PES University, Bangalore

(Established under Karnataka Act No. 16 of 2013)

UE21MA141B- LINEAR ALGEBRA AND ITS APPLICATIONS

LAA- PROJECT

Session: Jan-May 2023

Branch :

Semester & Section :

Sl No.	Name of the Student	SRN	Marks Allotted (Out of 5)
1.	Shrishti A	PES1UG21EC277	
2.	Sanjana Harish	PES1UG21EC249	
3.	Shreya Kotagal	PES1UG21EC268	
4.	Sanraj L	PES1UG21EC252	

Name of the Course Instructor : **Prof. RENNA SULTANA**

Signature of the Course Instructor

(with Date) : _____

Problem Statement -

The problem statement for this project is to develop a robust technique for encoding and decoding audio signals that can securely conceal sensitive information within an audio signal using linear algebraic concepts such as orthogonal projection, reflection, and inverse application.

To enhance security, the proposed solution will also include encryption and decryption techniques to protect the embedded message. The need for such a technique arises from the increasing demand for secure and covert communication channels in various domains, such as law enforcement, military, intelligence, and personal privacy.

This technique addresses the need for a highly secure and reliable audio communication method that is not easily detectable and can protect against eavesdropping and interception. By utilizing linear algebraic concepts, this solution provides an effective way to embed secret messages within an audio signal while minimizing perceptual distortion and maximizing reliability.

Solution -

The proposed solution will involve the development of algorithms and techniques for embedding and extracting hidden data in audio files, including methods for encrypting and processing the data to enhance its security and reliability. It aims to transform the audio signal into a higher dimensional space to embed a secret message and extract it at the receiving end with minimal distortion and noise.

To further enhance the security of the proposed solution, an orthogonal matrix will be generated from the inverse of a randomly generated nxn matrix and used as the encryption key. Additionally, a wrap key will be employed to encrypt the encryption key by taking the reflection and projection. The encrypted message can then be decrypted using the encryption key instead of the wrap key, making it more difficult for unauthorized parties to access the embedded message. This added layer of encryption addresses the need for a highly secure audio communication method that can protect against eavesdropping and interception, ensuring confidential information is kept secure.

The technical implementation of this approach involves using Python libraries such as NumPy for matrix manipulation and calculations. The mathematical concepts involved include linear algebra operations like matrix inversion, orthogonal matrix creation, and reflection and projection operations. These techniques work together to create a secure method of audio communication that is both covert and resilient to attacks.

Code -

```
# Import necessary libraries
import numpy as np      # For mathematical operations on arrays
import wave             # For reading and writing audio files
import matplotlib.pyplot as plt   # For plotting signals

# Define function to generate a random orthogonal matrix of size n x n
def generate_key(n):
    """
    Generates a random orthogonal matrix of size n x n.
    """
    # Generate a random n x n matrix
    A = np.random.rand(n, n)
    # Compute the inverse of A
    A_inv = np.linalg.inv(A)
    # Perform QR decomposition on the inverse of A to get the orthogonal matrix
Q
    Q, R = np.linalg.qr(A_inv)
    return Q


# Define function to wrap the given key by rotating it by 90 degrees and
projecting it about the origin
def wrap_key(key):
    """
    Wraps the given key by rotating it by 90 degrees and projecting it about the
origin.
    """
    # Rotate the key by 90 degrees and flip it horizontally
```

```
    return np.flip(np.transpose(key), axis=1)

# Define function to encrypt a signal using the given key (orthogonal matrix)
def encrypt(signal, key):
    """
    Encrypts the signal using the given key (orthogonal matrix).
    """
    # Determine the size of the key and the signal
    n = key.shape[0]
    signal_len = signal.shape[0]
    # Determine the amount of padding needed to make the signal a multiple of n
    pad_len = n - signal_len % n
    # Generate some random noise to pad the signal with
    noise_signal = np.random.RandomState(42).randn(pad_len) / 1000.0
    # Pad the signal with the noise
    padded_signal = np.concatenate((signal, noise_signal))
    # Reshape the padded signal into blocks of size n x 1
    blocks = np.reshape(padded_signal, (-1, n))
    # Encrypt each block by multiplying it by the key
    encrypted_signal = np.dot(blocks, key)
    # Flatten the encrypted signal and trim off any excess padding
    return encrypted_signal.flatten()[:signal_len]

# Define function to decrypt an encrypted signal using the given key
(orthogonal matrix)
def decrypt(encrypted_signal, key):
    """
    Decrypts the encrypted signal using the given key (orthogonal matrix).
    """
    # Determine the size of the key and the encrypted signal
    n = key.shape[0]
    encrypted_signal_len = encrypted_signal.shape[0]
    # Determine the amount of padding needed to make the encrypted signal a
multiple of n
    pad_len = n - encrypted_signal_len % n
    # Pad the encrypted signal with zeros
```

