

STEGANOGRAPHIC ANALYSIS OF AUDIO AND IMAGE MEDIA USING LSB AND RC4 ALGORITHMS

Ilham Firman Ashari*, Ikhsanudin Raka Siwi, Hafizh Londata, Ihtandiko Wicaksono

Department of Production and Industrial Technology, Institut Teknologi Sumatera, Lampung Selatan, Indonesia

e-mail: firman.ashari@if.itera.ac.id, ikhsanuddin.raka@gmail.com, hafiz.londata@gmail.com,
wicaksono.ihtandiko@gmail.com

Received: 21 June 2022 – Revised: 12 June 2023 – Accepted: 13 June 2023

ABSTRACT

In the current digital era, it is deemed essential to ensure security and confidentiality of information when exchanging information through communication networks. This is done to allow recipients to receive the information from senders in its entirety without any interference from third parties who are not entitled to the information. Cryptography and Steganography are some useful methods to secure a confidential message, including the RC4 algorithm as one type of applicable method to secure the original message into a random secret message to make it remain unknown to others. One of the methods used in steganography to secure messages, including images, audio, video, and documents, is the least significant bit (LSB) algorithm. This study aims to analyze the comparison of the two-storage media, namely audio and images using LSB and RC4 in order to see the effect of the LSB and RC4 algorithms on the container media based on the aspects of imperceptibility, fidelity, recovery, and capacity. Having tested the imperceptibility aspect as indicated by the histogram of the image and the audio spectrum, it is clear that there is no difference between the image and audio before and after insertion. The fidelity test of the PSNR (Peak Signal to Noise Ratio) resulted in an average value of > 30 dB, while the recovery test shows 100% success because there is no difference between the original message and after extraction. The capacity test indicates that the larger the size of the container media, the larger the message that can be inserted.

Keywords: Audio, Cryptography, Image, Steganography.

I. INTRODUCTION

THE currently rapid technological development has made it impossible to guarantee data or information security since they can be manipulated by other parties using various methods. As a result, often times, the data or information submitted by the sender does not reach the recipient and/or the contents of the information have changed. To overcome problems regarding confidential data assets, it is necessary to use security methods such as cryptography and steganography [1].

Cryptography studies how to protect data or information from attackers or other irresponsible parties [2]. Cryptography is related to data encryption or plaintext, which is arranged randomly using a key to generate random data known as ciphertext, in addition to a decryption to translate random data or ciphertext using the same key to see the actual data or information [3].

Cryptography can be performed using several approaches and groups, including symmetric key cryptography, public key cryptography, and hash functions [4]. The symmetric key is generally applicable using several cryptographic methods, such as block and stream ciphers, which are included in the modern cryptographic group. One of the algorithms in the stream cipher is the RC4 algorithm, which uses keystream generation by performing the Key Scheduling Algorithm and Pseudo Generate Random Algorithm sub-processes. [5][6][7].

Steganography is a method to hide a secret message in a container to prevent other parties without unwanted rights from knowing the existence of the message. Steganography can be done using several technical methods, including replacement, domain transformation, spread spectrum, statistics, warping, and cover generation [8]. Steganography has several algorithms, one of which is the LSB (Least Significant B), which inserts a message into the digital image media by replacing the last bit of the byte

arrangement containing 8 bits in the digital image [9][10][11]. The LSB algorithm is used due to its ability to insert more messages and to increase security combined with encryption [12]. Bit insertion is done at the end because the 8th or last bit has little effect on the result of reading the data, and thus it is not a big problem to insert the message in the LSB bit.

Research by Nidhi shows that RC4 has advantages over AES in terms of throughput, CPU processing time, memory utilization, encryption and decryption time [13]. Research by [14], which measured throughput, cpu work load, cost energy, disclosed that RC4 is faster and more efficient for large data packets. Another relevant research by Ari, et al, using video and LSB as insertion method, found that the video dimensions did not change in size despite the drawback of limited insertion depending on the number of frames in the video [15]. This study proposed LSB as encoding method and RC4 as the encryption method. To solve problems regarding the data or information security, this study aims to analyze and compare the two methods in order to see the steganographic aspects of the imperceptibility, fidelity, recovery, and capacity of image and audio.

