LAPORAN UAS

CAESAR CIPHER + STEGANOGRAFI LSB

Dosen Pengampu:

Saiful Nur Budiman, S.Kom., M.Kom

Disusun Oleh :

NABILLAH RACHELIA M.	(22104410050)
M. FIRZA KHOIRUDIN	(22104410067)
GILANG PRADANA	(22104410089)
LAILA IVONE BELLA P.	(22104410090)
ANGGI SATRIA WIRANATA	(22104410091)

PROGRAM STUDI TEKNIK DAN INFORMATIKA

FAKULTAS TEKNOLOGI INFORMASI

UNIVERSITAS ISLAM BALITAR

2026

DAFTAR ISI

DAFTAR GAMBAR

BAB I
PENDAHULUAN

1.1 Latar Belakang

Perkembangan teknologi informasi dan komunikasi yang semakin pesat menyebabkan pertukaran data digital menjadi sangat intensif, baik dalam bentuk teks, gambar, maupun media digital lainnya. Kondisi ini menimbulkan tantangan serius terkait keamanan informasi, khususnya dalam menjaga kerahasiaan pesan agar tidak dapat diakses oleh pihak yang tidak berwenang. Tanpa adanya mekanisme pengamanan yang baik, data digital sangat rentan terhadap penyadapan, pencurian, maupun manipulasi oleh pihak yang tidak bertanggung jawab.

Kriptografi merupakan salah satu solusi utama dalam menjaga keamanan informasi dengan cara menyandikan pesan asli (plaintext) menjadi pesan tersandi (ciphertext) sehingga hanya pihak tertentu yang memiliki kunci yang dapat mengembalikannya ke bentuk semula. Salah satu algoritma kriptografi klasik yang paling sederhana adalah Caesar Cipher, yaitu algoritma substitusi yang bekerja dengan menggeser setiap karakter dalam alfabet berdasarkan nilai kunci tertentu. Meskipun Caesar Cipher memiliki tingkat keamanan yang rendah, algoritma ini masih relevan digunakan sebagai media pembelajaran untuk memahami konsep dasar enkripsi dan dekripsi.

Namun, kriptografi saja belum sepenuhnya cukup untuk menjamin keamanan komunikasi karena keberadaan ciphertext masih dapat menimbulkan kecurigaan. Oleh karena itu, diperlukan teknik tambahan untuk menyembunyikan keberadaan pesan itu sendiri. Steganografi merupakan teknik penyembunyian pesan rahasia ke dalam media penampung, seperti citra digital, sehingga pesan tersebut tidak terlihat secara kasat mata. Salah satu metode steganografi yang paling umum digunakan adalah Least Significant Bit (LSB), yaitu dengan memodifikasi bit paling tidak signifikan pada piksel citra tanpa menurunkan kualitas visual secara signifikan.

Berdasarkan permasalahan tersebut, kombinasi antara kriptografi Caesar Cipher dan steganografi metode LSB dapat menjadi solusi keamanan berlapis, di mana pesan terlebih dahulu dienkripsi kemudian disembunyikan ke dalam citra digital. Pendekatan ini diharapkan dapat meningkatkan keamanan informasi, karena meskipun pesan berhasil diekstraksi, isinya tetap tidak dapat dibaca tanpa proses dekripsi. Oleh karena itu, pada proyek ini

dikembangkan sebuah aplikasi berbasis Python yang mampu melakukan proses enkripsi dan embedding LSB, serta proses kebalikannya yaitu ekstraksi LSB dan dekripsi, sesuai dengan ketentuan tugas yang telah ditetapkan.

1.2 Rumusan Masalah

1. Bagaimana cara mengimplementasikan algoritma Caesar Cipher untuk proses enkripsi dan dekripsi pesan teks menggunakan bahasa pemrograman Python?
2. Bagaimana cara menerapkan metode Steganografi Least Significant Bit (LSB) untuk menyisipkan pesan terenkripsi ke dalam citra digital berformat RGB?
3. Bagaimana proses ekstraksi pesan dari citra dan dekripsi kembali agar pesan asli dapat diperoleh dengan benar?
4. Bagaimana kelebihan dan kelemahan kombinasi Caesar Cipher dan Steganografi LSB dalam menjaga keamanan informasi?

1.3 Tujuan Proyek

1. Mengimplementasikan algoritma Caesar Cipher untuk melakukan enkripsi dan dekripsi pesan teks.
2. Mengembangkan aplikasi yang mampu melakukan penyisipan pesan terenkripsi ke dalam citra digital menggunakan metode LSB.
3. Mengimplementasikan proses ekstraksi pesan dari citra dan dekripsi untuk memperoleh kembali pesan asli.
4. Menganalisis kinerja, kelebihan, dan kekurangan dari kombinasi algoritma kriptografi Caesar Cipher dan steganografi LSB sebagai sistem keamanan informasi sederhana.

BAB II
LANDASAN TEORI

2.1 Kriptografi

Secara sederhana, kriptografi dapat diartikan sebagai metode untuk mengamankan pesan dengan cara mengubah bentuk aslinya (*plaintext*) menjadi bentuk acak (*ciphertext*) sehingga hanya pihak tertentu yang memiliki kunci yang dapat membaca isi pesannya. Dalam menjaga keamanan informasi, kriptografi memiliki empat fungsi utama, yaitu menjaga kerahasiaan agar isi pesan hanya dapat diketahui oleh pihak yang berhak, menjamin keutuhan data agar tidak diubah, dihapus, atau disisipkan oleh pihak lain tanpa izin, memastikan keaslian identitas pengirim dan penerima sehingga sumber data dapat dipercaya, serta mencegah terjadinya penyangkalan oleh pengirim atau penerima terhadap pesan yang telah dikirim atau diterima. Selain fungsi-fungsi tersebut, kriptografi juga berperan penting dalam mengamankan komunikasi digital seperti pesan teks, transaksi keuangan, dan pertukaran data agar tidak dapat diakses oleh pihak yang tidak berhak.

Kelebihan kriptografi terletak pada kemampuannya menjaga kerahasiaan data sehingga tidak dapat dibaca tanpa kunci yang benar, melindungi integritas pesan agar tidak dimodifikasi oleh pihak yang tidak berwenang, serta mudah diadaptasi dan dikembangkan dalam berbagai bentuk algoritma, baik klasik maupun modern. Namun, kriptografi juga memiliki beberapa kekurangan, seperti membutuhkan waktu dan sumber daya komputasi yang besar terutama pada algoritma kompleks atau data berukuran besar, tingkat keamanan yang sangat bergantung pada panjang dan kekuatan kunci yang digunakan, serta fakta bahwa tidak semua ancaman keamanan dapat diselesaikan hanya dengan kriptografi karena masih ada risiko lain seperti kesalahan manusia atau pencurian kunci secara sosial.

2.2 Caesar Cipher

Caesar Cipher adalah algoritma kriptografi klasik yang menggunakan teknik substitusi sederhana, di mana setiap huruf pada pesan asli (*plaintext*) digeser sejumlah posisi tertentu dalam alfabet untuk menghasilkan *ciphertext*. Metode ini pertama kali digunakan oleh Julius Caesar untuk mengamankan komunikasi militernya. Secara matematis, prosesnya dapat dijelaskan dengan rumus enkripsi $C = (P + K) \bmod 26$ dan dekripsi $P = (C - K) \bmod 26$, di mana P merupakan huruf plaintext, C adalah huruf ciphertext, dan K adalah jumlah pergeseran huruf. Caesar Cipher termasuk dalam jenis kriptografi simetris, karena proses enkripsi dan dekripsinya menggunakan kunci yang sama.

Fungsi utama dari Caesar Cipher adalah untuk mengamankan pesan teks sederhana agar tidak mudah dibaca oleh pihak yang tidak berhak, menyamarkan data agar kerahasiaan dan integritas pesan tetap terjaga, serta sebagai media pembelajaran dasar dalam memahami konsep enkripsi dan dekripsi karena metodenya yang mudah dipahami. Kelebihan dari Caesar Cipher terletak pada kesederhanaannya yang membuat algoritma ini mudah diimplementasikan, proses enkripsinya yang cepat sehingga cocok untuk pesan teks pendek, serta kemampuannya untuk dikombinasikan dengan metode lain guna meningkatkan keamanan. Namun, Caesar Cipher juga memiliki beberapa kelemahan, antara lain tingkat keamanannya yang sangat rendah karena hanya memiliki 26 kemungkinan kunci sehingga mudah dipecahkan melalui serangan *brute force*, kerentanannya terhadap analisis frekuensi karena pola kemunculan huruf ciphertext masih menyerupai *plaintext*, serta ketidakmampuannya dalam mengenkripsi karakter kompleks seperti simbol, rumus, atau file non-teks tanpa modifikasi tambahan. Akibatnya, metode ini tidak cocok digunakan untuk melindungi data yang membutuhkan tingkat keamanan tinggi seperti transaksi digital atau pesan rahasia berskala besar.

