z)=(p-1) *(q-1) Find a random number e satisfying 1<e<f(n) and relatively prime to f (n) i.e., gcd (e, f (z)) = 1
d. Calculate a number d such that d = e-1 mod f(n)

e. Encryption: Enter a message to get ciphertext. Ciphertext c= mod ((message. ^e), N)
f. Decryption: The ciphertext is decrypted by Message=mod ((c. ^d), N)

4.1.2 LSB ALGORITHM:

For embedding the message:

a. Input the Encrypted message using RSA Algorithm that is hidden in the cover image
b. Select the cover image
c. Take pixels from the cover image
d. Take the LSB bit from the pixel
e. Divide the encrypted message into two equal parts
f. Perform XOR of the first half of the encrypted message with the odd position pixel values
g. Perform XOR of the second half of the encrypted message with the even position pixel values
h. Get all the xored values of even and odd position pixels
i. Store the xored value of even in the even position LSB bit of pixels. And xored value of odd in odd positioned pixels.

For extracting the message:

a. Receive the Image with the embedded message
b. Convert the red, green, and blue values into binary format
c. Extracting data from the least significant bit of red pixel
d. Extracting data from the least significant bit of green pixel
e. Extracting data from the least significant bit of blue pixel
f. Split by 8-bits
g. Convert from bits to characters
h. Check if we have reached the delimiter which is "$t3g0"
i. Print(decoded_data)
j. Remove the delimiter to show the original hidden message.

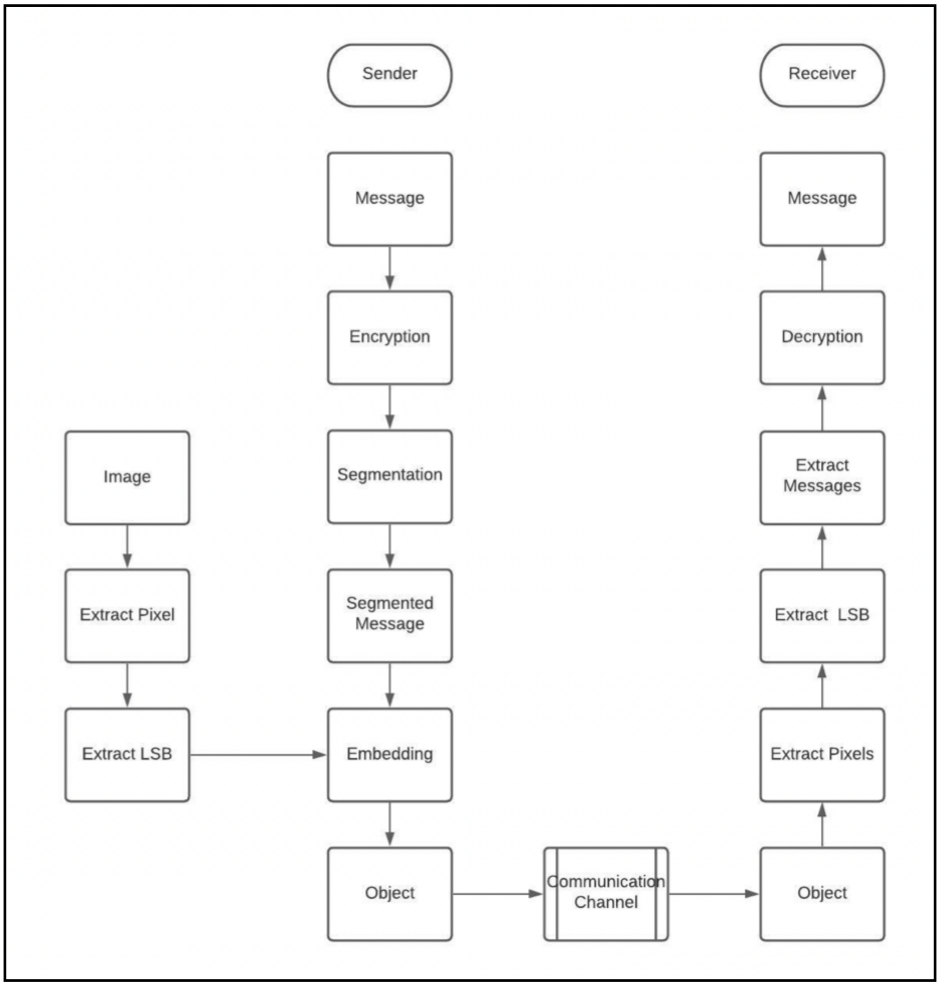


Fig 2. An architecture diagram of how the process works, with both algorithms combined

5. Metrics of Comparison:

We use five main metrics of comparison in order to determine the actual efficiency of the product we've created and how well the combination of the two algorithms is performing.

