

# Laporan Tugas Kecil 2 IF4020

## Kriptografi



**Disusun oleh:**

Farhan Nafis Rayhan (13522037)

Abdul Rafi Radityo Hutomo (13522089)

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA INSTITUT  
TEKNOLOGI BANDUNG**

**2025**

# **Daftar Isi**

<b>Daftar Isi</b>	<b>2</b>
<b>BAB 1: Teori Singkat</b>	<b>3</b>
Steganografi	3
Metode Least Significant Bit pada Steganografi	3
File MP3	4
Cipher Vigenère	5
<b>Bab 2: Perancangan dan Implementasi</b>	<b>6</b>
2.1 Arsitektur Sistem	6
2.2 Rancangan Algoritma Steganografi	9
2.2.1 Lokasi Penyisipan di MP3	9
2.2.2 Seleksi Kandidat & Masking	9
2.2.3 Penjadwalan Posisi (Pseudo-acak, Deterministik)	9
2.3 Format Header & Enkripsi	10
2.4 Perhitungan Kapasitas	10
2.5 Perhitungan PSNR	10
2.6 Parsing Header dan Huffman Bits pada MP3	11
<b>Bab 3: Pengujian program dan Analisis Hasil</b>	<b>14</b>
<b>Testcase 1: D4vd</b>	<b>14</b>
<b>Testcase 2: Quadeca</b>	<b>17</b>
<b>Testcase 3: Succession</b>	<b>20</b>
<b>Testcase 4: Inception</b>	<b>23</b>
<b>Analisis Testcase</b>	<b>24</b>
<b>Bab 4: Kesimpulan</b>	<b>26</b>
<b>Bab 5: Daftar Pustaka</b>	<b>27</b>
<b>Bab 6: Lampiran</b>	<b>28</b>
Repository Github: <a href="https://github.com/abdulrafirh/Tucil2-IF4020">https://github.com/abdulrafirh/Tucil2-IF4020</a>	28
Video Demo: <a href="https://youtu.be/cT--6sSWRBk?si=3PWdtK7VjeiZnIVu">https://youtu.be/cT--6sSWRBk?si=3PWdtK7VjeiZnIVu</a>	28
Pembagian Tugas	28

# BAB 1: Teori Singkat

## Steganografi

Secara singkat, Steganografi adalah ilmu dan seni menyembunyikan pesan rahasia sedemikian rupa sehingga tidak seorang pun mencurigai keberadaan pesan tersebut<sup>1</sup>. Istilah ini berasal dari Bahasa Yunani, dari kata steganos yang berarti tersembunyi dan graphien, yaitu tulisan, yang secara harfiah berarti tulisan tersembunyi. Tujuan utama steganografi adalah menyembunyikan keberadaan pesan untuk menghindari kecurigaan dari pihak ketiga atau pengamat, menjadikannya berbeda dari kriptografi yang bertujuan menyembunyikan makna atau isi pesan melalui enkripsi.

Dalam konteks digital, steganografi melibatkan penyembunyian pesan digital (*embedded message*) di dalam dokumen digital lain yang disebut *cover object* seperti teks, gambar, audio, atau video. Hasil dari proses ini adalah *stego object*, yang terlihat seperti *cover object* biasa, sehingga pesan tersembunyi di dalamnya tidak dapat dideteksi.

Kualitas steganografi yang baik mensyaratkan pesan rahasia tidak terdeteksi secara visual atau audial (*imperceptible*), kualitas cover-object tidak berubah jauh setelah penyisipan (*fidelity*), pesan harus dapat diekstraksi kembali (*recovery*), dan sedapat mungkin memiliki kapasitas besar untuk menyembunyikan pesan (*capacity*).

Steganografi sering digunakan bersama dengan kriptografi untuk meningkatkan keamanan pesan. pesan mula-mula dienkripsi, lalu pesan terenkripsi tersebut disembunyikan di dalam media lain. Namun, perlu diingat bahwa Steganografi bukanlah bagian dari kriptografi, melainkan sebagai pelengkap seni kriptografi saja.

## Metode Least Significant Bit pada Steganografi

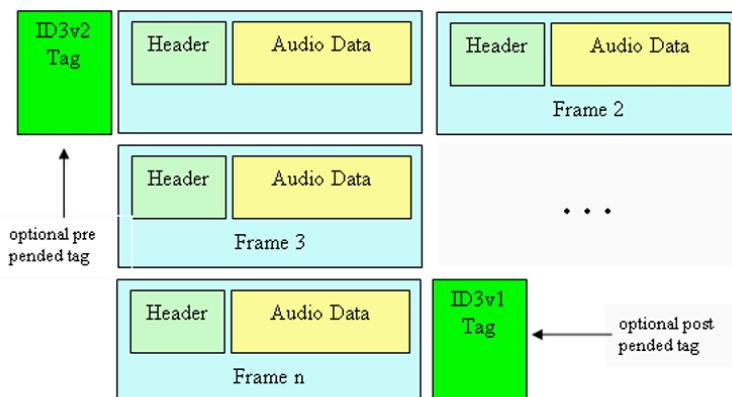
Metode Least Significant Bit (LSB) adalah teknik dasar steganografi domain spasial yang memanfaatkan fakta bahwa bit yang paling tidak signifikan pada data digital dapat diubah tanpa menyebabkan perubahan yang signifikan secara visual atau audial pada media pembawa. Dalam sebuah *byte* yang memiliki 8 bit, LSB hanya berkontribusi sedikit pada nilai keseluruhan. Dalam citra digital (*cover-object*), ini berarti mengganti LSB dari setiap komponen warna *pixel* dengan bit rahasia akan menghasilkan perubahan warna yang sangat minimal, umumnya tidak terdeteksi oleh mata manusia, sehingga mempertahankan fidelity. Embedded message disisipkan bit demi bit ke dalam *byte-byte* citra secara terstruktur.

Varian metode LSB meliputi metode sekuensial, yaitu penyisipan berurutan, ataupun metode acak dengan menggunakan *PRNG* dan *stego-key* untuk memilih *pixel* secara acak guna meningkatkan keamanan. Selain itu ada pula metode m-bit LSB dimana kita menggunakan lebih dari satu bit LSB per *byte* untuk meningkatkan capacity. Lebih tepatnya, digunakan m buah bit. Untuk mendapatkan kembali pesan, penerima membaca *byte-byte* *stego-object* dan mengekstrak bit LSB yang telah dimodifikasi, merangkainya kembali menjadi pesan asli.

## File MP3

File MP3 (MPEG-1 Audio Layer III) adalah salah satu format audio digital paling populer yang menggunakan metode kompresi lossy. Artinya, sebagian data asli dari rekaman suara akan dibuang untuk mengurangi ukuran file tanpa terlalu mengorbankan kualitas suara yang dapat didengar manusia. Hal ini berbeda dengan format WAV yang menyimpan audio dalam bentuk lossless atau tanpa kompresi, sehingga kualitasnya lebih tinggi tetapi ukuran file jauh lebih besar. Misalnya, lagu berdurasi 3 menit dalam format WAV bisa berukuran sekitar 30 MB, sementara dalam format MP3 hanya beberapa MB saja.

Dari sisi penyimpanan, file MP3 menyimpan data suara dalam bentuk frame yang berisi blok-blok kecil audio terkompresi. Setiap frame memiliki header yang berisi informasi penting seperti bitrate, frekuensi sampling, dan mode kanal (stereo/mono), serta data audio yang telah dikompresi dengan algoritma perceptual coding. Algoritma ini membuang frekuensi-frekuensi suara yang dianggap kurang signifikan bagi pendengaran manusia (berdasarkan prinsip psychoacoustics). Dengan struktur seperti ini, pemutar audio bisa membaca frame demi frame secara berurutan untuk merekonstruksi kembali suara yang bisa didengar.



file MP3 terdiri dari beberapa komponen utama: tag metadata, frame audio, serta header di dalam setiap frame. Pada bagian awal file dapat terdapat ID3v2 tag (opsional) yang berfungsi menyimpan informasi tambahan seperti judul lagu, artis, album, tahun, atau bahkan cover art. Sedangkan di bagian akhir file bisa terdapat ID3v1 tag (juga opsional) yang memiliki fungsi serupa meski dengan kapasitas lebih kecil. Tag ini tidak berhubungan langsung dengan suara, melainkan hanya berisi data pendukung untuk identifikasi dan pengelolaan file musik.

