# DAFTAR PUSTAKA

Rika Susanti. (2020). Rahasia Medis di Era Disruption. [Internet]. Tersedia di : https://fk.unand.ac.id/id/berita/item/744-rahasia-medis-di-era-disruption.html

Kementrian Kesehatan Republik Indonesia. (2020). *Pedoman Pencegahan dan Pengendalian Coronavirus Disease (COVID-19) Revisi ke-4.* Direktorat Jenderal Pencegahan dan Pengendalian Penyakit, Jakarta. 136 hal.

Sutoyo, T, E Mulyanto, V Suhartono, & OD Nurhayati. (2009). Teori Pengolahan Citra Digital. Yogyakarta: Andi.

Sulistiyanti, Sri Ratna, FX Arinto Setyawan, & Muhamad Komarudin. (2016). *Penglolahan Citra Dasar dan Contoh Penerapannya.*Yogyakarta: Teknosain.

Irnawati. (2018). Studi Dosis Radiasi Pada Pemeriksaan Computer Tomography (CT) Scan Dengan Nilai Computer Tomography Dose Index (CTDI) di Rumah Sakit Bhayangkara Makassar. JFT, 2(5): 177-183.

Irnawati. (2018). Studi Dosis Radiasi Pada Pemeriksaan Computer Tomography (CT) Scan Dengan Nilai Computer Tomography Dose Index (CTDI) di Rumah Sakit Bhayangkara Makassar. Skripsi. Makassar: UIN Alauddin Makassar.

Aditya, Yogie, Andhika Pratama, & Alfian Nurlifa. (2010). Studi Pustaka Untuk Steganografi Dengan Beberapa Metode. *Prosiding dari Seminar Nasional Aplikasi Teknologi Informasi 2010*, Yogyakarta: 19 Juni 2010. Hal.32-35.

Pranoto, Budi. (2011). Steganografi pada Citra Digital Menggunakan Metode Spread Spectrum dan Metode Least Significant Bit (LSB) Modification. Skripsi. Pekanbaru: Universitas Islam Negeri Sultan Syarif Kasim Riau.

Noviardhi, Albertus Guritno. (2008). Kompresi Citra Menggunakan METODE Discrete Cosine Transform ( DCT ). Skripsi. Yogyakarta: Universitas Sanata Dharma.

Huang,F., J.Huang, & Y.Q.Shi. (2012). New Channel Selection Rule for JPEG Steganography. IEEE Transaction On Information Forensics and Security, 7(4): 1181-1191.

Biantara, I Made Divya, I Made Sudana, Alfa Faridh Suni, Suryono, & Arimaz Hangga. (2015). Modifikasi Metode Linear Congruential Generator Untuk Optimalisasi Hasil Acak. *Prosiding dari Seminar Nasional Informatika 2015*, Yogyakarta: 14 November 2015. Hal.182-186.

Hernandes, A., Hartini, & Dewi Sartika. (2019). Steganografi Citra Menggunakan Metode Least Significant Bit (LSB) Dan Linear Congruential Generator (LCG). Jurnal Teknik Informatika dan Sistem Informasi, 5(2): 137-150.

Ridwan, Muhammad, & Yusi Yustikasari. (2020). Implementasi Steganografi Text to Image JPEG Menggunakan Discrete Cosine Transfrom. Tasikmalaya: Universitas Siliwangi. https://www.researchgate.net/publication/338355925_Implementasi_Steganografi_Text_to_Image_JPEG_Menggunakan_Discrete_Cosine_Transfrom

Doo, S.Y, S. Tena, & V.M. Ndolu. (2019). Implementasi Pengamanan Data Menggunakan Metode Kriptografi Hill Cipher dan Steganografi Least Significant Bit (LSB) Pada Media Citra Digital. Jurnal Media Elektro, 8(2): 93-99.

Fajrin, Hanifah Rahmi. (2016). Perbandingan Metode Untuk Perbaikan Kualitas Citra Mammogram. Jurnal SIMETRIS, 7(2): 657-664.

Handayani, H.H. (2010). Digital Image Matching Method Using Normalized Cross Correlation (NCC). Geodi, 6(1):1-5.

Saleh, A., M. Harahap, & E. Indra. (2020). Kombinasi Jaringan Learning Vector Quantization Dan Normalized Cross Correlation Pada Pengenalan Wajah. Jusikom Prima, 3(2):13-20.