1. Image size before and after the message is encoded into it
2. The difference in finding out the plaintext prior to and after RSA Encryption
3. The difference in the ratios of the RGB Pixels in the images before and after encoding the message into it
4. Mean Square Error
5. Peak Signal-to-Noise Ratio

The Mean Square Error (MSE) and Peak Signal to Noise Ratio (PSNR) between the steganographic image and its corresponding cover image have been studied and given below as eq. 2 and 3.

MSE=1$H*WPi$, −Si, Hi=12 .... (2)

Where MSE is Mean Square Error, H and W are height, width, and P (i, j) which represents the cover image, and S (i, j) represents its corresponding steganographic image.

$PSNR$=10log 10 $L2MSE$ .... (3)

Where PSNR is the peak signal to noise ratio, L is the peak signal level for a colour image have been taken as 255. In this technique of image steganography, eight bits of data are embedded in 3 pixels of the cover image. The mean square error (MSE) and the peak signal to noise ratio (PSNR) for different steganographic images are shown in Table I. By comparing the PSNR values of all the stego images, it has been analysed that only image(f) as a cover image has given the best PSNR value. The same is true in the case for the MSE values while comparing with different stego images, image(b) as a cover image has given the least MSE value.

The formulae used are as follows:

$$PSNR = 10\ \log\ 10 \frac{(2^n\ - 1)^2}{MSE}$$

$$MSE = \frac{\sum_{M,N}\ [I1\,(m,n) - I2\,(m,n)]^2}{M * N}$$

5.1 Parameter Testing results

Fig No.	Original Image Size	Embedded Image Size	PSNR Value	MSE Value	Payload Capacity	Image Dimensions
a) *Fig 4.1.1 - 4.1.4*	8.32 KB	84.5 KB	78.907510158902 97 dB	0.000836236933797 9094	18834 Bytes	(245, 205, 3)
b) *Fig 4.2.1 - 4.2.4*	8.40 KB	92.3 KB	79.1028602417265 dB	0.000799455577358 8895	18919 Bytes	(201, 251, 3)
c) *Fig 4.3.1 - 4.3.4*	3.57 KB	53.3 K	78.238647508896 89 dB	0.000975473801560 7581	18837 Bytes	(299, 168, 3)
d) *Fig 4.4.1 - 4.4.4*	1.28 KB	4.25 KB	79.082068211355 49 dB	0.000803292181069 9589	18984 Bytes	(225, 225, 3)
e) *Fig 4.5.1 - 4.5.4*	9.15 KB	90.2 KB	79.281569285562 17 dB	0.000767226212680 3974	18899 Bytes	(226, 223, 3)
f) *Fig 4.6.1 - 4.6.4*	237 KB	1.79 MB	98.510517441055 81 dB	0.4897	777600	(1080, 1920, 3)

6. Implementation

6.1 Encrypt.py File

```
import random
max_PrimLength = 1000000000000
def egcd(a, b):
    if a == 0:
        return (b, 0, 1)
    else:
        g, y, x = egcd(b % a, a)
        return (g, x - (b // a) * y, y)
def gcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a


def is_prime(num):
    if num == 2:
        return True
    if num < 2 or num % 2 == 0:
        return False
    for n in range(3, int(num**0.5)+2, 2):
        if num % n == 0:
            return False
    return True
def generateRandomPrim():
    while(1):
```

```
        ranPrime = random.randint(0,max_PrimLength)
        if is_prime(ranPrime):
            return ranPrime
def generate_keyPairs(p,q):
    n = p*q
    print("n ",n)
    phi = (p-1) * (q-1)
    print("phi ",phi)
    e = random.randint(1, phi)
    g = gcd(e,phi)
    while g != 1:
        e = random.randint(1, phi)
        g = gcd(e, phi)

    print("e=",e," ","phi=",phi)
    d = egcd(e, phi)[1]

    d = d % phi
    if(d < 0):
        d += phi
    print ("d= ",d)
    return (e,d)

def decrypt(ctext,n,key):
    try:
        print (ctext)
        text = [chr(pow(char,key,n)) for char in ctext]
```

```
        return "".join(text)
    except TypeError as e:
        print(e)
def encrypt(text,key,n):
    ctext = [pow(ord(char),key,n) for char in text]
    return ctext
def rsa(a,n,p,q):
    public_key,private_key = generate_keyPairs(p,q)
    print("Public: ",public_key)
    print("Private: ",private_key)

    ctext = encrypt(a,public_key,n)
    print("encrypted  =",ctext)
    plaintext = decrypt(ctext,n, private_key)
    print("decrypted =",plaintext)
p=int(input("Enter prime number 1 : "))
q=int(input("Enter prime number 2 : "))
a="sender"
n=p*q
rsa(a,n,p,q)
```

