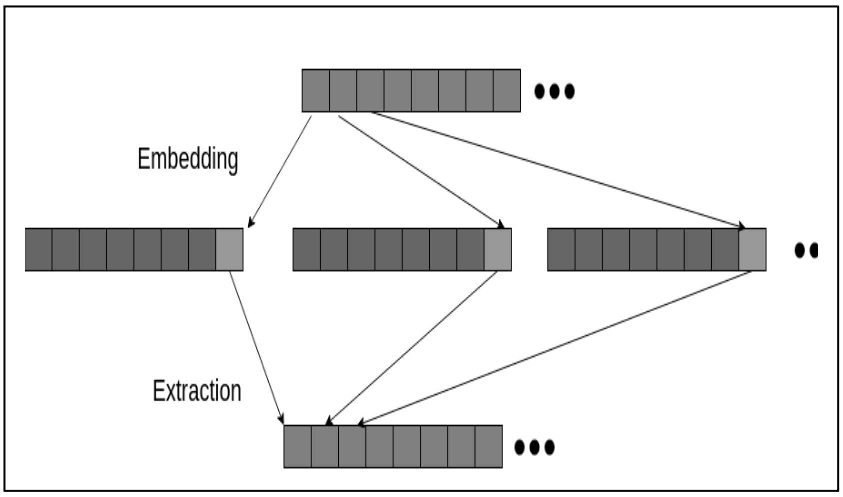
KATHMANDU UNIVERSITY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL & ELECTRONICS ENGINEERING

EEEG 221 SIGNALS AND SYSTEMS

COURSE PROJECT REPORT



AUDIO-IMAGE STEGANOGRAPHY: HIDING AUDIO SIGNALS INSIDE IMAGES

Submitted by:

Samip Aryal (22025)

Yagya Bahadur Shahi (22073)

Date of Submission:

26 November 2023

A. Introduction

This project, integrating audio signal processing and digital image processing, aimed to embed audio files into images using steganography techniques. It was initiated to demonstrate the application of classroom concepts in real-world scenarios. Motivated by the rising need for secure data transmission, our goal was to explore steganography's potential in safeguarding digital communications. Ultimately, we successfully developed a MATLAB-based tool that covertly embeds and extracts audio from images.

B. Project Description and Design

The project entailed processing audio signals into binary data and embedding them within digital images. We employed MATLAB for development, focusing on minimal alteration of the image's appearance while embedding the audio. Our design included features for both embedding and extracting audio, with a user-friendly MATLAB GUI interface highlighting steganography's practicality. The project comprises an embedding process where audio is modified based on a hashed password and hidden within the least significant bits of image pixels. Subsequently, an extraction process recovers the hidden audio using the same password. The user interacts with the system by providing a password during both embedding and extraction. The project's features include file I/O operations for reading and writing audio and image files, user input prompts for password entry, and basic security measures.