# LAMPIRAN

```
from tkinter import *
from tkinter import ttk, messagebox
from tkinter import filedialog
import cv2 as cv
from PIL import ImageTk,Image
import numpy as np
import bitstring
import time

# Internal
import image_secret_e as enk
import image_secret_d as dek
import blok_img as blok
import zigzag as zz
import embed

## Source File
SECRET_IMAGE_FILE    = "D:\\GUI\\berkas\\Secret\\secret.png"
STEGO_IMAGE_FILE     = "D:\\GUI\\berkas\\Stego\\stego.png"
OUTPUT_IMAGE_FILE     = "D:\\GUI\\berkas\\Ekstrak\\Hasil.png"

class gui:

    def __init__(self):

        self.window = Tk()
        self.window.title("Steganografi")
        self.window.geometry('1150x590')
        self.window.resizable(0,0)

        self.frm1 = ttk.LabelFrame(self.window, text="LCG", padding=10)
        self.frm1.grid(row=0,column=0, padx=20, pady=20, sticky=NW)
        self.frm2 = ttk.LabelFrame(self.window, text="LCG Image", padding=10)
        self.frm2.grid(row=1,column=0, padx=20, pady=0, sticky=W)
        self.frm3 = ttk.LabelFrame(self.window, text="Penyisipan", padding=10)
        self.frm3.grid(row=0,column=1, pady=20, sticky=N)
        self.frm4 = ttk.LabelFrame(self.window, text="Stego Image", padding=10)
        self.frm4.grid(row=1,column=1, sticky=W)
        self.frm5 = ttk.LabelFrame(self.window, text="Ekstraksi", padding=10)
        self.frm5.grid(row=0,column=2, padx=20, pady=20, sticky=NW)
        self.frm6 = ttk.LabelFrame(self.window, text="Secret Image", padding=10)
        self.frm6.grid(row=1,column=2, padx=20, sticky=W)

        paddings = { 'padx' :5 , 'pady' :5}

        # Variable
        self.h = 0
```

```
        self.w = 0
        self.brs_a = StringVar()
        self.brs_b = StringVar()
        self.klm_a = StringVar()
        self.klm_b = StringVar()
        self.image = []
        self.cover = []
        self.stego = []
        self.bin_secret = ""

        # Kunci
        self.k_brs_a = StringVar()
        self.k_brs_b = StringVar()
        self.k_brs_m = StringVar()
        self.k_klm_a = StringVar()
        self.k_klm_b = StringVar()
        self.k_klm_m = StringVar()

        # Frame 1
        ttk.Label(self.frm1, text="Kunci Baris").grid(row=1, column=0, sticky=W,
**paddings)
        ttk.Label(self.frm1, text=":").grid(row=1, column=1, sticky=W,
**paddings)
        ttk.Label(self.frm1, text="a= ").grid(row=1, column=2, sticky=W,
**paddings)
        ttk.Label(self.frm1, text="b= ").grid(row=1, column=4, sticky=W,
**paddings)
        ttk.Label(self.frm1, text="m= ").grid(row=1, column=6, sticky=W,
**paddings)

        baris_a = ttk.Entry(self.frm1, textvariable=self.brs_a
,width=6).grid(row=1, column=3, sticky=W)
        baris_b = ttk.Entry(self.frm1, textvariable=self.brs_b,
width=6).grid(row=1, column=5, sticky=W)
        baris_m = ttk.Entry(self.frm1, width=6, state=DISABLED)
        baris_m.grid(row=1, column=7, sticky=W)

        ttk.Label(self.frm1, text="Kunci Kolom").grid(row=2, column=0, sticky=W,
**paddings)
        ttk.Label(self.frm1, text=":").grid(row=2, column=1, sticky=W,
**paddings)
        ttk.Label(self.frm1, text="a= ").grid(row=2, column=2, sticky=W,
**paddings)
        ttk.Label(self.frm1, text="b= ").grid(row=2, column=4, sticky=W,
**paddings)
        ttk.Label(self.frm1, text="m= ").grid(row=2, column=6, sticky=W,
**paddings)
```

```
        kolom_a = ttk.Entry(self.frm1, textvariable=self.klm_a,
width=6).grid(row=2, column=3, sticky=W)
        kolom_b = ttk.Entry(self.frm1, textvariable=self.klm_b,
width=6).grid(row=2, column=5, sticky=W)
        kolom_m = ttk.Entry(self.frm1, width=6, state=DISABLED)
        kolom_m.grid(row=2, column=7, sticky=W)

        ttk.Button(self.frm1, text="Buka Gambar", command=self.upload_img ,
width=15).grid(row=3, column=3, columnspan=4, sticky=W)
        ttk.Button(self.frm1, text="Ok", command=self.acak).grid(row=3,
column=6, columnspan=6, sticky=W, **paddings)

        # Frame 2
        canvas_lcg = Canvas(self.frm2, width=330, height=330, bg="white")