6.2 Image Steganography.py

```
import cv2
import numpy as np
def Binary_convertor(msg):
    if type(msg) == str:
        return ''.join([ format(ord(i), "08b") for i in msg])
```

```python
    elif type(msg) == bytes or type(msg) == np.ndarray:
        return [format(i, "08b") for i in msg]
    elif type(msg) == int or type(msg) == np.uint8:
        return format(msg, "08b")
    else:
        raise TypeError("Input type not supported")
def hide_data(img, sec_msg):
    # Max Byte
    n_bytes = img.shape[0] * img.shape[1] * 3 // 8
    print("Maximum bytes to encode:", n_bytes)

    # Checking if the number of bytes to encode is less than the maximum bytes in the image
    if len(sec_msg) > n_bytes:
        raise ValueError("Error encountered insufficient bytes, need bigger image or less data !!")
    sec_msg += "$t3g0"  #delimeter
    index = 0
    # convert input data to binary format
    bin_sec_msg = Binary_convertor(sec_msg)
    data_len = len(bin_sec_msg)  # Find the length of data that needs to be hidden
    for values in img:
        for pixel in values:
            # convert RGB values to binary format
            r, g, b = Binary_convertor(pixel)
            # modifying the least significant bit only if there is still data to store
            if index < data_len:
                # hide the data into least significant bit of red pixel
                pixel[0] = int(r[:-1] + bin_sec_msg[index], 2)
```

```
            index += 1
        if index < data_len:
            # hide the data into least significant bit of green pixel
            pixel[1] = int(g[:-1] + bin_sec_msg[index], 2)
            index += 1
        if index < data_len:
            # hide the data into least significant bit of  blue pixel
            pixel[2] = int(b[:-1] + bin_sec_msg[index], 2)
            index += 1
        # if data is encoded, break out of the loop
        if index >= data_len:
            break
    return img
def present_data(img):
    binary_data = ""
    for values in img:
        for pixel in values:
            r, g, b = Binary_convertor(pixel)  # convert the red,green and blue values into binary format
            binary_data += r[-1]  # extracting data from the least significant bit of red pixel
            binary_data += g[-1]  # extracting data from the least significant bit of red pixel
            binary_data += b[-1]  # extracting data from the least significant bit of red pixel
    # split by 8-bits
    all_bytes = [binary_data[i: i + 8] for i in range(0, len(binary_data), 8)]
    # convert from bits to characters
    decoded_data = ""
    for byte in all_bytes:
        decoded_data += chr(int(byte, 2))
```

```
        if decoded_data[-5:] == "$t3g0":  # check if we have reached the delimeter which is "#####"
            break
    # print(decoded_data)
    return decoded_data[:-5]  # remove the delimeter to show the original hidden message
def encode_text(data):
    image_name = input("Enter image name(with extension): ")
    image = cv2.imread(image_name)  # Read the input image using OpenCV
    # details of the image
    print("The shape of the image is: ", image.shape)
    resized_image = cv2.resize(image, (500, 500))
    if (len(data) == 0):
        raise ValueError('Data is empty')
    filename = input("Enter the path and name of new encoded image(with extension): ")
    encoded_image = hide_data(image,data)
    cv2.imwrite(filename, encoded_image)
def decode_text():
    # read the image that contains the hidden image
    image_name = input("Enter the path of the image to be decode (with extension) :")
    image = cv2.imread(image_name)
    text = present_data(image)
    return text
def Steganography(userinput,txt):
    if (userinput == 1):
        print("\nEncoding....")
        encode_text(txt)
    elif (userinput == 2):
        print("\nDecoding....")
```

```
        a=decode_text()
        print("Decoded message is " + a)
        li = list((a.split(" ")))
        print(li)
    else:
        raise Exception("Enter correct input")
```