2.3 Kelebihan dan Kekurangan Ceasar Cipher

Caesar Cipher memiliki beberapa kelebihan yang menjadikannya masih relevan untuk digunakan dalam pembelajaran dan penelitian dasar kriptografi. Kelebihan utama dari algoritma ini terletak pada kesederhanaan metode yang digunakan, sehingga mudah dipahami dan diimplementasikan baik secara manual maupun menggunakan program komputer. Proses enkripsi dan dekripsi pada Caesar Cipher juga berlangsung dengan cepat karena memiliki kompleksitas yang rendah serta tidak memerlukan sumber daya komputasi yang besar. Selain itu, Caesar Cipher dapat dikombinasikan dengan metode keamanan lain, seperti steganografi atau algoritma kriptografi yang lebih kompleks, sebagai lapisan keamanan tambahan.

Namun demikian, Caesar Cipher memiliki kekurangan yang cukup signifikan, terutama dari sisi keamanan. Ruang kunci yang dimiliki sangat terbatas, yaitu hanya sebanyak 26 kemungkinan pergeseran huruf, sehingga algoritma ini sangat rentan terhadap serangan Exhaustive Key Search atau brute force attack. Selain itu, Caesar Cipher juga mudah dianalisis menggunakan teknik analisis frekuensi karena pola kemunculan huruf pada ciphertext masih menyerupai pola bahasa aslinya. Kelemahan lainnya adalah keterbatasan dalam mengenkripsi karakter non-alfabet seperti angka, simbol, maupun data non-teks tanpa adanya modifikasi tambahan. Oleh karena itu, Caesar Cipher tidak direkomendasikan untuk

mengamankan data yang membutuhkan tingkat keamanan tinggi, melainkan lebih sesuai digunakan sebagai sarana pembelajaran dan penelitian dasar kriptografi.

2.4 Exhaustive Search Key

Exhaustive Key Search, yang juga dikenal sebagai brute force attack, adalah metode kriptanalisis yang digunakan untuk membobol ciphertext dengan mencoba semua kemungkinan kunci yang mungkin digunakan dalam proses enkripsi hingga ditemukan hasil dekripsi yang benar atau *plaintext* asli. Algoritma ini bekerja dengan asumsi bahwa penyerang mengetahui algoritma enkripsi yang digunakan, namun tidak mengetahui kuncinya. Oleh karena itu, penyerang akan mencoba setiap kunci satu per satu sampai menemukan hasil dekripsi yang bermakna. Exhaustive Key Search merupakan salah satu metode kriptanalisis paling umum dan efektif untuk menyerang algoritma kriptografi, terutama yang memiliki panjang kunci pendek. Karena alasan tersebut, metode ini sering digunakan untuk menguji tingkat kekuatan suatu algoritma enkripsi.

Fungsi utama dari Exhaustive Key Search adalah untuk mengukur tingkat keamanan algoritma kriptografi, yaitu seberapa sulit dan lama waktu yang dibutuhkan untuk memecahkan *ciphertext* dengan mencoba semua kemungkinan kunci. Selain itu, metode ini digunakan untuk menguji ketahanan sistem enkripsi terhadap serangan brute force, serta menjadi dasar analisis performa algoritma kriptografi klasik seperti Caesar Cipher dan Playfair Cipher, terutama dalam konteks pembelajaran kriptanalisis.

Kelebihan dari Exhaustive Key Search antara lain kemampuannya untuk selalu menemukan kunci yang benar karena semua kemungkinan akan dicoba, kemudahannya dalam implementasi karena tidak memerlukan analisis matematis yang kompleks, serta kegunaannya dalam pengujian keamanan algoritma baru untuk menilai ketahanan cipher terhadap serangan *brute force*. Metode ini juga menjadi standar dasar dalam kriptanalisis dan sering dijadikan pembanding dalam penelitian kriptografi, khususnya untuk menilai panjang kunci dan waktu pemrosesan. Namun, Exhaustive Key Search juga memiliki beberapa kekurangan, seperti ketidakefisienannya dalam menghadapi algoritma modern yang menggunakan panjang kunci besar dan struktur kompleks sehingga serangan brute force menjadi tidak praktis. Metode ini juga hanya efektif pada algoritma dengan ruang kunci kecil seperti Caesar Cipher, serta tidak mempertimbangkan analisis struktur cipher karena hanya bergantung pada kekuatan perangkat keras dan waktu yang tersedia.

2.5 Steganografi

Steganografi merupakan teknik penyembunyian informasi rahasia ke dalam suatu media penampung sehingga keberadaan informasi tersebut tidak dapat diketahui oleh pihak yang tidak berwenang. Media penampung yang digunakan dapat berupa citra digital, audio, video, maupun teks. Tujuan utama steganografi adalah menyamarkan keberadaan pesan agar pesan tersebut tidak menimbulkan kecurigaan, sehingga komunikasi dapat dilakukan secara aman.

Istilah steganografi berasal dari bahasa Yunani, yaitu *steganos* yang berarti tersembunyi dan *graphos* yang berarti tulisan. Dalam steganografi digital, pesan rahasia disisipkan ke dalam media digital dengan cara tertentu sehingga perubahan yang terjadi tidak terlihat secara visual maupun tidak mudah terdeteksi oleh pengamatan biasa.

Berbeda dengan kriptografi yang berfokus pada pengamanan isi pesan melalui proses penyandian, steganografi lebih menekankan pada penyembunyian keberadaan pesan itu sendiri. Oleh karena itu, steganografi sering digunakan sebagai pelengkap kriptografi untuk meningkatkan tingkat keamanan informasi dalam proses pengiriman data.

2.6 Tujuan dan Kriteria Steganografi

Tujuan utama steganografi adalah untuk melindungi kerahasiaan informasi dengan cara menyembunyikan pesan rahasia ke dalam media penampung yang tampak normal. Dengan demikian, hanya pihak pengirim dan penerima yang mengetahui keberadaan serta isi pesan tersebut. Steganografi banyak digunakan dalam pengamanan data, komunikasi rahasia, serta perlindungan informasi digital.

Keberhasilan suatu teknik steganografi dapat diukur berdasarkan beberapa kriteria utama. Kriteria pertama adalah *imperceptibility*, yaitu perubahan pada media penampung tidak dapat dibedakan secara visual oleh pancaindra manusia. Kriteria kedua adalah *fidelity*, yaitu kualitas media setelah disisipi pesan tetap terjaga dan tidak mengalami perubahan yang signifikan dibandingkan dengan media asli. Kriteria ketiga adalah *recovery*, yaitu pesan rahasia yang disisipkan dapat diekstraksi kembali secara utuh tanpa mengalami kerusakan.

Selain itu, steganografi juga harus mempertimbangkan kapasitas penyisipan pesan dan ketahanan terhadap manipulasi media. Pemilihan metode steganografi yang tepat diharapkan mampu menghasilkan keseimbangan antara keamanan pesan, kualitas media, dan kemampuan penyimpanan data rahasia.

2.7 Metode Least Significant Bit (LSB)

Metode *Least Significant Bit* (LSB) merupakan salah satu teknik steganografi yang paling sederhana dan banyak digunakan dalam penyembunyian pesan pada citra digital. Metode ini bekerja dengan cara mengganti bit paling tidak signifikan (bit terakhir) dari setiap piksel citra dengan bit pesan rahasia. Karena perubahan dilakukan pada bit dengan nilai terendah, perbedaan antara citra asli dan citra hasil steganografi sangat sulit dibedakan secara visual.