        canvas_lcg.grid(row=0, column=0)

        # Frame 3
        ttk.Button(self.frm3, text="Gambar Cover",
command=self.upload_cover).grid(row=0, column=0, sticky=W, **paddings)
        ttk.Button(self.frm3, text="Sisipkan", command=self.embed).grid(row=0,
column=1, columnspan=4, sticky=W, **paddings)
        ttk.Label(self.frm3,
text="                                                ").grid(row=0, column=5,
sticky=W, **paddings)
        ttk.Label(self.frm3, text="Status Cover").grid(row=1, column=0 ,
sticky=W, **paddings)
        ttk.Label(self.frm3, text=" :").grid(row=1, column=2, sticky=W,
**paddings)
        ttk.Label(self.frm3, text="").grid(row=2, column=0, columnspan=2,
sticky=W, **paddings)

        # Frame 4

        canvas_lcg = Canvas(self.frm4, width=330, height=330, bg="white")
        canvas_lcg.grid(row=0, column=0)

        # Frame 5
        ttk.Label(self.frm5, text="Kunci Baris").grid(row=1, column=0, sticky=W,
**paddings)
        ttk.Label(self.frm5, text=":").grid(row=1, column=1, sticky=W,
**paddings)
        ttk.Label(self.frm5, text="a= ").grid(row=1, column=2, sticky=W,
**paddings)
        ttk.Label(self.frm5, text="b= ").grid(row=1, column=4, sticky=W,
**paddings)
```

```
        ttk.Label(self.frm5, text="m= ").grid(row=1, column=6, sticky=W,
**paddings)

        baris_a = ttk.Entry(self.frm5, textvariable=self.k_brs_a
,width=6).grid(row=1, column=3, sticky=W)
        baris_b = ttk.Entry(self.frm5, textvariable=self.k_brs_b,
width=6).grid(row=1, column=5, sticky=W)

        baris_m = ttk.Entry(self.frm5, textvariable=self.k_brs_m, width=6)
        baris_m.grid(row=1, column=7, sticky=W)


        ttk.Label(self.frm5, text="Kunci Kolom").grid(row=2, column=0, sticky=W,
**paddings)
        ttk.Label(self.frm5, text=":").grid(row=2, column=1, sticky=W,
**paddings)
        ttk.Label(self.frm5, text="a= ").grid(row=2, column=2, sticky=W,
**paddings)
        ttk.Label(self.frm5, text="b= ").grid(row=2, column=4, sticky=W,
**paddings)
        ttk.Label(self.frm5, text="m= ").grid(row=2, column=6, sticky=W,
**paddings)

        kolom_a = ttk.Entry(self.frm5, textvariable=self.k_klm_a,
width=6).grid(row=2, column=3, sticky=W)
        kolom_b = ttk.Entry(self.frm5, textvariable=self.k_klm_b,
width=6).grid(row=2, column=5, sticky=W)

        kolom_m = ttk.Entry(self.frm5, textvariable=self.k_klm_m, width=6)
        kolom_m.grid(row=2, column=7, sticky=W)

        ttk.Button(self.frm5, text="Gambar Stego", command=self.upload_stego ,
width=15).grid(row=3, column=3, columnspan=4, sticky=W)
        ttk.Button(self.frm5, text="Ok", command=self.ekstrak).grid(row=3,
column=6, columnspan=6, sticky=W, **paddings)

        # Frame 6
        canvas_lcg = Canvas(self.frm6, width=330, height=330, bg="white")
        canvas_lcg.grid(row=0, column=0)

        self.window.mainloop()

    @staticmethod
    def upload():
        path_imgsec = filedialog.askopenfilename(title='Choose image')
        return path_imgsec
```

```
# FPB
def FPB(self,x,y):
    while (y):
        x, y = y, x%y

    return x

# Faktor-faktor m
def faktorisasiprima(self,x):
    faktor_list = []
    loop = 2

    while loop <= x :
        if x % loop == 0 :
            x/=loop
            faktor_list.append(loop)

        else : loop+=1

    return faktor_list

def upload_img(self):
    path_imgsec = self.upload()
    secret_image = cv.imread(path_imgsec,cv.IMREAD_GRAYSCALE)
    self.h, self.w = secret_image.shape[:2]

    baris_m = ttk.Entry(self.frm1, width=6)
    baris_m.insert(0,self.h)
    baris_m.config(state=DISABLED)
    baris_m.grid(row=1, column=7, sticky=W)

    kolom_m = ttk.Entry(self.frm1, width=6)
    kolom_m.insert(0, self.w)
    kolom_m.config(state=DISABLED)
    kolom_m.grid(row=2, column=7, sticky=W)

    self.image = secret_image

def acak(self):

    acak_start = time.time()

    # Variabel Baris
    baris_a = int(self.brs_a.get())
    baris_b = int(self.brs_b.get())
    baris_m = self.h
```

```