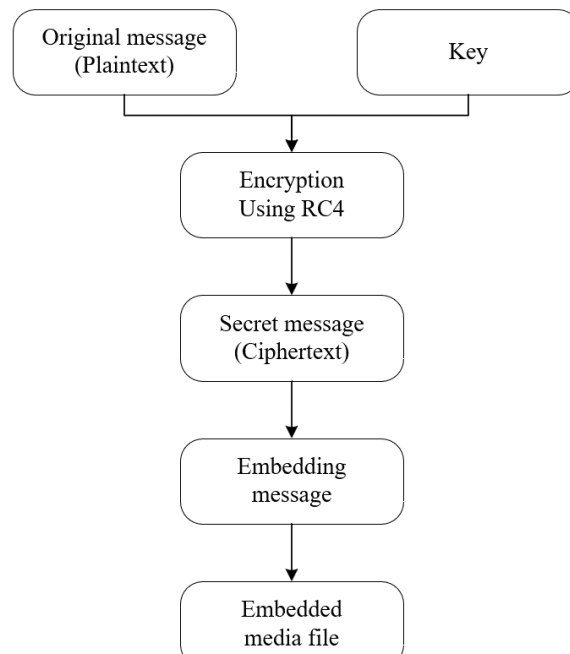


Figure 1. Encoding Process

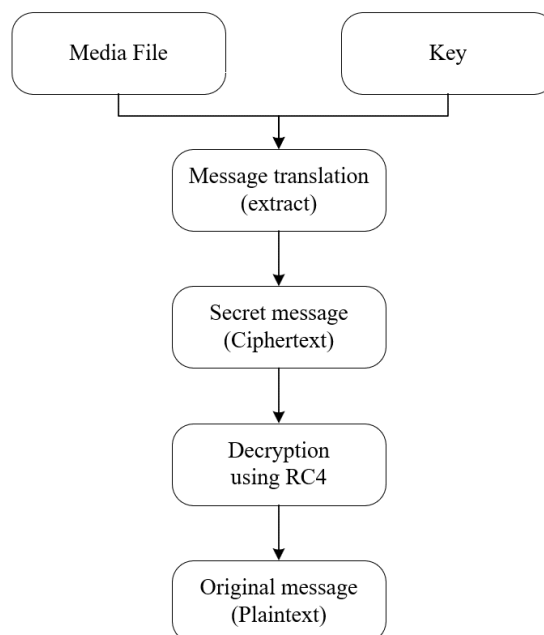


Figure 2. Message decoding

II. RESEARCH METHOD

The cryptographic stage involved the use of the RC4 algorithm combined with LSB for message insertion. It treated a text message as a secret message and the container media as an image with a .png extension and audio with a .wav extension. The program flow was divided into two parts, namely the encoding process and the decoding process. The encoding and decoding processes are presented in Figure 1 and 2.

A. Message Encoding

In the encryption process, the plaintext was encrypted with the RC4 algorithm along with the entered password, followed by inserting a message into a file using the LSB (see Figure 1).

In the encoding process, the original message and the key are entered by users to be then encrypted with the RC4 algorithm so as to produce a secret message in ciphertext form. The secret message was then inserted in the selected media file using the least significant bit algorithm (LSB).

B. Message Decoding

In the decoding process, the user inputted a media file containing the ciphertext message and key before extracting them with LSB to obtain the ciphertext. Subsequently, the ciphertext was decrypted using the RC4 algorithm to produce plaintext (see Figure 2).

In the decoding process, the user inputted a media file containing ciphertext messages and keys entered by the user before extracting the previously inserted media file to obtain a secret message stored in the media file in ciphertext form. This stage was followed by decrypting ciphertext into plaintext using the RC4 algorithm, which generated a plaintext string or original message as an output.