Pada citra digital berwarna (RGB), setiap piksel terdiri dari tiga komponen warna, yaitu merah (*Red*), hijau (*Green*), dan biru (*Blue*), yang masing-masing direpresentasikan dalam 8 bit. Dengan metode LSB, bit terakhir dari setiap komponen warna tersebut dapat dimanfaatkan untuk menyisipkan pesan. Selama jumlah bit pesan tidak melebihi kapasitas bit citra penampung, penyisipan dapat dilakukan tanpa menurunkan kualitas citra secara signifikan.

Metode LSB memiliki beberapa keunggulan, antara lain proses implementasi yang sederhana, waktu ekstraksi yang cepat, serta kapasitas penyisipan pesan yang relatif besar. Namun demikian, metode ini memiliki kelemahan, yaitu tingkat ketahanan yang rendah terhadap manipulasi citra seperti kompresi, cropping, atau perubahan format file. Oleh karena itu, dalam beberapa penelitian, metode LSB sering dikombinasikan dengan algoritma kriptografi untuk meningkatkan keamanan pesan yang disisipkan.

2.8 Kelebihan dan Kekurangan Metode *Least Significant Bit* (LSB)

Metode *Least Significant Bit* (LSB) memiliki sejumlah kelebihan yang menjadikannya populer dalam implementasi steganografi citra digital. Kelebihan utama metode ini adalah kemudahan implementasi, proses penyisipan dan ekstraksi yang cepat, serta perubahan visual citra yang sangat kecil sehingga sulit dideteksi oleh mata manusia.

Namun demikian, metode LSB juga memiliki beberapa kelemahan. Metode ini relatif tidak tahan terhadap manipulasi citra seperti kompresi, pemotongan (*cropping*), perubahan ukuran (*resizing*), maupun konversi format file. Selain itu, metode LSB juga rentan terhadap serangan steganalisis yang secara khusus menganalisis perubahan pada bit-bit piksel. Oleh karena itu, dalam pengembangan lebih lanjut, metode LSB sering dikombinasikan dengan teknik kriptografi atau metode steganografi lain untuk meningkatkan keamanan dan ketahanan pesan rahasia yang disisipkan.

BAB III
PEMBAHASAN

3.1 Perancangan Sistem

Sistem keamanan informasi yang dikembangkan terdiri dari dua proses utama yaitu enkripsi-embedding dan ektraksi-dekripsi. Arsitektur system dapat digambarkan sebagi berikut :

Plainteks ⟶ Enkripsi Caesar Cipher ⟶ Ciphertext ⟶ Konversi Ke Biner ⟶ Embed LSB ke Gambar ⟶ Stego Image ⟶ Ekstrasi LSB ⟶ Cipherteks ⟶ Dekripsi Caesar Cipher ⟶ Plainteks Asli

3.1.1 Inisialisasi Program dan Input Data

Saat program dijalankan, sistem menampilkan judul aplikasi dan meminta pengguna memasukkan pesan rahasia serta path gambar yang akan dijadikan media penyimpanan. Jika pengguna tidak mengisi path gambar, program otomatis menggunakan gambar default. Pada tahap ini dilakukan pengecekan keberadaan file gambar untuk memastikan proses dapat berjalan tanpa error.

3.1.2 Proses Enkripsi Caesar Cipher

Setelah pesan dimasukkan, program melakukan enkripsi menggunakan metode Caesar Cipher dengan pergeseran 7 karakter. Setiap huruf dalam plaintext diubah ke posisi alfabet, kemudian ditambah dengan nilai shift. Jika hasil pergeseran melewati huruf Z atau z, maka posisi huruf akan diputar kembali ke awal alfabet. Proses ini menghasilkan ciphertext yang sudah tidak dapat dibaca secara langsung.

3.1.3 Konversi Cipherteks ke Bentuk Biner

Ciphertext yang dihasilkan kemudian dikonversi ke dalam bentuk biner. Setiap karakter diubah ke nilai ASCII dan direpresentasikan dalam 8 bit biner. Proses ini menghasilkan deretan bit panjang yang akan disisipkan ke dalam gambar.

3.1.4 Pembuatan Header Panjang Pesan

Sebelum pesan disisipkan, program menambahkan header sepanjang 32 bit di bagian awal data biner. Header ini berisi informasi panjang pesan sehingga saat proses ekstraksi, program mengetahui berapa banyak bit yang harus dibaca dari gambar stego.

3.1.5 Proses Embedding Dengan Metode LSB

Program membaca seluruh pixel gambar dan mengubahnya menjadi array satu dimensi. Setiap bit pesan kemudian disisipkan ke dalam bit paling tidak signifikan atau Least Significant Bit dari setiap pixel. Perubahan ini sangat kecil sehingga tidak memengaruhi kualitas visual gambar. Setelah semua bit pesan tertanam, gambar disimpan sebagai stego image.

3.1.6 Input Stego Image Untuk Ekstrasi

Pada tahap dekripsi, pengguna diminta memasukkan path gambar stego. Jika tidak diisi, program menggunakan file stego default. Program memastikan file tersebut tersedia sebelum melanjutkan proses.

3.1.7 Pembacaan Header dan Ekstrasi Bit Pesan

Program membaca 32 bit pertama dari LSB gambar untuk mendapatkan panjang pesan. Setelah itu, program mengekstrak bit-bit berikutnya sesuai dengan panjang pesan yang telah diperoleh dari header. Bit-bit ini dikumpulkan kembali menjadi sebuah rangkaian biner.

3.1.8 Konversi Biner Menjadi Cipherteks

Rangkaian biner yang telah diekstrak dibagi menjadi kelompok 8 bit. Setiap kelompok dikonversi ke karakter ASCII sehingga menghasilkan kembali ciphertext yang sebelumnya telah disisipkan ke dalam gambar.

3.1.9 Proses Dekripsi Caesar Cipher

Ciphertext hasil ekstraksi kemudian didekripsi menggunakan metode Caesar Cipher dengan pergeseran 7. Setiap karakter dikurangi dengan nilai shift untuk mengembalikannya ke huruf aslinya sehingga diperoleh plaintext.

3.1.10 Output dan Penyimpanan Hasil

Plaintext hasil dekripsi ditampilkan ke layar dan juga disimpan ke dalam file teks. Dengan demikian, seluruh proses mulai dari enkripsi, embedding, ekstraksi, hingga dekripsi berhasil dilakukan dan pesan rahasia dapat dikembalikan dengan sempurna.

3.2 Implementasi Caesar Cipher (Shift 7)

a. Enkripsi Caesar Cipher

   Caesar Cipher bekerja dengan menggeser setiap karakter alfabet sebanyak 7 posisi menggunakan rumus **C = (P + 7) mod 26**. Implementasi dalam Python :

```python
def caesar_cipher_encrypt(plaintext, shift=7):
    encrypted = ""

    print("\n=== PROSES ENKRIPSI CAESAR CIPHER ===")
    print(f"Plaintext: {plaintext}")
    print(f"Shift: {shift}")
    print("\nProses per karakter:")

    for i, char in enumerate(plaintext):
        if char.isalpha():
            base = ord('A') if char.isupper() else ord('a')
            pos = ord(char) - base
            new_pos = (pos + shift) % 26
            encrypted_char = chr(base + new_pos)
            encrypted += encrypted_char
            print(f"  {i+1}. '{char}' (pos {pos}) + {shift} = '{encrypted_char}' (pos {new_pos})")
        else:
            encrypted += char
            print(f"  {i+1}. '{char}' -> '{char}' (tidak berubah)")

    print(f"\nCiphertext: {encrypted}")
    return encrypted

```

Gambar 3.1 Implementasi Caesar Cipher (Shift 7)

b. Dekripsi Caesar Cipher

Dekripsi menggunakan rumus **P = (C - 7) mod 26** untuk menggeser karakter 7 posisi ke belakang dan mengembalikan plaintext asli.