        # Variabel Kolom
        kolom_a = int(self.klm_a.get())
        kolom_b = int(self.klm_b.get())
        kolom_m = self.w

        # Cek a-1 habis dibagi semua fator m
        faktor_bm = self.faktorisasiprima(baris_m) # Baris
        uji_baris = True
        for i in range(len(faktor_bm)):
            if (baris_a-1) % faktor_bm[i] != 0:
                uji_baris = False
                break
            uji_baris = True

        faktor_km = self.faktorisasiprima(kolom_m) # Kolom
        uji_kolom = True
        for i in range(len(faktor_km)):
            if (kolom_a-1) % faktor_km[i] != 0:
                uji_kolom = False
                break
            uji_kolom = True

        # Tampilkan Warning
        if baris_a<=0 or kolom_a<=0 or baris_b<=0 or kolom_b<=0 :
            messagebox.showwarning("Informasi", "Syarat : a,b > 0")

        elif baris_m <= max(baris_a,baris_b) or kolom_m <= max(kolom_a,kolom_b)
:
            messagebox.showwarning("Informasi", "Syarat : m > maks(a,b)")

        elif self.FPB(baris_b,baris_m)!=1 or self.FPB(kolom_b,kolom_m)!=1:
            messagebox.showwarning("Informasi", "Syarat : (b,m) = 1")

        elif uji_baris == False or uji_kolom==False :
            messagebox.showwarning("Informasi", "Syarat : a-1 habis dibagi semua
faktor m")

        elif baris_m%4==0 and (baris_a-1)%4!=0 or kolom_m%4==0 and (kolom_a-
1)%4!=0 :
            messagebox.showwarning("Informasi", "Syarat : jika m kelipatan 4,
maka (a-1) harus kelipatan 4")

        # Enkripsi
        else :
            e_img = enk.Row(self.image,baris_m,kolom_m,baris_a,baris_b)
#height,width
```

```
            e_img = enk.Column(e_img.e_rimg,baris_m,kolom_m,kolom_a,kolom_b)
            enk_image = e_img.e_cimg

        # Pixel -> Biner
            for i in range(baris_m):
                for j in range(kolom_m):
                    self.bin_secret += bitstring.pack('uint:8',enk_image[i,j])

        # Output
            cv.imwrite(SECRET_IMAGE_FILE, enk_image)
            global my_image
            imge = Image.open(SECRET_IMAGE_FILE)
            resize = imge.resize((330,330), Image.ANTIALIAS)
            my_image = ImageTk.PhotoImage(resize)
            label_img = ttk.Label(self.frm2, image=my_image)
            label_img.grid(row=0, column=0)

        acak_end = time.time()
        print("Waktu LCG = " + str(acak_end-acak_start) + " Seconds")

    def upload_cover(self):
        paddings = { 'padx' :5 , 'pady' :5}
        path_cover = self.upload()
        cover_img = cv.imread(path_cover, flags=cv.IMREAD_COLOR)
        self.cover = cover_img
        ttk.Label(self.frm3, text="Cover Uptodate    ").grid(row=1, column=3,
columnspan=3, sticky=W)
        print("Cover")

    def embed(self):
        embed_start = time.time()
        for i in range(9):

            if i==0:
                print("Mulai Penyisipan")

            # Blok Img 8x8
            cover_image_f32 = np.float32(self.cover)
            cover_image_YCC = blok.Blok_Image(cv.cvtColor(cover_image_f32,
cv.COLOR_BGR2YCrCb))

            # Gambar Kosong
            stego_image = np.empty_like(cover_image_f32)

            # DCT
```

```
            dct_blocks = [cv.dct(block) for block in
cover_image_YCC.channels[0]] # channel 0 = komponen Y

            # Kuantisasi
            dct_kuantisasi = [np.round(np.divide(item, blok.TABEL_KUANTISASI))
for item in dct_blocks]
            dct_kuantisasi = [zz.zigzag(block) for block in dct_kuantisasi]

            # Penyisipan
            blok_embed = embed.penyisipan(self.bin_secret, dct_kuantisasi)
            dct_embed = [zz.inverse_zigzag(block, vmax=8,hmax=8) for block in
blok_embed]

            # Dekuantisasi
            dct_dekuantisasi = [np.multiply(data, blok.TABEL_KUANTISASI) for
data in dct_embed]

            # IDCT
            idct_blocks = [cv.idct(block) for block in dct_dekuantisasi]

            # Full Image
            stego_image[:,:,0] =
np.asarray(blok.stitch_blocks(cover_image_YCC.lebar, idct_blocks))
