EE323 Digital Signal Processing

Assignment - I Report

Assignment Title: Secure & Intelligent Speech Communication with Motion-Sensor Integration

Objectives

1. To develop a Simulink-based Android application that works on two phones(sender[phone A] and receiver[phone B]).
2. Recording a voice on phone A using the device microphone.
3. Encrypt the recorded voice signal using Frequency Scrambling with Key-Based Permutation to make it unintelligible.
4. Design a configurable encryption key within the application for both encryption and decryption.
5. Enable wireless transmission of the encrypted audio signal from phone A to phone B.
6. Integrate motion sensors (gyroscope and accelerometer) on phone A to capture device movement data simultaneously with the audio recording.
7. Time-synchronize the motion sensor data with the audio stream to allow for correlation between voice and movement patterns.
8. Transmit the synchronized motion data along with the encrypted audio stream.
9. Implement a receiving mechanism on phone B to decrypt the audio and receive the motion data.
10. Create a basic UI on phone B to display the received motion data in a chart or log it for analysis.

Theory

- Frequency scrambling using key-based permutation reorders frequency sub-bands of a signal, like speech or images, according to a secret key to make it unintelligible, with the scrambled signal being restorable only with the correct inverse permutation key. This method involves transforming the signal into the frequency domain, applying a frequency-based permutation using a key derived from a seed, and then transforming the signal back to the time domain.

- The security of this method relies entirely on the **key**. A longer and more complex key makes it more difficult for an attacker to guess the correct permutation sequence.

- The key is the crucial link between the scrambled and unscrambled signal. It's not just a password; it is the seed that generates the precise, secret rule used to shuffle the frequency components of the signal. Without the correct key, the scrambled signal cannot be restored to its original, intelligible form.

- The key acts as the input to a pseudorandom number generator (PRNG). A PRNG is a deterministic algorithm that, when given the same starting value (the key), will produce the exact

same sequence of seemingly random numbers every time. This sequence is then used to create the permutation map.

- The permutation map, which is unique to your key, is applied to the frequency components of the signal after a Fast Fourier Transform (FFT). This rearranges the spectral information, destroying the natural order and making the signal incomprehensible. The scrambled signal is now ready for transmission.

- Decryption: On the receiver's end, the same process is performed in reverse.

- The receiver uses the exact same key to seed the same PRNG. This generates the exact same permutation sequence used for scrambling. The receiver then creates the inverse permutation map, which is the exact opposite of the scrambling map. This map tells the system how to put each frequency sub-band back in its original position. The received, scrambled signal is first converted to the frequency domain (FFT) and then its sub-bands are reordered according to the inverse permutation map.

- Finally, an Inverse Fast Fourier Transform (IFFT) converts the signal back into the time domain, restoring the original audio.[1]

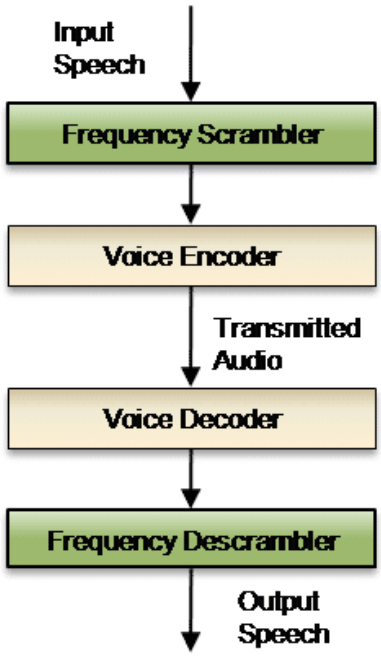

Figure-1

Our Implementation:-

1. **Using the SimuLink Android package**: The Simulink Support Package for Android Devices enables developing, simulating, and deploying models from Simulink directly onto Android smartphones and tablets. It provides specialized blocks to access device sensors such as the accelerometer, gyroscope, magnetometer, GPS, microphone, and camera, as well as to control outputs like the speaker and display. By generating Android-compatible code automatically, it allows rapid prototyping and real-time testing of algorithms on actual hardware. This makes it useful for building interactive applications, performing data acquisition, and implementing signal processing or control systems using the built-in capabilities of Android devices.

