

# Steganografia applicata ai segnali audio

Luigi Priano 1000002081. Professori: Dario Allegra - Filippo Stanco.

Dipartimento: D.M.I. Informatica LM-18 A.A. 2023/2024

Data documento: 05/02/2024.

# Indice

<b>1 Introduzione alla steganografia</b>	<b>4</b>
1.1 Definizione di segnale . . . . .	4
1.2 Definizione di onda . . . . .	4
1.3 Definizione di audio . . . . .	4
1.3.1 Grafico Waveform (Forma d'onda) . . . . .	5
1.3.2 Spettrogramma . . . . .	5
1.4 Approfondimenti sulla steganografia . . . . .	6
1.4.1 LSB Coding (Least Significant Bit) . . . . .	7
1.5 Steganalisi . . . . .	7
1.6 Steganografia combinata alla crittografia . . . . .	8
<b>2 Introduzione all'elaborato</b>	<b>9</b>
<b>3 Sviluppo degli script - LSB Coding</b>	<b>9</b>
3.1 Steganography Hidden . . . . .	10
3.1.1 Text to Bits . . . . .	10
3.1.2 Modify bit . . . . .	10
3.1.3 Hide message . . . . .	11
3.2 Steganography Extract . . . . .	12
3.2.1 Extract bit . . . . .	12
3.2.2 Bits to Text . . . . .	13
3.2.3 Extract message . . . . .	13
3.3 Steganography Spectrogram . . . . .	14
3.3.1 Calculate spectrogram . . . . .	14
3.4 Steganography Detect . . . . .	15
3.4.1 Load audio . . . . .	15
3.4.2 Detect audio . . . . .	15
<b>4 Fase di test</b>	<b>16</b>
4.1 Nascondere il messaggio . . . . .	16
4.2 Verificare la presenza del messaggio . . . . .	17
4.3 Monitorare gli spettrogrammi . . . . .	18
<b>5 Steganalisi</b>	<b>19</b>
5.1 Uso dell'algoritmo Steganography Detect . . . . .	19

5.2 Steganalisi visuale . . . . .	20
5.2.1 Uso del grafico a forma d'onda (Waveform) . . . . .	20
5.2.2 Uso dello spettrogramma . . . . .	21
<b>6 Compressione del file wav in mp3</b>	<b>23</b>
6.1 Steganography Hidden Mp3 . . . . .	23
6.2 Steganography Extract Mp3 . . . . .	24
6.3 Steganography Spectrogram Mp3 . . . . .	24
6.4 Fase di test . . . . .	25
6.4.1 Nascondere il messaggio e comprimerlo . . . . .	25
6.4.2 Estrarre il messaggio dal file Mp3 . . . . .	25
6.4.3 Steganalisi tramite spettrogrammi . . . . .	26
<b>7 Steganografia LSB Multilivello</b>	<b>27</b>
7.1 Primo caso . . . . .	28
7.2 Secondo caso . . . . .	28
7.3 Terzo caso . . . . .	29
<b>8 Phase Coding</b>	<b>30</b>
8.1 Script utilizzati . . . . .	31
8.1.1 Encode . . . . .	31
8.1.2 Decode . . . . .	32
8.2 Fase di test . . . . .	32
8.3 Steganalisi visuale . . . . .	33
<b>9 Conclusione</b>	<b>35</b>
<b>10 Riferimenti</b>	<b>36</b>

# 1 Introduzione alla steganografia

La steganografia è una pratica utilizzata per garantire la sicurezza delle comunicazioni. A differenza della crittografia, che mira a rendere un messaggio incomprensibile a chi non è autorizzato, la steganografia si concentra sull'atto stesso della comunicazione, cercando di renderlo invisibile. Un metodo comune di steganografia coinvolge la nascita di informazioni all'interno di file immagine o audio. Questa tecnica si basa sulla premessa che le modifiche apportate ai bit meno significativi delle immagini o dei file audio siano praticamente impercettibili all'occhio umano e difficili da rilevare mediante analisi statistica, a causa del rumore di fondo intrinseco presente nei tipi di file utilizzati come contenitori. In questo progetto, esploreremo il mondo della steganografia applicata ai file audio.

## 1.1 Definizione di segnale

Un segnale è una variazione di una grandezza fisica rispetto a un'altra, spesso in funzione del tempo, che può essere il risultato di un processo sia aleatorio che deterministico. Nel caso di un processo deterministico, il segnale ha la capacità di trasportare informazioni.

## 1.2 Definizione di onda

Un'onda è una variazione o perturbazione di una grandezza fisica che si diffonde attraverso lo spazio e il tempo, trasportando con sé energia o quantità di moto. Le onde possono essere classificate in base alla loro forma, direzione di propagazione e al mezzo attraverso cui si diffondono. Inoltre, possono essere categorizzate come onde periodiche, in cui le oscillazioni si ripetono a intervalli regolari, oppure come onde non periodiche, in cui le oscillazioni non seguono un pattern prevedibile.

## 1.3 Definizione di audio

L'audio è una rappresentazione digitale dei segnali sonori, che può includere suoni, voci umane, musica e altri dati sonori. Principalmente l'audio è analogico quindi per trasformarlo in audio digitale abbiamo bisogno di seguire un processo preciso. Quest'ultimo coinvolge la conversione di un segnale sonoro continuo e analogico in una forma discrinizzata digitale che può essere elaborata e memorizzata dai computer e dai dispositivi audio digitali. I passi che vengono eseguiti durante la conversione sono tre:

- Campionamento: il segnale continuo viene campionato a intervalli regolari nel tempo per ottenere una serie di punti discreti noti come "campioni". Questi campioni rappresentano l'ampiezza del segnale audio in momenti specifici nel tempo. La frequenza con la quale vengono presi questi intervalli viene detta frequenza di campionamento;
- Quantizzazione: è il processo mediante il quale un segnale a valori continui viene trasformato in un segnale a valori discreti. In pratica, ogni valore del segnale viene mappato su un valore all'interno di un insieme discreto di livelli. Questo processo introduce inevitabilmente un certo errore, poiché valori originalmente diversi possono essere arrotondati allo stesso livello, diventando indistinguibili tra loro.

- Codifica: Dopo il campionamento e la quantizzazione, i campioni vengono codificati digitalmente in un formato specifico, come un file audio WAV o MP3.

I segnali audio digitali vengono spesso rappresentati graficamente in due modi principali: Grafico di forma d'onda e Spettrogramma.

### 1.3.1 Grafico Waveform (Forma d'onda)

Il grafico di forma d'onda, noto anche come grafico di forma del segnale o semplicemente forma d'onda, è una rappresentazione grafica di un segnale audio nel dominio del tempo. Questo tipo di grafico visualizza come varia l'ampiezza del segnale audio rispetto al tempo. Ecco alcuni dettagli importanti sul grafico di forma d'onda:

- Asse delle ordinate: L'asse verticale rappresenta l'ampiezza o l'intensità del segnale audio;
- Asse delle ascisse: L'asse orizzontale rappresenta il tempo;
- Linea tracciata: La linea tracciata all'interno del grafico stesso rappresenta il percorso che viene seguito dall'ampiezza del segnale audio nel tempo.

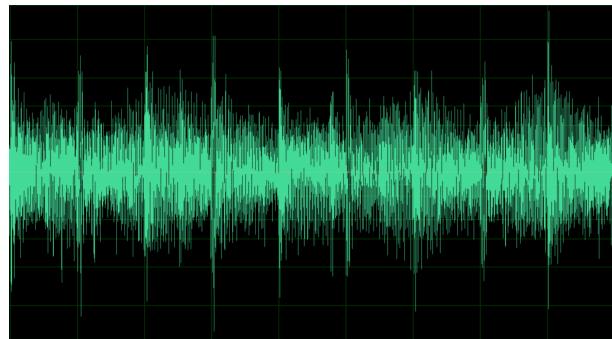


Fig. 1: Esempio di grafico Waveform

### 1.3.2 Spettrogramma

Lo spettrogramma, nell'ambito dell'analisi audio e della fonetica acustica, è la rappresentazione grafica dell'intensità di un suono in funzione del tempo e della frequenza. Uno spettrogramma è una rappresentazione visuale di un segnale audio che viene ottenuta suddividendo di solito l'intervallo di tempo totale (corrispondente all'intera forma d'onda da analizzare) in sottointervalli di durata uniforme, solitamente da 5 a 10 milisecondi ciascuno. In ciascuna di queste finestre temporali, si calcola la trasformata di Fourier del segmento di forma d'onda contenuto in quella finestra. Questa trasformata di Fourier fornisce informazioni sull'intensità delle frequenze presenti nel suono in funzione del tempo. Le diverse trasformate di Fourier, relative alle diverse finestre temporali, vengono poi assemblate per creare lo spettrogramma complessivo. In altre parole, lo spettrogramma mostra come le diverse frequenze contribuiscono al suono nel corso del tempo, consentendo di identificare variazioni nella frequenza e nell'intensità nel segnale audio.

Ecco cosa bisogna sapere per leggere uno spettrogramma:

- Asse delle ascisse: è riportato il tempo in scala lineare;
- Asse delle ordinate: è riportata la frequenza in scala lineare o logaritmica;
- A ciascun punto di data ascissa e data ordinata è assegnata una tonalità di grigio, o un colore, rappresentante l'intensità del suono in un dato istante di tempo e a una data frequenza; la relazione fra l'intensità del suono e la scala di grigi o di colori può essere lineare o logaritmica.

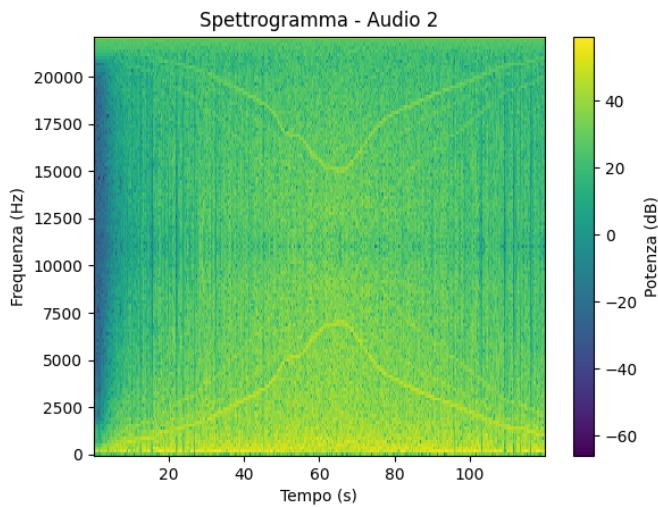


Fig. 2: Esempio di spettrogramma

## 1.4 Approfondimenti sulla steganografia

Come abbiamo già accennato la steganografia applicata agli audio è una branca della steganografia che si occupa di nascondere informazioni all'interno di file audio. Questo processo consente di trasmettere messaggi in modo occulti, sfruttando le proprietà dei file audio per mascherare la presenza di dati nascosti. La steganografia audio può essere utilizzata per vari scopi, tra cui la protezione della proprietà intellettuale, la comunicazione sicura e il watermarking digitale. Questo metodo di nascondere messaggi si basa su tecniche che modificano leggermente il segnale audio in modo che le modifiche siano impercettibili all'ascolto umano, ma possano essere rilevate e estratte con l'uso di algoritmi specifici. Questi metodi sfruttano la limitata percezione umana dei suoni, in particolare la capacità di occultare suoni deboli (messaggi da nascondere) sotto i suoni forti. Le tecniche di steganografia audio sono diverse, in questo progetto useremo la tecnica chiamata "LSB Coding (Least Significant Bit)". Nella steganografia è importante saper bilanciare la robustezza delle informazioni nascoste con la loro impercettibilità, inoltre, bisogna sapere che la quantità di dati che possono essere nascosti in un file audio è limitata dalla sua lunghezza e dalla qualità del segnale.