Setiap frame MP3 terdiri dari dua bagian: header dan audio data. Header menyimpan informasi teknis penting seperti bitrate (berapa banyak data yang digunakan tiap detik audio), frekuensi sampling (misalnya 44,1 kHz), mode kanal (stereo atau mono), serta informasi sinkronisasi untuk memastikan pemutar audio bisa membaca frame secara berurutan. Setelah header, terdapat audio data yang berisi potongan kecil suara yang telah dikompresi. MP3

biasanya menyimpan ribuan frame secara berurutan; dengan membaca frame demi frame, pemutar dapat merekonstruksi kembali keseluruhan lagu.

Proses pembacaan MP3 mengikuti alur ini: pemutar audio terlebih dahulu mengecek apakah terdapat tag ID3 di awal file, kemudian mulai membaca frame satu per satu. Dari setiap frame, header diproses untuk mengetahui bagaimana mendekode data, lalu audio data didekompresi menjadi sinyal digital PCM. Pada bagian akhir file, jika terdapat tag ID3v1, pemutar juga dapat membacanya untuk melengkapi informasi metadata. Dengan struktur seperti ini, MP3 menjadi sangat fleksibel: data audio bisa diputar tanpa harus membaca keseluruhan file sekaligus (streaming-friendly), sementara metadata tetap bisa disimpan untuk kenyamanan pengguna.

## Cipher Vigenère

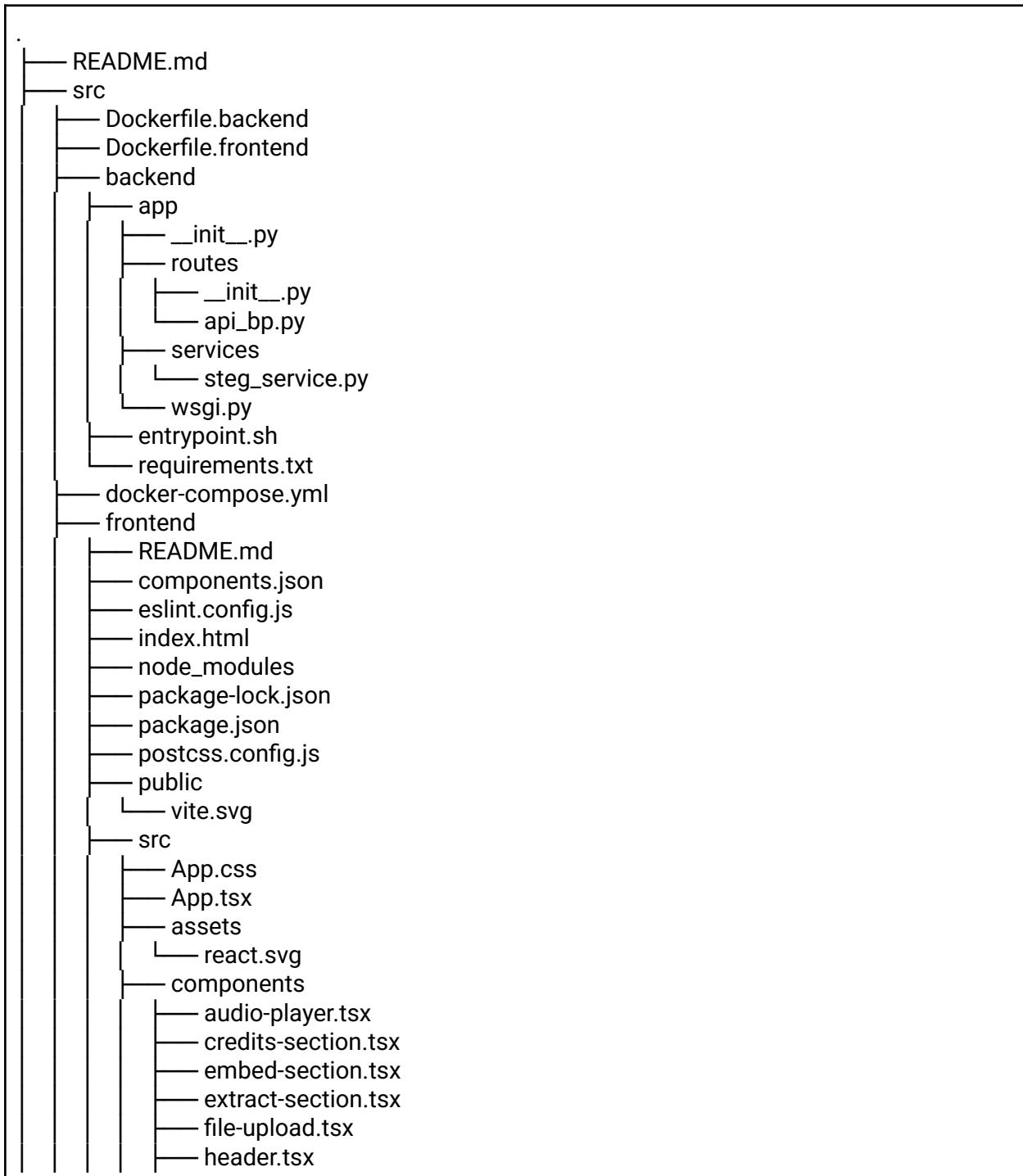
Vigenère Cipher adalah salah satu algoritma kriptografi klasik yang termasuk dalam kategori polyalphabetic substitution cipher. Teknik ini menggunakan sebuah kata kunci (key) untuk menentukan pergeseran huruf dalam plaintext. Setiap huruf pada pesan asli akan dienkripsi dengan menggunakan huruf pada kunci secara berulang. Misalnya, jika kuncinya adalah "KEY", maka huruf pertama dienkripsi dengan pergeseran berdasarkan 'K', huruf kedua dengan 'E', huruf ketiga dengan 'Y', lalu kembali ke 'K' untuk huruf keempat, dan seterusnya. Keunggulan Vigenère dibanding cipher sederhana seperti Caesar adalah lebih sulit ditebak karena menggunakan banyak abjad pengganti, sehingga pola perulangan lebih tersembunyi.

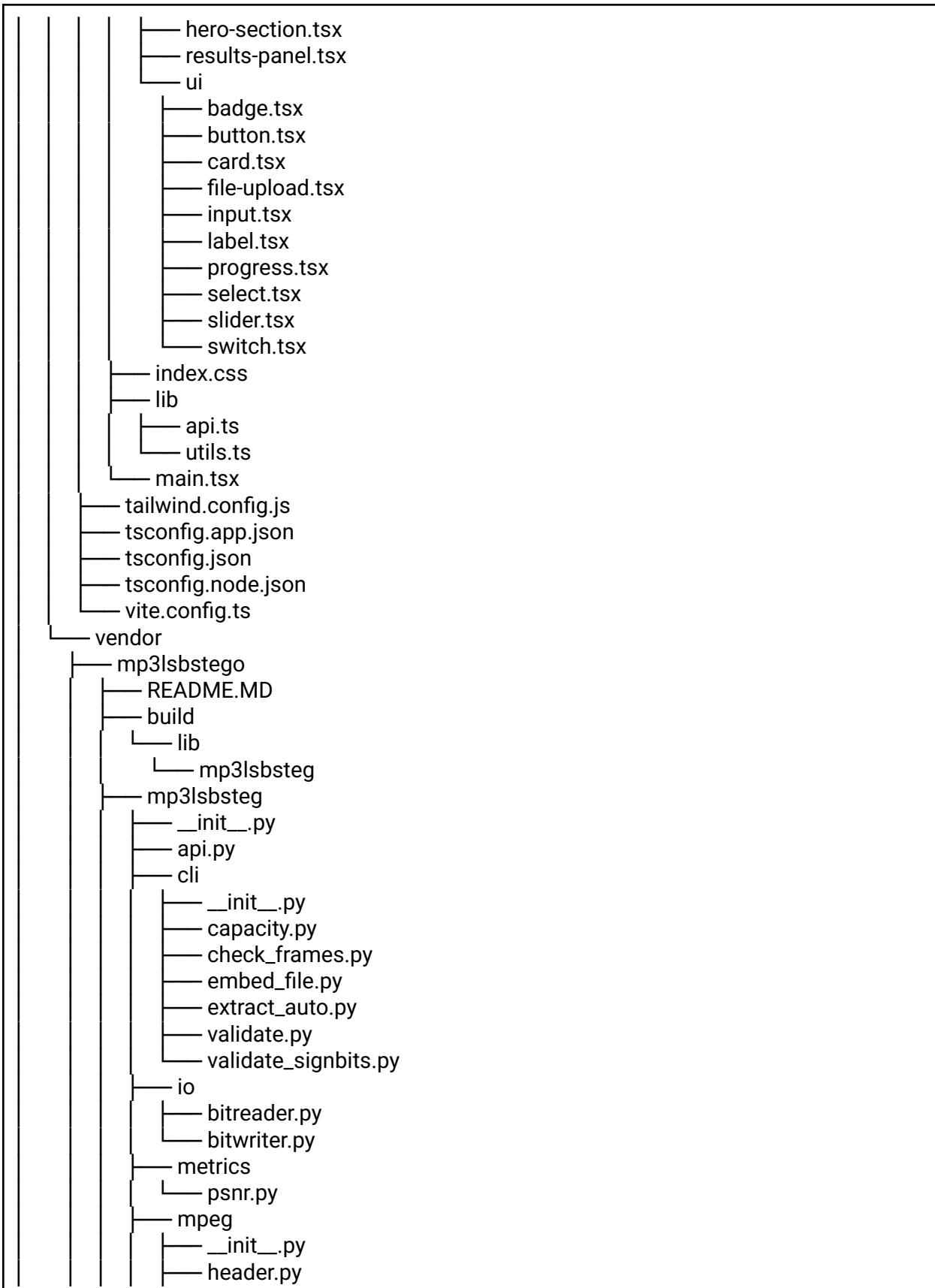
Salah satu variasi dari Vigenère Cipher adalah Extended Vigenère Cipher, yang memperluas ruang karakter dari hanya huruf alfabet menjadi 256 karakter ASCII. Dengan cara ini, cipher dapat mengenkripsi tidak hanya huruf, tetapi juga angka, simbol, spasi, hingga karakter khusus lainnya. Proses enkripsinya serupa dengan Vigenère klasik, hanya saja perhitungan dilakukan menggunakan nilai desimal ASCII. Setiap karakter plaintext ditambahkan dengan nilai karakter kunci (mod 256), sedangkan proses dekripsi dilakukan dengan pengurangan (juga mod 256). Dengan dukungan penuh terhadap seluruh karakter ASCII, variasi ini membuat cipher lebih fleksibel dan dapat digunakan untuk mengenkripsi teks maupun file biner sederhana.