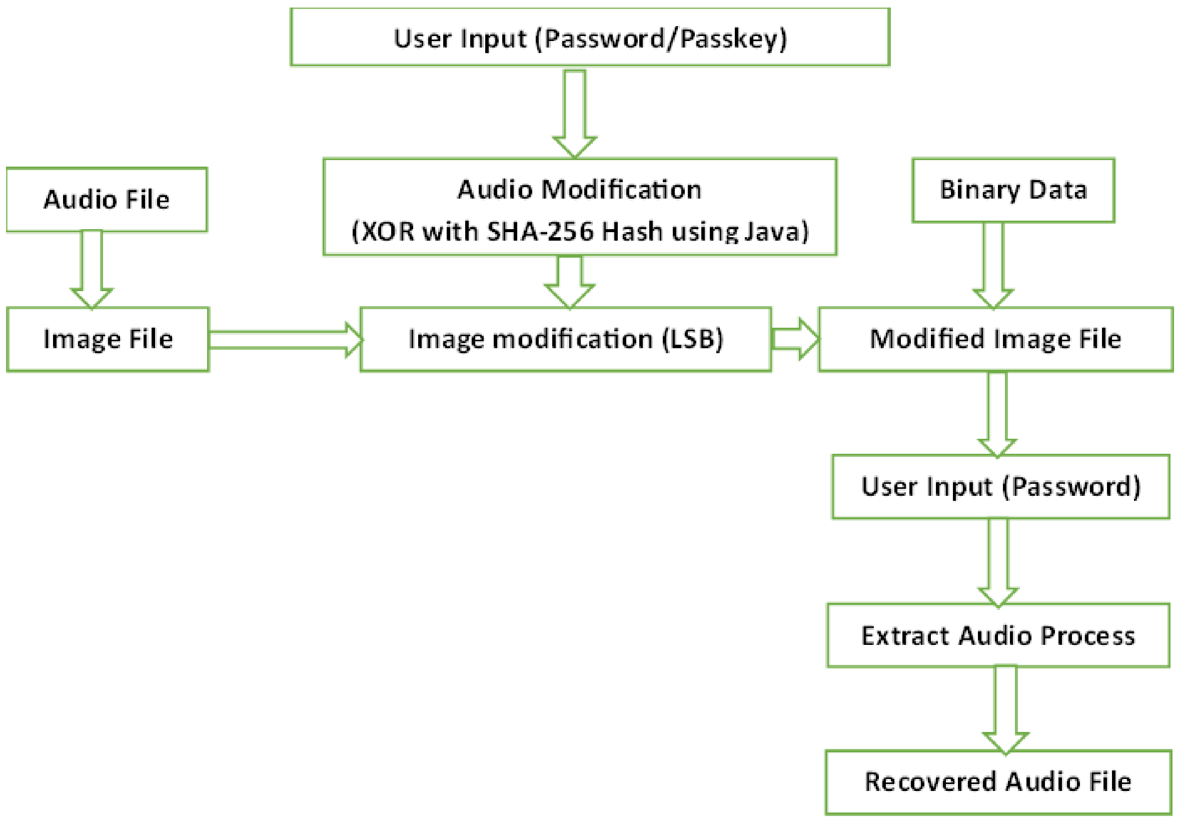


Fig: Block Diagram of the Project

Here’s the brief description of each stage:

1. **User Input (Password):** The process initiates with the user inputting a password. This password is vital for both embedding and extracting the audio. It acts as a key to ensure that only authorized users can access the hidden audio.
2.
3. **Audio Modification (XOR with Hashed Password):** The provided password undergoes a hashing process, typically using a secure algorithm like SHA-256, to generate a unique numeric hash. This hash is then used to modify the audio file through a bitwise exclusive OR (XOR) operation. By XORing the audio data with the hash, the audio undergoes a unique transformation, making the process secure and the audio data indiscernible without the correct hash.
4. **Conversion to Binary Data:** The XOR-modified audio data is then converted into a binary format. This binary representation is essential as it simplifies the process of embedding this data into the image, allowing for a straightforward modification of the image’s pixel values.
5. **Image Modification (LSB Substitution):** The least significant bits (LSBs) of the image pixels are carefully altered to incorporate the binary audio data. This method, known as LSB substitution, involves modifying the last bit of each pixel's color value so it represents a bit of the audio data. This change is typically imperceptible to the human eye, thus maintaining the original appearance of the image while embedding the audio data.
6. **Creation of Modified Image File:** The image, now carrying the hidden audio data within its pixels, is saved as a new file. This modified image file looks visually identical to the original but contains embedded audio information within its structure.
7. **User Input for Audio Extraction:** During the audio extraction phase, the user is prompted to input the password again. This step is crucial to ensure that the extraction process uses the correct password, which is necessary to accurately retrieve and decrypt the hidden audio data.
8. **Extraction of Audio Data:** The extraction process begins by reading the modified image file. The LSBs of each pixel are then extracted to reconstruct the binary representation of the hidden audio. This binary data is converted back into a decimal format and decrypted using an XOR operation with the hashed password, reversing the initial modification process.
9. **Recovery of the Original Audio File:** The final step involves normalizing and saving the decrypted audio data as a new file. Ideally, if the correct password is used, this recovered audio file closely resembles the original audio in terms of quality and content, demonstrating the effectiveness of the steganography process.

Each of these steps showcases a blend of cryptography and steganography, ensuring both the security and stealthiness of the embedded audio data. The use of hashing and XOR operations provides a robust layer of security, while LSB substitution ensures the concealment of the audio within the image without affecting its visual integrity.

C. Project Implementation

We implemented the project using MATLAB. The inputs were audio files and digital images.