#### 1.4.1 LSB Coding (Least Significant Bit)

La tecnica impiegata è concettualmente molto semplice, consiste nel sostituire i bit meno significativi (LSB least significant bit) dei file digitalizzati con i bit che costituiscono il messaggio segreto. Quello che succede quindi è che il file contenitore risultante, dopo un'iniezione steganografica, si presenta in tutto e per tutto simile all'originale, con differenze difficilmente percettibili e quindi, a meno di confronti approfonditi con il file originale (non effettuabili ad occhio nudo) è difficile dire se le eventuali perdite di qualità siano da imputare al rumore od alla presenza di un messaggio segreto.

I vantaggi principali sono:

- La semplicità del metodo in sè;
- La possibilità di nascondere una grande quantità di dati all'interno dell'audio;
- Bassa percezione da parte di un udito umano, le modifiche apportate sono difficilmente rilevabili all'ascolto.

Mentre gli svantaggi sono i seguenti:

- Vulnerabilità a manipolazioni: Operazioni come la compressione, il cropping o anche una semplice modifica del volume possono distruggere i dati nascosti.
- Bassa sicurezza: Senza l'uso di tecniche aggiuntive come la crittografia, i dati nascosti possono essere facilmente estratti da chiunque conosca e rilevi la tecnica utilizzata.
- Qualità audio: Sebbene l'impatto sulla qualità percepita dell'audio sia minimo, in alcuni casi le modifiche possono essere rilevate, soprattutto se vengono nascosti grandi volumi di dati.

Gli svantaggi possono essere risolti in diversi modi come: l'uso della crittografia, distribuire i bit modificati in modo più casuale oppure usare la tecnica di stenografia chiamata "Steganografia Adattiva" che modifica i bit meno significativi in base alle caratteristiche del segnale audio.

### 1.5 Steganalisi

La steganalisi è il campo di studio che si concentra sulla rivelazione e, talvolta, sull'estratto di informazioni nascoste attraverso l'uso della steganografia, senza necessariamente decifrarle. Questa disciplina gioca un ruolo critico nella sicurezza informatica e nella crittografia, agendo come controparte della steganografia. Mentre la steganografia mira a nascondere l'esistenza di un messaggio, la steganalisi cerca di rilevare tali messaggi nascosti, valutando se un file contiene o meno dati occultati. La steganalisi si basa su principi di analisi statistica, analisi del contenuto, e teoria dell'informazione. Può essere classificata in due categorie principali:

- Steganalisi specifica: Mirata a rilevare informazioni nascoste tramite tecniche specifiche di steganografia. Richiede la conoscenza del metodo steganografico utilizzato.

- Steganalisi universale o Cieca: Non si basa sulla conoscenza del metodo specifico di steganografia utilizzato per nascondere il messaggio. Si focalizza sull'analisi delle anomalie statistiche o delle incongruenze nei dati che potrebbero indicare la presenza di informazioni nascoste.

Esistono diversi tipi di analisi da applicare per la steganalisi, in questo progetto utiliziamo le due tipologie descritte di seguito:

- Analisi statistica: Questa tecnica si concentra sull'analisi delle distribuzioni statistiche dei dati di un file. Ad esempio, nella steganografia basata sul LSB, l'alterazione dei bit meno significativi può alterare la distribuzione statistica dei campioni in un file audio;
- Analisi del contenuto, delle anomalie o del rumore: Esamina le caratteristiche intrinseche del contenuto come le caratteristiche temporali e spettrali di un file audio. In pratica analizzando lo spettrogramma o altri valori del file audio possiamo intercettare la presenza di eventuali messaggi nascosti.

Lo sviluppo di tecniche di steganalisi è fondamentale non solo per combattere usi illeciti della steganografia ma anche per promuovere l'avanzamento della stessa steganografia.

## 1.6 Steganografia combinata alla crittografia

La combinazione di steganografia e crittografia rappresenta un potente approccio per rafforzare la sicurezza e la privacy delle comunicazioni digitali. Mentre la crittografia si concentra sulla trasformazione dei dati in una forma incomprensibile (cifrata) senza la chiave appropriata, la steganografia nasconde il fatto che si sta svolgendo una comunicazione. I passi da eseguire sono quattro:

1. Cifratura dei dati: I dati da trasmettere vengono prima cifrati utilizzando un algoritmo di crittografia;
2. Applicazione della steganografia: Il testo cifrato viene quindi nascosto all'interno di un file multimediale;
3. Trasmissione: Il file contenente il testo cifrato viene trasmesso al destinatario;
4. Estrazione e decifratura: Il destinatario, conoscendo la tecnica steganografica utilizzata e possedendo la chiave di decifratura, può estrarre il testo cifrato e decifrarlo per leggerne il contenuto.

L'uso congiunto di queste due tecniche consente di ottenere un livello di sicurezza superiore rispetto all'impiego di ciascuna tecnica separatamente.

## 2 Introduzione all’elaborato

Lo scopo di questo elaborato è implementare un metodo per inserire un messaggio nascosto all’interno di un file audio, utilizzando tecniche di steganografia audio per occultare il testo nello spettrogramma del segnale. Si è reso necessario anche lo sviluppo di un meccanismo efficace per l’estrazione del testo nascosto. Per valutare la robustezza e la non rilevabilità della tecnica proposta, verranno applicate metodologie di steganalisi specifiche, al fine di determinare se il messaggio nascosto può essere facilmente individuato o meno. L’analisi dell’efficacia del metodo si basa su una serie di test, condotti considerando diverse variabili. Il progetto è strutturato in tre fasi principali:

1. Sviluppo degli script per l’inserimento e l’estrazione del messaggio, per la creazione di spettrogrammi e un algoritmo per verificare la robustezza tramite la steganalisi;
2. Fase di test, per valutare la performance dei metodi sviluppati, sotto l’aspetto del funzionamento e verificando l’impercettibilità del messaggio;
3. Analisi e interpretazione dei risultati, con un focus particolare sulla steganalisi per verificare l’effettiva capacità di nascondere il messaggio all’interno del file audio.
4. Verifica della compressione dell’audio e relativi risultati sul messaggio.

## 3 Sviluppo degli script - LSB Coding

In questa sezione verranno descritti gli script sviluppati per l’elaborato. Tutti i codici sono stati realizzati utilizzando il linguaggio di programmazione Python e sono accessibili, insieme ai risultati delle diverse sperimentazioni e analisi, tramite un repository su GitHub. Le librerie usate con le relative versioni sono le seguenti:

Libreria	Versione
Wave	3.11.5
Numpy	1.26.0
Pydub	0.25.1
Matplotlib	3.8.0
Scipy	1.12.0

### 3.1 Steganography Hidden

L'obiettivo dell'algoritmo Steganography Hidden è quello di accettare un file di testo, un file audio e un livello di bit specifico come input. Utilizzando tre funzioni principali, l'algoritmo inserirà il messaggio contenuto nel file di testo all'interno del file audio, più precisamente nel livello di bit specificato. Di seguito verranno illustrati i tre metodi dell'algoritmo.

#### 3.1.1 Text to Bits

La funzione text to bits prende una stringa di testo come input e la converte in una rappresentazione binaria sotto forma di stringa di bit. I passaggi che effettua sono i seguenti:

1. Converte la stringa di testo in una stringa di byte utilizzando l'encoding UTF-8;
2. Viene convertita la sequenza di byte in un intero, specificando "big" ovvero l'ordine dei byte (mettendo il byte più significativo prima);
3. La sequenza di byte ottenuta viene quindi convertita in una rappresentazione binaria. Infine, per garantire che la lunghezza sia di otto caratteri, aggiungiamo zeri a sinistra fino a raggiungere la lunghezza corretta. Alla fine del metodo restituisce questa sequenza modificata.

```
def text_to_bits(text):
    bits = bin(int.from_bytes(text.encode(), 'big'))[2:]
    return bits.zfill(8 * ((len(bits) + 7) // 8))
```

Fig. 3: Metodo Text to Bits

#### 3.1.2 Modify bit

Questa funzione è responsabile della modifica dei bit all'interno dei campioni audio. L'utente specifica la posizione del bit da modificare e il nuovo valore del bit desiderato. L'algoritmo quindi verifica se la posizione del bit è valida (compresa tra 0 e 15, in quanto si tratta di campioni audio di 16 bit) e crea una maschera per isolare il bit desiderato. Successivamente, il bit viene modificato nel campione audio in base al nuovo valore specificato. I passi eseguiti sono i seguenti:

1. Verifica se "bit\_position" è un valore valido, ovvero compreso tra 0 e 15. Se non è valido, viene sollevata un'eccezione con un messaggio di errore e ferma l'algoritmo;
2. Successivamente calcola una maschera (mask) con un bit impostato a 1 nella posizione specificata da bit\_position. Questa maschera viene utilizzata per isolare il bit desiderato nel campione audio;

3. Adesso bisogna verificare se il valore "bit" è pari a 0 o ad 1. Nel caso fosse pari a 0, la funzione imposta il bit nella posizione specificata a 0 nel campione audio utilizzando l'operatore di negazione bitwise e l'operatore and. Nel caso fosse pari a 1, la funzione imposta il bit nella posizione specificata a 1 nel campione audio utilizzando l'operatore OR. Per finire il metodo restituisce il campione audio modificato.

```
def modify_bit(audio_data, bit_position, bit):
    if bit_position < 0 or bit_position > 15:
        raise ValueError("Posizione del bit non valida. Deve essere compresa tra 0 e 15.")

    mask = 1 << bit_position
    if bit == 0:
        audio_data &= ~mask
    elif bit == 1:
        audio_data |= mask
    else:
        raise ValueError("Il bit deve essere 0 o 1.")

    return audio_data
```

Fig. 4: Metodo Modify bit

### 3.1.3 Hide message

Questo algoritmo svolge il processo di nascondere il testo da nascondere dentro la traccia audio selezionata. L'inizio di questo processo implica la traduzione del messaggio di testo in una forma binaria. Successivamente, il messaggio viene inserito nell'audio di destinazione in modo completamente impercettibile, con modifiche mirate ai bit specifici assegnati dallo sviluppatore. Alla conclusione di questa operazione, viene generato un nuovo file audio che contiene il messaggio occultato in modo sicuro e discreto. Il processo viene eseguito in questo modo:

1. La prima cosa è la lettura del file di testo che viene memorizzato nella variabile "message";
2. Successivamente viene convertito il "message" in bit tramite l'algoritmo descritto in precedenza;
3. A questo punto viene letto in forma binaria il file audio, di tipo wav. L'algoritmo estrae diverse informazioni utili durante le prossime fasi dell'algoritmo;
4. La fase successiva è quella di inserire i dati del file audio dentro un array di tipo NumPy a 16 bit, faremo un controllo se la frase è più lunga della dimensione dell'array se non lo è allora continuamo l'algoritmo;
5. Adesso tramite l'algoritmo Modify bit andremo a modificare i bit necessari per inserire il nostro testo. Una volta finito questo passaggio salveremo il risultato in un nuovo file wav.

```

def hide_message(audio_file, message_file, output_file, bit_position=0):
    with open(message_file, 'r') as f:
        message = f.read()
    bits = text_to_bits(message)

    with wave.open(audio_file, 'rb') as wav:
        framerate = wav.getframerate()
        nframes = wav.getnframes()
        channels = wav.getnchannels()
        sampwidth = wav.getsampwidth()
        audio_format = np.int16 if sampwidth == 2 else np.int8
        audio_data = np.frombuffer(wav.readframes(nframes), dtype=audio_format).copy()
    if len(bits) > len(audio_data) * 16:
        raise ValueError("Il messaggio è troppo lungo per essere nascosto nell'audio.")

    for i, bit in enumerate(bits):
        audio_data[i] = modify_bit(audio_data[i], bit_position, int(bit))

    with wave.open(output_file, 'w') as out_wav:
        out_wav.setparams((channels, sampwidth, framerate, nframes, 'NONE', 'not compressed'))
        out_wav.writeframes(audio_data.tobytes())

```

Fig. 5: Metodo Hide message

## 3.2 Steganography Extract

La funzione principale dell'algoritmo Steganography Extract consiste nel ricevere come input un file audio, specificare il livello di bit in cui è nascosto il messaggio e determinare il numero di bit da estrarre. Successivamente, l'algoritmo estrae questi bit specifici dal file audio e restituisce in output il messaggio nascosto. I metodi principali sono tre e vengono mostrati di seguito.