Dimana :

P = posisi huruf plaintext dalam alfabet (0-25)

C = posisi huruf ciphertext dalam alfabet (0-25)

7 = nilai shift yang digunakan

26 = jumlah huruf dalam alfabet

Contoh Proses Enkripsi :

Plaintext : “HELLO”

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Huruf	Posisi	Perhitungan	Posisi Baru	Ciphertext
H	7	(7+7) MOD 26 = 14	14	O
E	4	(4+7) MOD 26 = 11	11	L

L	11	(11+7) MOD 26 = 18	18	S
L	11	(11+7) MOD 26 = 18	18	S
0	14	(14+7) MOD 26 = 21	21	V

Hasil Ciphertext = **“OLSSV”**

3.3 Implementasi Steganografi LSB

Metode LSB bekerja dengan mengganti bit terakhir (bit paling tidak signifikan) dari setiap byte nilai piksel citra dengan bit pesan yang akan disembunyikan. Pada citra RGB, setiap piksel terdiri dari 3 komponen warna (Red, Green, Blue) yang masing-masing bernilai 0-255 (8 bit)

Proses Embedding

Metode LSB menyisipkan pesn dengan mengganti bit terakhir setiap pixel citra berikut adalah prosesnya :

1. Konversi Teks ke Binary: Setiap karakter diubah menjadi 8 bit ASCII
2. Pembuatan Header: 32 bit pertama menyimpan panjang pesan
3. Embedding Bit: Operasi bitwise (pixel & 0xFE) | bit untuk menyisipkan bit pesan
4. Penyimpanan: Stego image disimpan dalam format PNG

Contoh jika nilai pixel RGB Adalah (154,25,76) dalam binary :

Red : 10011010

Green : 00011001

Blue : 01001100

Bit pesan yang disisipkan Adalah “101”

Maka setelah di embedding akan menghasilkan nikai pixel baru (155,24,77) :

Red : 10011011 LSM dirubah menjadi angka 1

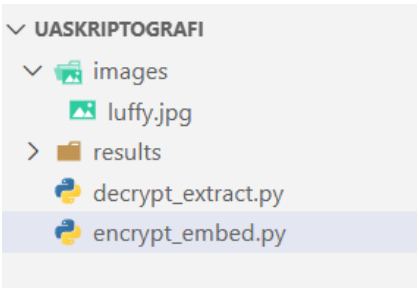
Green : 00011000 LSM dirubah menjadi angka 0

Blue : 01001101 LSM dirubah menjadi angka 1

Perubahanya adalah Red +1 , Grenn -1, Blue +1 (tidak terlihat secar avisual)

3.4 Proses Enkripsi dan Embedding LSB

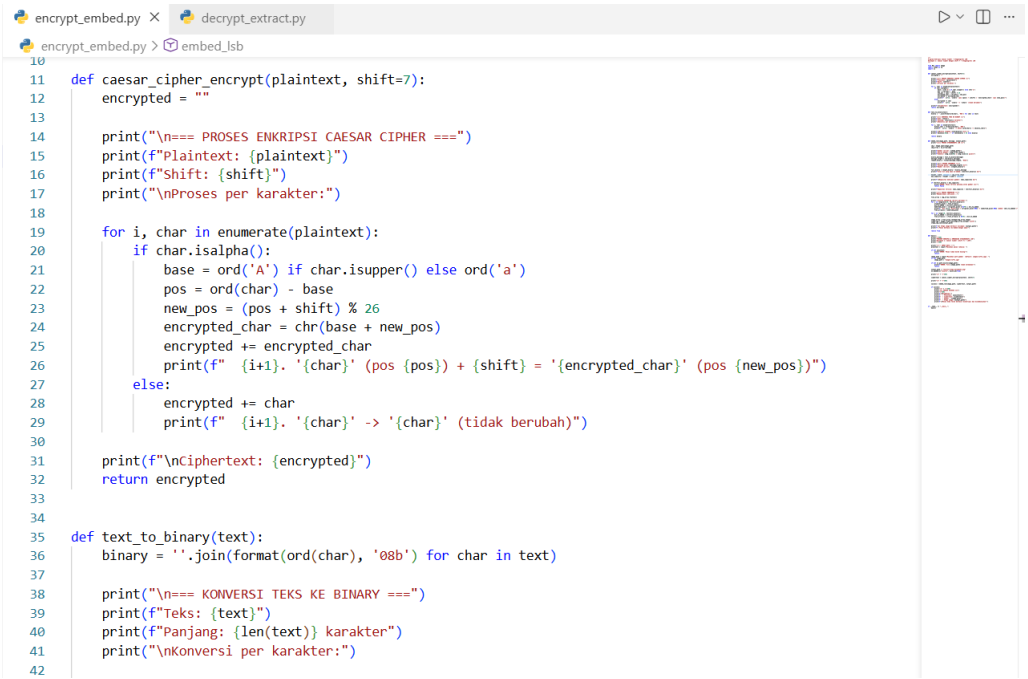
1. Pastikan sudah mempunyai folder dengan nama UASKRIPTOGRAFI yang didalamnya terdapat gambar yang akan digunakan dan folder results untuk menyimpan hasil setelah diberikan kode rahasinya



Gambar 3.2 Proses Enkripsi dan Embedding LSB

Setelah itu buat file di project dengan nama encrypt_embed.py pastikan sudah menginstall python terlebih dulu agar dapat menjalankan programnya.

2. Selanjutnya isikan kode yang berisi tentang bagaimana cara untuk menyembunyikan pesan rahasia yang akan kita gunakan.

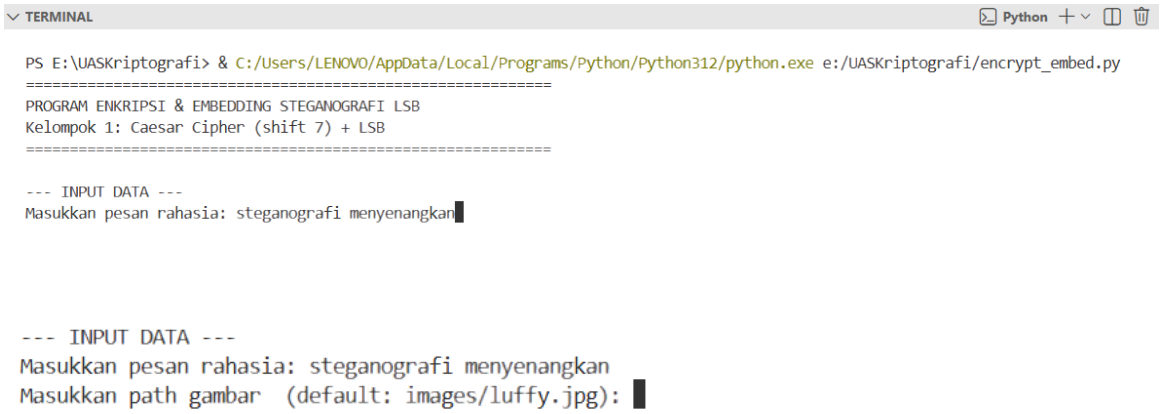


Gambar 3.3 Proses Enkripsi dan Embedding LSB

3. Setelah itu klik kanan untuk menjalankan terminalnya, pilih pada menu *Run Phyton* kemudian *Run Python File in Terminal*.

Gambar 3.4 Proses Enkripsi dan Embedding LSB

4. Selanjutnya layar akan menampilkan halaman terminal yang menyuruh kita untuk memasukkan pesan rahasia yang akan kita gunkakan , disini kelompok kami menciba memasukkan pesan rahasia **" steganografi menyenangkan "** setelah di ENTER maka program akan menampilkan masukkan gampar path yang sudh kita punya seperti contoh disini sudah ada gambar luffy.jpg