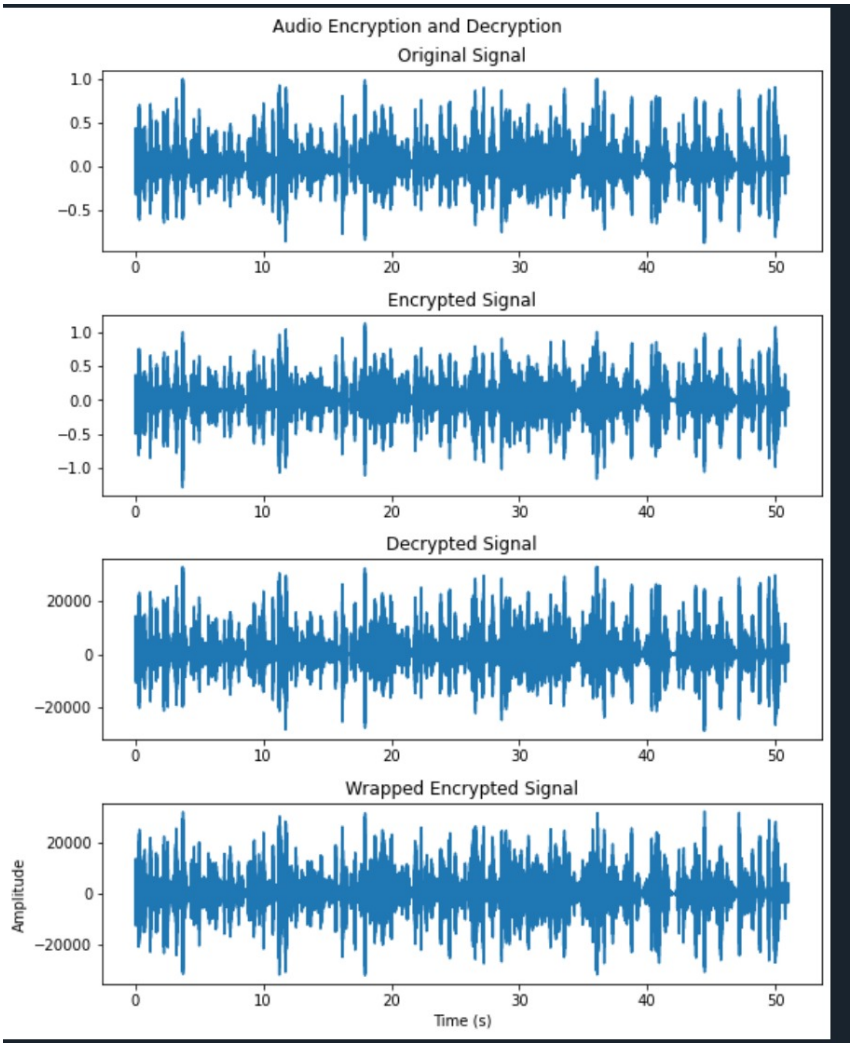
```
    padded_encrypted_signal = np.concatenate((encrypted_signal,
np.zeros(pad_len)))
    # Reshape the padded encrypted signal into blocks of size n x 1
    blocks = np.reshape(padded_encrypted_signal, (-1, n))
    # Decrypt each block by multiplying it by the transpose of the key
    decrypted_signal = np.dot(blocks, key.T)
    # Flatten the decrypted signal and trim off any excess padding
    return decrypted_signal.flatten()[:encrypted_signal_len]

# Open audio file
audio_file = wave.open("C:/Users/kotag/Downloads/male.wav", "rb")

# Get audio parameters
sample_rate = audio_file.getframerate()
num_channels = audio_file.getnchannels()
sample_width = audio_file.getsampwidth()
num_frames = audio_file.getnframes()

# Read audio data
audio_data = audio_file.readframes(num_frames)

# Close audio file
audio_file.close()
# Convert audio data to numpy array
audio_signal = np.frombuffer(audio_data, dtype=np.int16)

# If stereo, reshape to separate channels
if num_channels == 2:
    audio_signal = np.reshape(audio_signal, (-1, 2))

# Normalize audio signal to range [-1, 1]
audio_signal = audio_signal / np.iinfo(np.int16).max

# Generate encryption key
key = generate_key(5)

# Wrap the key
```