### 3.2.1 Extract bit

Questa funzione estrae un singolo bit per volta andandolo a reperire dalla traccia audio. Il funzionamento è abbastanza semplice ed è il seguente:

1. La prima istruzione verifica che il bit che bisogna analizzare è presente nel range tra 0 e 15, poichè gli algoritmi lavorano su campioni audio a 16 bit;
2. Successivamente viene creata una maschera che presenta una sequenza di 0 e un solo 1, esattamente nella posizione specificata dentro "bit\_position";
3. A questo punto bisogna estrapolare dall'audio preso in input, chiamato "sample", il bit richiesto applicando la maschera precedentemente creata. Quest'ultimo una volta trovato verrà restituito come risultato della funzione.

```

def extract_bit(sample, bit_position):
    if bit_position < 0 or bit_position > 15:
        raise ValueError("Posizione del bit non valida. Deve essere compresa tra 0 e 15.")

    mask = 1 << bit_position
    extracted_bit = (sample & mask) >> bit_position
    return extracted_bit

```

Fig. 6: Metodo Extract bit

### 3.2.2 Bits to Text

La funzione Bits to Text ha il compito di convertire una sequenza di bit in una stringa di testo. Possiamo vederlo come l'opposto del metodo Text to Bits descritto in precedenza. Il metodo esegue i seguenti passi:

1. Prende una sequenza di bit e la converte in intero, successivamente viene convertita da intero ad una sequenza di byte. Anche qui viene specificato l'ordinamento "big" ovvero il byte più significativo si trova all'inizio;
2. Utilizzando l'encoding UTF-8 l'algoritmo decodifica la sequenza di byte per ottenere una stringa di testo leggibile. In caso di byte corrotti, questi ultimi verranno scartati ma non bloccano l'algoritmo. Da questo si può dedurre che in output verrà sempre inviata una stringa di caratteri anche se all'interno dell'input del metodo erano presenti dei byte non conformi.

```
def bits_to_text(bits):
    n = int(bits, 2)
    byte_sequence = n.to_bytes((n.bit_length() + 7) // 8, 'big')
    try:
        return byte_sequence.decode('utf-8')
    except UnicodeDecodeError:
        return byte_sequence.decode('utf-8', errors='ignore')
```

Fig. 7: Metodo Bits to Text

### 3.2.3 Extract message

L'utilizzo della funzione Extract message, come dice la stessa traduzione, è quello di estrarre il messaggio dal file audio. In sintesi quello che fa l'algoritmo è prendere un file audio in input, converte i bit in caratteri leggibili e prende in base al livello di bit specificato il messaggio nascosto. I passaggi da effettuare sono i seguenti:

1. Il primo passo è quello di leggere il file audio, con estensione wav, sotto forma di lettura binaria. Subito dopo si calcolano i frame audio del file e la larghezza del campione audio, ovvero quanti byte vengono utilizzati per rappresentare ciascun campione audio;
2. Si memorizzano i dati dell'audio dentro un vettore NumPy (in questo caso a 16 bit);
3. Viene chiamato il metodo "Extract bit" per estrarre i bit relativi al messaggio nascosto e successivamente trasformiamo il testo da bit a caratteri facilmente leggibili;
4. Alla fine dell'algoritmo stampa il messaggio trovato.

```

def extract_message(audio_file, bit_length, lsb_position=0):
    with wave.open(audio_file, 'rb') as wav:
        nframes = wav.getnframes()
        sampwidth = wav.getsampwidth()
        audio_format = np.int16 if sampwidth == 2 else np.int8
        audio_data = np.frombuffer(wav.readframes(nframes), dtype=audio_format)

        bits = ''.join(str(extract_bit(audio_data[i], lsb_position)) for i in range(bit_length))

    message = bits_to_text(bits)
    return message

```

Fig. 8: Metodo Extract message

### 3.3 Steganography Spectrogram

L'algoritmo Steganography Spectrogram è progettato per calcolare e visualizzare gli spettrogrammi di segnali audio forniti in input. Questa funzione è utile per esaminare graficamente la distribuzione spettrale del segnale e identificare eventuali variazioni o modifiche al suo contenuto. Tutto questo avviene tramite il metodo "Calculate spectrogram".

#### 3.3.1 Calculate spectrogram

1. Come negli altri algoritmi il primo passo è leggere il file audio tramite una lettura binaria, durante la lettura reperiamo dati come: framerate, tutti i campioni audio e viene creato un array NumPy (16 bit);
2. Tramite la chiamata "spectrogram" viene calcolato lo spettrogramma del segnale audio;
3. Usando la libreria "matplotlib.pyplot" si inizia a creare il grafico da mostrare in output.

```

def calculate_spectrogram(audio_file, title):
    with wave.open(audio_file, 'rb') as wav:
        framerate = wav.getframerate()
        audio_data = wav.readframes(-1)
        audio_data = np.frombuffer(audio_data, dtype=np.int16)

    f, t, Sxx = spectrogram(audio_data, fs=framerate)

    plt.figure()
    plt.pcolormesh(t, f, 10 * np.log10(Sxx), shading='auto')
    plt.title(title)
    plt.xlabel('Tempo (s)')
    plt.ylabel('Frequenza (Hz)')
    plt.colorbar(label='Potenza (dB)')
    plt.show()

```

Fig. 9: Metodo Calculate spectrogram

## 3.4 Steganography Detect

Questo codice fornisce una semplice verifica per rilevare la presenza di steganografia in un file audio basandosi sulle variazioni nei campioni audio. Se la variazione supera una certa soglia, il file viene considerato sospetto. Il controllo viene effettuato tramite due funzioni di seguito descritte.

### 3.4.1 Load audio

Questa funzione apre un file audio specificato nei dati in input, successivamente legge tutti i frame audio e li converte in un array NumPy di tipo int16 (interi a 16 bit). Infine restituisce sia l'oggetto audio aperto che l'array dei dati audio.

```
def load_audio(file_path):
    audio = wave.open(file_path, 'rb')
    frames = audio.readframes(-1)
    signal = np.frombuffer(frames, dtype=np.int16)
    return audio, signal
```

Fig. 10: Metodo Load audio

### 3.4.2 Detect audio

Questa funzione calcola la possibilità di trovare un messaggio nascosto in un audio, tramite l'uso di una soglia detta "detection\_threshold".

1. In primo luogo usa l'algoritmo sopra descritto per aprire l'audio e generare il vettore di interi, successivamente calcola la differenza tra campioni audio consecutivi per identificare le variazioni nel segnale audio;
2. Viene calcolata una soglia basata sulla massima variazione nel segnale audio, questa soglia è impostata a metà della massima variazione rilevata. Successivamente viene contato il numero di campioni audio che superano la soglia di variazione, questi campioni sono considerati "sospetti" in quanto potrebbero contenere informazioni nascoste;
3. A questo punto bisogna verificare se la percentuale di campioni sospetti supera la soglia prestabilita all'inizio dell'algoritmo. Se questa condizione è vera siamo in presenza di un messaggio sospetto, se questa condizione è falsa allora non è stato rilevato alcun messaggio sospetto.

```

def detect_audio_steganography(file_path):
    audio, signal = load_audio(file_path)
    diff = np.diff(signal)
    threshold = np.max(np.abs(diff)) / 2
    suspicious_samples = np.sum(np.abs(diff) > threshold)
    detection_threshold = 0.035
    if suspicious_samples > detection_threshold * len(signal):
        print("Sospetta presenza di steganografia nell'audio.")
    else:
        print("Nessuna steganografia rilevata nell'audio.")

    audio.close()

```

Fig. 11: Metodo Detect audio steganography

## 4 Fase di test

Per condurre la fase di test, è stato utilizzato un set di dati comprendente due file audio, ciascuno della durata di un minuto, uno rappresentante il suono di un jet e l'altro il suono di un'orchestra. Il testo da inserire è presente in quattro file in formato .txt, questi documenti di testo hanno rispettivamente lunghezze di 1000, 10000, 100000 e 500000 caratteri. Durante i test, sono stati considerati tre diversi livelli di Least Significant Bit (LSB) per l'inserimento del messaggio all'interno dell'audio. I livelli selezionati sono 0, 7 e 15, rappresentanti il numero di bit meno significativi del segnale audio utilizzati per incorporare il messaggio. Questi test sono stati eseguiti al fine di valutare le prestazioni dell'algoritmo di steganografia in vari scenari, consentendo una completa analisi dell'efficacia e della rilevabilità del messaggio nascosto.

### 4.1 Nascondere il messaggio

La fase iniziale del processo ha coinvolto l'utilizzo dell'algoritmo Steganography Hidden per incorporare il messaggio all'interno dei file audio. Questa fase è stata suddivisa in diverse iterazioni, in cui ciascun file audio è stato trattato separatamente, nascondendo uno dopo l'altro i quattro file di testo in formato .txt. Inoltre, per ciascun file di testo, sono stati esaminati tutti e tre i livelli di bit precedentemente specificati nell'introduzione. Questo approccio ha comportato una serie di passaggi mirati in cui il messaggio è stato nascosto in modo sequenziale nei file audio utilizzando i diversi livelli di LSB. Tale metodologia è stata adottata al fine di esaminare attentamente il comportamento dell'algoritmo in varie configurazioni, fornendo una panoramica completa delle capacità di nascondere il messaggio in differenti contesti audio.

```
File testo: 1000.txt
Testo trasformato in bit
File audio: jetengine1.wav
Creato l'array np.int16
Audio modificato
Audio salvato con il nome: jetengine1000(15).wav
Tempo di esecuzione: 0.017817020416259766 secondi
```

Fig. 12: Test steganografia effettuato con 1000 caratteri

```
File testo: 500000.txt
Testo trasformato in bit
File audio: jetengine1.wav
Creato l'array np.int16
Audio modificato
Audio salvato con il nome: jetengine500000(15).wav
Tempo di esecuzione: 3.6367461681365967 secondi
```

Fig. 13: Test steganografia effettuato con 500000 caratteri

Come si può notare in questi due esempi, l'aumento della dimensione del testo comporta un incremento nella durata del tempo di esecuzione. Questa variazione è attribuibile alla presenza di un maggior numero di caratteri, pari a 499000 in questo esempio. L'aumento è dovuto perchè per ogni bit viene chiamata la funzione "modify bit" che ne aumenta il tempo di esecuzione totale. È importante notare che l'applicazione della steganografia tramite la tecnica dell'LSB a un file audio non comporta variazioni nelle dimensioni del file. Di conseguenza, anche l'aumento del numero di caratteri nel messaggio nascosto non determina cambiamenti nelle dimensioni del file audio, come visibile nell'esempio sottostante.

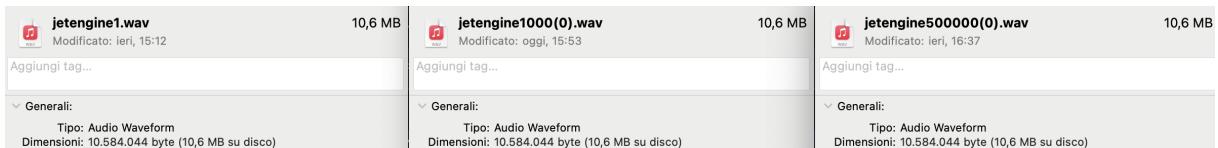


Fig. 14: A sinistra il file originale, al centro il file contenente 1000 caratteri, a destra il file contenente 500000 caratteri

## 4.2 Verificare la presenza del messaggio

Nella fase successiva dei test, è stata eseguita un'analisi dettagliata della presenza dei messaggi nascosti all'interno degli audio creati nella fase precedente. Questo processo ha coinvolto l'utilizzo dell'algoritmo Steganography Extract su ciascun file audio, esaminando attentamente i diversi livelli di Least Significant Bit (LSB), ovvero 0, 7 e 15. L'obiettivo principale di questa fase è quello di valutare l'individuabilità dei messaggi nascosti in base alla variazione dei livelli di LSB utilizzati per l'inserimento. Questa analisi ha consentito di comprenderne la robustezza e la rilevabilità in scenari diversificati. Una volta verificato che il messaggio è stato nascosto correttamente dalla fase precedente e questa fase è riuscita perfettamente a identificarlo, si può passare all'ultima fase.

```
Apertura file: jetengine1000(0).wav
Creazione array np.int16
Messaggio trovato in bit
Messaggio trasformato in testo
Tempo di esecuzione: 0.01200413703918457 secondi
```

Fig. 15: Test estrazione messaggio con 8000 bit

```
Apertura file: jetengine500000(0).wav
Creazione array np.int16
Messaggio trovato in bit
Messaggio trasformato in testo
Tempo di esecuzione: 3.4771101474761963 secondi
```

Fig. 16: Test estrazione messaggio con 4000000 bit

Nella fase attuale, come anche nella precedente, la dimensione del messaggio svolge un ruolo determinante. In questo caso, è la lunghezza in bit del messaggio a incidere sull'aumento del tempo di esecuzione, come chiaramente evidenziato nelle figure 15 e 16.