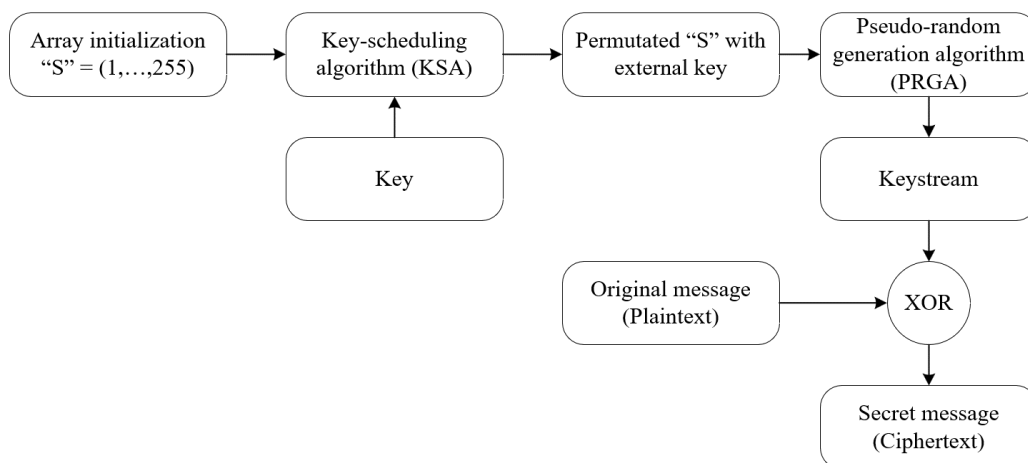


Figure 3. Message Encryption using RC4 algorithm

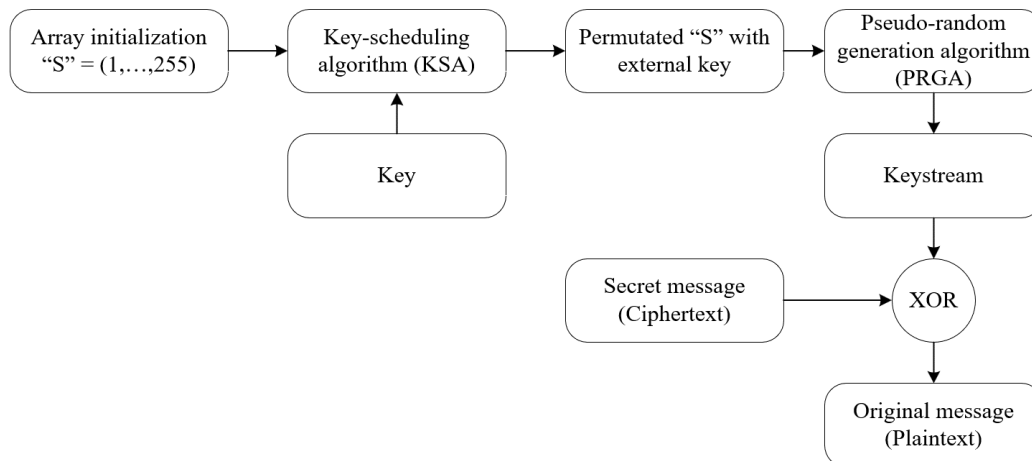


Figure 4. Message Decryption

TABLE 2
 PIXEL BINARY INITIALIZATION

R	G	B	R	G	B	R	G	B	R	G	B
11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111
11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111
11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111
11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111
11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111
11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111
11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111
11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111
11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111
11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111
11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111
11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111
11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111
11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111
11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111
11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111
11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111
11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111
11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111
11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111

TABLE 3
 BINARY INSERTION OF MESSAGES INTO PIXELS

R	G	B	R	G	B	R	G	B	R	G	B
11111111	11111110	11111110	11111111	11111111	11111111	11111111	11111111	11111111	11111110	11111111	11111110
11111110	11111110	11111111	11111110	11111110	11111110	11111110	11111110	11111110	11111110	11111110	11111110
11111111	11111110	11111111	11111111	11111110	11111110	11111111	11111110	11111110	11111110	11111110	11111110
11111110	11111111	11111111	11111110	11111110	11111111	11111110	11111110	11111110	11111110	11111110	11111110
11111111	11111111	11111110	11111110	11111110	11111111	11111110	11111110	11111110	11111110	11111111	11111111
11111110	11111110	11111111	11111111	11111110	11111110	11111110	11111111	11111110	11111111	11111110	11111110
11111111	11111111	11111111	11111110	11111111	11111111	11111111	11111111	11111110	11111110	11111110	11111110
11111111	11111111	11111110	11111111	11111110	11111110	11111110	11111111	11111110	11111110	11111111	11111111
11111110	11111110	11111111	11111110	11111110	11111111	11111110	11111111	11111110	11111110	11111110	11111111
11111110	11111110	11111110	11111111	11111110	11111111	11111110	11111110	11111111	11111110	11111110	11111110
11111110	11111111	11111110	11111111	11111111	11111110	11111111	11111111	11111110	11111110	11111110	11111111
11111110	11111110	11111110	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111
11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111
11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111
11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111
11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111

TABLE 4
 BINARY CHANGES AFTER INSERTION INTO DECIMAL

R	G	B	R	G	B	R	G	B	R	G	B
255	254	254	255	255	255	255	255	255	254	255	254
254	254	255	254	254	254	254	254	254	254	254	254
255	254	255	255	255	254	255	254	254	254	254	254
254	255	255	254	254	255	254	254	254	254	254	254
255	255	254	254	254	255	254	254	254	254	255	255
254	254	255	255	254	254	254	255	254	255	254	254
255	255	255	254	255	255	255	255	254	254	254	254
255	255	254	255	254	254	254	255	254	254	255	255
254	254	255	254	254	255	254	255	254	254	254	255
254	254	254	255	254	255	254	254	255	254	254	254
254	255	254	255	255	254	255	254	254	254	254	255
254	254	254	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255

III. RESULT AND DISCUSSION

The manual calculation determines the RMS (Root Mean Square) and PSNR (Peak Signal to Noise Ratio) image files in Figure 5. The encryption results obtain the ciphertext before being converted into hexadecimal form, which is as follows: 9FA200BA0640C43314EF0D132511485A11. The subsequent stage was changing the ciphertext from hexadecimal to binary, which resulted the following:

1001111110100010000000001011101000000110010000001100010000110011000101001110111100
 001101000100110010010100010001010010000101101000010001.

The required 46 pixels were taken from a total of 136-bit message bits available pixels, according to the number of binary ciphertexts, where each pixel consists of RGB bits, before changing them from decimal form to binary form as Table 1 and 2. After that, sequential insertion of binary ciphertext with

```
import cv2
import numpy as np
import types
```

Figure 7. Libraries used for the entire program

```
mod = 256

# Key Scheduling Algorithm
def KSA(key):
    key_length = len(key)
    # create the array (list in python) "S"
    S = list(range(mod)) # [0,1,2, ..., 255]
    j = 0
    for i in range(mod):
        j = (j + S[i] + key[i % key_length]) % mod
        S[i], S[j] = S[j], S[i] # swap values
    return S

def PRGA(S):
    i = j = 0
    while True:
        i = (i + 1) % mod
        j = (j + S[i]) % mod
        S[i], S[j] = S[j], S[i] # swap values
        K = S[(S[i] + S[j]) % mod]
        yield K

def RC4(key):
    S = KSA(key)
    return PRGA(S)
```