Gambar 3.5 Proses Enkripsi dan Embedding LSB

5. Setelah di ENTER program akan menampilkan sebuah kode benkripsi Caesar cipher yang baru dengan menampilkan 25 karakter yang menghasilkan sebuah Ciphertext : **"zalnhuvnyhmp tlufluhunrhu"** yang disertai dengan hasil steganografi LSB sebagai konversi teks ke binary , dengan binary yang lengkap sebanyak (200 bit).

```
=== PROSES ENKRIPSI CAESAR CIPHER ===
Plaintext: steganografi menyenangkan
Shift: 7

Proses per karakter:
  1. 's' (pos 18) + 7 = 'z' (pos 25)
  2. 't' (pos 19) + 7 = 'a' (pos 0)
  3. 'e' (pos 4) + 7 = 'l' (pos 11)
  4. 'g' (pos 6) + 7 = 'n' (pos 13)
  5. 'a' (pos 0) + 7 = 'h' (pos 7)
  6. 'n' (pos 13) + 7 = 'u' (pos 20)
  7. 'o' (pos 14) + 7 = 'v' (pos 21)
  8. 'g' (pos 6) + 7 = 'n' (pos 13)
  9. 'r' (pos 17) + 7 = 'y' (pos 24)
  10. 'a' (pos 0) + 7 = 'h' (pos 7)
  11. 'f' (pos 5) + 7 = 'm' (pos 12)
  12. 'i' (pos 8) + 7 = 'p' (pos 15)
  13. ' ' -> ' ' (tidak berubah)
  14. 'm' (pos 12) + 7 = 't' (pos 19)
  15. 'e' (pos 4) + 7 = 'l' (pos 11)
  16. 'n' (pos 13) + 7 = 'u' (pos 20)
  17. 'y' (pos 24) + 7 = 'f' (pos 5)
  18. 'e' (pos 4) + 7 = 'l' (pos 11)
  19. 'n' (pos 13) + 7 = 'u' (pos 20)
  20. 'a' (pos 0) + 7 = 'h' (pos 7)
  21. 'n' (pos 13) + 7 = 'u' (pos 20)
  22. 'g' (pos 6) + 7 = 'n' (pos 13)
  23. 'k' (pos 10) + 7 = 'r' (pos 17)
  24. 'a' (pos 0) + 7 = 'h' (pos 7)
  25. 'n' (pos 13) + 7 = 'u' (pos 20)

Ciphertext: zalnhuvnyhmp tlufluhunrhu
```

```
=== PROSES STEGANOGRAFI LSB ===
Gambar carrier: images/luffy.jpg
Ukuran gambar: (345, 622, 3)
Dimensi: 622 x 345 pixels

=== KONVERSI TEKS KE BINARY ===
Teks: zalnhuvnyhmp tlufluhunrhu
Panjang: 25 karakter

Konversi per karakter:
  1. 'z' -> ASCII 122 -> 01111010
  2. 'a' -> ASCII 97 -> 01100001
  3. 'l' -> ASCII 108 -> 01101100
  4. 'n' -> ASCII 110 -> 01101110
  5. 'h' -> ASCII 104 -> 01101000
  6. 'u' -> ASCII 117 -> 01110101
  7. 'v' -> ASCII 118 -> 01110110
  8. 'n' -> ASCII 110 -> 01101110
  9. 'y' -> ASCII 121 -> 01111001
  10. 'h' -> ASCII 104 -> 01101000
  11. 'm' -> ASCII 109 -> 01101101
  12. 'p' -> ASCII 112 -> 01110000
  13. ' ' -> ASCII 32 -> 00100000
  14. 't' -> ASCII 116 -> 01110100
  15. 'l' -> ASCII 108 -> 01101100
  16. 'u' -> ASCII 117 -> 01110101
  17. 'f' -> ASCII 102 -> 01100110
  18. 'l' -> ASCII 108 -> 01101100
  19. 'u' -> ASCII 117 -> 01110101
  20. 'h' -> ASCII 104 -> 01101000
  21. 'u' -> ASCII 117 -> 01110101
  22. 'n' -> ASCII 110 -> 01101110
  23. 'r' -> ASCII 114 -> 01110010
  24. 'h' -> ASCII 104 -> 01101000
  25. 'u' -> ASCII 117 -> 01110101

Binary lengkap (200 bit):
01111010011000010110110001101110011010000111010101110110011011010...
```

Gambar 3.6 Hasil Enkripsi dan Embedding LSB

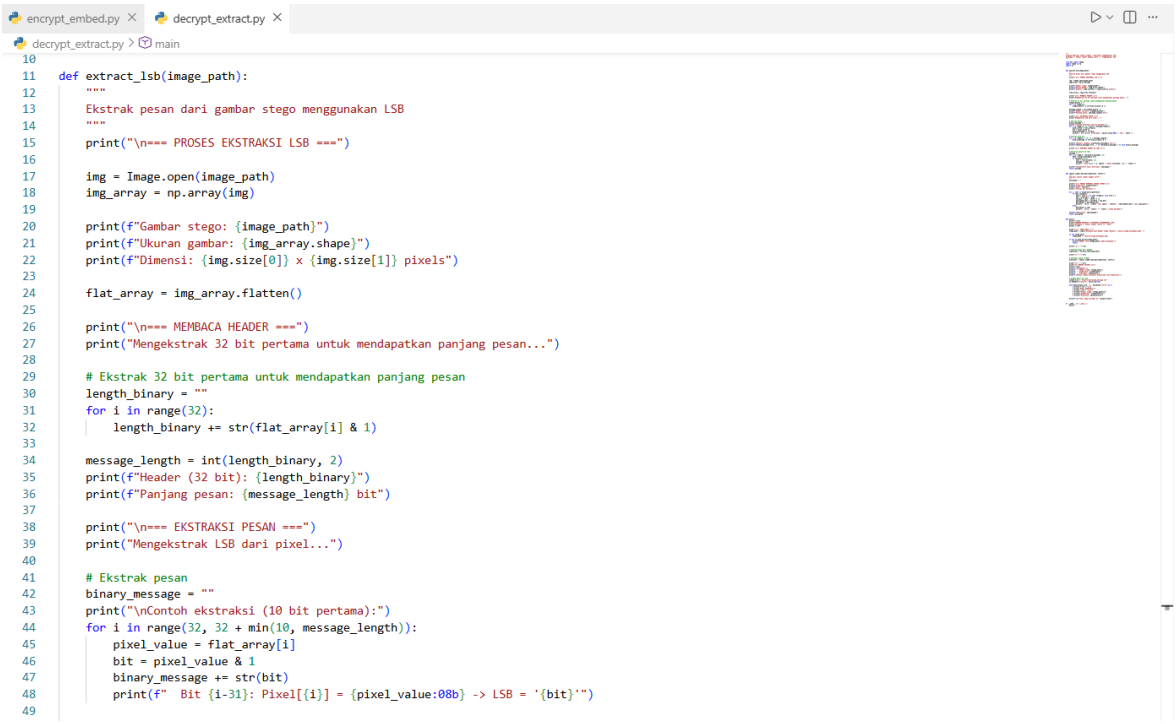
Proses Embedding modifikasi LSB yang sudah di embed dengan aman dan tersimpan.

```
=== HEADER INFORMASI ===
Panjang pesan: 200 bit
Header (32 bit): 00000000000000000000000011001000
Total bit yang akan di-embed: 232 bit

Kapasitas maksimal gambar: 643770 bit
Kapasitas tersisa: 643538 bit

=== PROSES EMBEDDING ===
Memodifikasi LSB pixel...

Contoh embedding (10 bit pertama):
  Bit 1: Pixel[0] = 00001010 -> 00001010 (embed '0')
  Bit 2: Pixel[1] = 01011100 -> 01011100 (embed '0')
  Bit 3: Pixel[2] = 10110010 -> 10110010 (embed '0')
  Bit 4: Pixel[3] = 00001010 -> 00001010 (embed '0')
  Bit 5: Pixel[4] = 01011100 -> 01011100 (embed '0')
  Bit 6: Pixel[5] = 10110010 -> 10110010 (embed '0')
  Bit 7: Pixel[6] = 00001010 -> 00001010 (embed '0')
  Bit 8: Pixel[7] = 01011100 -> 01011100 (embed '0')
  Bit 9: Pixel[8] = 10110010 -> 10110010 (embed '0')
  Bit 10: Pixel[9] = 00001010 -> 00001010 (embed '0')

✓ Stego image berhasil disimpan: results/stego_kelompok1.png
✓ Pesan berhasil di-embed dengan aman!

============================================================
√√√ PROSES SELESAI √√√
============================================================

Ringkasan:
  - Plaintext: steganografi menyenangkan
  - Ciphertext: zalnhuvnyhmp tlufluhunrhu
  - Gambar : images/luffy.jpg
  - Stego image: results/stego_kelompok1.png

Pesan Anda telah berhasil dienkripsi dan disembunyikan!
PS E:\UASKriptografi>
```