            stego_image[:,:,1] =
np.asarray(blok.stitch_blocks(cover_image_YCC.lebar,
cover_image_YCC.channels[1]))
            stego_image[:,:,2] =
np.asarray(blok.stitch_blocks(cover_image_YCC.lebar,
cover_image_YCC.channels[2]))

            # Convert RGB (BGR) Colorspace
            stego_image_BGR = cv.cvtColor(stego_image, cv.COLOR_YCR_CB2BGR)

            # Pixel Values [0 - 255]
            final_stego_image = np.uint8(np.clip(stego_image_BGR, 0, 255))
            self.cover = final_stego_image

            print("Pass "+str(i+1))

        # Write stego image
        cv.imwrite(STEGO_IMAGE_FILE, final_stego_image)

        global gambar
        hstego = Image.open(STEGO_IMAGE_FILE)
        resize_gambar = hstego.resize((330,330), Image.ANTIALIAS)
        gambar = ImageTk.PhotoImage(resize_gambar)
        hlabel = ttk.Label(self.frm4, image=gambar)
```

```
        hlabel.grid(row=0, column=0)

        ttk.Label(self.frm3, text="Penyisipan selesai").grid(row=1, column=3,
columnspan=3, sticky=W)
        self.bin_secret = ""

        embed_end = time.time()

        print("Waktu Embed = " + str(embed_end-embed_start) + " Seconds")

    def upload_stego(self):
        path_stego = self.upload()
        self.stego = cv.imread(path_stego, flags=cv.IMREAD_COLOR)
        print("Stego")

    def ekstrak(self):
        ekstrak_start = time.time()
        print("Mulai Ekstrak")
        # Baris
        kbaris_a = int(self.k_brs_a.get())
        kbaris_b = int(self.k_brs_b.get())
        kbaris_m = int(self.k_brs_m.get())

        # Kolom
        kkolom_a = int(self.k_klm_a.get())
        kkolom_b = int(self.k_klm_b.get())
        kkolom_m = int(self.k_klm_m.get())

        # Variable
        ekstrak_data = ''
        secret_image = np.zeros((kbaris_m, kkolom_m),dtype='uint8')

        # Blok Img 8x8
        stego_image_f32 = np.float32(self.stego)
        stego_image_YCC = blok.Blok_Image(cv.cvtColor(stego_image_f32,
cv.COLOR_BGR2YCrCb))

        # DCT
        dct_blocks = [cv.dct(block) for block in stego_image_YCC.channels[0]]

        # Kuantisasi
        dct_kuantisasi = [np.round(np.divide(item, blok.TABEL_KUANTISASI)) for
item in dct_blocks]
        dct_kuantisasi = [zz.zigzag(block) for block in dct_kuantisasi]

        # Ekstraksi bit rahasia
        for block in dct_kuantisasi:
```

```
            for indeks in range(1, len(block)):
                curr_coeff = np.int32(block[indeks])
                if (curr_coeff > 1):
                    ekstrak_data += bitstring.pack('uint:1',
np.uint8(block[indeks]) & 0x01)

        data_len = int(ekstrak_data.read('uint:32') / 8)
        print(ekstrak_data.bin[:32])
        print(f"Panjang bit : {data_len}")

        # Rebuild
        for k in range(kbaris_m):
            for b in range(kkolom_m):
                secret_image[k,b] = ekstrak_data.read('uint:8')

        secret_image_c = dek.Column(secret_image, kbaris_m, kkolom_m, kkolom_a,
kkolom_b)
        secret_image_final = dek.Row(secret_image_c.d_cimg, kbaris_m, kkolom_m,
kbaris_a, kbaris_b)

        print("Selesai")

        cv.imwrite(OUTPUT_IMAGE_FILE,secret_image_final.d_rimg)

        global output
        hstego = Image.open(OUTPUT_IMAGE_FILE)
        resize_gambar = hstego.resize((330,330), Image.ANTIALIAS)
        output = ImageTk.PhotoImage(resize_gambar)

        ttk.Label(self.frm6, image=output).grid(row=0, column=0)

        ekstrak_data = ''

        ekstrak_end = time.time()

        print("Waktu Ekstrak = " + str(ekstrak_end-ekstrak_start) + " Seconds")

if __name__ == '__main__':
    gui()
```

```
## External Lib
import cv2 as cv
import numpy as np

# Enkripsi Baris
class Row:
    def __init__(self,image, height, width, a, b):
        self.image = image
        self.height = height
        self.width = width
        self.a = a
        self.b = b

        self.e_rimg = np.zeros((self.height,self.width),dtype="uint8")

        m = self.height
        e_lcg = 0

        # LCG