Figure 8. Program code for the RC4 function consisting of KSA and PRGA

```
def hideData(image, secret_message):
    n_bytes = image.shape[0] * image.shape[1] * 3 // 8
    print("Maximum byte yang dapat dimasukkan", n_bytes)
    if len(secret_message) > n_bytes:
        raise ValueError("Kesalahan Jumlah Bytes, Gunakan Media Gambar Lebih Besar Atau Kurangi Data Yang Dimasukkan!")
    secret_message += "#####" # you can use any string as the delimiter
    print(secret_message)
    data_index = 0
    binary_secret_msg = messageToBinary(secret_message)
    #print("biner", binary_secret_msg)

    data_len = len(binary_secret_msg) #Find the length of data that needs to be hidden
    for values in image:
        for pixel in values:
            r, g, b = messageToBinary(pixel)

            if data_index < data_len:
                pixel[0] = int(r[-1] + binary_secret_msg[data_index], 2)
                data_index += 1
            if data_index < data_len:
                pixel[1] = int(g[-1] + binary_secret_msg[data_index], 2)
                data_index += 1
            if data_index < data_len:
                pixel[2] = int(b[-1] + binary_secret_msg[data_index], 2)
                data_index += 1
            # If data is encoded, just break out of the loop
            if data_index >= data_len:
                break
    return image
```

Figure 9. The main function of embedding messages into pictures

```
def showData(image):
    binary_data = ""
    for values in image:
        for pixel in values:
            r, g, b = messageToB:
                binary_data += r[-1]
                binary_data += g[-1] #ekstraksi data dari bit terakhir dari pixel
                binary_data += b[-1] #ekstraksi data dari bit terakhir dari pixel
    # split menjadi 8 bit

    all_bytes = [binary_data[i:i+8] for i in range(0, len(binary_data), 8)]
    # konversi bit menjadi karakter
    decoded_data = ""
    for byte in all_bytes:
        decoded_data += chr(int(byte, 2))
        if decoded_data[-5:] == "####": #check if we have reached the delimeter
            break
    #print(decoded_data)
    return decoded_data[:-5] #remove the delimeter to show the original hidden
```

Figure 10. The main function of extracting messages from within images

```
def encode_text():
    image_name = input("Masukkan nama file (dengan ekstensi PNG): ")
    image = cv2.imread(image_name) # Read the image
    #It is a library of Python bindings designed to solve computer vision problems.

    #details of the image
    print("Ukuran File Gambar : ", image.shape) #check the shape of image to calculate the

    plaintext = input("Masukkan pesan yang ingin disisipkan : ")

    key = input("Masukkan kunci yang ingin digunakan: ")
    if (len(plaintext) == 0):
        raise ValueError('Data Kosong')

    data = encrypt(key, plaintext)

    print(data)
    filename = input("Masukkan nama file gambar yang telah disisipi (dengan ekstensi): ")
    encoded_image = hideData(image, data) # call the hideData function to hide the secret
    cv2.imwrite(filename, encoded_image)
```

Figure 11. The function of encoding data to images

```
def decode_text():
    # read the image that contains the hidden image
    image_name = input("Masukkan nama file gambar yang akan dilakukan decoding (dengan ekstensi PNG): ")
    image = cv2.imread(image_name) #read the image using cv2.imread()

    key = input("Masukkan kunci yang digunakan : ")
    text = showData(image)

    data = decrypt(key, text)
    return data
```

Figure 12. The function of decoding data from images

SB was performed (see Table 3). This process would change the bit value into decimal form to be used for RMS and PSNR calculations, as written as Table 4.

The first step before the process of encryption, decryption, insertion, and extraction was to import the required libraries, namely cv, numpy, and types. The library used along with the code can be seen in Figure 7. Another function was for RC4, which consists of KSA (key scheduling algorithm) and PRGA (Pseudo random generator automation) functions, which can be seen in Figure 8. The main function to embed messages into images is depicted in Figure 9. The main function for extracting messages from images is illustrated in Figure 10. The functions to change messages and insert them into images can be seen in the data encoding function to images in Figure 11. The function to retrieve messages from images

```
def encode_audio_data():
    import wave

    nameoffile=input("Masukkan nama file (dengan ekstensi) : ")
    song = wave.open(nameoffile, mode='rb')

    nframes=song.getnframes()
    frames=song.readframes(nframes)
    frame_list=list(frames)
    frame_bytes=bytearray(frame_list)

    plaintext = input("Masukkan pesan yang ingin disisipkan : ")
    key = input("Masukkan kunci yang ingin digunakan : ")
    data_input = encrypt(key, plaintext)

    res = ''.join(format(i, '08b') for i in bytearray(data_input, encoding='utf-8'))
    # print("nBit hasil konversi :- " + (res))
    length = len(res)
    # print("\nPanjang Bit hasil konversi :- ",length)

    data = data_input + '####'

    result = 1

    for c in data:
        bits = bin(ord(c))[2:].zfill(8)
        result.extend([int(b) for b in bits])

    j = 0
    for i in range(0,len(result),1):
        res = bin(frame_bytes[j])[2:].zfill(8)
        if res[len(res)-4]== result[i]:
            frame_bytes[j] = (frame_bytes[j] & 253) #253: 11111101
        else:
            frame_bytes[j] = (frame_bytes[j] & 253) | 2
            frame_bytes[j] = (frame_bytes[j] & 254) | result[i]
        j = j + 1

    frame_modified = bytes(frame_bytes)

    stegofile=input("Masukkan Nama File Yang Telah Disisipkan (dengan ekstensi) :- ")
    with wave.open(stegofile, 'wb') as fd:
        fd.setparams(song.getparams())
        fd.writeframes(frame_modified)
    print("\nData berhasil di sisipkan pada file audio.")
    song.close()
```