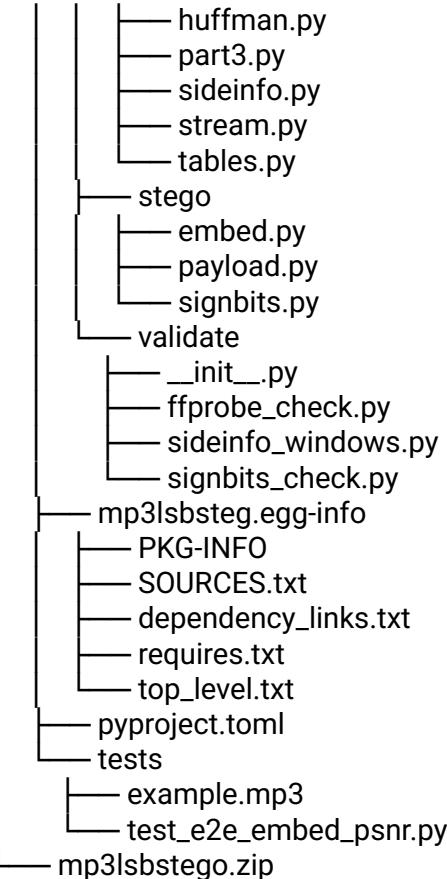
# Bab 2: Perancangan dan Implementasi

## 2.1 Arsitektur Sistem

Untuk struktur folder kami adalah sebagai berikut







Sistem terdiri dari tiga komponen utama:

## 1. Library mp3lsbsteg

Library untuk processing steganografi yang diimplementasikan dengan python yang menyediakan API untuk embed, extract, estimate capacity, dan PSNR. Pustaka ini bekerja langsung pada bitstream MP3 (tanpa re-encode), sehingga perubahan bersifat halus dan tahan ukuran.

## 2. Flask Backend

Menyediakan REST API:

- POST /api/capacity – hitung kapasitas sisip (bit/byte) pada MP3 untuk parameter yang diberikan; juga mengembalikan fits bila diberi ukuran payload.
  - POST /api/embed – menerima MP3 (carrier) + file payload + parameter, mengembalikan **stego MP3** dan header X-PSNR-dB.
  - POST /api/extract – menerima **stego MP3** + parameter, mengembalikan file hasil ekstraksi (nama/ekstensi mengikuti header).
- Nilai **fraction=1.0**, **mask\_percentile=0.75**, dan **max\_frames=None** dikunci di

layer services (konsisten dengan implementasi).

### 3. Frontend – Vite + React + Tailwind

Fitur utama:

- Embed Section: unggah carrier + secret, pilih bits per frame (1–4), key, dan Vigenère; tombol Check Capacity sebelum embed.  
Extract Section: unggah stego, masukkan key, pilih bits per frame dan Vigenère.
- Results Panel: kartu Capacity, Embed (PSNR), Extract (download).  
Frontend berkomunikasi ke backend via VITE\_API\_BASE.

Containerization: docker-compose menjalankan backend (Python 3.11 + ffmpeg) dan frontend (Node 22) dalam dua kontainer. Pustaka mp3lsbsteg di-mount dari vendor/ lalu di-install saat start.

## 2.2 Rancangan Algoritma Steganografi

### 2.2.1 Lokasi Penyisipan di MP3

MP3 tiap frame berisi: Header (4B), opsional CRC, Side-Info, dan Main-Data (bit reservoir). Payload kita ditulis pada Main-Data → Part3 (Huffman spectral data), tepatnya pada bit tanda (sign bits) milik koefisien MDCT yang tidak nol. Membalik sign bit tidak mengubah panjang codeword Huffman, sehingga ukuran frame tetap (reservoir aman).

Konsep LSB di mp3 yang kami berbeda dari LSB pada PCM, melainkan LSB di sini adalah bit yang memiliki signifikansi kecil terhadap hasil audio di domain terkompresi, bukan strictly LSB seperti lsb pada sebuah integer. Bit yang memenuhi kriteria tersebut dan kami digunakan di sini adalah sign bit pada koefisien kecil di frame yang keras (berdasarkan global\_gain). Ini memenuhi multiple-LSB yang menekankan distorsi minimal pada komponen yang kurang terdengar.

### 2.2.2 Seleksi Kandidat & Masking

- Enumerasi kandidat per frame: semua sign bit koefisien non-zero yang terdekripsi dari Part3 (big values dan count1).
- Masking (perceptual ringan): gunakan global\_gain dari Side-Info sebagai proksi kekerasan/sensitivitas. Hanya frame dengan  $\text{global\_gain} \geq \text{P60}$  ( $\text{mask\_percentile}=0.60$ ) yang dipakai.
- Batas per frame:  $\text{bits\_per\_frame} \in \{1,2,3,4\}$  menentukan maksimal sign bit yang dipakai per frame (analog “n-LSB”).

### 2.2.3 Penjadwalan Posisi (Pseudo-acak, Deterministik)

- Key → PRNG seed: *key* (kata kunci ≤25 karakter) dijadikan seed PRNG untuk men-*shuffle* daftar kandidat di setiap frame (opsional menggabungkan indeks frame ke seed).
- Ambil berurutan  $\leq$  bits\_per\_frame posisi dari daftar yang sudah di-*shuffle*.
- fraction=1.0 → semua kandidat yang lolos masking dipertimbangkan sebelum dipotong oleh batas per frame.
- Header 16 byte (lihat §2.3) dikirimkan terlebih dulu melalui urutan posisi global (frame ke frame), lalu diikuti payload.
- Deterministik → ekstraksi merekonstruksi urutan posisi yang sama dari parameter yang sama (key, bpf, masking).

## 2.3 Format Header & Enkripsi

- Header 16 byte:  
 $\text{magic}[4] = \text{"MP3S"} | \text{size}[4] (\text{uint32, byte}) | \text{ext}[8] (\text{ASCII, } \leq 8, \text{ NUL-pad})$   
Ekstensi (misal bin, png, dst.) disimpan agar fitur *save as* langsung mengetahui *extension file*
- Enkripsi (opsional): Extended Vigenère 256 (repeating-key XOR pada byte payload).  
Key yang sama dipakai untuk seed PRNG dan Vigenère.

## 2.4 Perhitungan Kapasitas

Fungsi `estimate_capacity` menghitung jumlah bit kandidat yang tersedia untuk parameter tertentu:

- jalankan frame → masking global\_gain  $\geq P60$  → enumerasi sign bit non-zero → ambil  $\leq$  bits\_per\_frame per frame.
- capacity\_bits → capacity\_bytes =  $\text{floor}(\text{bits}/8)$ .  
usable\_payload\_bytes = capacity\_bytes - 16 (dikurangi header).