        for i in range(self.height):
            e_lcg = ((self.a*e_lcg)+self.b) % m # pers 2.5
            self.e_rimg[e_lcg,:] = self.image[i,:]

# Enkripsi Kolom
class Column:
    def __init__(self,image, height, width, a, b):
        self.image = image
        self.height = height
        self.width = width
        self.a = a
        self.b = b

        self.e_cimg = np.zeros((self.height,self.width),dtype="uint8")

        m = self.width
        e_lcg = 0

        # LCG
        for i in range(self.width):
            e_lcg = ((self.a*e_lcg)+self.b) % m # pers 2.5
            self.e_cimg[:,e_lcg] = self.image[:,i]
```

```
## External Lib
from re import A
import cv2 as cv
import numpy as np

# Dekripsi Kolom
class Column:
    def __init__(self,image, height, width, a, b):
        self.image = image
        self.height = height
        self.width = width
        self.a = a
        self.b = b
        self.m = width

        self.d_cimg = np.zeros((self.height,self.width),dtype="uint8")

        d_lcg = 0

        for i in range(self.width):
            d_lcg = ((self.a*d_lcg)+self.b) % self.m # Pers 2.5
            self.d_cimg[:,i] = self.image[:,d_lcg]

# Dekripsi Baris
class Row:
    def __init__(self,image, height, width,a ,b):
        self.image = image
        self.height = height
        self.width = width
        self.a = a
        self.b = b

        self.d_rimg = np.zeros((self.height,self.width),dtype="uint8")

        self.m = height

        d_lcg = 0

        for i in range(self.height):
            d_lcg = ((self.a*d_lcg)+self.b) % self.m # Pers 2.5
            self.d_rimg[i,:] = self.image[d_lcg,:]
```

```
import numpy as np

# Kuantisasi standar JPEG
TABEL_KUANTISASI = np.asarray([
                                [16, 11, 10, 16,  24, 40,   51,  61],
                                [12, 12, 14, 19,  26, 58,   60,  55],
                                [14, 13, 16, 24,  40, 57,   69,  56],
                                [14, 17, 22, 29,  51, 87,   80,  62],
                                [18, 22, 37, 56,  68, 109, 103,  77],
                                [24, 36, 55, 64,  81, 104, 113,  92],
                                [49, 64, 78, 87, 103, 121, 120, 101],
                                [72, 92, 95, 98, 112, 100, 103,  99]
                            ], dtype = np.float32)

class Blok_Image:
    def __init__(self, cover_img):
        self.tinggi, self.lebar = cover_img.shape[:2]
        self.channels = [
            split_image(cover_img[:,:,0]),
            split_image(cover_img[:,:,1]),
            split_image(cover_img[:,:,2]),
        ]

# Memecah image ke dalam blok-blok 8x8
def split_image(image):
    blocks = []
    for vert_slice in np.vsplit(image, int(image.shape[0] / 8)):
        for horiz_slice in np.hsplit(vert_slice, int(image.shape[1] / 8)):
            blocks.append(horiz_slice)
    return blocks

# Menggabungkan blok-blok
def stitch_blocks(Nc, block_segments):

    image_rows = []
    temp = []
    for i in range(len(block_segments)):
        if i > 0 and not(i % int(Nc / 8)):
            image_rows.append(temp)
            temp = [block_segments[i]]
        else:
            temp.append(block_segments[i])
    image_rows.append(temp)

    return np.block(image_rows)
```

```
import numpy as np

def zigzag(input):
    #----------------------------------
    h = 0
    v = 0
    vmin = 0
    hmin = 0
    vmax = input.shape[0]
    hmax = input.shape[1]
    i = 0
    output = np.zeros(( vmax * hmax))
    #----------------------------------

    while ((v < vmax) and (h < hmax)):

        if ((h + v) % 2) == 0:                 # going up
            if (v == vmin):
                #print(1)
                output[i] = input[v, h]        # if we got to the first line
                if (h == hmax):
                    v = v + 1
                else:
                    h = h + 1
                i = i + 1
            elif ((h == hmax -1 ) and (v < vmax)):   # if we got to the last