Gambar 3.7 Hasil Enkripsi dan Embedding LSB

3.5 Proses Extraction LSM dan Dekripsi

1. Buat file baru dengan nama decrypt_extract.py yang digunakan untuk mengekstrak dan mendekripsi pesan dari gambar stego sebelumnya, kemudian buat kodenya untuk menghasilkan hasil yang akan ditampilkan di terminalnya nanti.



```python
def extract_lsb(image_path):
    """
    Ekstrak pesan dari gambar stego menggunakan LSB
    """
    print("\n=== PROSES EKSTRAKSI LSB ===")

    img = Image.open(image_path)
    img_array = np.array(img)

    print(f"Gambar stego: {image_path}")
    print(f"Ukuran gambar: {img_array.shape}")
    print(f"Dimensi: {img.size[0]} x {img.size[1]} pixels")

    flat_array = img_array.flatten()

    print("\n=== MEMBACA HEADER ===")
    print("Mengekstrak 32 bit pertama untuk mendapatkan panjang pesan...")

    # Ekstrak 32 bit pertama untuk mendapatkan panjang pesan
    length_binary = ""
    for i in range(32):
        length_binary += str(flat_array[i] & 1)

    message_length = int(length_binary, 2)
    print(f"Header (32 bit): {length_binary}")
    print(f"Panjang pesan: {message_length} bit")

    print("\n=== EKSTRAKSI PESAN ===")
    print("Mengekstrak LSB dari pixel...")

    # Ekstrak pesan
    binary_message = ""
    print("\nContoh ekstraksi (10 bit pertama):")
    for i in range(32, 32 + min(10, message_length)):
        pixel_value = flat_array[i]
        bit = pixel_value & 1
        binary_message += str(bit)
        print(f"  Bit {i-31}: Pixel[{i}] = {pixel_value:08b} -> LSB = '{bit}'")
```

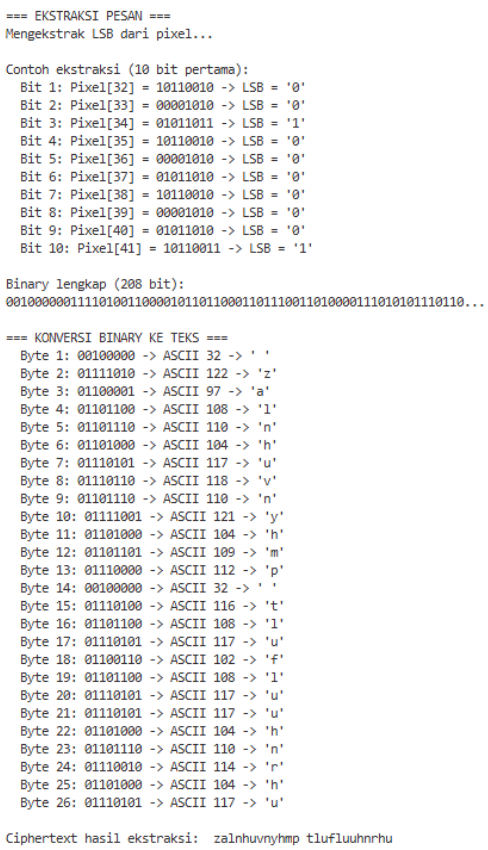
Gambar 3.8 Proses Extraction LSM dan Dekripsi

2. Selanjutnya ulangi cara yang sama dengan menjalankan run python untuk di terminal, setelah itu klik ENTER maka program akan otomatis menampilkan hasil input data jika sudah sesuai dalam dilanjutkan dengan klik enter maka proes ekstrksi LSB akan keluar dengan penjelasan ukuran gambar stego serta dimensi pixelnya, terdapat juga membaca header untuk mengekstrak 32 bit pertama sebagai Panjang pesan yang didapatkan hingga 208 bit.

```
PS E:\UASKriptografi> & C:/Users/LENOVO/AppData/Local/Programs/Python/Python312/python.exe e:/UASKriptografi/decrypt_extract.py
============================================================
PROGRAM DEKRIPSI & EKSTRAKSI STEGANOGRAFI LSB
Kelompok 1: Caesar Cipher (shift 7) + LSB
============================================================

--- INPUT DATA ---
Masukkan path gambar stego (default: results/stego_kelompok1.png):

============================================================

=== PROSES EKSTRAKSI LSB ===
Gambar stego: results/stego_kelompok1.png
Ukuran gambar: (345, 622, 3)
Dimensi: 622 x 345 pixels

=== MEMBACA HEADER ===
Mengekstrak 32 bit pertama untuk mendapatkan panjang pesan...
Header (32 bit): 00000000000000000000000011010000
Panjang pesan: 208 bit
```

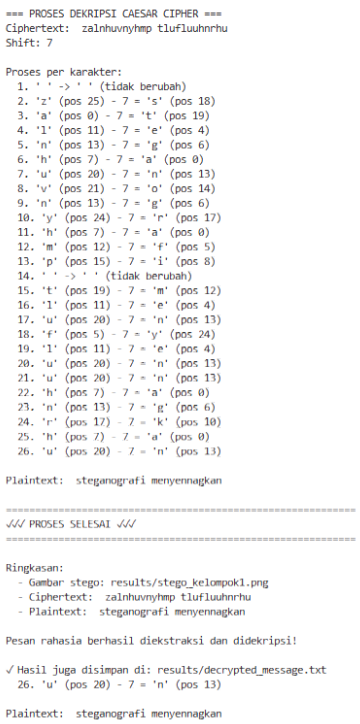
Gambar 3.9 Proses Extraction LSM dan Dekripsi

3. Selanjutnya hasil dari ektraksi pesan yang telah mengektrak LSB dari pixel dan konversi binary ke teks yang akan menghasilkan Ciphertext dari hasil ektraksi:



Gambar 3.10 hasil dari ektraksi pesan yang telah mengektrak LSB dari pixel

4. Selanjutnya proses dekripsi Caesar Cipher dengan Ciphertext :“zalnhuvnyhmp tlufluuhnrhu” yang menghasilkan Plaintext: “steganografi menyenangkan”



Gambar 3.11 Proses Dekripsi Caesar Cipher

3.6 Kelebihan dan Kelemahan Implementasi

a. Kelebihan

1. Keamanan Berlapis: Kombinasi kriptografi dan steganografi memberikan dua lapisan perlindungan. Meskipun pesan berhasil diekstrak, isinya tetap tidak terbaca tanpa dekripsi.
2. Imperceptibility Tinggi: Perubahan pada citra stego tidak terdeteksi oleh mata manusia karena modifikasi hanya pada bit terakhir dengan perubahan nilai maksimal ±1.
3. Kapasitas Besar: Citra RGB berukuran 600×600 piksel dapat menyimpan hingga 1.080.000 bit atau sekitar 135 KB data.
4. Implementasi Sederhana: Kedua algoritma mudah dipahami, diimplementasikan, dan memiliki performa yang cepat.

b. Kelemahan

1. Keamanan Caesar Cipher Rendah: Hanya memiliki 26 kemungkinan kunci sehingga sangat rentan terhadap serangan brute force yang dapat dilakukan dalam hitungan detik.

2. Rentan terhadap Manipulasi Citra: Kompresi JPEG, cropping, resizing, atau konversi format dapat merusak atau menghilangkan pesan yang tersembunyi.
3. Deteksi Steganalisis: Metode LSB rentan terhadap analisis statistik seperti chi-square attack yang dapat mendeteksi keberadaan pesan tersembunyi.
4. Format Output Terbatas: Harus menggunakan format lossless (PNG) untuk menjaga integritas pesan, sehingga ukuran file lebih besar dibanding format terkompresi.

3.7 Kesimpulan Pembahasan

Implementasi kombinasi Caesar Cipher shift 7 dan Steganografi LSB berhasil dilakukan dengan baik menggunakan Python. Sistem mampu melakukan enkripsi pesan, menyisipkannya ke dalam citra digital, kemudian mengekstrak dan mendekripsi kembali dengan hasil yang akurat. Metode ini cocok digunakan untuk pembelajaran konsep keamanan berlapis, namun untuk aplikasi produksi disarankan menggunakan algoritma kriptografi yang lebih kuat seperti AES atau RSA untuk meningkatkan keamanan sistem.