Figure 13. The main function of encoding data into audio

```
def decode_aud_data():
    import wave

    nameoffile=input("Masukkan nama file yang akan dilakukan decoding (dengan ekstensi) : ")
    song = wave.open(nameoffile, mode='rb')

    key = input("Masukkan kunci yang digunakan : ")

    nframes=song.getnframes()
    frames=song.readframes(nframes)
    frame_list=list(frames)
    frame_bytes=bytearray(frame_list)

    extracted = ""
    text = ""
    data = ""
    p=0
    for i in range(len(frame_bytes)):
        if (p==1):
            break
        sec_bda[frame_bytes[i][2:1+3]]+=1

    if res[len(res)-2]==0:
        extracted+=res[len(res)-4]
    else:
        extracted+=res[len(res)-1]

    all_bytes = [ extracted[i: i+8] for i in range(0, len(extracted), 8) ]
    decoded_data = ""
    for byte in all_bytes:
        decoded_data += chr(int(byte, 2))
        if decoded_data[-5:] == "#####": #check if we have reached the delimiter which is #####
            p=1
            break
        # print("The Encoded data was :—",decoded_data[:-5])
        # p+=1
    # print(decoded_data)
    text = decoded_data[:-5] #remove the delimiter to show the original hidden message

    data = decrypt(key, text)
    # print(data)
    print("Pesan Yang Tersembunyi Adalah", data)
```

Figure 14. The main function of decoding data from audio files.

```
def Steganography_Image():  
    print("\n\t\t IMAGE STEGANOGRAPHY OPERATIONS")  
    print("1. Encoding Pesan")  
    print("2. Decoding Pesan")  
    print("3. Keluar")  
    userInput = int(input("Masukkan Pilihan :"))  
    if (userInput == 1):  
        print("\nEncoding....")  
        encode_text()  
    elif (userInput == 2):  
        print("\nDecoding....")  
        print("Pesan Yang Tersembunyi Adalah " + decode_text())  
    elif (userInput == 3):  
        exit()  
    else:  
        raise Exception("Masukkan Pilihan")
```

Figure 15. The main function of the image steganography menu

```
def Steganography_Audio():
    print("\n\t\t\tAUDIO STEGANOGRAPHY OPERATIONS")
    print("1. Encoding Pesan")
    print("2. Decoding Pesan")
    print("3. Keluar")
    choice1 = int(input("Masukkan Pilihan : "))
    if choice1 == 1:
        _print("\n\t\t\tEncoding...")
        encode_audio_data()
    elif choice1 == 2:
        print("\n\t\t\tDecoding...")
        decode_audio_data()
    elif choice1 == 3:
        _exit()
    else:
        raise Exception("Pilihan Salah")
```

Figure 16. The main function of the audio steganography menu



Figure 17. Original Image Files

and display the contents of the message is the decode text function which is presented in Figure 12. The main function for encoding messages into audio is displayed in Figure 13. The main function for decoding data from audio files is demonstrated in Figure 14. The main menu for encoding and decoding messages in image files, is depicted in the steganography_image function as shown in Figure 15. The process of encoding and decoding messages from audio files using the audio steganography function is displayed in Figure 16.

A. Plaintext Encryption System Scheme and Insertion on Image Media

This secret message insertion test phase is run three times to check the image file size. The image file uses a PNG image with an original file size of 5.56 KB and a size (pixel) of 225x225. The original image file is illustrated in Figure 17.

TABLE 5
 RESULTS OF ENCRYPTION AND STEGANOGRAPHY TESTS ON IMAGE MEDIA

File Name	Plaintext	Key	Size	Pixel	Bit Depth
Gambar_asli.png	-	-	24.3 KB	225 x 225	24
Enkripsi 1.png	teknikinformatika	kriptografi	24.3 KB	225 x 225	24
Enkripsi 2.png	institutteknologisumatera	tugasbesarkriptografi	24.3 KB	225 x 225	24

TABLE 6
 RESULTS OF ENCRYPTION AND STEGANOGRAPHY TESTS ON AUDIO MEDIA

File Name	Plaintext	Key	Size (bytes)	Size on disk (bytes)	Bit rate (kbps)
Audion Asli.wav	-	-	5.119.532	5.120.000	1411
Enkripsi 1.wav	teknikinformatika	kriptografi	5.119.532	5.120.000	1411
Enkripsi 2.wav	institutteknologisumatera	tugasbesarkriptografi	5.119.532	5.120.000	1411

TABLE 7
 DECODING EXPERIMENT RESULTS ON AUDIO MEDIA

Name	Previous Plaintext	Key	Final Plaintext	Description
Audion Asli	-	-	-	Success
Decode 1	teknikinformatika	kriptografi	teknikinformatika	Success
Decode 2	institutteknologisumatera	tugasbesarkriptografi	institutteknologisumatera	Success