Embedding Audio Data within an Image

1. **Preparing the Audio:**
   - Reading the Audio: The project starts by reading an MP3 file using audioread. This file is then processed into 16-bit integers, essentially a more computer-friendly format, using int16.
   - Converting to Binary: These audio samples are transformed into a binary format with dec2bin and typecast, turning the audio into a series of 0s and 1s.

2. **Securing with a Password:**
   - Creating a Digital Key: A password is selected and encrypted using Java's SHA-256 MessageDigest function, producing a unique hash.
   - Formatting the Key: This hash is then converted into binary and reshaped to align with the length of the binary audio data, ensuring each piece of audio data will be altered by the password.

3. **Merging Audio with Password:**
   - XOR Operation: The binary audio data is mixed with the binary hashed password using an XOR operation. This step is crucial as it intertwines the security of the password with the audio data.

4. **Preparing the Image:**
   - Image Input: An image (in this case, 'demo.png') is read using imread and its pixel values are flattened. This creates a long string of pixel values, ready to have the audio data embedded within.

5. **Embedding the Audio:**
   - Steganography: The binary audio is carefully inserted into the least significant bits (LSBs) of the image's pixels. This is done through bit-level operations (bitand and bitor), ensuring the audio data is embedded without visibly altering the image.

6. **Saving the Modified Image:**
   - Final Output: The image, now containing hidden audio data, is reshaped to its original dimensions, and saved as 'modified.png'. This image looks the same as before but holds the secret audio within its pixels.

Extracting Audio Data from the Image

1. **Accessing the Modified-Image (stego Image) :**
    - Image Retrieval: The process begins by reading the modified image (‘modified.png’) with imread and flattening its pixel values.

2. **Extracting Hidden Data:**
    - Uncovering Secrets: The least significant bits, which now contain the audio data, are extracted from each pixel using bitget. These bits form the binary representation of the original audio.

3. **Decrypting the Audio:**
    - Password Verification: The user is prompted to input the correct password. This password is then hashed in the same manner as before.
    - Reversing the XOR: The binary audio data is recovered by reversing the XOR process using the hashed password. This step is crucial as it decrypts the audio data.

4. **Restoring the Audio:**
    - Binary to Audio: The binary string is converted back into audio format using uint16, bin2dec, and typecast.
    - Normalization and Output: The audio is normalized to maintain its quality and then saved as ‘recovered.mp3’.

D. Output and Analysis

The outputs were images with embedded audio, visually identical to the originals.

1. Output Conditions:

- Environment: Generated in the MATLAB environment.
- File Formats: Utilized MP3 for audio and PNG for images.
- Audio Processing: Incorporated the steganography process and denoising code.

2. Output Features:

- Modified Image: Remains visually identical to the original, concealing the audio.
- Recovered Audio: Slight noise and quality variances from the original were observed but was hearable decent and identical to the original.
- Recovered Audio (Post-Denoising): Showed improved clarity and reduced noise.
- Security Aspect: Notably, incorrect password input during extraction results in no audio recovery, highlighting the effective security implementation.