Backend `/api/capacity` menerima MP3 + bits\_per\_frame (+opsional payload\_size) dan mengembalikan metrik kapasitas serta fits bila ukuran payload diberikan terlebih dahulu.

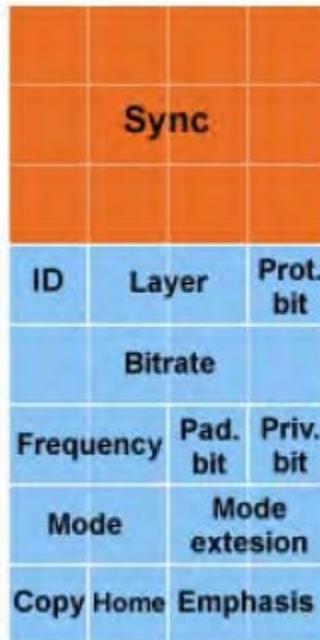
## 2.5 Perhitungan PSNR

Untuk menilai kualitas, backend menghitung PSNR antara audio sebelum dan sesudah penyisipan:

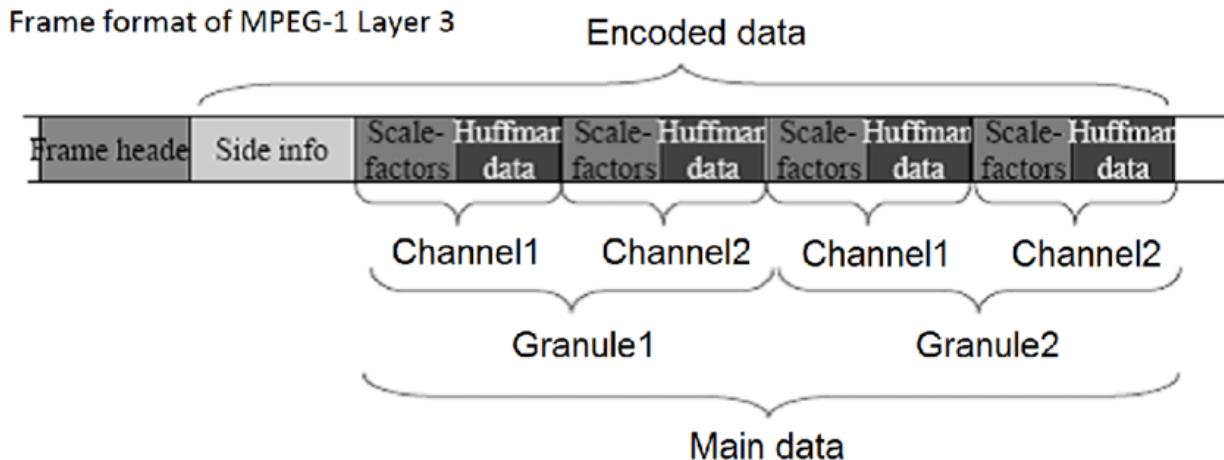
1. Decode kedua MP3 ke PCM *float*  $[-1,1][-1,1][-1,1]$  (via ffmpeg/pydub), mono opsional, align = min-length.
2. Hitung MSE, lalu hitung PSNR dengan rumus berikut

$$PSNR = 10 \cdot \log_{10} \left( \frac{MAX^2}{MSE} \right)$$

## 2.6 Parsing Header dan Huffman Bits pada MP3



Pada setiap frame MP3, proses dimulai dari header 32-bit di awal frame untuk mendapatkan *metadata* yang menentukan tata letak bagian berikutnya. Dari header ini kita memvalidasi bit sinkronisasi, membaca versi MPEG dan layer (kita pada Layer III), serta mengekstrak indeks bitrate, laju sampling, bit padding, dan mode *channel*. Nilai-nilai tersebut dipetakan melalui tabel standard sehingga panjang frame (dalam byte) dan ukuran blok side-info dapat dihitung secara deterministik; jika bit proteksi menunjukkan adanya CRC, parser melewati 16 bit CRC setelah header. Dengan informasi ini penunjuk baca dapat diposisikan tepat ke awal main-data (payload terkompresi) untuk pemrosesan berikutnya.



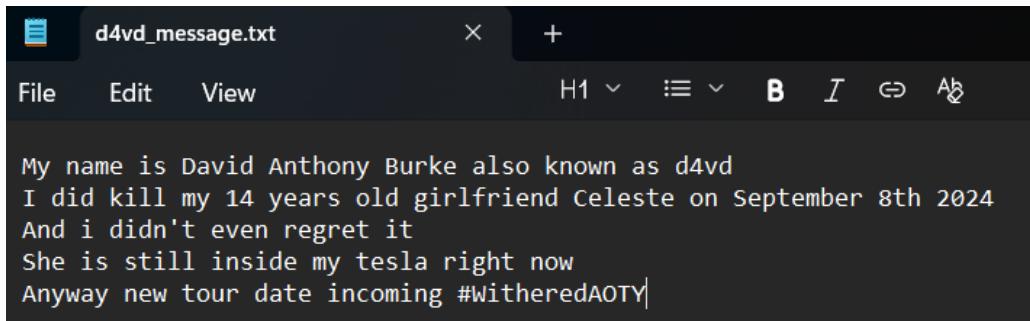
Sesudah header, parser membaca side-info untuk memperoleh parameter per granule×channel, khususnya main\_data\_begin yang merealisasikan bit reservoir. Karena main-data suatu frame dapat dimulai pada frame sebelumnya, implementasi menyusun jendela bit kontinu berdasarkan main\_data\_begin dan batas part2\_3\_length. Jendela inilah yang dipakai untuk membaca Part2 (scalefactors) lalu Part3 (Huffman spectral) meskipun data menyebar lintas frame. Dari side-info pula diambil global\_gain yang di tahap stego dimanfaatkan sebagai *hardness proxy* untuk masking.

Pada main-data, parser terlebih dahulu menghitung panjang Part2 (scalefactors) sesuai block mode dan parameter kompresinya agar pointer mendarat tepat di awal Part3 (Huffman spectral data). Fungsionalitas di Part3 bukan untuk merekonstruksi audio, tetapi hanya sebatas untuk menemukan posisi bit tanda (sign) pada koefisien MDCT non-nol—posisi inilah yang menjadi carrier saat embed. Implementasi menelusuri region big-values menggunakan tabel Huffman dari side-info (serta *linbits* untuk nilai besar), membaca pasangan magnitudo (x, y) dan untuk setiap magnitudo non-nol mengambil 1 bit sign yang disimpan terpisah dari codeword, lalu mencatat posisi bit absolut di dalam jendela. Setelah big-values, parser memasuki count1 region (kuartet 0/1) dan kembali mencatat sign bit untuk tiap magnitudo non-nol. Seluruh pembacaan dilakukan sesuai dengan part2\_3\_length sehingga tidak pernah melampaui windows yang valid.