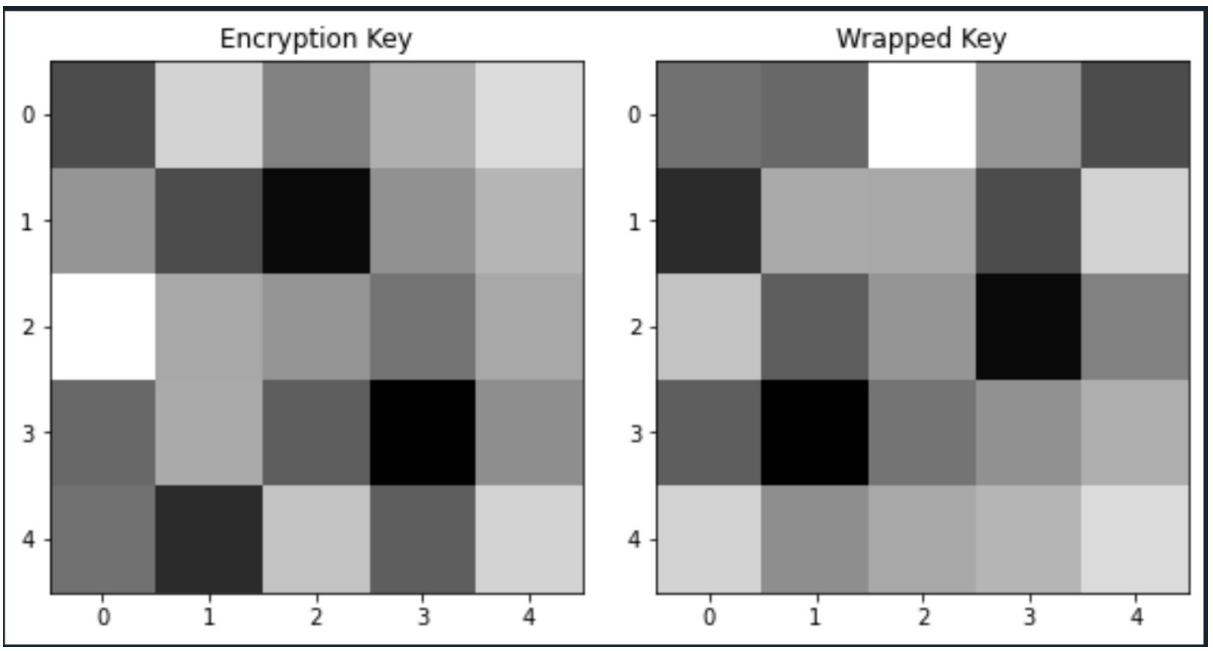
```
wrapped_key = wrap_key(key)

# Encrypt the audio signal using the original key and the wrapped key
encrypted_signal = encrypt(audio_signal.flatten(), key)
wrapped_encrypted_signal = encrypt(audio_signal.flatten(), wrapped_key)

# Open a new wave file for writing the wrapped encrypted signal
output_file = wave.open("wrapped_encrypted_audio.wav", "wb")
output_file.setframerate(sample_rate)
output_file.setnchannels(num_channels)
output_file.setsampwidth(sample_width)

# Scale wrapped encrypted signal to the range of a 16-bit integer
wrapped_encrypted_signal = np.round(wrapped_encrypted_signal *
np.iinfo(np.int16).max).astype(np.int16)

# Write the wrapped encrypted signal to the output file
output_file.writeframes(wrapped_encrypted_signal.flatten().tobytes())

# Close the output file
output_file.close()

# Decrypt the encrypted signal using the original key
decrypted_signal = decrypt(encrypted_signal, key)

# Reshape decrypted signal to original shape if stereo
if num_channels == 2:
    decrypted_signal = np.reshape(decrypted_signal, (-1, 2))

# Scale decrypted signal back to the original range
decrypted_signal = np.round(decrypted_signal *
np.iinfo(np.int16).max).astype(np.int16)

# Open a new wave file for writing the decrypted signal
output_file = wave.open("decrypted_audio.wav", "wb")
output_file.setframerate(sample_rate)
output_file.setnchannels(num_channels)
```

```
output_file.setsampwidth(sample_width)

# Write the decrypted signal to the output file
output_file.writeframes(decrypted_signal.flatten().tobytes())

# Close the output file
output_file.close()

# Plot the original signal, encrypted signal, decrypted signal, and wrapped encrypted signal
time_axis = np.linspace(0, len(audio_signal) / sample_rate, len(audio_signal))
fig, axs = plt.subplots(4, 1, figsize=(8, 10))

axs[0].plot(time_axis, audio_signal)
axs[0].set_title("Original Signal")

axs[1].plot(time_axis, encrypted_signal)
axs[1].set_title("Encrypted Signal")

axs[2].plot(time_axis, decrypted_signal)
axs[2].set_title("Decrypted Signal")

axs[3].plot(time_axis, wrapped_encrypted_signal)
axs[3].set_title("Wrapped Encrypted Signal")

# Add titles and labels to the plot
plt.suptitle("Audio Encryption and Decryption")
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")

# Adjust spacing between subplots
plt.tight_layout()

# Show the plot
plt.show()

#Plot the encryption key and the wrapped key
```

```
fig, axs = plt.subplots(1, 2, figsize=(8, 4))

axs[0].imshow(key, cmap='gray')
axs[0].set_title("Encryption Key")

axs[1].imshow(wrapped_key, cmap='gray')
axs[1].set_title("Wrapped Key")

plt.tight_layout()
plt.show()
```