### 4.3 Monitorare gli spettrogrammi

Dopo aver completato con successo le prime due fasi dei test, durante le quali abbiamo confermato l'efficacia degli algoritmi Steganography Hidden ed Extract, siamo pronti per procedere alla terza fase di analisi. In questa fase, faremo uso dell'algoritmo Steganography Spectrogram per esaminare attentamente tutti gli spettrogrammi generati durante il processo di steganografia. L'obiettivo principale è quello di rilevare eventuali differenze significative tra i vari grafici spettrografici. Questa fase rappresenta un passo cruciale per valutare l'impatto della steganografia sull'aspetto visuale dello spettrogramma audio. Analizzeremo le differenze tra gli spettrogrammi risultanti da audio con e senza messaggi nascosti, al fine di comprendere come l'inserimento dei messaggi influenzzi la rappresentazione grafica dell'audio stesso. Questa analisi ci consentirà di valutare l'individuabilità dei messaggi nascosti anche da un punto di vista visuale. Di seguito inserisco gli spettrogrammi originali dei due audio presenti nel dataset, tramite la steganalisi andremo a studiare gli spettrogrammi presenti negli audio contenenti messaggi nascosti.

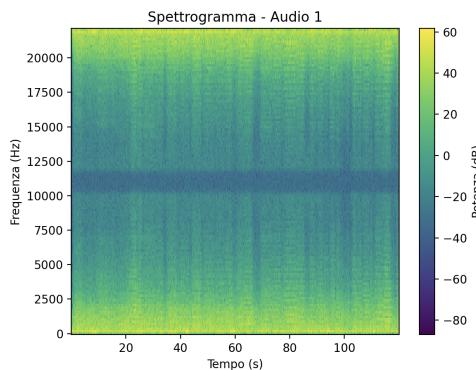


Fig. 17: Spettrogramma audio orchestra.wav

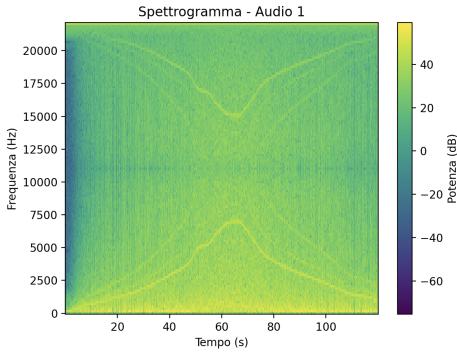


Fig. 18: Spettrogramma audio jetengine1.wav

## 5 Steganalisi

Dopo aver calcolato gli spettrogrammi nell'ultima fase di test, entriamo ora nell'ultima e fondamentale fase del processo: la steganalisi. In questa fase, faremo uso dell'algoritmo Steganography Detect per condurre un'analisi statistica mirata al fine di valutare la possibilità di rilevare la presenza di messaggi nascosti all'interno degli audio steganografati. In parallelo, utilizzeremo gli spettrogrammi precedentemente calcolati nella fase di test finale. Attraverso un'analisi dettagliata degli spettrogrammi, verificheremo attentamente le differenze significative tra gli audio con e senza messaggi nascosti. Questo ci consentirà di valutare visivamente se è presente un contenuto nascosto. Inoltre, nell'ambito dell'analisi manuale, adotteremo l'uso dei grafici a forma d'onda (waveform) per monitorare la presenza di messaggi nascosti. Questi due approcci ci forniranno una visione completa sulla possibile presenza di contenuti nascosti nell'audio.

### 5.1 Uso dell'algoritmo Steganography Detect

L'algoritmo Steganography Detect è progettato per rilevare la presenza di messaggi nascosti all'interno di file audio. Utilizzando una soglia preimpostata determinata dallo sviluppatore, l'algoritmo analizza l'audio per individuare eventuali anomalie di frequenza che potrebbero indicare la presenza di informazioni nascoste. Una volta individuata la presenza di un possibile messaggio nascosto, è possibile usare l'algoritmo Steganography Extract per tentare l'estrazione del messaggio. Durante la fase di test, è stata stabilita una soglia di rilevamento pari a "0.035". Utilizzando questa soglia, l'algoritmo è in grado di riconoscere la presenza di messaggi nascosti quando si trovano a partire dal livello di bit 12 in su all'interno dell'audio. Tuttavia, riducendo ulteriormente questa soglia, si è osservato che il segnale dell'audio originale stesso dava indicazioni errate sulla presenza di messaggi nascosti, anche quando questi non erano presenti. Di seguito è presente un output del segnale quando il messaggio nascosto non è stato rilevato e quando invece è stato trovato. È facilmente osservabile che all'aumentare del livello dei bit utilizzati (come nel caso sopra, dove il livello è di 7, mentre nel secondo esempio è di 12), si osserva un aumento dei campioni sospetti. Questo aumento fa superare la soglia precedentemente definita, rendendo quindi il campione sospetto.

```
luigipriano@Luigis-MacBook-Air Steganography % python3 SteganographyDetect.py
Audio usato: jetengine1000(7).wav
Calcolo il threshold medio 16383.5
Calcolo il numero di campioni sospetti pari a 182010
Calcolo usando la soglia scelta dal programmatore: 185220.00000000003
Nessuna steganografia rilevata nell'audio.
luigipriano@Luigis-MacBook-Air Steganography % python3 SteganographyDetect.py
Audio usato: jetengine1000(12).wav
Calcolo il threshold medio 16383.5
Calcolo il numero di campioni sospetti pari a 194906
Calcolo usando la soglia scelta dal programmatore: 185220.00000000003
Sospetta presenza di steganografia nell'audio.
```

Fig. 19: Due esecuzioni dell'algoritmo Steganography Detect

## 5.2 Steganalisi visuale

Per condurre l'analisi visuale nella steganalisi, si possono impiegare due tipi di rappresentazioni grafiche precedentemente menzionate: il grafico delle forme d'onda (waveform) e lo spettrogramma. Qui di seguito verrà spiegato come sono stati utilizzati i due approcci di analisi in modo differenziato.

### 5.2.1 Uso del grafico a forma d'onda (Waveform)

Per eseguire la steganalisi visuale utilizzando il grafico delle forme d'onda, è stato impiegato uno strumento specifico chiamato "Sonic Visualizer". Utilizzando questo software, è stato possibile recuperare il grafico corrispondente a ciascuna onda sonora. Successivamente, confrontando tra loro i risultati ottenuti, è stato possibile condurre un'analisi comparativa.

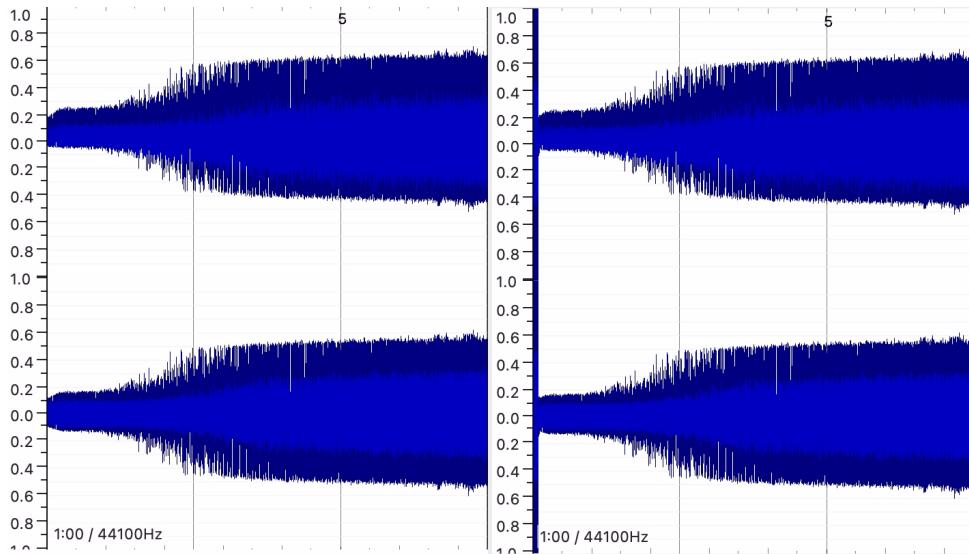


Fig. 20: A sinistra audio con messaggio 1000 caratteri nel livello 0, a destra audio con messaggio 1000 caratteri nel livello 15

Come evidenziato nella figura, quando viene nascosto un messaggio all'interno dell'audio utilizzando il livello di bit 0, risulta praticamente indistinguibile all'occhio umano e non solleva sospetti. Tuttavia, quando il messaggio viene incorporato nel bit più significativo, ossia il bit 15, si nota chiaramente una banda iniziale molto ampia che suscita sospetti

riguardo alla presenza di steganografia. Un altro esempio può facile da riconoscere è il rumore presente nell'audio con 500.000 caratteri al livello di bit 15, come mostrato nella figura sottostante. Inoltre avendo picchi così alti anche l'uditore percepisce il rumore del messaggio.

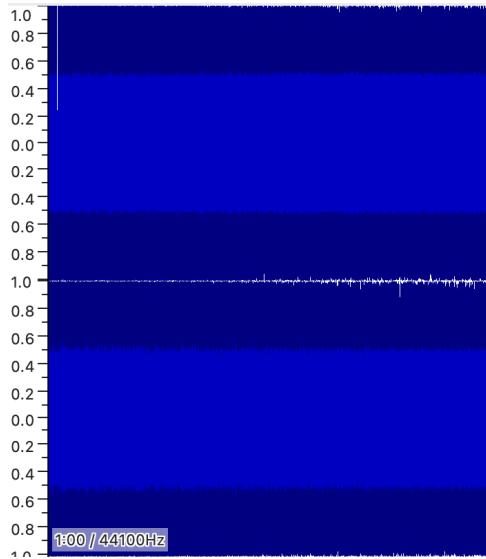


Fig. 21: Audio con 500.000 caratteri nel livello 15

Gli stessi risultati sono facilmente visibili anche nell'audio "Orchestra".

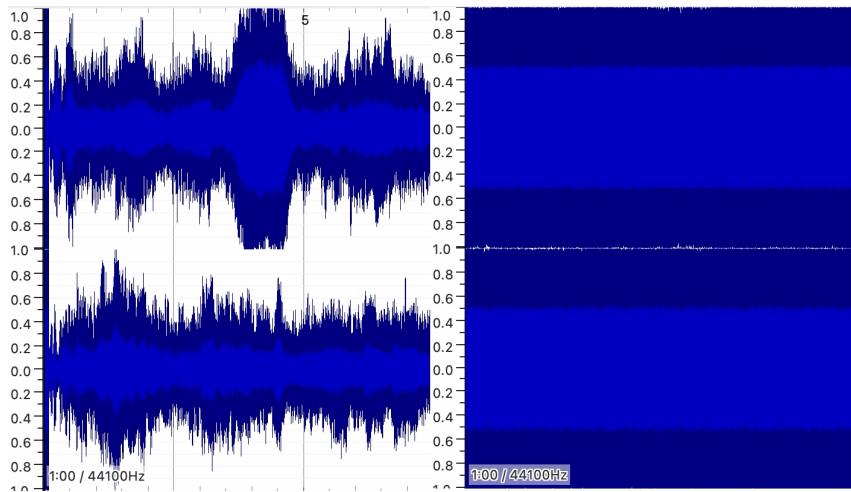


Fig. 22: Audio orchestra, a sinistra 1000 caratteri al livello di bit 15, a destra 500.000 caratteri al livello di bit 15

### 5.2.2 Uso dello spettrogramma

Un'altra metodologia per condurre la steganalisi visuale consiste nell'analizzare lo spettrogramma dell'audio. Attraverso l'utilizzo di un algoritmo specifico come lo "steganography spectrogram", è possibile calcolare gli spettrogrammi di vari file audio steganografati e valutarne le differenze. Questo approccio consente di individuare eventuali pattern o anomalie nei dati spettrali, facilitando l'identificazione di messaggi nascosti.