# Bab 3: Pengujian program dan Analisis Hasil

## Testcase 1: D4vd

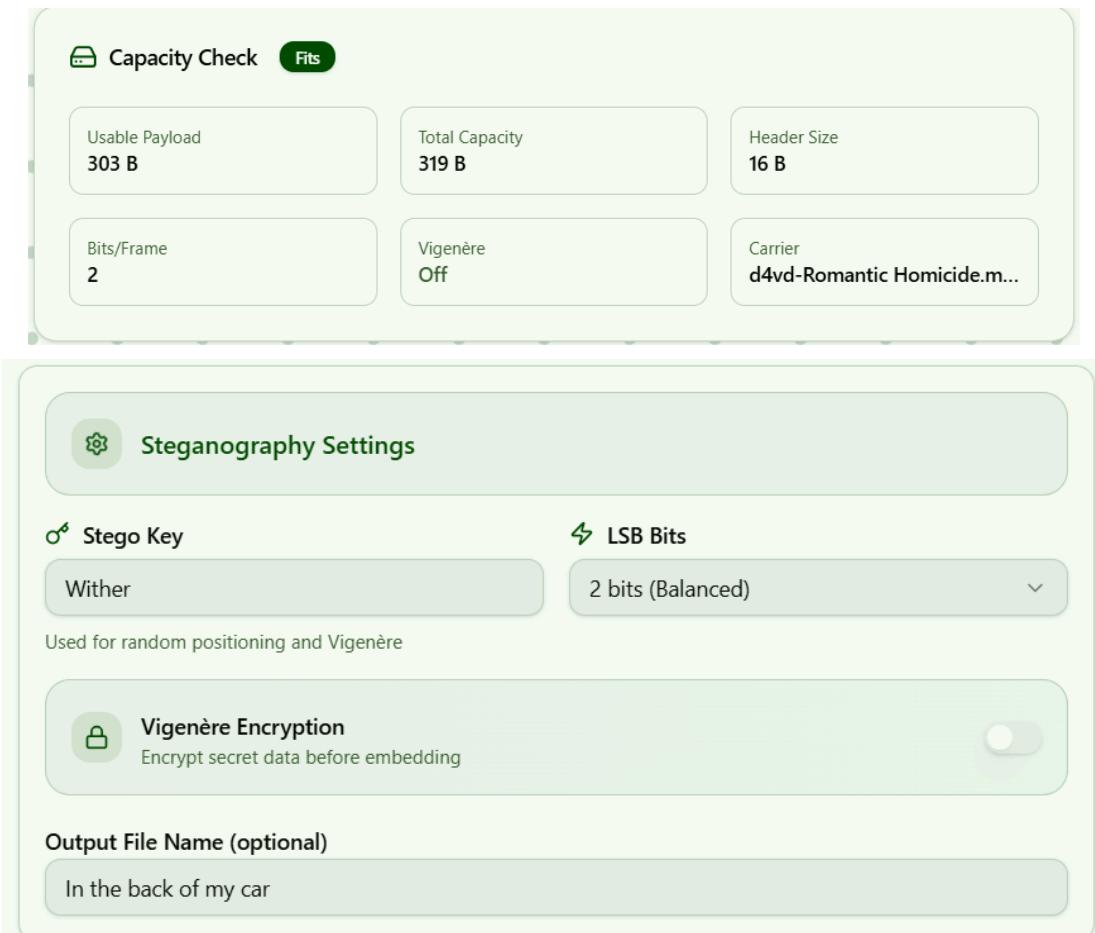
Pada kasus uji kali ini *cover object* yang digunakan adalah file mp3 lagu Romantic Homicide karya d4vd yang bergenre *Bedroom Pop*. Stego object yang digunakan berupa file txt berikut



The screenshot shows a text editor window titled "d4vd\_message.txt". The content of the file is as follows:

```
My name is David Anthony Burke also known as d4vd
I did kill my 14 years old girlfriend Celeste on September 8th 2024
And i didn't even regret it
She is still inside my tesla right now
Anyway new tour date incoming #WitheredAOTY
```

Embedding dilakukan tanpa menggunakan Cipher Vigenere, namun tetap digunakan kunci untuk seeding titik awal sisip, yaitu **Wither**. Pada kasus ini digunakan 2-LSB karena pengujian menunjukkan bahwa 1-LSB tidak memiliki kapasitas yang cukup.



The screenshot shows the D4vd software interface. The top section is titled "Capacity Check" and displays the following information:

Usable Payload 303 B	Total Capacity 319 B	Header Size 16 B
Bits/Frame 2	Vigenère Off	Carrier d4vd-Romantic Homicide.m...

The bottom section is titled "Steganography Settings" and includes the following settings:

- Stego Key:** Wither (Used for random positioning and Vigenère)
- LSB Bits:** 2 bits (Balanced)
- Vigenère Encryption:** Encrypt secret data before embedding (disabled)
- Output File Name (optional):** In the back of my car

Setelah sekian waktu, akan muncul hasil embedding yang dapat dimainkan. Nama file pun sudah sesuai dengan apa yang kami inginkan

Audio Player

In the back of my car

0:47 / 2:12

PSNR: 40.6 dB

Embedding Complete

Original Size 5.1 MB	Stego Size 5.1 MB	PSNR 40.6 dB
Bits/Frame 2	Vigenère Off	Output Name In the back of my car

Download Processed Audio

Download Stego Audio

Selanjutnya akan dicoba ekstraksi kembali pesan menggunakan *stego audio* yang baru saja dihasilkan. Setting yang digunakan sama seperti saat embedding

Stego Audio (MP3)

 In the back of my car.mp3  
5.07 MB X

 Extraction Settings

 Stego Key       LSB Bits  
Wither      2 bits (Balanced) ▼  
Must match the embedding settings.

 Vigenère Encryption  
For already Encrypted Secret Data using the same key for steganography

Extracted File Name (optional)  
d4vd\_message

Berikut adalah hasil ekstraksinya

 Extraction Complete

File Name d4vd_message.txt	File Size 0.2 KB	Type application/octet-stream
Bits/Frame 2	Vigenère Off	Extracted 10/4/2025, 9:42:41 PM

 Download Extracted File

d4vd\_message (1).txt X +

File Edit View H1 ☰ B I A

```
My name is David Anthony Burke also known as d4vd
I did kill my 14 years old girlfriend Celeste on September 8th 2024
And i didn't even regret it
She is still inside my tesla right now
Anyway new tour date incoming #WitheredAOTY
```

Hasil ekstraksinya sama persis seperti file txt stego object yang diujikan.

## Testcase 2: Quadeca

Kasus uji selanjutnya menggunakan *cover object* berupa file mp3 lagu GODSTAINED karya Quadeca yang bergenre *Bossa Nova/Experimental Hip Hop*. Stego object yang digunakan adalah suatu gambar dengan tipe png sebagai berikut



Embedding dilakukan dengan menggunakan Cipher Vigenere juga menggunakan kunci, yaitu **BULGARIA**. Pada kasus ini digunakan 4-LSB karena kapasitas dengan LSB lebih sedikit tidak ada yang cukup.

Carrier Audio (MP3)

QUADECA - GODSTAINED.mp3  
4.81 MB

Secret File

thundrrr.png  
967 Bytes

Steganography Settings

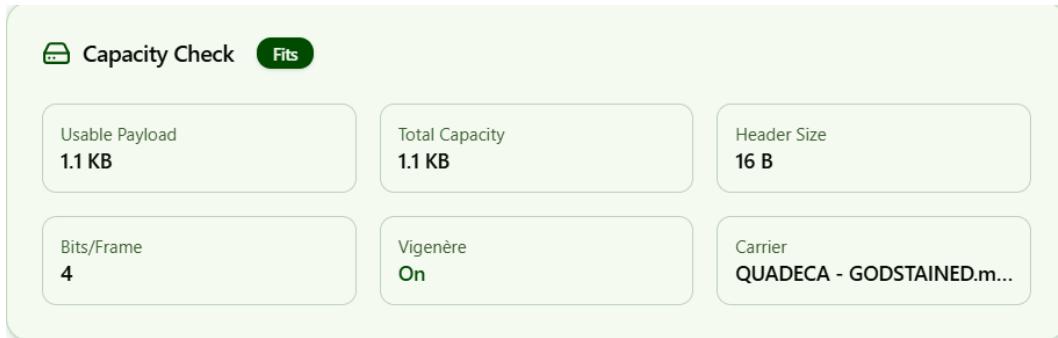
Stego Key: BULGARIA  
Used for random positioning and Vigenère

LSB Bits: 4 bits (Maximum capacity)

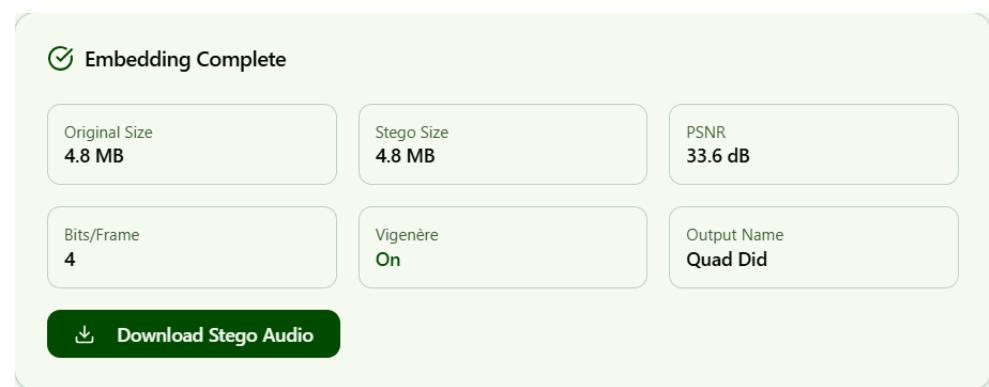
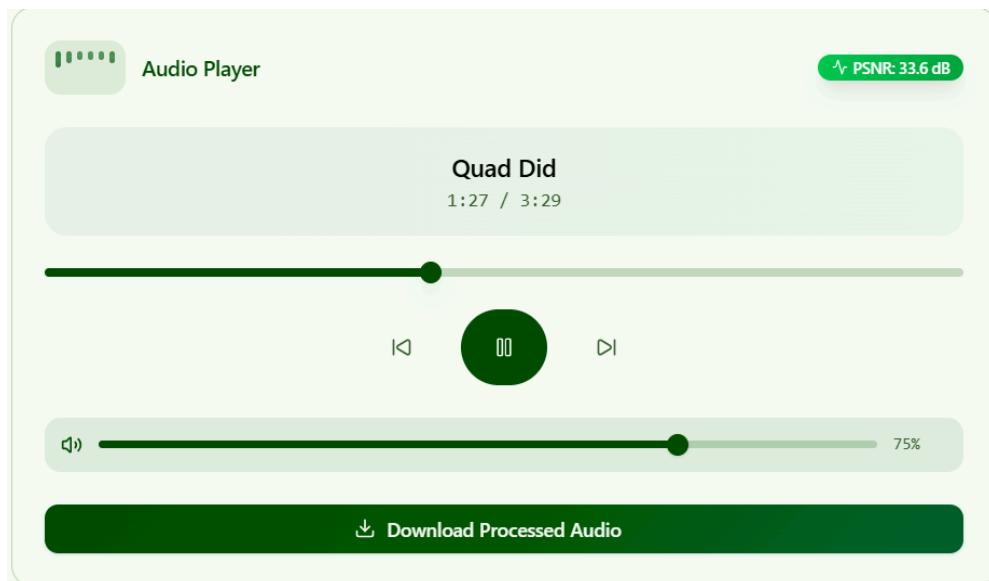
Vigenère Encryption: Encrypt secret data before embedding