3.8 Jobdesk Masing – Masing Anggota

Nama	Jobdesk
ANGGI SATRIA WIRANATA	Laporan & Coding
NABILLAH RACHELIA M	Laporan Bab 1 & 4
M. FIRZA KHOIRUDIN	Editing Video & Laporan
LAILA IVONE BELLA P	Laporan & Coding
GILANG PRADANA	Laporan Bab 2 & Referensi

BAB IV
PENUTUP

4.1 Kesimpulan

Berdasarkan hasil pembahasan dan analisis yang telah dilakukan, dapat disimpulkan bahwa algoritma Caesar Cipher dapat di-implementasikan dengan baik untuk melakukan proses enkripsi dan dekripsi pesan teks menggunakan bahasa pemrograman Python. Algoritma ini mampu menyamarkan pesan asli menjadi bentuk ciphertext melalui teknik pergeseran karakter sehingga pesan tidak dapat dibaca secara langsung oleh pihak yang tidak berwenang. Selain itu, metode steganografi Least Significant Bit (LSB) berhasil digunakan untuk menyisipkan pesan terenkripsi ke dalam citra digital berformat RGB dengan perubahan visual yang sangat kecil dan sulit dibedakan secara kasat mata. Proses ekstraksi pesan dari citra steganografi dan dekripsi kembali menggunakan Caesar Cipher dapat mengembalikan pesan asli secara utuh, selama citra tidak mengalami manipulasi dan kunci enkripsi yang digunakan sesuai. Kombinasi antara kriptografi dan steganografi ini membentuk sistem keamanan berlapis, di mana pesan tidak hanya diamankan isinya melalui enkripsi, tetapi juga disembunyikan keberadaannya di dalam media citra. Namun demikian, tingkat keamanan sistem ini masih terbatas karena Caesar Cipher rentan terhadap serangan brute force akibat ruang kunci yang kecil, serta metode LSB memiliki ketahanan yang rendah terhadap manipulasi citra digital.

4.2 Saran

Berdasarkan hasil proyek yang telah dilaksanakan, disarankan agar pengembangan selanjutnya menggunakan algoritma kriptografi yang lebih kuat dan aman, seperti Advanced Encryption Standard (AES) atau algoritma kriptografi modern lainnya, untuk meningkatkan tingkat keamanan pesan. Metode steganografi LSB juga dapat dikembangkan dengan menambahkan teknik pengacakan posisi bit atau dikombinasikan dengan metode steganografi lain agar lebih tahan terhadap serangan steganalisis dan manipulasi citra. Selain itu, pengujian sistem sebaiknya dilakukan pada berbagai format citra dan ukuran pesan yang berbeda untuk mengetahui batas kapasitas penyisipan serta ketahanan sistem terhadap perubahan kualitas citra. Pengembangan aplikasi juga dapat ditingkatkan dengan penambahan antarmuka pengguna yang lebih interaktif dan ramah pengguna agar sistem lebih mudah digunakan. Penelitian selanjutnya diharapkan mampu mengombinasikan teknik kriptografi dan steganografi yang lebih modern sehingga menghasilkan sistem pengamanan informasi yang lebih aman, efisien, dan andal.

DAFTAR PUSTAKA

Alfinatul Hasanah, M. T. (2023). Implementasi Algoritma Caesar Cipher untuk Pengamanan Pesan Menggunakan Java NetBeans. *Digital Transformation Technology (Digitech)*, 11-19.

Amin, M. M. (2016). IMPLEMENTASI KRIPTOGRAFI KLASIK PADA KOMUNIKASI BERBASIS TEKS. *Jurnal Pseudocode*.

Arum Purbaningrum, K. S. (2023). Penerapan Metode Least Significant Bit (LSB) dalam Menyisipkan Pesan Rahasia pada Citra Digital: Sebuah Pendekatan Steganograf. *SEMINAR NASIONAL AMIKOM SURAKARTA*, 176 - 183.

Ary Hidayatullah, E. I. (2016). PENGENALAN KRIPTOGRAFI DAN PEMAKAIANYA SEHARI-HARI. *Jurnal Pengenalan Kriptografi Dan Pemakaianya Sehari-Hari*.

Chyan, P. (2018). PENERAPAN SISTEM KRIPTOGRAFI ENKRIPSI JAMAK DAN TANDA TANGAN DIGITAL DALAM MENDUKUNG KEAMANAN INFORMASI. *JURNAL TEMATIKA*, 39-46.

E.Haodudin Nurkifli, D. W. (2016). ANALISIS CRYPTANALYS EXHAUSTIVE SEARCH DAN STATISTICAL ATTACK PADA ALGORITMA PLAYFAIR CIPHER. *Jurnal Hasil Riset*.

Fachrul Dhika Ardiansyah, A. D. (2023). IMPLEMENTASI KRIPTOGRAFI CAESAR CHIPER PADA APLIKASI ENKRIPSI DAN DEKRIPSI . *JURNAL ILMIAH SISTEM INFORMASI DAN ILMU KOMPUTER*, 105-112.

Fitri Yanti, K. B. (2023). Implementasi Steganografi Menggunakan Metode Least Significant Bit (Lsb) dalam Pengamanan Informasi pada Citra Digital. *Jurnal Vocational Teknik Elektronika dan Informatika*, 64 - 70.

Melvayana Manik, S. S. (2024). Pengujian dan Analisis Teknik Steganografi Menggunakan Metode Playfair, ElGamal, dan LSB untuk Penyembunyian Data pada Gambar Digital dalam Aplikasi Modern. *Jurnal Quancom*, 17 - 22.

Mesran, S. D. (2020). Peningkatan Keamanan Kriptografi Caesar Cipher dengan Menerapkan Algoritma Kompresi “Stout Codes”. *JURNAL RESTI (Rekayasa Sistem dan Teknologi Informasi)*, 1209 –1215.

Muhammad Ferdiansyah Hidayat, M. R. (2024). Implementasi Metode Caesar Chiper Dalam Kriptografi Untuk Keamanan Data Pesan. *Jurnal Multidisiplin Indonesia*.

Nanda Amalya, S. M. (2023). Kriptografi dan Penerapannya Dalam Sistem Keamanan Data. *JURNAL MEDIA INFORMATIKA [JUMIN]*, 90-93.

Rinaldi, M. (2017). *Pengantar Kriptografi: Konsep Dasar dan Aplikasi pada Keamanan Data*. Yogyakarta: Andi.

Sari, D. (2020). *Keamanan Data Menggunakan Teknik Kriptografi*. Jakarta: Mitra Wacana Media.

Yudhi Darmawan, N. F. (2024). OPTIMASI STEGANOGRAFI BERBASIS LEAST SIGNIFICANT BIT (LSB) METODE OPERASI PERGESERAN BIST. *Jurnal Mahasiswa Teknik Informatika*, 11055 - 11059.