6.3 Final.py

```
# from RSA2 import *
from image_steg_newest import *
from encrypt import *
if __name__=='__main__':
    a = input("Image Steganography \n 1. Encode the data \n 2. Decode the data \n Choose Option: ")
    userinput = int(a)
    if (userinput == 1):
        print ("--->RSA ENCRYPTION<---")
        p=int(input("Enter prime number 1 : "))
        q=int(input("Enter prime number 2 : "))
        n=p*q
        print ("Generating Public/Private keypairs...")
        public, private=generate_keyPairs(p,q)
        print ("Public key : ",public," and private key : ",private)
        msg=input("Enter message to be encrypted : ")
        enc_msg=encrypt(msg,public,n)
        print (enc_msg)
        txt=""
        for a in enc_msg:
```

```
        txt+=str(a)+" "
    print (txt)
    Steganography(userinput,txt)
elif (userinput == 2):
    # print("\nDecoding....")
    # image_name = input("Enter image name(with extension): ")
    enc=decode_text()
    print("Decoded message is " + enc)
    li = list((enc.split(" ")))
    li.pop()
    ab=[]
    for i in li:
        ab.append(int(i))
    print(ab)
    private=int(input("Enter the private Key"))
    n=int(input("Enter n value"))
    dec_msg=decrypt(ab,n,private)
    print(dec_msg)
else:
    raise Exception("Enter correct input")
print ('.join(map(lambda x: str(x), enc_msg)))
print ("Decrypting msf with public key", public, "...")
print ("YOur message : ")
print (decrypt(public, enc_msg))
```

7. Results

```
PS C:\Users\Yash\Desktop\ISAA_PROJECT> python3 final.py
Image Steganography
 1. Encode the data
 2. Decode the data
 Choose Option: 1
--->RSA ENCRYPTION<---
Enter prime number 1 : 11
Enter prime number 2 : 13
Generating Public/Private keypairs...
n  143
phi  120
e= 73   phi= 120
d=  97
Public key :  73  and private key :  97
Enter message to be encrypted : rajaspsir
[75, 58, 2, 58, 37, 8, 37, 40, 75]
75 58 2 58 37 8 37 40 75

Encoding....
Enter image name(with extension): kitten.png
The shape of the image is:  (251, 384, 3)
Enter the path and name of new encoded image(with extension): kitt.png
Maximum bytes to encode: 36144
PS C:\Users\Yash\Desktop\ISAA_PROJECT> python3 final.py
Image Steganography
 1. Encode the data
 2. Decode the data
 Choose Option: 2
Enter the path of the image to be decode (with extension) :kitt.png
Decoded message is 75 58 2 58 37 8 37 40 75
[75, 58, 2, 58, 37, 8, 37, 40, 75]
Enter the private Key97
Enter n value143
[75, 58, 2, 58, 37, 8, 37, 40, 75]
rajaspsir
PS C:\Users\Yash\Desktop\ISAA_PROJECT>
```

Fig 3. Screenshot of the Code Running and giving the Desired Output

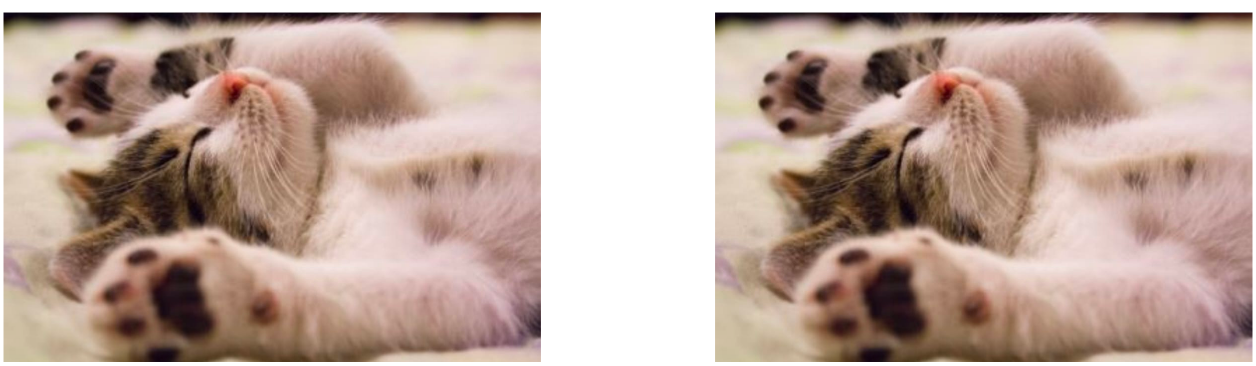
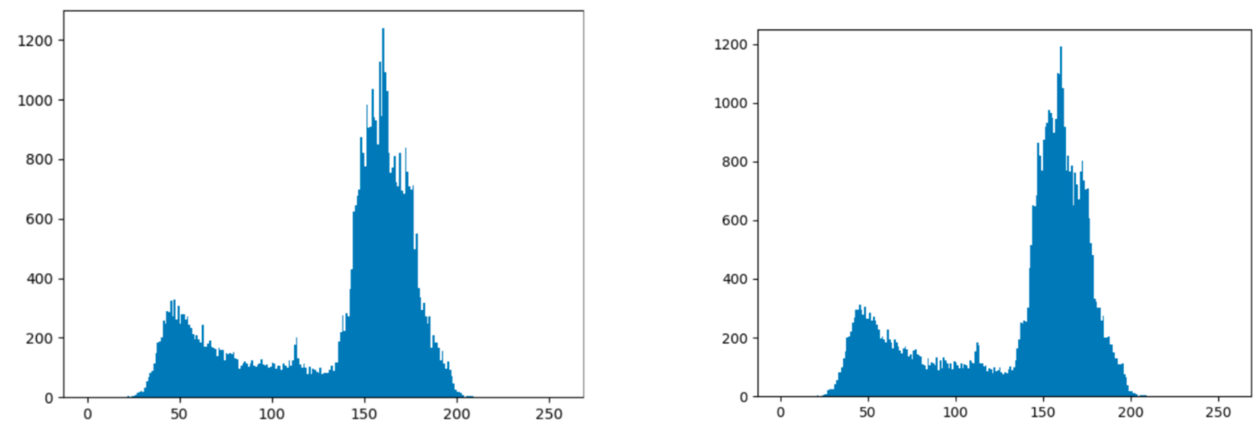