column
                #print(2)
                output[i] = input[v, h]
                v = v + 1
                i = i + 1
            elif ((v > vmin) and (h < hmax -1 )):    # all other cases
                #print(3)
                output[i] = input[v, h]
                v = v - 1
                h = h + 1
                i = i + 1

        else:                                    # going down
            if ((v == vmax -1) and (h <= hmax -1)):       # if we got to the
last line
                #print(4)
                output[i] = input[v, h]
                h = h + 1
                i = i + 1
            elif (h == hmin):                  # if we got to the first column
                #print(5)
```

```
                output[i] = input[v, h]
                if (v == vmax -1):
                    h = h + 1
                else:
                    v = v + 1
                i = i + 1
            elif ((v < vmax -1) and (h > hmin)):     # all other cases
                #print(6)
                output[i] = input[v, h]
                v = v + 1
                h = h - 1
                i = i + 1

        if ((v == vmax-1) and (h == hmax-1)):          # bottom right element
            #print(7)
            output[i] = input[v, h]
            break

    #print ('v:',v,', h:',h,', i:',i)
    return output

def inverse_zigzag(input, vmax, hmax):

    #--------------------------------
    h = 0
    v = 0
    vmin = 0
    hmin = 0
    output = np.zeros((vmax, hmax))
    i = 0
    #--------------------------------

    while ((v < vmax) and (h < hmax)):
        #print ('v:',v,', h:',h,', i:',i)
        if ((h + v) % 2) == 0:                 # going up
            if (v == vmin):
                #print(1)
                output[v, h] = input[i]        # if we got to the first line
                if (h == hmax):
                    v = v + 1
                else:
                    h = h + 1
                i = i + 1

            elif ((h == hmax -1 ) and (v < vmax)):   # if we got to the last
column
                #print(2)
```

```
                output[v, h] = input[i]
                v = v + 1
                i = i + 1

            elif ((v > vmin) and (h < hmax -1 )):    # all other cases
                #print(3)
                output[v, h] = input[i]
                v = v - 1
                h = h + 1
                i = i + 1

        else:                                   # going down
            if ((v == vmax -1) and (h <= hmax -1)):       # if we got to the
last line
                #print(4)
                output[v, h] = input[i]
                h = h + 1
                i = i + 1
            elif (h == hmin):                  # if we got to the first column
                #print(5)
                output[v, h] = input[i]
                if (v == vmax -1):
                    h = h + 1
                else:
                    v = v + 1
                i = i + 1
            elif((v < vmax -1) and (h > hmin)):     # all other cases
                output[v, h] = input[i]
                v = v + 1
                h = h - 1
                i = i + 1

        if ((v == vmax-1) and (h == hmax-1)):          # bottom right element
            #print(7)
            output[v, h] = input[i]
            break

    return output
```

```
import bitstring
import numpy as np

def penyisipan(bin_secret, dct_blok):
    secret_data_len = bitstring.pack('uint:32', len(bin_secret))
    new_blocks = []

    for block in dct_blok:
        for index in range(1, len(block)):
            curr_coef = np.int32(block[index])
            if (curr_coef > 1):
                curr_coef = np.uint8(block[index])
                if (bin_secret.pos == len(bin_secret)) : break
                pack_coef = bitstring.pack('uint:8', curr_coef)
                if (secret_data_len.pos <= len(secret_data_len) - 1) :
pack_coef[-1] = secret_data_len.read(1)
                else: pack_coef[-1] = bin_secret.read(1)