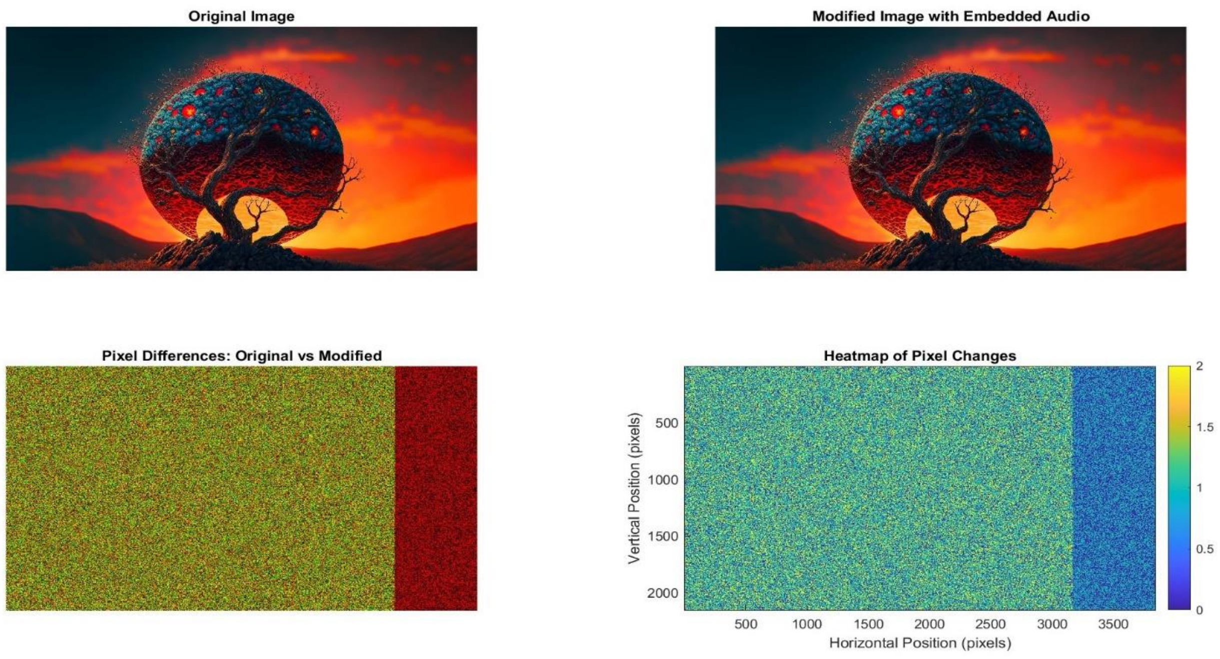


Fig: Pixel comparison between two images

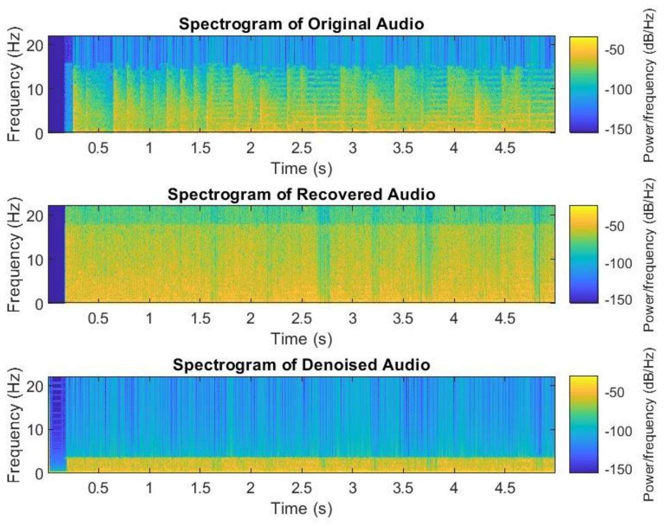


Fig: Comparison of signal strength of the audio signals

3. Critical Features in Outputs:

- Steganography Success: High effectiveness in audio embedding without altering image appearance.
- Audio Quality Pre and Post-Denoising: The quality of the extracted audio decently matched that of the original files, demonstrating the effectiveness of our steganography technique but there were some noises. But Initial quality issues addressed effectively with denoising.
- Enhanced Security: Password-dependent extraction process is a critical security feature, ensuring audio is recoverable only with correct authorization.

E. Challenges Faced and Limitations

Challenges included managing audio size and image capacity. We overcame these by optimizing the embedding algorithm. However, limitations in data size handling, susceptibility to steganalysis, dependence on a strong password, potential quality degradation were noted.

F. Conclusions and Recommendations

The project successfully demonstrated audio steganography's potential in secure communications. The provided MATLAB code successfully demonstrates basis audio-image steganography through LSB substitution, offering insight into the integration of audio data into image. However, its security is limited, relying heavily on the strength of the password and lacking robustness against advanced steganalysis. The capacity to hide large audio files is constrained, and user guidance is minimal. We recommend exploring more advanced encryption techniques and expanding the tool's capacity for larger data sizes, introduce error correction mechanism, consideration of security trade-offs, alternative tools, and a user-friendly interface would further enhance the overall functionality and usability of the code.

G. References

[1] F. S. Mohamad, "Information Hiding Based on Audio Steganography using Least Significant Bit," *International Journal of Engineering & Technology* , 2018.