Output File Name (optional): Quad Did

A screenshot of a software interface for performing steganography. It shows the 'Carrier Audio (MP3)' section containing 'QUADECA - GODSTAINED.mp3' (4.81 MB), the 'Secret File' section containing 'thundrrr.png' (967 Bytes), and the 'Steganography Settings' section. In the settings, 'Stego Key' is set to 'BULGARIA' and 'LSB Bits' is set to '4 bits (Maximum capacity)'. Below these, there is a note about using the key for random positioning and Vigenère. A 'Vigenère Encryption' option is available with a toggle switch that is turned on. At the bottom, there is an 'Output File Name (optional)' field containing 'Quad Did'.



Setelah sekian waktu, akan muncul hasil embedding yang dapat dimainkan. Nama file pun sudah sesuai dengan apa yang kami ingingkan. Statistik file hasil embedding seperti PSNR pun dapat dilihat.



Selanjutnya akan dicoba ekstraksi kembali pesan menggunakan *stego audio* yang baru saja dihasilkan. Setting yang digunakan sama seperti saat embedding, dimana dilakukan pula dekripsi Vigenère cipher dengan kunci BULGARIA

 Extract Secret Message

Recover a hidden file from a steganography embedded MP3

Stego Audio (MP3)

 Quad Did.mp3  
4.81 MB 

 Extraction Settings

 Stego Key  
BULGARIA

 LSB Bits  
4 bits (Maximum capacity)   
Must match the embedding settings.

 Vigenère Encryption  
For already Encrypted Secret Data using the same key for steganography 

Extracted File Name (optional)  
Quad Did Extraction

 Extraction Complete

File Name Quad Did Extraction.png	File Size 0.9 KB	Type application/octet-stream
Bits/Frame 4	Vigenère On	Extracted 10/4/2025, 10:07:26 PM

 Download Extracted File

Diperoleh file gambar png baru yang merupakan hasil ekstraksi. Berikut adalah gambar yang dihasilkan proses ekstraksi



Integritas pada kasus uji ini terbukti telah terjaga

### Testcase 3: Succession

Kasus uji ini menggunakan *cover object* berupa file mp3 yaitu lagu pembuka acara TV *Succession* yang bergenre Classical/Hip Hop. Stego object yang digunakan adalah suatu file JSON dengan isi sebagai berikut

```
[{"deceased": {"name": "Lester", "age": 78, "status": "deceased", "employment": {"company": "Waystar", "years": 40}}, {"survived_by": {"spouse": {"name": "Maria", "relationship": "wife", "marriage_years": 15}}}, {"sentiment": "sad", "full_eulogy": "Hello. I'm here as a fellow human"}]
```

Embedding dilakukan tanpa menggunakan Cipher Vigenere juga tanpa menggunakan kunci. Penyisipan tetap dilakukan secara acak namun tidak menggunakan seeding yang ditentukan pengguna. Pada kasus ini digunakan 1-LSB saja karena kapasitas sangat mencukupi untuk mensisipi file JSON ini.

### Carrier Audio (MP3)



Succession Theme Extended.mp3  
42.83 MB



### Secret File



connor eulogy.json  
754 Bytes



### Steganography Settings

#### ♂ Stego Key

Optional Secret Key

#### ⚡ LSB Bits

1 bit (Highest quality)



Used for random positioning and Vigenère



#### Vigenère Encryption

Encrypt secret data before embedding



#### Output File Name (optional)

Uncle Mo

#### 📄 Capacity Check

Fits

Usable Payload  
2.4 KB

Total Capacity  
2.4 KB

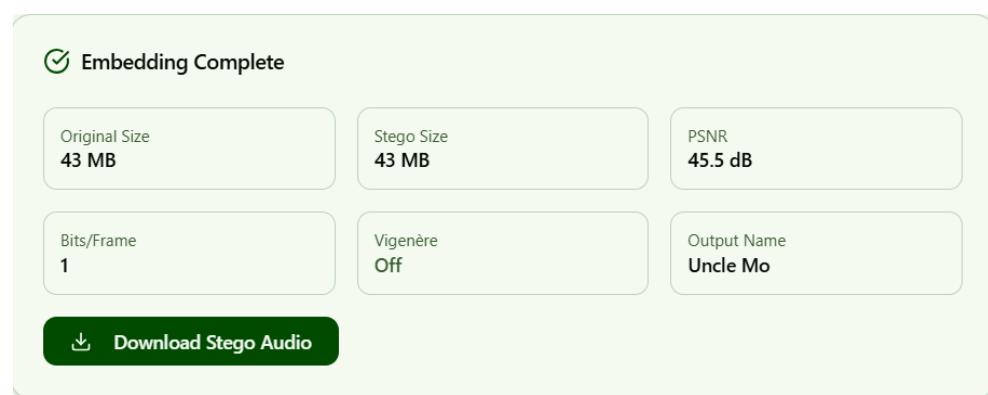
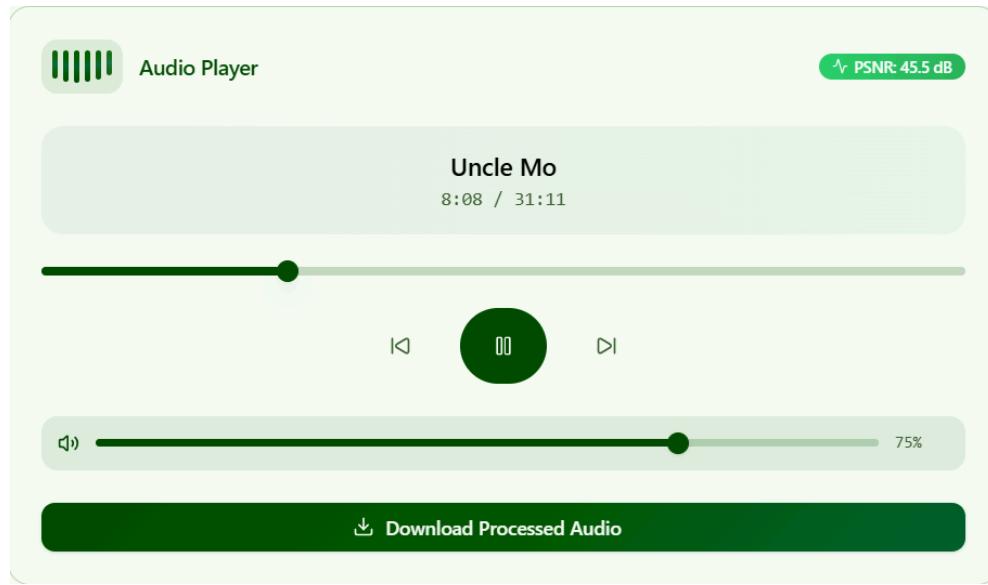
Header Size  
16 B

Bits/Frame  
1

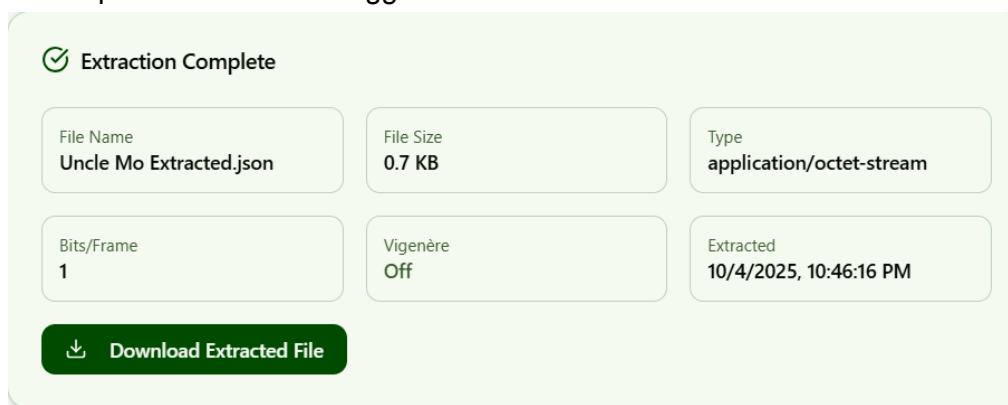
Vigenère  
Off

Carrier  
Succession Theme Extended...