Fig 4.1 Image of Kitten.png before and after Steganography



Histogram **Histogram**

Fig 4.2 Image of the histograms of the comparative images' metrics

8. Conclusion

Clearly, a secured LSB technique and an RSA algorithm for image steganography. Through the use of steganography, we were able to successfully transfer messages as well as a variety of file types without any loss of data or information. Even though the files are of a substantial size, they have been successfully incorporated into the cover image without any data being lost. The RSA encryption algorithm is a robust one that has proven to be somewhat resistant to the effects of time. Because RSA is an asymmetric algorithm, it requires users to have both a public key and a private key in order to facilitate secure communication. Through the utilisation of this method, a successful communication channel between two users was established. The LSB technique has been utilised so that the RGB pixel values can be collected. RSA has been used to encrypt the data, and then that encrypted data was given to LSB. After that, the steganographic image is crafted by applying the LSB technique, which involves the transfer of data from the cover image to the hidden one. Following the successful transmission of this image along with the message that is embedded within it across the network, the message can then be successfully extracted.

9. References

[1] Bose, Abhranil, et al. "Steganography Method Using Effective Combination of RSA Cryptography and Data Compression." 2022 First International Conference on Electrical, Electronics, Information and Communication Technologies (ICEEICT). IEEE, 2022.

[2] Ali, U. A. M. E., Md Sohrawordi, and Md Palash Uddin. "A robust and secured image steganography using LSB and random bit substitution." American Journal of Engineering Research (AJER) 8.2 (2019): 39-44.

[3] Loan, Nazir A., et al. "Secure and robust digital image watermarking using coefficient differencing and chaotic encryption." IEEE Access 6 (2018): 19876-19897.

[4] Vaidya, S. Prasanth, and PVSSR Chandra Mouli. "Adaptive digital watermarking for copyright protection of digital images in wavelet domain." Procedia Computer Science 58 (2015): 233-240.

[5] Shelke, Falesh M., Ashwini A. Dongre, and Pravin D. Soni. "Comparison of different techniques for Steganography in images." International Journal of Application or Innovation in Engineering & Management 3.2 (2014): 171-176.

[6] Gupta, Shailender, Ankur Goyal, and Bharat Bhushan. "Information hiding using least significant bit steganography and cryptography." International Journal of Modern Education and Computer Science 4.6 (2012): 27.

[7] Akhtar, Nadeem, Pragati Johri, and Shahbaaz Khan. "Enhancing the security and quality of LSB-based image steganography." 2013 5th International Conference and Computational Intelligence and Communication Networks. IEEE, 2013.

[8] Karim, SM Masud, Md Saifur Rahman, and Md Ismail Hossain. "A new approach for LSB-based image steganography using a secret key." 14th international conference on computer and information technology (ICCIT 2011). IEEE, 2011.[10] Ghaemmaghami, Shahrokh, and Seyed Mojtaba Seyedhosseini Seyedhosseini Tarzjani. "Detection of LSB replacement and LSB matching steganography using gray level run length matrix." 2009 Fifth International Conference on Intelligent Information Hiding and Multimedia Signal Processing. IEEE, 2009. APA