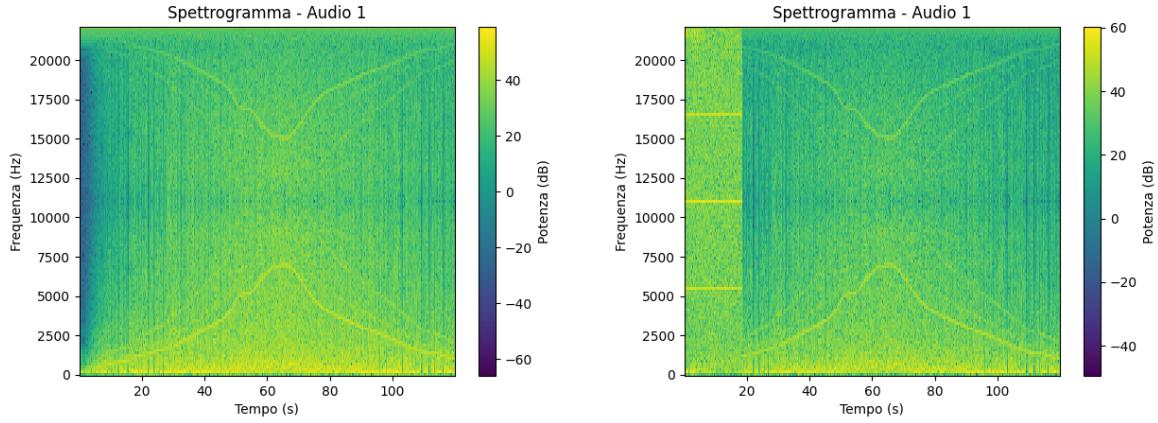


Fig. 23: Spettrogrammi dell'audio contenente 100.000 caratteri: a sinistra il livello di bit usato pari a 0, mentre a destra il livello di bit usato è 15

In quest'esempio, è facilmente osservabile che tutte le frequenze vengono uniformemente ridotte durante la presenza del messaggio quando questo viene incorporato nel bit più significativo. Questo fenomeno mette in evidenza un'alterazione più evidente dello spettrogramma, causata dalla presenza del messaggio nascosto nella componente audio di maggiore rilevanza. Al contrario, nel caso dell'utilizzo del livello di bit meno significativo, non sono osservabili anomalie evidenti. Un esempio ancora più marcato nella presenza di un messaggio è quando il quest'ultimo è quello di 500.000 caratteri come mostrato di seguito:

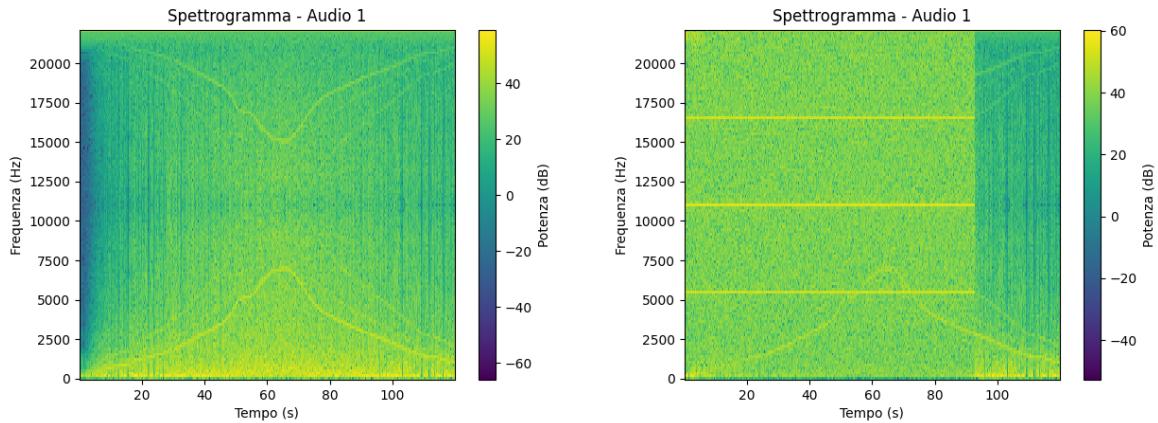


Fig. 24: Spettrogrammi dell'audio contenente 500.000 caratteri: a sinistra il livello di bit usato pari a 0, mentre a destra il livello di bit usato è 15

Nell'audio "orchestra" contenente il messaggio relativo a 500.000 caratteri, si osserva un comportamento simile a quello descritto nella figura precedente quando il messaggio è inserito nel bit 15. Ciò significa che anche in questo caso, l'effetto sulla rappresentazione dello spettrogramma è evidente e consistente con quanto osservato in precedenza. Questa uniformità di risultato sottolinea la ripetibilità del fenomeno e la sua consistenza rispetto al posizionamento del messaggio all'interno del livello di bit più significativo. Come visibile nell'immagine sottostante.

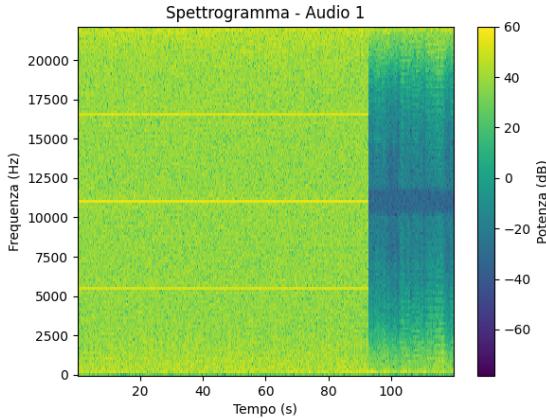


Fig. 25: Spettrogramma dell'audio orchestra contenente 500.000 caratteri al livello di bit 15

## 6 Compressione del file wav in mp3

In questa sezione del lavoro, ci concentriamo sulla domanda fondamentale: "Quali sono le conseguenze quando un audio contenente un messaggio nascosto viene sottoposto a compressione?". Al fine di rispondere a questa interrogativo, abbiamo sviluppato tre algoritmi distinti. Il primo algoritmo è stato progettato per eseguire la steganografia su un file audio in formato wav e successivamente applicare una compressione portandolo in formato mp3. Il secondo algoritmo è stato ideato per estrarre il messaggio nascosto da un file audio compresso in formato mp3. Infine, il terzo algoritmo ha lo scopo di generare uno spettrogramma da un file audio compresso per poter effettuare una steganalisi visuale degli audio risultanti. Questi algoritmi sono stati sviluppati per fornire una panoramica completa sulle implicazioni della compressione, sull'integrità e sulla rivelazione di messaggi nascosti all'interno di un file audio. Di seguito verranno esposte le modifiche applicate agli algoritmi precedentemente esposti.

### 6.1 Steganography Hidden Mp3

L'algoritmo è sostanzialmente identico al precedente, con l'unica differenza rappresentata dall'aggiunta della fase di compressione del file una volta applicato il messaggio al file audio in formato wav.

```
compress_audio(output_file, output_file.replace('.wav', '.mp3'))

def compress_audio(input_file, output_file):
    sound = AudioSegment.from_wav(input_file)
    sound.export(output_file, format="mp3", bitrate="128k")
```

Fig. 26: Metodo Compress audio

Semplicemente prende il file audio modificato in formato wav e lo comprime nello stesso file audio in Mp3, si può anche specificare il bitrate in base alle esigenze di compressione. Anche in questo caso si può specificare il livello di bit nella quale inserire il messaggio.

## 6.2 Steganography Extract Mp3

Anche questo algoritmo è simile al precedente ma adattato al nuovo formato Mp3. Questa modifica avviene nella chiamata al metodo "Extract message" ora chiamata "Extract message from mp3".

```
def extract_message_from_mp3(audio_file, bit_length, lsb_position=0):
    audio_segment = AudioSegment.from_mp3(audio_file)

    samples = np.array(audio_segment.get_array_of_samples())
    if audio_segment.channels == 2:
        samples = samples[:,::2]

    bits = ''.join(str(extract_bit(samples[i], lsb_position)) for i in range(bit_length))

    # Converti i bit estratti in testo
    message = bits_to_text(bits)
    return message
```

Fig. 27: Metodo Extract message from mp3

Per prima cosa prendiamo il file Mp3 in input ed estraie i campioni audio. Se il file dovesse essere stereo viene estratto un solo canale per semplificare l'estrazione del messaggio. Anche questa estrazione viene effettuata sul livello di bit specificato dal programmatore.

## 6.3 Steganography Spectrogram Mp3

Nel codice, l'unica modifica apportata riguarda la funzione utilizzata per la lettura dei file MP3, mentre il resto del processo rimane invariato, in sostanza, si tratta di un adattamento per permettere al codice di manipolare correttamente i file audio in formato MP3.

```
def calculate_spectrogram_mp3(audio_file, title):
    audio_segment = AudioSegment.from_mp3(audio_file)

    samples = np.array(audio_segment.get_array_of_samples())
    if audio_segment.channels == 2:
        samples = samples[:,::2]

    framerate = audio_segment.frame_rate

    f, t, Sxx = spectrogram(samples, fs=framerate)

    plt.figure()
    plt.pcolormesh(t, f, 10 * np.log10(Sxx), shading='auto')
    plt.title(title)
    plt.xlabel('Tempo (s)')
    plt.ylabel('Frequenza (Hz)')
    plt.colorbar(label='Potenza (dB)')
    plt.show()
```

Fig. 28: Metodo Calculate spectrogram mp3

## 6.4 Fase di test

Per la fase di test ho utilizzato il testo contenente 1000 caratteri da iniettare nel file audio "jetengine1".

### 6.4.1 Nascondere il messaggio e comprimerlo

Anche in questa situazione, utilizzando l'algoritmo "Steganography Hidden Mp3", si osserva un aumento del tempo di esecuzione dell'algoritmo in base al numero di caratteri inseriti. È importante notare che, nonostante l'aumento della dimensione del messaggio influenzi il tempo di elaborazione, la compressione finale del file audio non è direttamente correlata a questa variabile. Questo perché, come già detto in precedenza, durante il processo di steganografia LSB in un file WAV, la dimensione del file rimane costante, indipendentemente dalla quantità di dati nascosti. La compressione finale avviene solo dopo l'applicazione della steganografia, generando così il file MP3 compresso.