Beberapa menit kemudian akan muncul hasil embedding yang dapat dimainkan. Nama file pun sudah sesuai dengan apa yang kami ingingkan, yaitu Uncle Mo.



Selanjutnya akan dicoba ekstraksi kembali pesan menggunakan *stego audio* yang baru saja dihasilkan. Setting yang digunakan sama seperti saat embedding, dimana tidak dilakukan dekripsi Vigenère cipher serta tidak menggunakan kunci sama sekali.



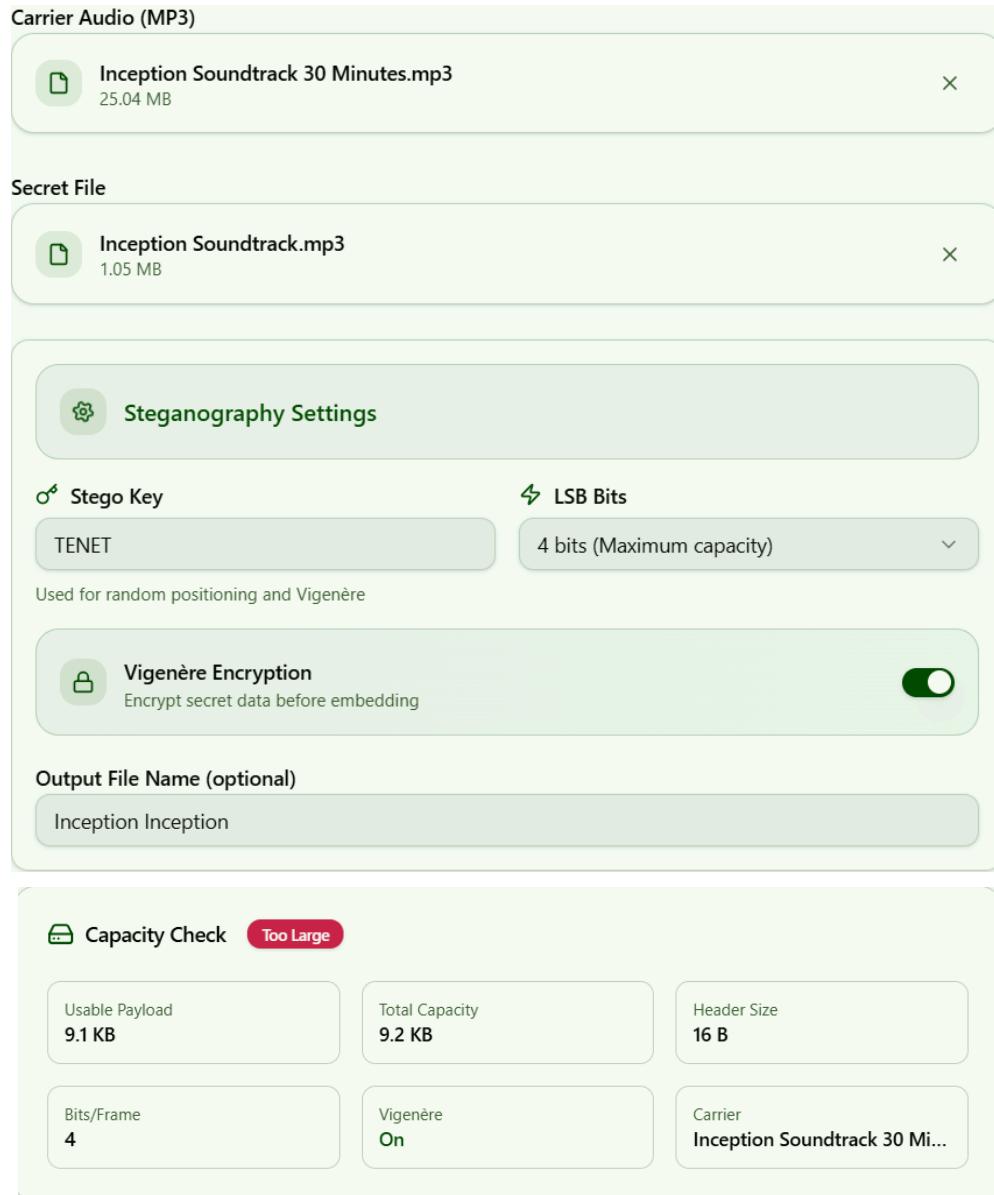
Dan ini lah file JSON yang dihasilkan oleh proses ekstraksi

```
{ Uncle Mo Extracted.json X
C: > Users > Farhan Nafis Rayhan > Downloads > Uncle Mo Extracted.json > .
1  {
2    "deceased": {
3      "name": "Lester",
4      "age": 78,
5      "status": "deceased",
6      "employment": {
7        "company": "Waystar",
8        "years": 40
9      }
10     },
11    "survived_by": {
12      "spouse": {
13        "name": "Maria",
14        "relationship": "wife",
15        "marriage_years": 15
16      }
17    },
18    "sentiment": "sad",
19    "full_eulogy": "Hello. I'm here as a fellow human
20 }
```

Hasilnya sesuai dengan file JSON sebelumnya

## Testcase 4: Inception

Pada kasus uji terakhir ini *cover object* yang digunakan adalah file mp3 lagu soundtrack film Inception (2011) yang bergenre *classical*. File mp3 ini cukup besar karena lagu diulang berkali-kali hingga berdurasi 30 menit. Sedangkan stego object yang digunakan berupa file mp3 lagu soundtrack film inception juga, tetapi tidak diulang. Maka pada kasus ini kami mencoba *embedding* suatu lagu kedalam lagu yang sama.



Dapat dilihat bahwa embedding ini tidak mungkin dilakukan. Apabila menggunakan 4 LSB pun payload yang dapat digunakan hanya sebesar 9.1 Kilobyte. Padahal, *stego object* memiliki ukuran sebesar 1,05 Megabyte. Artinya, kapasitas pada *cover object* tidak cukup untuk disisipi oleh *stego object* ini. Akibatnya, aplikasi menolak untuk melakukan penyisipan ini.

## Analisis Testcase

TC	Ukuran Cover Object	Ukuran Stego Object	Bit LSB	Usable Payload	PSNR	Stego Result Size	Extract Result Size
1	5.1 Mb	1 Kb	2	303 b	40.6 dB	5.1 Mb	0.2 Kb
2	4.81 Mb	987 b	4	1.1 Kb	33.6 dB	4.81 Mb	0.9 Kb
3	42.38 Mb	784 b	1	2.4 Kb	45.5 dB	43 Mb	0.7 Kb

Berdasarkan hasil pengujian, imperceptibility (PSNR) dari implementasi aplikasi audio steganography MP3 yang kami buat sudah sangat baik. Stego object berhasil disisipkan kedalam file MP3 tanpa mudah disadari ketika audio didengar. Walaupun masih terdapat sedikit kerusakan audio, secara garis besar pesan berhasil disembunyikan. Akan tetapi, terdapat masalah kritis pada fungsi utama ekstraksi data. File yang diekstrak selalu tidak lengkap, sehingga menunjukkan adanya bug serius pada logika penyisipan maupun ekstraksi data.