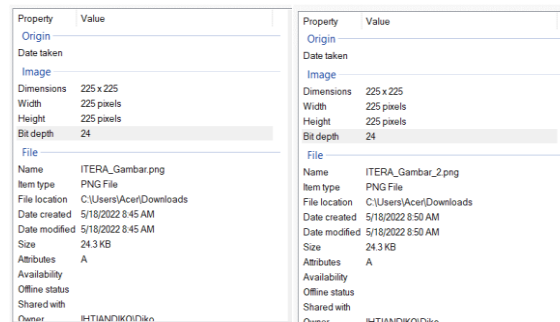


Figure 18. Results of Encryption and Steganography Tests on Image Media

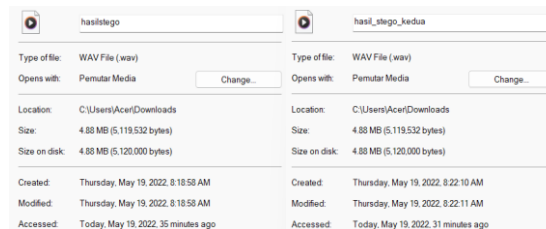


Figure 19. Results of Encryption and Steganography Tests on Audio Media



Figure 20. First Audio Cut

```
Decoding....
Enter name of the file to be decoded :- pemotongan1.wav
Enter data key to be decoded : kriptografi
teknikinformatika
Decoded message is teknikinformatika
```

Figure 21. Decode Results at First Cut

After the encryption stage combined with RC4 cryptography, an image file is inserted using the LSB method with the image media extension .PNG, which was performed through three trials on different plaintexts. The results of the test are presented in Table 5.

B. Plaintext Encryption System Scheme and Insertion on Audio Media

In the test phase, the insertion of this secret message was carried out three times to identify the difference level in the size of the audio file. Audio files were in the WAV format with an original file

TABLE 8
 IMPERCEPTIBILITY ANALYSIS RESULTS BEFORE EMBEDDING


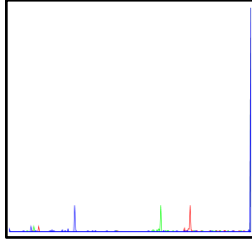
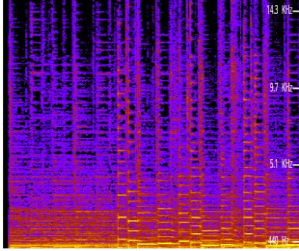
Image before embedding	Histogram before embedding	Audio detail before embedding	Audio spectrum before embedding
		<div>Length 00:00:29 Audio Bit rate 1411kbps</div>	

TABLE 9
 RESULTS OF IMPERCEPTIBILITY ANALYSIS AFTER EMBEDDING


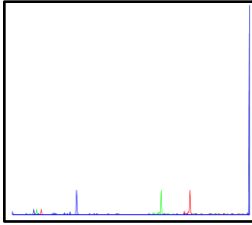
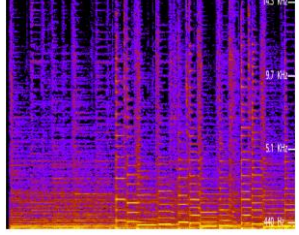
Image after embedding	Histogram after embedding	Audio detail after embedding	Audio spectrum after embedding
		<div>Length 00:00:29 Audio Bit rate 1411kbps</div>	

TABLE 10
 FIDELITY ANALYSIS RESULTS

No	File name	Message size (byte)	Previous size (byte)	Final size (byte)	PSNR
1	ITERA.png	1000	5012	5012	65.32 dB
2	Peta.png	512	28580	28580	78.64 dB
3	Nokia.wav	1000	5.119.466	5.119.466	62.56 dB

TABLE 11
 RECOVERY ANALYSIS RESULTS

No	File name	Message size (byte)	Embedding	Extraction	Final size (byte)
1	ITERA.png	5012	Success	Success	5012
2	Hapis.png	1362621	Success	Success	1362621
3	nokia.wav	5119566	Success	Success	5119566
4	samsung.wav	3732746	Success	Success	3732746

TABLE 12
 CAPACITY ANALYSIS RESULTS

No	File name	File size	Maximum bit value
1	ITERA.png	5012	18984
2	Peta.png	1362621	114345
3	Sepatu.png	5119566	513012
4	Bird.png	3732746	871112

size of 5.119.566 bytes, an on-disk size of 5.124.096 bytes, and a bit rate of 1411 kbps. Once the encryption process was carried out using a combination of cryptographic techniques after being inserted into an audio file using the RC4 and LSB methods, the file was generated in WAV format and tested three times with different plain text. The test results are shown in Table 6.

The test results in Figure 19 highlight the different size of the audio file before and after testing the ciphertext insertion into the audio media using the LSB method. The original audio file size was 5.119.532 bytes, which was exactly the same as the audio after testing of 5.119.532 bytes. However, the size for the first and second encryption did not change even though they had different plaintext and keys. It was also revealed that there was no bit rate change for each test.

C. Scheme of Ciphertext Extraction System in Audio and Ciphertext Decryption

1) Testing decode method

The decode method was tested to obtain secret messages from audio files that have been embedded in the previous stage. The test was carried out 3 times with the following results shown in Table 7.