### 6.4.2 Estrarre il messaggio dal file Mp3

Durante questa fase, viene eseguita l'estrazione dei dati precedentemente incorporati all'interno del file audio in formato WAV, che è stato successivamente compresso in formato MP3. L'algoritmo utilizzato è "Steganography Extract Mp3" ed il risultato ottenuto è il seguente:

```
Messaggio estratto: oX鍏B'U
e0! '_#D>7J|ucb[%">M|yr''47 ]%&[3
ssEzi6('k(g?n("Wuzp>+>i{NJ66 &mb]:h=^u(2RE^if智X\LC%E{boI;uhwjlaEhP_XsJrV$o_ e ly
!x_
8jMM/wH4hhb4P!a~B?WYyNjW!5□` 繁Z8T*0>UZ
`+c5yDKEdLNEw3*?
ZTw<
0fzo:+2:I^DH8-!B;50a
o!mh8Dh..{mY6RlHuxSq
"}` 9T/KV%u%æ
y{#
Kô+e(Rf,TmX<]@AS·0/
{
}cZP&AY+T?{*0mZ
/l/
(h#i*y
ki .g ys2@OKIgu|XmAP
EPf/?Q-}z%W^x
R
```

Fig. 29: Estrazione messaggio nel livello di bit 0

Poiché la compressione agisce principalmente sui livelli di bit meno significativi, è evidente che il messaggio in output differirà significativamente da quello inserito in input. Lo stesso esempio lo possiamo osservare usando come livello di bit quello più significativo ovvero il 15, il risultato è il seguente:

```
owo` o~f>o7g60fw3g~vgx;83 0@v|2yey;sw7~sn>7?vo0`"@"/wwwv7{r7wgofop
$`$ng6gw?}0;;Cd7g>>o @A$ '71<ro'Óx1sw%#@'~s~ngg??3pfP@ ?o'3o6o2g>f>o?
v/|@0`'$~|`wa16?{$`" 0#w~wl&NwoOL@0 @@"f1?
```

Dai due risultati ottenuti, è evidente che il contenuto del messaggio viene compromesso e che non importa il livello di bit in cui viene inserito: verrà comunque alterato o distrutto durante il processo di compressione.

#### 6.4.3 Steganalisi tramite spettrogrammi

Per condurre l'analisi visiva steganalitica, facciamo uso dell'algoritmo denominato "Steganography Spectrogram Mp3", appositamente sviluppato per questa finalità e precedentemente descritto.

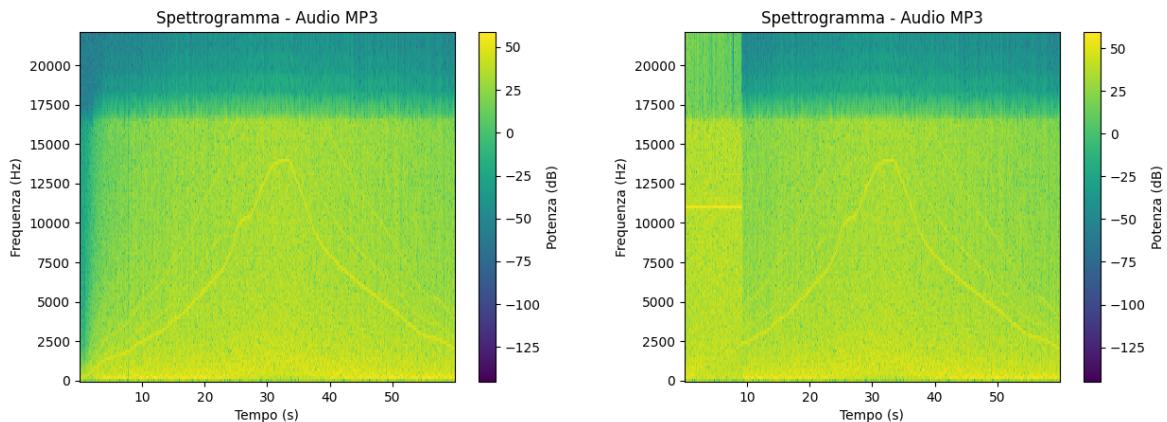


Fig. 30: Spettrogrammi audio mp3 contenente un messaggio di 100.000 caratteri: a sinistra nel livello di bit 0, a destra nel livello di bit 15

Come evidenziato nell'esempio precedente, è possibile notare innanzitutto la differenza tra uno spettrogramma di un file WAV e uno di un file MP3. Inoltre, osserviamo che le frequenze in cui è stato incorporato il messaggio sono chiaramente distinguibili nello spettrogramma quando il testo è stato inserito nel bit più significativo. Usando più caratteri, come nell'esempio sottostante, si nota ancora di più la presenza del messaggio usando il livello di bit pari a 15.

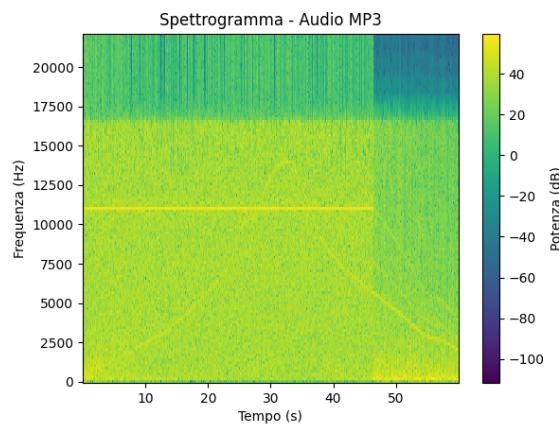


Fig. 31: Spettrogramma audio mp3 contenente un messaggio di 500.000 caratteri nel livello di bit 15

Utilizzando il secondo audio del dataset (Orchestra), osserviamo gli stessi risultati. Anche in questo caso, è evidente l'effetto della compressione sulle frequenze più elevate, e i messaggi risultano visibili nello spettrogramma solo quando sono inseriti nei livelli di bit più significativi, come visibile nella figura sottostante.

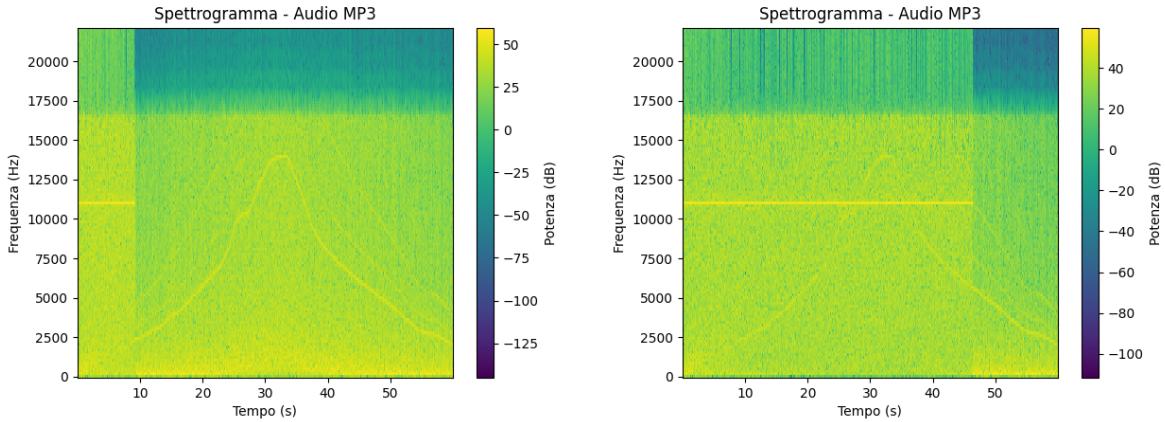


Fig. 32: Spettrogrammi "Orchestra" mp3 contenenti rispettivamente un messaggio di 100.000 e 500.000 caratteri, entrambi nel livello di bit 15

La steganalisi visuale tramite il grafico a forma d'onda (Waveform) evidenzia gli stessi cambiamenti riscontrati nell'analisi dei file WAV non compressi. Tuttavia, si osserva che la compressione influisce sulle frequenze più elevate, abbassandole. Nonostante ciò, i risultati complessivi rimangono sostanzialmente identici. Poiché non si evidenziano variazioni significative al di là di questo effetto, tali aspetti non sono oggetto di studio approfondito nel presente elaborato.

## 7 Steganografia LSB Multilivello

La tecnica predominante adottata per la steganografia nell'ambito di questo studio è stata quella del coding LSB (Least Significant Bit). Tale metodo, tradizionalmente associato alla manipolazione del bit meno significativo in una sequenza di dati, offre un'ampia flessibilità che va ben oltre l'applicazione a un singolo livello di bit. In effetti, la robustezza e l'efficacia della steganografia LSB risiedono nella sua capacità di essere adattata e applicata a diversi livelli di bit, senza limitarsi esclusivamente al livello meno significativo. Durante la fase sperimentale, ho condotto una serie di test per valutare l'efficienza e la discrezione del metodo LSB Coding, inserendo lo stesso messaggio segreto all'interno di vari livelli di bit dei dati ospitanti. Questo approccio sperimentale ha avuto l'obiettivo di esplorare in profondità il potenziale della tecnica LSB, valutando non solo la sua capacità di nascondere efficacemente l'informazione, ma anche di mantenere l'integrità del messaggio segreto attraverso differenti gradi di manipolazione. La fase di test per questa tecnica multilivello è stata suddivisa in tre casi:

- Il primo è il caso di un audio contenente due messaggi nei livelli di bit 0 e 1;
- Il secondo caso è un audio contenente due messaggi nei livelli medio-alto (12/15);
- L'ultimo caso comprende un messaggio su tutti e 16 i livelli dell'audio.

## 7.1 Primo caso

Il primo caso comprende due messaggi nascosti di 500.000 caratteri su due livelli di bit differenti, precisamente i più bassi ovvero 0 e 1.

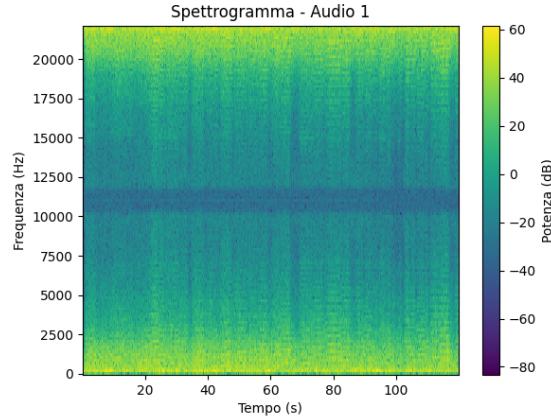


Fig. 33: Primo caso di LSB Multilivello

In questo caso entrambi i messaggi sono inseriti correttamente e non è visibile nulla nello spettrogramma dell'audio.

## 7.2 Secondo caso

All'interno del secondo caso la steganografia viene applicata al livello 12 e 15. Anche in questo inserimento il testo è pari a 500.000 caratteri.

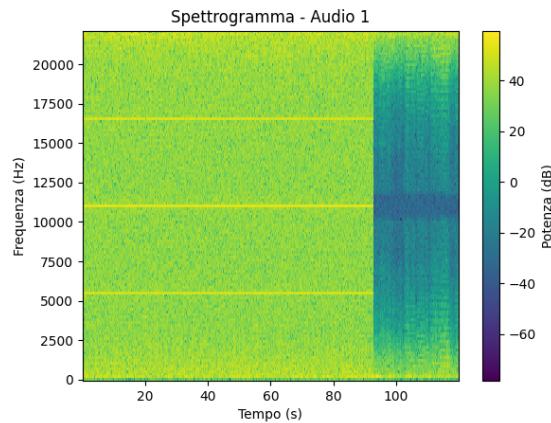


Fig. 34: Secondo caso di LSB Multilivello

Il messaggio applicato al livello di bit 15 sovrasta completamente quello al livello di bit 12; in questo caso, è evidente e facile osservare la presenza del primo messaggio nascosto. D'altra parte, risulta più difficile individuare il secondo messaggio, in quanto è coperto dal primo e si trova a un livello inferiore all'ultimo.

### 7.3 Terzo caso

Nel terzo e ultimo caso, ho applicato la tecnica multilivello a tutti i livelli di bit, creando uno spettrogramma per ogni passaggio, secondo lo schema seguente. È degno di nota che nell'ultimo passaggio, una volta riempiti tutti i livelli di bit disponibili, le frequenze diventano tutte uguali al contenuto del testo inserito. Di conseguenza, nello spettrogramma, il risultato si traduce in frequenze tutte dello stesso colore. Durante la riproduzione audio, l'udito percepisce silenzio fino a quando non viene completata la trasmissione del messaggio.