LAMPIRAN

A. Source Code encrypt_embed.py

```
from PIL import Image
   import numpy as np
   import os

   def caesar_cipher_encrypt(plaintext, shift=7):
       encrypted = ""

       print("\n=== PROSES ENKRIPSI CAESAR CIPHER ===")
       print(f"Plaintext: {plaintext}")
       print(f"Shift: {shift}")
       print("\nProses per karakter:")

       for i, char in enumerate(plaintext):
           if char.isalpha():
               base = ord('A') if char.isupper() else ord('a')
               pos = ord(char) - base
               new_pos = (pos + shift) % 26
               encrypted_char = chr(base + new_pos)
               encrypted += encrypted_char
               print(f"  {i+1}. '{char}' (pos {pos}) + {shift} =
   '{encrypted_char}' (pos {new_pos})")
            else:
               encrypted += char
               print(f"  {i+1}. '{char}' -> '{char}' (tidak berubah)")

       print(f"\nCiphertext: {encrypted}")
       return encrypted

   def text_to_binary(text):
       binary = ''.join(format(ord(char), '08b') for char in text)

       print("\n=== KONVERSI TEKS KE BINARY ===")
       print(f"Teks: {text}")
       print(f"Panjang: {len(text)} karakter")
       print("\nKonversi per karakter:")

       for i, char in enumerate(text):
           binary_char = format(ord(char), '08b')
           print(f"  {i+1}. '{char}' -> ASCII {ord(char)} ->
   {binary_char}")

       print(f"\nBinary lengkap ({len(binary)} bit):")
       print(f"{binary[:64]}..." if len(binary) > 64 else binary)

       return binary
```

```
def embed_lsb(image_path, message, output_path):
    print("\n=== PROSES STEGANOGRAFI LSB ===")

    img = Image.open(image_path)
    img_array = np.array(img)

    print(f"Gambar carrier: {image_path}")
    print(f"Ukuran gambar: {img_array.shape}")
    print(f"Dimensi: {img.size[0]} x {img.size[1]} pixels")

    binary_message = text_to_binary(message)
    message_length = len(binary_message)
    length_binary = format(message_length, '032b')

    print(f"\n=== HEADER INFORMASI ===")
    print(f"Panjang pesan: {message_length} bit")
    print(f"Header (32 bit): {length_binary}")

    full_binary = length_binary + binary_message
    print(f"Total bit yang akan di-embed: {len(full_binary)} bit")

    height, width, channels = img_array.shape
    max_capacity = height * width * channels

    print(f"\nKapasitas maksimal gambar: {max_capacity} bit")

    if len(full_binary) > max_capacity:
        print("ERROR: Pesan terlalu panjang untuk gambar ini!")
        return False

    print(f"Kapasitas tersisa: {max_capacity - len(full_binary)}
bit")

    print("\n=== PROSES EMBEDDING ===")
    print("Memodifikasi LSB pixel...")

    flat_array = img_array.flatten()

    print("\nContoh embedding (10 bit pertama):")
    for i in range(min(10, len(full_binary))):
        original_pixel = flat_array[i]
        bit_to_embed = int(full_binary[i])
        modified_pixel = (original_pixel & 0xFE) | bit_to_embed
        print(f"  Bit {i+1}: Pixel[{i}] = {original_pixel:08b} ->
{modified_pixel:08b} (embed '{bit_to_embed}')")
        flat_array[i] = modified_pixel
```

```
    for i in range(10, len(full_binary)):
        bit_to_embed = int(full_binary[i])
        flat_array[i] = (flat_array[i] & 0xFE) | bit_to_embed

    stego_array = flat_array.reshape(img_array.shape)
    stego_img = Image.fromarray(stego_array.astype('uint8'))
    stego_img.save(output_path)

    print(f"\n✓ Stego image berhasil disimpan: {output_path}")
    print(f"✓ Pesan berhasil di-embed dengan aman!")

    return True

def main():
    print("="*60)
    print("PROGRAM ENKRIPSI & EMBEDDING STEGANOGRAFI LSB")
    print("Kelompok 1: Caesar Cipher (shift 7) + LSB")
    print("="*60)

    print("\n--- INPUT DATA ---")
    plaintext = input("Masukkan pesan rahasia: ")

    if not plaintext:
        print("ERROR: Pesan tidak boleh kosong!")
        return

    image_path = input("Masukkan path gambar  (default:
images/luffy.jpg): ")
    if not image_path:
        image_path = "images/luffy.jpg"

    if not os.path.exists(image_path):
        print(f"ERROR: File {image_path} tidak ditemukan!")
        return

    output_path = "results/stego_kelompok1.png"
    os.makedirs("results", exist_ok=True)

    print("\n" + "="*60)

    ciphertext = caesar_cipher_encrypt(plaintext, shift=7)

    print("\n" + "="*60)

    success = embed_lsb(image_path, ciphertext, output_path)

    if success:
        print("\n" + "="*60)
```

```
        print("✓✓✓ PROSES SELESAI ✓✓✓")
        print("="*60)
        print(f"\nRingkasan:")
        print(f"  - Plaintext: {plaintext}")
        print(f"  - Ciphertext: {ciphertext}")
        print(f"  - Gambar : {image_path}")
        print(f"  - Stego image: {output_path}")
        print(f"\nPesan Anda telah berhasil dienkripsi
dandisembunyikan!")

if __name__ == "__main__":
    main()
```

B. Source Code decrypt_extract.py

```
from PIL import Image
import numpy as np
import os

def extract_lsb(image_path):
    """
    Ekstrak pesan dari gambar stego menggunakan LSB
    """
    print("\n=== PROSES EKSTRAKSI LSB ===")


    img = Image.open(image_path)
    img_array = np.array(img)

    print(f"Gambar stego: {image_path}")
    print(f"Ukuran gambar: {img_array.shape}")
    print(f"Dimensi: {img.size[0]} x {img.size[1]} pixels")

    flat_array = img_array.flatten()

    print("\n=== MEMBACA HEADER ===")
    print("Mengekstrak 32 bit pertama untuk mendapatkan Panjang
pesan...")

    # Ekstrak 32 bit pertama untuk mendapatkan panjang pesan
    length_binary = ""
    for i in range(32):
        length_binary += str(flat_array[i] & 1)

    message_length = int(length_binary, 2)
    print(f"Header (32 bit): {length_binary}")
    print(f"Panjang pesan: {message_length} bit")

    print("\n=== EKSTRAKSI PESAN ===")
    print("Mengekstrak LSB dari pixel...")

    # Ekstrak pesan
    binary_message = ""
    print("\nContoh ekstraksi (10 bit pertama):")
    for i in range(32, 32 + min(10, message_length)):
        pixel_value = flat_array[i]
        bit = pixel_value & 1
        binary_message += str(bit)
        print(f"  Bit {i-31}: Pixel[{i}] = {pixel_value:08b} -> LSB=
'{bit}'")

    # Ekstrak sisa bit
```

```
    for i in range(32 + 10, 32 + message_length):
        binary_message += str(flat_array[i] & 1)

    print(f"\nBinary lengkap ({len(binary_message)} bit):")
    print(f"{binary_message[:64]}..." if len(binary_message) > 64
else binary_message)

    print("\n=== KONVERSI BINARY KE TEKS ===")

    # Konversi binary ke teks
    message = ""
    for i in range(0, len(binary_message), 8):
        byte = binary_message[i:i+8]
        if len(byte) == 8:
            char = chr(int(byte, 2))
            message += char
            print(f"  Byte {i//8 + 1}: {byte} -> ASCII {int(byte,
2)} -> '{char}'")

    print(f"\nCiphertext hasil ekstraksi: {message}")
    return message

def caesar_cipher_decrypt(ciphertext, shift=7):
    """
    Dekripsi Caesar Cipher dengan shift 7
    """
    decrypted = ""

    print("\n=== PROSES DEKRIPSI CAESAR CIPHER ===")
    print(f"Ciphertext: {ciphertext}")
    print(f"Shift: {shift}")
    print("\nProses per karakter:")

    for i, char in enumerate(ciphertext):
        if char.isalpha():
            base = ord('A') if char.isupper() else ord('a')
            pos = ord(char) - base
            new_pos = (pos - shift) % 26
            decrypted_char = chr(base + new_pos)
            decrypted += decrypted_char
            print(f"  {i+1}. '{char}' (pos {pos}) - {shift} =
'{decrypted_char}' (pos {new_pos})")
        else:
            decrypted += char
            print(f"  {i+1}. '{char}' -> '{char}' (tidak berubah)")

    print(f"\nPlaintext: {decrypted}")
    return decrypted
```

```
def main():
    print("="*60)
    print("PROGRAM DEKRIPSI & EKSTRAKSI STEGANOGRAFI LSB")
    print("Kelompok 1: Caesar Cipher (shift 7) + LSB")
    print("="*60)

    print("\n--- INPUT DATA ---")
    stego_path = input("Masukkan path gambar stego (default:
results/stego_kelompok1.png): ")

    if not stego_path:
        stego_path = "results/stego_kelompok1.png"

    if not os.path.exists(stego_path):
        print(f"ERROR: File {stego_path} tidak ditemukan!")
        return

    print("\n" + "="*60)

    # Ekstrak pesan dari gambar
    ciphertext = extract_lsb(stego_path)

    print("\n" + "="*60)

    # Dekripsi Caesar Cipher
    plaintext = caesar_cipher_decrypt(ciphertext, shift=7)

    print("\n" + "="*60)
    print("✓✓✓ PROSES SELESAI ✓✓✓")
    print("="*60)
    print(f"\nRingkasan:")
    print(f"  - Gambar stego: {stego_path}")
    print(f"  - Ciphertext: {ciphertext}")
    print(f"  - Plaintext: {plaintext}")
    print(f"\nPesan rahasia berhasil diekstraksi dan didekripsi!")

    # Simpan hasil ke file
    output_file = "results/decrypted_message.txt"
    os.makedirs("results", exist_ok=True)

    with open(output_file, 'w', encoding='utf-8') as f:
        f.write("="*60 + "\n")
        f.write("HASIL DEKRIPSI\n")
        f.write("="*60 + "\n\n")
        f.write(f"Gambar stego: {stego_path}\n")
        f.write(f"Ciphertext: {ciphertext}\n")
        f.write(f"Plaintext: {plaintext}\n")
```

```
    print(f"\n✓ Hasil juga disimpan di: {output_file}")

if __name__ == "__main__":
    main()
```