[2] "https://www.mathworks.com/," Mathworks, 25 Dec 2014. [Online]. Available: https://www.mathworks.com/matlabcentral/answers/168062-i-want-to-hide-an-image-into-audio-file.

H. Appendix

The appendix includes MATLAB scripts, detailed code documentation, graphical outputs, and additional test results.

```
% CODE to execute at Transmitter Side (Hiding Audio into an image)

clear; clc;

% Read the audio file
[audio, fs] = audioread('audio_file.mp3');

% Convert the audio to 16-bit integers
audio = int16(audio * 32767);

% Convert the audio data to binary
binaryAudio = dec2bin(typecast(audio(:), 'uint16'), 16);

% Password input and hashing using Java Security
password = 'passkey'; % Replace with your desired password
md = java.security.MessageDigest.getInstance('SHA-256');
hash = md.digest(double(password));
hashedPassword = dec2bin(typecast(hash, 'uint8'), 8);

% Reshape hashedPassword to have the same number of columns as binaryAudio
hashedPassword = repmat(hashedPassword(:),
ceil(numel(binaryAudio)/numel(hashedPassword)), 1);
hashedPassword = reshape(hashedPassword, size(binaryAudio, 1), []);

% XORing the binary audio data with the hashed password
binaryAudioXORed = char(xor(binaryAudio-'0',
hashedPassword(:,1:size(binaryAudio,2))-'0') + '0');

% Read the image
img = imread('demo.png');

% Flatten the image and modified audio
flatImg = img(:);
flatAudio = reshape(binaryAudioXORed, [], 1);

% Hide the audio inside the image
for i = 1:numel(flatAudio)
    if flatAudio(i) == '0'
        flatImg(i) = bitand(flatImg(i), 254); % Set LSB to 0
    else
        flatImg(i) = bitor(flatImg(i), 1);    % Set LSB to 1
    end
end

% Reshape the image back to its original size
modImg = reshape(flatImg, size(img));

% Write the stego image to a file
imwrite(modImg, 'modified.png');
```

```matlab
% CODE to execute at the Receiver’s Side (Recovering the Audio From Image)

clear; clc;

% Read the audio file
[audio, fs] = audioread('audio_file.mp3');

% Convert the audio to 16-bit integers
audio = int16(audio * 32767);

% Convert the audio data to binary
binaryAudio = dec2bin(typecast(audio(:), 'uint16'), 16);

% Password input and hashing using Java Security
password = 'passkey'; % Replace with your desired password
md = java.security.MessageDigest.getInstance('SHA-256');
hash = md.digest(double(password));
hashedPassword = dec2bin(typecast(hash, 'uint8'), 8);

% Reshape hashedPassword to have the same number of columns as binaryAudio
hashedPassword = repmat(hashedPassword(:),
ceil(numel(binaryAudio)/numel(hashedPassword)), 1);
hashedPassword = reshape(hashedPassword, size(binaryAudio, 1), []);

% XORing the binary audio data with the hashed password
binaryAudioXORed = char(xor(binaryAudio-'0',
hashedPassword(:,1:size(binaryAudio,2))-'0') + '0');

% Read the image
img = imread('demo.png');

% Flatten the image and modified audio
flatImg = img(:);
flatAudio = reshape(binaryAudioXORed, [], 1);

% Hide the audio inside the image
for i = 1:numel(flatAudio)
    if flatAudio(i) == '0'
        flatImg(i) = bitand(flatImg(i), 254); % Set LSB to 0
    else
        flatImg(i) = bitor(flatImg(i), 1);    % Set LSB to 1
    end
end

% Reshape the image back to its original size
modImg = reshape(flatImg, size(img));

% Write the stego image to a file
imwrite(modImg, 'modified.png');
```

```
% CODE for basic denoising of the recovered audio

clear; clc;

% Read the recovered audio
[recoveredAudio, fs] = audioread('recovered.mp3');

% Apply a bandpass filter
lowerBound = 300;
upperBound = 3400;
filteredAudio = bandpass(recoveredAudio, [lowerBound, upperBound], fs);

% Normalize the audio
normalizedAudio = filteredAudio - min(filteredAudio(:));
normalizedAudio = normalizedAudio / max(normalizedAudio(:));

% Write the processed audio to a file
audiowrite('processed_recovered.mp3', normalizedAudio, fs);
```