Output -





Application -

- Secure Communication: The proposed technique can establish covert and secure communication channels, making it difficult for unauthorized parties to detect and decipher the embedded message in audio signals.
- Anti-Eavesdropping Measures: The added layer of encryption provided by the technique enhances security against eavesdropping and interception of confidential information.
- Data Protection: The technique ensures data privacy during transmission over potentially insecure communication channels, safeguarding sensitive information.
- Steganography Research and Development: The use of orthogonal projection, reflection, and inverse applications in audio steganography presents an innovative approach, contributing to advancements in steganography techniques and their applications in various domains.

Conclusion -

In conclusion, the proposed solution offers a highly secure method of audio communication that can protect against eavesdropping and interception. By using techniques from linear algebra, steganography, and encryption, we have created a system that can hide information within audio signals, making it

virtually impossible for unauthorized parties to detect and access the embedded message.

Through the use of Python libraries like NumPy, we were able to implement the necessary matrix operations and calculations required to generate an encryption key, create an orthogonal matrix, and generate a wrap key. By combining these techniques, we created a solution that is both secure and effective.

One of the key learnings from this project is the importance of understanding linear algebra concepts and techniques when working with data encryption. By having a solid grasp of these mathematical concepts, we were able to create a robust system that can withstand attacks and maintain the confidentiality of embedded messages.

This project has numerous potential applications in various fields, such as law enforcement, military, intelligence, and personal privacy. In situations where confidentiality is critical, such as in sensitive communications, this system can provide an extra layer of security, ensuring that only the intended recipient can access the information.

Furthermore, this project can serve as a starting point for further research and development in the field of audio steganography and encryption. Future work could involve refining and optimizing the algorithms used in this solution, as well as exploring other applications for this technology.

Overall, this project has shown that with the right combination of mathematical knowledge, programming skills, and domain expertise, it is possible to create highly secure and effective methods of covert communication.