                block[index] = np.float32(pack_coef.read('uint:8'))
        new_blocks.append(block)

    bin_secret.pos = 0
    secret_data_len.pos = 0

    return new_blocks
```

```
from tkinter import *
from tkinter import ttk, messagebox
from tkinter import filedialog
import cv2 as cv
import numpy as np
from math import log10,sqrt

class gui:
    def __init__(self):
        self.window = Tk()
        self.window.title("Steganografi")
        self.window.geometry('255x260')
        self.window.resizable(0,0)

        self.frm1 = ttk.LabelFrame(self.window, text="MSE & PNSR", padding=10)
        self.frm1.grid(row=0,column=0, padx=20, pady=15, sticky=NW)
        self.frm2 = ttk.LabelFrame(self.window, text="NCC", padding=10)
        self.frm2.grid(row=1,column=0, padx=20, sticky=NW)

        # Variable
        self.cover = []
        self.stego = []
        self.awal = []
        self.akhir = []
        # Frame 1

        ttk.Button(self.frm1, text="Cover", command=self.upload_cover ,
width=10).grid(row=0, column=0, sticky=W)
        ttk.Button(self.frm1, text="Stego", command=self.upload_stego ,
width=10).grid(row=0, column=1, sticky=W, padx=5)
        ttk.Button(self.frm1, text="OK", command=self.mp , width=5).grid(row=0,
column=2, sticky=W)

        self.n_frame = ttk.Frame(self.frm1)
        self.n_frame.grid(row=1,column=0,columnspan=2, sticky=W)

        ttk.Label(self.n_frame, text="MSE ", padding=5).grid(row=1, column=0,
sticky=W)
        ttk.Label(self.n_frame, text="= ", padding=5).grid(row=1, column=1,
sticky=W)
        ttk.Label(self.n_frame, text="PSNR ", padding=5).grid(row=2, column=0,
sticky=W)
        ttk.Label(self.n_frame, text="= ", padding=5).grid(row=2, column=1,
sticky=W)

        # Frame 2
```

```
        ttk.Button(self.frm2, text="Awal", command=self.upload_awal ,
width=10).grid(row=0, column=0, sticky=W)
        ttk.Button(self.frm2, text="Akhir", command=self.upload_akhir ,
width=10).grid(row=0, column=1, sticky=W, padx=5)
        ttk.Button(self.frm2, text="OK", command=self.ncc , width=5).grid(row=0,
column=2, sticky=W)

        self.n_frame1 = ttk.Frame(self.frm2)
        self.n_frame1.grid(row=1,column=0,columnspan=2, sticky=W)

        ttk.Label(self.n_frame1, text="NCC ", padding=5).grid(row=1, column=0,
sticky=W)
        ttk.Label(self.n_frame1, text="= ", padding=5).grid(row=1, column=1,
sticky=W)

        self.window.mainloop()

    @staticmethod
    def upload():
        path_imgsec = filedialog.askopenfilename(title='Choose image')
        return path_imgsec

    def upload_cover(self):
        path_cover = self.upload()
        cover_img = cv.imread(path_cover)
        self.cover = cover_img

    def upload_stego(self):
        path_stego = self.upload()
        stego_img = cv.imread(path_stego)
        self.stego = stego_img

    def upload_awal(self):
        path_awal = self.upload()
        awal_img = cv.imread(path_awal)
        self.awal = awal_img

    def upload_akhir(self):
        path_akhir = self.upload()
        akhir_img = cv.imread(path_akhir)
        self.akhir = akhir_img

    def mp(self):
        mse = np.mean((self.cover - self.stego) ** 2)

        if(mse == 0):  # MSE is zero means no noise is present in the signal .
```

```
                    # Therefore PSNR have no importance.
            psnr = 100

        elif(mse!=0):
            max_pixel = 255.0
            psnr = 20 * log10(max_pixel / sqrt(mse))

        ttk.Label(self.n_frame, text=f"{mse:.4f}", padding=5).grid(row=1,
column=2, sticky=W)
        ttk.Label(self.n_frame, text=f"{psnr:.4f}", padding=5).grid(row=2,
column=2, sticky=W)

        self.cover = []
        self.stego = []

    def ncc(self):
        result = cv.matchTemplate(self.awal, self.akhir ,cv.TM_CCOEFF_NORMED)
        ttk.Label(self.n_frame1, text=f"{result[0,0]:.6f}",
padding=5).grid(row=1, column=2, sticky=W)
        print((result[0,0]))

        self.awal = []
        self.akhir = []

if __name__ == '__main__':
    gui()
```