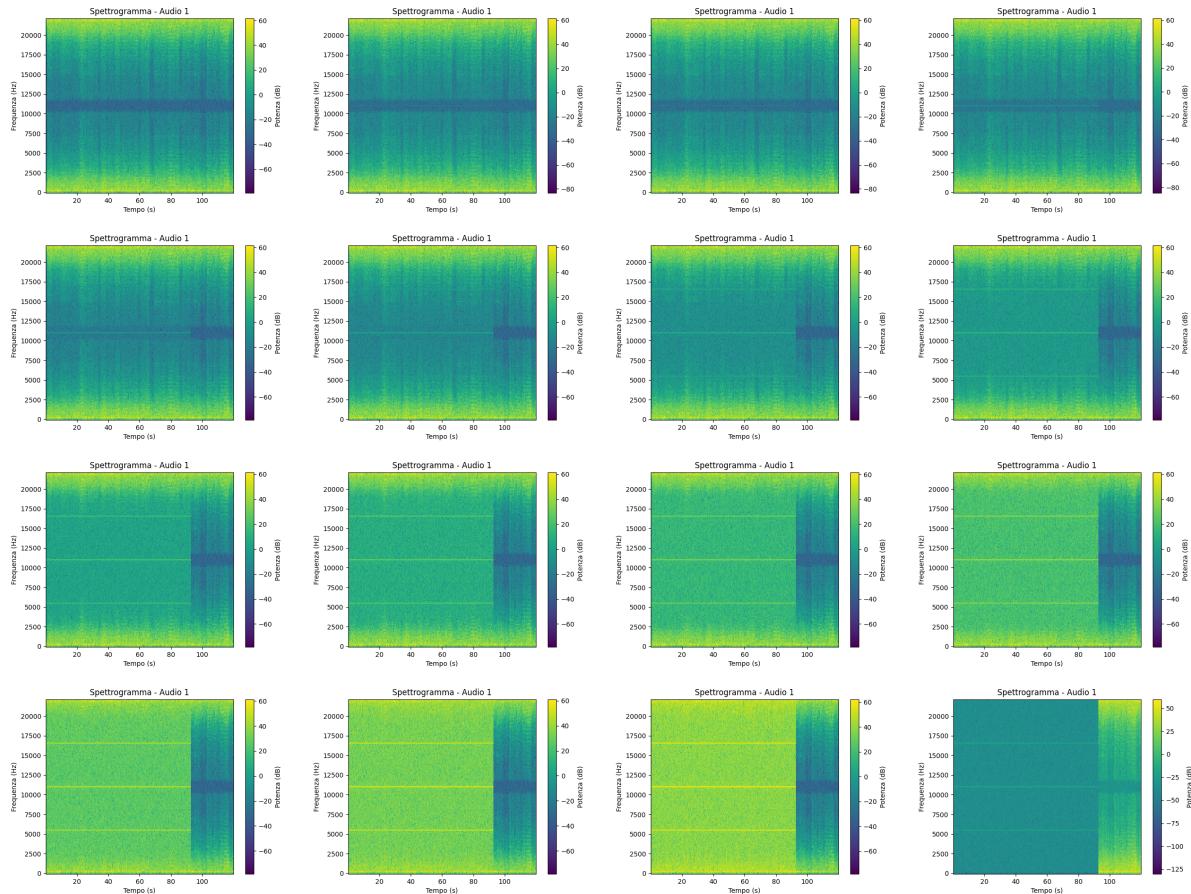


Fig. 35: Tutti i passi da 1 a 16

## 8 Phase Coding

Il phase coding è una tecnica utilizzata nella steganografia audio per nascondere informazioni all'interno di un segnale audio modificando le variazioni di fase del segnale stesso. Questa tecnica si basa sull'idea che le variazioni di fase del segnale audio possano essere modificate in modo impercettibile per rappresentare dati nascosti, consentendo così di incorporare informazioni all'interno del segnale senza comprometterne significativamente la qualità uditiva. Per applicare il phase coding, viene spesso utilizzata una trasformata matematica chiamata trasformata di Fourier, in particolare la sua versione discreta (DFT) o la sua variante veloce (FFT). La trasformata di Fourier è un'operazione matematica che scomponete un segnale audio complesso in una serie di componenti sinusoidali. Essa è utilizzata per analizzare e manipolare le componenti di fase del segnale audio, modificando in modo accurato la fase di queste componenti. Il phase coding offre diversi vantaggi per la steganografia audio, tra cui:

- Resistenza alla compressione: Poiché le variazioni di fase non dipendono dall'ampiezza del segnale, il phase coding può essere più resistente alla compressione audio rispetto ad altre tecniche di steganografia;
- Robustezza: Se applicato correttamente, il phase coding può essere robusto contro tentativi di rilevazione.

Presenta anche diversi svantaggi, come:

- Sensibilità alle distorsioni del segnale: Anche piccole variazioni nella fase del segnale audio possono causare distorsioni udibili;
- Complessità computazionale: La codifica e la decodifica delle variazioni di fase richiedono operazioni computazionali significative, specialmente quando si lavora con segnali audio ad alta risoluzione;
- Limitazioni nella capacità di inserimento: il phase coding potrebbe avere limitazioni sulla quantità di informazioni che possono essere nascoste in modo efficace senza compromettere la qualità dell'audio.

La codifica della fase avviene tramite i seguenti passaggi:

1. Codifica del messaggio: il messaggio da nascondere viene codificato in binario;
2. Modifica della fase: vengono apportate modifiche alla fase del segnale audio per nascondere le informazioni. Queste modifiche devono essere sufficientemente sottili da non essere udibili all'orecchio umano ma devono essere ancora rilevabili e decodificabili per recuperare il messaggio nascosto;
3. Introduzione del messaggio nel segnale audio: il segnale audio modificato, contenente le variazioni di fase che rappresentano il messaggio nascosto, viene combinato con il segnale audio originale. Idealmente, il segnale audio risultante non deve mostrare alcun segno evidente di manipolazione;
4. Decodifica: Per estrarre il messaggio nascosto dal segnale audio, è necessario eseguire il processo inverso di codifica. Le variazioni di fase nel segnale audio vengono analizzate e decodificate per determinare i bit del messaggio nascosto.

## 8.1 Script utilizzati

Nel contesto di questa tecnica di steganografia audio, sono state implementate strategie provenienti dalla repository "audiostego" di GitHub. In particolare, focalizzeremo la nostra attenzione sui due principali metodi utilizzati per la codifica e la decodifica dei messaggi nascosti all'interno dei segnali audio.

### 8.1.1 Encode

Di seguito viene mostrato l'algoritmo di codifica del messaggio dentro un file audio tramite la tecnica del phase coding. Questo eseguirà una codifica del messaggio dentro un file audio attraverso le fasi spettrali.

```
def encode(pathToAudio, stringToEncode):
    rate, audioData1 = wavfile.read(pathToAudio)
    stringToEncode = stringToEncode.ljust(100, '~')
    textLength = 8 * len(stringToEncode)
    blockLength = int(2 * 2 ** np.ceil(np.log2(2 * textLength)))
    blockNumber = int(np.ceil(audioData1.shape[0] / blockLength))
    audioData = audioData1.copy()
    if len(audioData1.shape) == 1:
        audioData.resize(blockNumber * blockLength, refcheck=False)
        audioData = audioData[np.newaxis]
    else:
        audioData.resize((blockNumber * blockLength, audioData.shape[1]), refcheck=False)
        audioData = audioData.T
    blocks = audioData[0].reshape((blockNumber, blockLength))
    blocks = np.fft.fft(blocks)
    magnitudes = np.abs(blocks)
    phases = np.angle(blocks)
    phaseDiffs = np.diff(phases, axis=0)
    textInBinary = np.ravel([[int(y) for y in format(ord(x), "08b")] for x in stringToEncode])
    textInPi = textInBinary.copy()
    textInPi[textInPi == 0] = -1
    textInPi = textInPi * -np.pi / 2
    blockMid = blockLength // 2
    phases[0, blockMid - textLength: blockMid] = textInPi
    phases[0, blockMid + 1: blockMid + 1 + textLength] = -textInPi[::-1]
    for i in range(1, len(phases)):
        phases[i] = phases[i - 1] + phaseDiffs[i - 1]
    blocks = (magnitudes * np.exp(1j * phases))
    blocks = np.fft.ifft(blocks).real
    audioData[0] = blocks.ravel().astype(np.int16)
    wavfile.write("encoded.wav", rate, audioData.T)
    return "encoded.wav"
```

Fig. 36: Metodo encode phase coding

Il funzionamento di questo metodo è il seguente:

1. Per prima cosa il codice legge il file audio, esso viene diviso in blocchi della dimensione della stringa inserita e successivamente vengono convertiti utilizzando la Trasformata di Fourier (DFT);
2. Il messaggio da inserire viene letto in binario e convertito in fasi, successivamente viene inserito all'interno dei blocchi spettrali;
3. Essendo che le modifiche apportate alle fasi sono invertite viene applicata la Trasformata inversa (IDFT) per ottenere il segnale audio modificato;
4. Infine viene salvato l'audio modificato dentro un file.

### 8.1.2 Decode

Questo algoritmo è progettato per estrarre un messaggio nascosto all'interno delle fasi spettrali di un file audio, di seguito viene illustrato il funzionamento dell'algoritmo di estrazione del messaggio da un file audio codificato. Mediante un'analisi delle fasi spettrali, l'algoritmo riesce a identificare e recuperare il messaggio precedentemente nascosto all'interno del segnale audio. I passi dell'algoritmo sono i seguenti:

```
def decode(audioLocation):
    rate, audioData = wavfile.read(audioLocation)
    textLength = 800
    blockLength = 2 * int(2 ** np.ceil(np.log2(2 * textLength)))
    blockMid = blockLength // 2
    if len(audioData.shape) == 1:
        secret = audioData[:blockLength]
    else:
        secret = audioData[:blockLength, 0]
    secretPhases = np.angle(np.fft.fft(secret))[blockMid - textLength:blockMid]
    secretInBinary = (secretPhases < 0).astype(np.int8)
    secretInIntCode = secretInBinary.reshape((-1, 8)).dot(1 << np.arange(8 - 1, -1, -1))
    return secretInIntCode
```

Fig. 37: Metodo decode phase coding

1. Una prima fase di lettura dell'audio, da quest'ultimo si seleziona un range detto "block length" ovvero il blocco dove sarà presente il messaggio;
2. Vengono calcolate le fasi spettrali del segmento selezionato utilizzando la Trasformata di Fourier (FFT). Successivamente, vengono estratte le fasi corrispondenti al messaggio nascosto dal punto medio del blocco fino alla lunghezza del messaggio;
3. Le fasi estratte vengono convertite in una rappresentazione binaria, i valori binari vengono quindi tradotti in caratteri ASCII per ottenere il messaggio nascosto originale;
4. Viene mostrato in output il messaggio nascosto.

## 8.2 Fase di test

Per condurre il test, è stato utilizzato il file audio "orchestra.wav" (durata 60 secondi). Successivamente, sono stati inseriti separatamente diversi testi di lunghezza variabile, contenenti rispettivamente 1000, 10000, 100.000 e 500.000 caratteri. Di seguito, sono riportati i risultati ottenuti durante il processo di steganografia e l'estrazione del messaggio nascosto. Durante il test condotto con il messaggio contenente 1000 caratteri, non sono stati riscontrati problemi significativi: il messaggio è stato nascosto perfettamente all'interno dell'audio senza comprometterne la qualità. Al momento della decodifica, il messaggio non ha subito variazioni e risulta identico all'originale. Inoltre, dal punto di vista uditorio, l'audio appare inalterato e non si percepiscono differenze rispetto alla versione originale. In contrasto, per i restanti tre file di testo, si è manifestato uno degli svantaggi associati alla tecnica di codifica delle fasi. In tutti e tre i casi, i messaggi sono stati incorporati nell'audio, tuttavia, hanno superato la lunghezza temporale dell'audio originale, provocando un'estensione della durata complessiva attraverso l'aggiunta di parti

vuote di suono. Il tutto sarà visibile successivamente tramite la steganalisi degli audio. Per verificare se i messaggi più grandi potessero essere inseriti in audio di dimensioni superiori a quello usato per il test, è stato utilizzato un audio di 7 minuti. Tuttavia, il risultato è rimasto invariato: tutti e tre i file di testo non possono essere inseriti senza modificare le dimensioni dell'audio. Da ciò si deduce che, indipendentemente dalla dimensione dell'audio, tramite questa tecnica la capacità di inserire testo è limitata e non può aumentare all'aumentare del minutaggio dell'audio. In altre parole, la tecnica del phase coding nella steganografia audio presenta delle limitazioni intrinseche che rendono difficoltoso l'inserimento di messaggi di grandi dimensioni senza influenzare la durata dell'audio. Si possono inserire circa 9500 caratteri senza modificare le dimensioni temporali dell'audio.