2) *Audio Trimming Test*

In the audio trimming test, the encryption and steganography results of the image are cut in Encryption 1. The process was conducted 4 times by cutting the audio every 8 seconds. In the subsequent stage was the process of checking the results of the message decryption contained in the audio file, resulting in some cuts as presented in Figure 20.

The contents of the message were checked based on the results of the audio cut in the first 8 seconds. The secret message embedded in the audio for the first 8 seconds could be detected and read (see figure 21). Based on the several trials above, an analysis of aspects of steganography including imperceptibility, fidelity, recovery, and capacity is obtained.

3) *Imperceptibility*

The existence of a message in the container media cannot be detected either by the senses of hearing or sight on the image and audio media after the message is inserted. To measure imperceptibility, histogram and amplitude visualization of image files and audio files are used. The results are shown in Tables 8 and 9.

The trials indicates that the existence of messages in images and audio cannot be visualized, which was attributed to the fact that the RGB histogram and the image showed no difference, which was also the case for the audio spectrum. The imperceptibility aspect generated very good results.

4) *Fidelity*

Stego file quality also generates good result if the quality of the stego file is not much different from the original file as seen from the comparison between the image files or audio files before and after insertion. The RMS (Root Mean Square) and PSNR (Peak Signal Noise Ratio) standards are used in measuring image files. The RMS was calculated using (1) and PSNR was calculated using (2) where *pixel1* is wide dimension, *pixel2* is high dimension, and *RGB* is number of RGB layers. The results of the fidelity analysis can be seen in Table 10.

$$RMS = \sqrt{\frac{1}{pixel1 \times pixel2 \times RGB} \times ((255 - 255)^2 + (255 - 254)^2 + \dots + (255 - 255)^2)} \quad (1)$$

$$PSNR = 20 \times \log_{10}\left(\frac{256}{RMS}\right) \quad (2)$$

It is obvious that the above experiment indicates no difference in terms of fidelity or the size of the image file before and after insertion with an average PSNR result of 68.84dB. These results indicate that the image and audio quality of the steganography is very good, with the minimum standard of ≥ 30 dB [16].

5) *Recovery*

The recovery indicates that the image files and audio files with inserted message can be extracted again which can be accessed by the recipient. The results of the analysis of recovery can be seen in Table 11.

Based on the trials above, it can be seen that the recovery or image and audio files that have been inserted with messages can be extracted again so that they form the original message that was originally inserted into the media file.

6) *Capacity*

At capacity where this aspect is very important because as many messages as possible must be inserted without affecting the quality of the cover file in audio and image files. The results of the analysis of capacity can be seen in Table 12.

Based on the trials, it can be seen that the maximum capacity or number of image and audio files depends on the number of pixels or the size of the image file and the duration of the audio file used. The larger the file size the greater the number of bits generated. The results of the analysis and conclusions can be seen in Table 13.

It is conclusive that with the same number of message sizes, messages are better inserted in media images, but image files have a smaller size than audio files, because image files are rather difficult to perceive compared to audio files that are perceived by hearing.

TABLE 13
 CONCLUSION ANALYSIS OF TESTS OF IMPERCEPTIBILITY, FIDELITY, RECOVERY, AND CAPACITY ASPECTS

Media	Imperceptible	Fidelity	Recovery	Capacity
Image	The existence of messages in images cannot be visualized because there is almost no change in the image after the message is inserted, this is proven through visualization through the RGB histogram of the image	The quality of the images used before and after the message is inserted does not look different. From observations, it was found that the average PSNR value was above > 30 dB	Messages that have been inserted and hidden in media images can be extracted again, with a 100% recovery rate	The size of the message that can be inserted into an image can be said to depend on the container media. The larger the size of the container media, the larger the bits that can be inserted
Audio	The existence of the message in the image cannot be visualized because there is almost no change in the image after the message is inserted, this is proven through visualization through the audio spectrum	The audio quality used before and after the message is inserted doesn't look different. From observations, it was found that the average PSNR value was above > 30 dB	Messages that have been inserted and hidden in audio media can be extracted again, with a 100% recovery rate	The size of the message that can be inserted into an audio can be said to depend on the receiving media. The larger the size of the container media, the larger the number of bits that can be inserted

Several aspects need to be analyzed in the cryptographic method, namely the RC4 algorithm, including computational speed and algorithm security. In the RC4 algorithm, computational speed is affected by the program code used, such as the use of libraries and bit processing, computational speed is also influenced by external factors, such as the processor used in a device. In encryption, the time needed to form a secret message or ciphertext is around 1.2 ms. In using the RC4 Algorithm the keystream is generated by xoring between the external key and the initialization S-Box, which is generated randomly to make it possible to have the same S-Box when generated repeatedly to translate the generated original message. In addition, RC4 encryption is an XOR operation between plaintext data bytes and a random byte stream generated from the keystream, so that an attacker may determine some bytes of the original message by XORing two cipher byte sets and looking at the pattern used in the two cipher sets regardless of the sequence key.