Dari sisi integritas data, seluruh uji coba gagal menghasilkan pesan rahasia secara utuh. Ukuran file hasil ekstraksi tidak sesuai dengan ukuran file asli yang disisipkan. Misalnya, pada TC1 disisipkan 1 Kb tetapi hanya berhasil diekstrak 0,2 Kb . Pada TC2 disisipkan 987 byte, namun hanya berhasil diekstrak sebesar 0,9 Kb . Sementara pada TC3 disisipkan 784 byte, namun hasil ekstraksi mengecil menjadi 0,7 Kb. Pola kegagalan ini konsisten, sehingga kemungkinan penyebabnya ada pada cara penulisan atau pembacaan nilai ukuran payload pada header atau masalah pada generator posisi acak yang tidak sinkron antara proses embed dan extract.

Meskipun demikian, dari sisi imperceptibility performa aplikasi sangat baik. *Nilai Peak Signal-to-Noise Ratio* (PSNR) yang dihasilkan berada pada kisaran 40,6 dB, 33,6 dB, dan 45,5 dB, seluruhnya jauh di atas ambang batas 30 dB yang dianggap masih berkualitas baik. Hasil ini juga sesuai ekspektasi terhadap parameter Bit LSB, di mana PSNR tertinggi tercapai pada n-LSB=1 (45,5 dB) dan terendah pada n-LSB=4 (33,6 dB), karena semakin banyak bit yang digunakan maka distorsi semakin besar.

Selain itu, ada potensi bug pada perhitungan kapasitas penyisipan. Sebagai contoh, pada TC1 berhasil menyisipkan file rahasia sebesar 1 Kb, padahal kapasitas yang dihitung sebelumnya hanya 303 byte. Hal ini mengindikasikan perhitungan kapasitas yang tidak konsisten atau proses pengecekan kapasitas yang terlewati. Pada uji coba lainnya, hasil kapasitas terlihat lebih wajar.

Secara keseluruhan, dapat disimpulkan bahwa kinerja aplikasi belum berhasil. Walaupun kualitas audio terjaga dengan sangat baik, fungsi utama untuk mengekstrak data secara utuh gagal tercapai. Hal ini memang bisa dimaklumi sebab MP3 memiliki sifat *lossy*, dimana bit yang dianggap tidak penting oleh kompresi akan menghilang. Hal inilah yang menjadi tantangan selama pengembangan. Namun, kami percaya bahwa kami sudah menemukan metode terbaik untuk melakukan steganography pada file MP3.

## Bab 4: Kesimpulan

Berdasarkan hasil analisis, aplikasi audio steganography berbasis MP3 yang dikembangkan belum sepenuhnya berhasil. Dari sisi kualitas audio, hasilnya sangat baik dengan nilai PSNR tinggi sehingga suara tetap jernih dan tidak mudah dibedakan dari file asli. Namun, fungsi utama yaitu ekstraksi data secara utuh masih gagal karena file yang diambil selalu tidak lengkap. Hal ini dipengaruhi oleh sifat lossy pada MP3 yang membuang sebagian bit serta kemungkinan bug pada mekanisme header atau sinkronisasi posisi. Selain itu, terdapat ketidakkonsistenan pada perhitungan kapasitas penyisipan yang menunjukkan perlunya perbaikan pada proses validasi. Meski begitu, hasil uji membuktikan bahwa metode penyisipan sudah tepat karena tidak merusak kualitas audio. Tantangan utama ke depan adalah memastikan data dapat dipertahankan meski melalui proses kompresi lossy. Dengan perbaikan lebih lanjut, aplikasi ini berpotensi menjadi solusi efektif untuk steganography pada file MP3.

## Bab 5: Daftar Pustaka

Seady, A., Shay, T., & Zaid, L. mp3-steganography-lib (MP3-Steganography). GitHub. URL: <https://github.com/tomershay100/mp3-steganography-lib>. Diakses 1 Oktober 2025.

Munir, R. 08 – Steganografi (Bagian 1 - 3). Informatika STEI ITB. URL: <https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2025-2026/08-Steganografi-Bagian1-2025.pdf>. Diakses 29 September 2025.

Alarood, A. A., Alghamdi, A. M., Alzahrani, A. O., Alzahrani, A., & Alsolami, E. Audio Steganography Method Using Least Significant Bit (LSB) Encoding Technique. ResearchGate. URL: [https://www.researchgate.net/publication/369708559\\_Audio\\_Steganography\\_Method\\_Using\\_Least\\_Significant\\_Bit LSB\\_Encoding\\_Technique](https://www.researchgate.net/publication/369708559_Audio_Steganography_Method_Using_Least_Significant_Bit LSB_Encoding_Technique). Diakses 29 September 2025.

# Bab 6: Lampiran

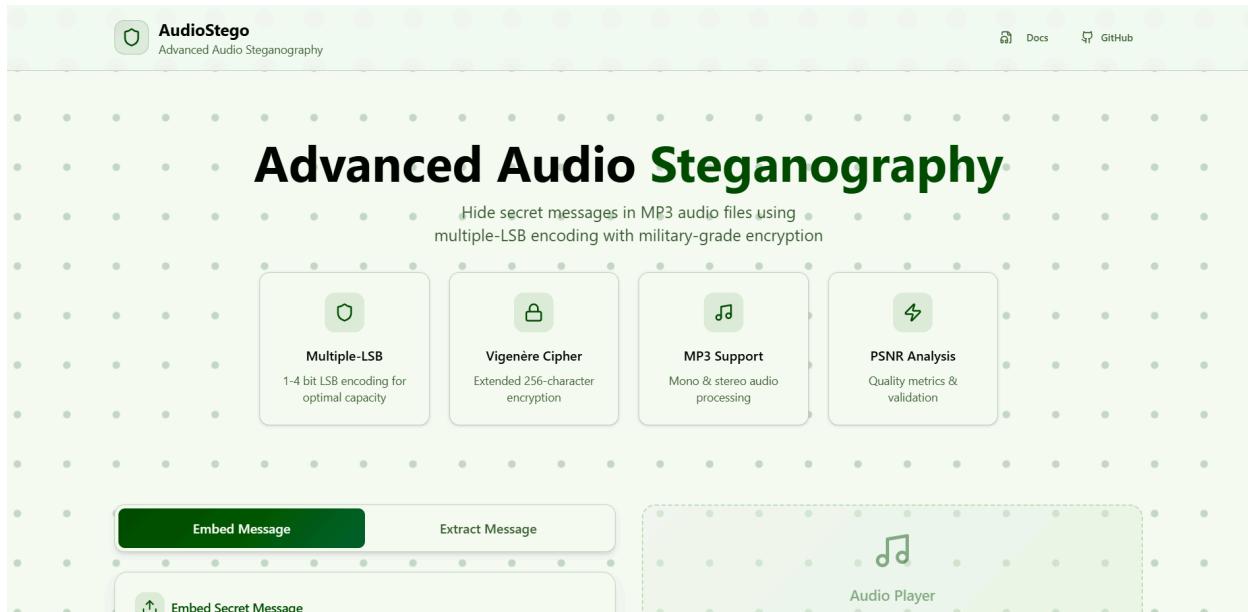
Repository Github: <https://github.com/abdulrafirh/Tucil2-IF4020>

Video Demo: <https://youtu.be/cT--6sSWRBk?si=3PWdtK7VjeiZnIVu>

## Pembagian Tugas

NIM	Nama	Tugas
13522037	Farhan Nafis Rayhan	Frontend, Documentation, and Researching for Steganography Methods
13522089	Abdul Rafi Radityo Hutomo	Backend, Steganography Implementation, Audio Library Bonus

## Halaman Utama Aplikasi



## Tampilan mode *Embed Message*

**Carrier Audio (MP3)**

QUADECA - GODSTAINED.mp3  
4.81 MB

**Secret File**

thundrrr.png  
967 Bytes

**Steganography Settings**

**Stego Key**: BULGARIA    **LSB Bits**: 4 bits (Maximum capacity)

Used for random positioning and Vigenère

**Vigenère Encryption**: On

**Output File Name (optional)**: Quad Did

**Quad Did**  
0:52 / 3:29

**Download Processed Audio**

## Tampilan mode Extract Message

Recover a hidden file from a steganography embedded MP3

**Stego Audio (MP3)**

Quad Did.mp3  
4.81 MB

**Extraction Settings**

**Stego Key**: BULGARIA    **LSB Bits**: 4 bits (Maximum capacity)

Must match the embedding settings.

**Vigenère Encryption**: On

For already Encrypted Secret Data using the same key for steganography

**Extracted File Name (optional)**: Quad Did Extraction

**Extract Secret Message**

**embedding**

**Extraction Complete**

**File Name**: Quad Did Extraction.png    **File Size**: 0.9 KB    **Type**: application/octet-stream

**Bits/Frame**: 4    **Vigenère**: On    **Extracted**: 10/4/2025, 10:07:26 PM

**Download Extracted File**