### 8.3 Steganalisi visuale

In questa fase della steganalisi, considerando i risultati dei test precedenti, procederemo con un confronto tra l'audio "orchestra" originale e l'audio in cui è stato inserito il messaggio di 1000 caratteri. Analizzeremo attentamente gli spettrogrammi di entrambi gli audio per individuare eventuali differenze significative. Successivamente, esamineremo anche gli spettrogrammi dei tre audio in cui è stata modificata la dimensione temporale, confrontandoli con l'audio originale e valutando le variazioni nelle caratteristiche spettrali.

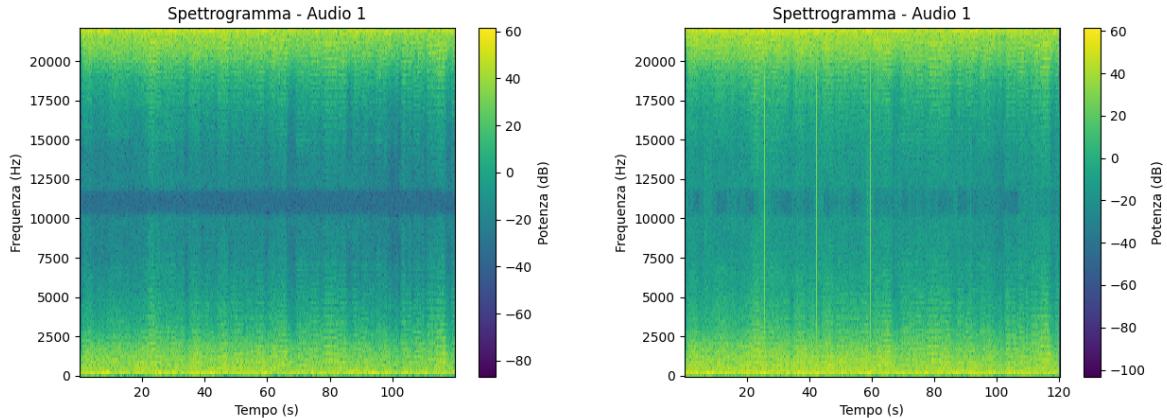


Fig. 38: Spettrogrammi phase coding, a sinistra audio originale, a destra audio col messaggio nascosto

La disamina dell'immagine evidenzia con chiarezza la presenza di frequenze enfatizzate che emergono a intervalli temporali regolari. In aggiunta, l'intero spettro di frequenze mostra una densità energetica notevolmente elevata, così come si rilevano picchi di potenza che raggiungono quasi il livello massimo, distribuiti distintamente sull'asse temporale. In questo modo la steganalisi riesce a rilevare la possibile presenza di un messaggio nascosto all'interno dell'audio.

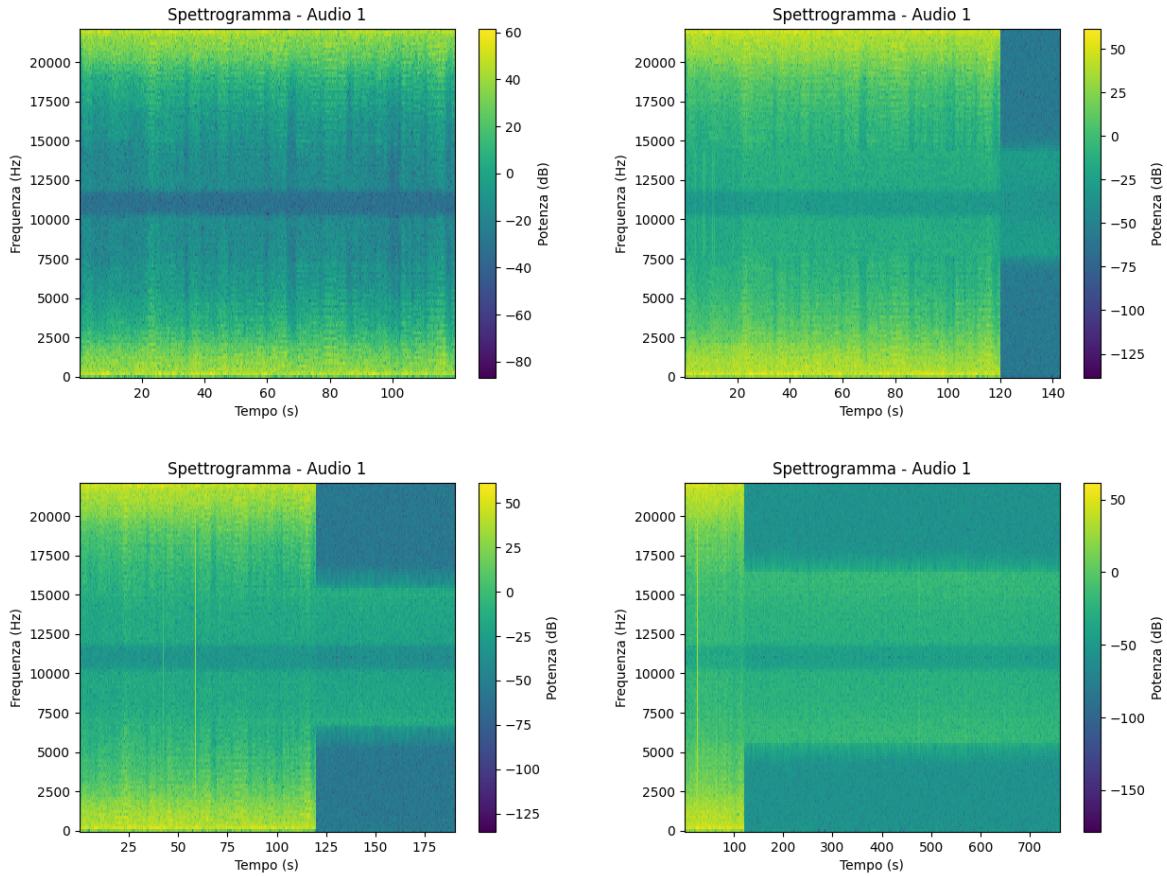


Fig. 39: Audio originale, audio con messaggio di 10.000 caratteri, audio con messaggio di 100.000 caratteri, audio con messaggio di 500.000 caratteri

I tre file audio in esame sono quelli alla quale la tecnica del phase coding ha influenzato significativamente la durata del file audio originario. Si osserva che l'inserimento di un testo di 10.000 caratteri determina un incremento di 5 secondi nella lunghezza complessiva dell'audio. Un messaggio più esteso, contenente 100.000 caratteri, provoca un'estensione di 11 secondi. Il caso più estremo riguarda l'aggiunta di un testo di 500.000 caratteri, che si traduce in un aumento della durata di 5 minuti e 20 secondi rispetto all'audio originale. Questi dati riflettono la relazione diretta tra la quantità di testo steganografato e l'ampliamento temporale del file audio. L'analisi delle immagini degli spettrogrammi consente di identificare visivamente l'incremento nella durata dei file audio esaminati. Tale variazione è riconoscibile osservando le "bande blu" posizionate sul lato destro degli spettrogrammi, in confronto con il primo spettrogramma, che rappresenta il file audio originale. Le bande blu denotano intervalli di silenzio o di bassa intensità sonora, che non erano presenti nell'audio originale e che indicano l'aggiunta di contenuto steganografico. La presenza e la lunghezza di queste bande forniscono una rappresentazione grafica dell'estensione temporale introdotta dalla steganografia nel file audio, inoltre tale variazione effettua anche un oneroso incremento delle dimensioni del file.

Per quanto riguarda l'analisi dei grafici a forma d'onda (Waveform) non si riesce a intravedere alcuna variazione dall'originale al file audio modificato. D'altra parte, dal punto di vista percettivo, si registra una discrepanza. La tecnica della codifica della fase, inserisce del rumore periodico facilmente riconoscibile dall'udito umano. Da questo si può dedurre la presenza di qualche rumore riconducibile a un messaggio nascosto.

## 9 Conclusione

In conclusione, l'elaborato ha esaminato l'impiego di due distinte tecniche di steganografia:

- LSB Coding: anche detta codifica del bit meno significativo, consente l'inserimento di una notevole quantità di dati in un file in maniera discreta, modificando il bit meno significativo di un segnale digitale. Questa tecnica di steganografia minimizza la percezione delle modifiche e riduce il rischio di rilevamento. Tuttavia, se si optasse per la modifica di bit a un livello superiore, si osserverebbe un aumento della vulnerabilità del metodo di fronte a tecniche di steganalisi, poiché tali alterazioni sarebbero più percettibili e quindi più facilmente identificabili attraverso l'analisi del file audio. Possibilità di decodificare più livelli per ampliare ulteriormente la capacità di invio dei messaggi;
- Phase Coding: è una tecnica steganografica che si avvale della trasformata di Fourier per manipolare la fase dei componenti di un segnale audio senza alterarne l'ampiezza. Sebbene efficace, questa metodologia presenta una limitazione intrinseca nella quantità massima di dati che può occultare. Inoltre, mediante la steganalisi visuale, che esamina le anomalie nelle rappresentazioni spettrali del segnale, la presenza di messaggi nascosti può essere rivelata con relativa facilità.

Avendo fatto un riepilogo breve delle due tecniche adottate possiamo dire che, malgrado LSB Coding mostri vulnerabilità quando si modificano bit di livello superiore, risulta essere preferibile in termini di furtività, poiché le modifiche a livello del bit meno significativo sono generalmente meno rilevabili e mantengono l'integrità percettiva del segnale audio. Nonostante il Phase Coding risulti essere debole contro la steganalisi, è comunque una buona alternativa che, utilizzando le componenti di fase del segnale audio, riesce a mascherare il messaggio in modo discreto.

I possibili sviluppi futuri, hanno l'intenzione di raffinare le tecniche di LSB Coding e Phase Coding, mirando a rafforzare significativamente la resistenza di tali metodologie agli avanzati strumenti di steganalisi attualmente in uso. I possibili approcci si orientano verso l'adozione e l'integrazione di sistemi crittografici, operazione che si prefigge di incrementare la sicurezza intrinseca dei dati celati, creando un doppio strato di protezione che combina le proprietà di occultamento della steganografia con la solidità della crittografia. Inoltre, è possibile un eventuale approfondimento nell'ambito delle tecnologie basate sull'intelligenza artificiale: l'obiettivo è quello di esplorare e implementare soluzioni AI per perfezionare e semplificare le operazioni di inserimento e estrazione delle informazioni steganografate, ottimizzando il rapporto tra la capacità di carico dei dati e la loro impercettibilità. L'obiettivo di integrare la steganografia con tecniche crittografiche o con l'intelligenza artificiale è di potenziare la sicurezza dei messaggi celati all'interno di file audio. Questa sinergia mira a garantire che, anche nell'eventualità in cui un messaggio nascosto venga scoperto, non sia automaticamente accessibile o decifrabile. In altre parole, questa strategia non solo nasconde l'esistenza del messaggio ma si assicura anche che, senza le chiavi o gli algoritmi specifici, il contenuto rimanga incomprensibile agli occhi degli individui non autorizzati. Questo approccio doppio fornisce un ulteriore livello di protezione, aumentando la confidenzialità e la sicurezza delle informazioni trasmesse attraverso il file audio.

## 10 Riferimenti

1. Steganografia - Sebastiano Battiato
2. Steganografia nei file Audio - Simona Di Lenno
3. Audio Steganography using Phase Encoding - Achyuta Katta
4. Repository GitHub Phase Coding
5. Repository GitHub progetto