IV. CONCLUSION

Based on the testing and analysis regarding the implementation of steganography with the LSB method on audio and image media using the RC4 algorithm, it can be concluded that from the imperceptibility aspect, the insertion of secret messages in images and audio is very safe because there is no visible difference in eye sight and ear hearing as evaluated from the image histogram and audio spectrum. From the aspect of fidelity, when inserting messages on images and audio, the file size increases, but the increase in file size does not affect the number of characters inserted. PSNR produces an average value of > 30 dB. From the recovery aspect, secret messages or plaintext in the insertion of images and audio obtain the same results in the decoding process. This shows that the recovery aspect was 100% successful. The aspect of capacity indicates that the larger the size of the container media, the larger the message that can be inserted. This system makes it possible to exchange information more safely, where information is inserted into image and audio media. Further research is expected to use other cryptographic algorithms, because the flow key or keystream used in the encryption and decryption of the RC4 algorithm is the same and can be solved by an attacker.

REFERENCES

- [1] A. P. Ratnasari and F. A. Dwiyanto, "Metode steganografi citra digital," *Sains, Apl. Komputasi dan Teknol. Inf.*, vol. 2, no. 2, 2020.
- [2] A. E. Setiawan, A. Pasaribu, and A. E. Setiawan, "Penerapan Steganografi Pada Citra Digital Menggunakan Metode Least Significant Bit (LSB) Kombinasi RC4 Berbasis Mobile Android," *Aisyah J. Informatics Electr. Eng.*, vol. 2, no. 1, pp. 18–28, 2020.
- [3] R. Saha, G. Geetha, G. Kumar, T.-H. Kim, and W. J. Buchanan, "MRC4: a modified rc4 algorithm using symmetric random function generator for improved cryptographic features," *IEEE Access*, vol. 7, pp. 172045–172054, 2019.
- [4] S. W. Siahaan, L. C. Purba, K. D. R. Sianipar, and I. Gunawan, "Pengamanan Data Teks Menggunakan Algoritma Kriptografi RC4 Dari Serangan Brute Force," *TECHSI-Jurnal Tek. Inform.*, vol. 11, no. 2, pp. 229–236, 2019.
- [5] D. Novianto, "Implementasi Keamanan Berkas Menggunakan Teknik Steganografi dan Algoritma Kriptografi menggunakan Metode Least Significant Bit (LSB) dan Algoritma Rivest Code 4 (RC4)," *JUTIM (Jurnal Tek. Inform. Musirawas)*, vol. 3, no. 2, pp. 92–98, 2018.
- [6] R. Rifki, A. Septiarini, and H. R. Hatta, "Cryptography using random Rc4 stream cipher on SMS for android-based smartphones," *IJACSA) Int. J. Adv. Comput. Sci. Appl.*, vol. 9, no. 12, 2018.
- [7] A. E. D. Riad, H. K. Elminir, A. R. Shehata, and T. R. Ibrahim, "Security evaluation and encryption efficiency analysis of rc4 stream cipher for converged network applications," *J. Electr. Eng.*, vol. 64, no. 3, pp. 196–200, 2013.

- [8] N. Nursobah, S. Lailiyah, and A. Kurnia, "Implementasi Steganografi Pesan Teks ke dalam File Aaudio (. MP3) dengan Algoritma Advanced Encryption Standard dan Least Significant Bit," *J. IT*, vol. 10, no. 2, pp. 84–100, 2019.
- [9] O. Soleh, F. Alfiah, and B. Yusuf, "Perancangan Aplikasi Steganografi Dengan Teknik LSB dan Algoritma RC4 & Base64 Encoding," *Technomedia J.*, vol. 3, no. 1 Agustus, pp. 1–15, 2018.
- [10] H. Antonio, P. W. C. Prasad, and A. Alsadoon, "Implementation of cryptography in steganography for enhanced security," *Multimed. Tools Appl.*, vol. 78, no. 23, pp. 32721–32734, 2019.
- [11] I. F. Ashari, "The Evaluation of Audio Steganography To Embed Image Files Using Encryption and Snappy Compression," *Indones. J. Comput. Sci.*, vol. 11, no. 2, 2022.
- [12] I. F. Ashari, A. W. Bhagaskara, J. M. Cakrawarty, and P. R. Winata, "Image Steganography Analysis Using GOST Algorithm and PRNG Based on LSB," *Techno. Com*, vol. 21, no. 3, pp. 700–713, 2022.
- [13] N. Singhal and J. P. S. Raina, "Comparative analysis of AES and RC4 algorithms for better utilization," *Int. J. Comput. Trends Technol.*, vol. 2, no. 6, pp. 177–181, 2011.
- [14] P. Prasithsangaree and P. Krishnamurthy, "Analysis of energy consumption of RC4 and AES algorithms in wireless LANs," in *GLOBECOM'03. IEEE Global Telecommunications Conference (IEEE Cat. No. 03CH37489)*, 2003, vol. 3, pp. 1445–1449.
- [15] U. A. Anti, A. H. Kridalaksana, and D. M. Khairina, "Steganografi Pada Video Menggunakan Metode Least Significant Bit (LSB) Dan End Of File (EOF)," 2017.
- [16] I. F. Ashari, "The Evaluation of Image Messages in MP3 Audio Steganography Using Modified Low-Bit Encoding," *Evaluation*, vol. 14, no. 2, 2021.