   We connected our Simulink with the Android suite, which contains the necessary SDK (Software Development Kit) files, to enable communication between Simulink and Android devices. The SDK provides the essential tools, libraries, and platform files required for building and deploying applications. By linking Simulink with this Android suite, we were able to generate Android-compatible code from our Simulink models and deploy it directly onto the connected Android device for real-time testing and execution.

2. **Making two Simulink Pipelines**: To achieve our objective of transmitting a speech signal from one phone to another, we created two separate Simulink pipelines: a **Sender pipeline** and a **Receiver pipeline**. The Sender pipeline captured the speech signal from the microphone of the first phone, processed it, and transmitted the data over a wireless connection using the **TCP/IP protocol**. The Receiver pipeline, running on the second phone, received this data through the same protocol, decoded it, and played it back through the phone's speaker. This setup allowed real-time wireless audio communication between the two Android devices using Simulink.

3. **Sender Simulink Model**
   - The role of the sender is to capture the audio signal, scramble its frequency with the key-based permutation, and send that scrambled audio along with the motion data (3 axes of accelerometer and 3 axes of gyroscope) via TCP/IP block

   - To capture the audio, the Audio Capture block from the Simulink Support Package for Android Devices is used to record audio from the microphone of an Android device and provide it as input to a Simulink model for further processing. It allows configuration of parameters such as the sampling rate, number of channels (mono or stereo), and frame size, which determine how the audio data is captured and streamed into the model. The block outputs the recorded audio as frame-based signals, enabling real-time acquisition of speech

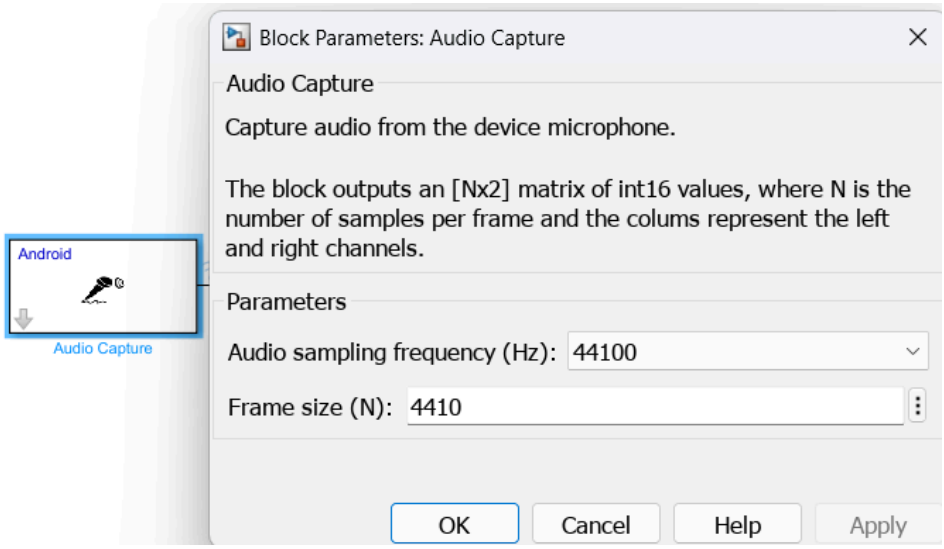

Figure 2

The speech signal of humans can go up to 4.4Khz; so the sampling frequency is taken 10 times the maximum frequency in order to avoid aliasing

Frame size refers to the number of audio samples that are grouped together and sent at once from the Audio Capture block to Simulink during each simulation step.

For a 4410 frame size, a 44.1KHz sampling frequency implies each frame contains 100ms of data

- The Selector block is used after the Audio Capture block to extract specific portions of the captured audio signal, such as selecting a single channel from a multi-channel (stereo) input or picking certain elements from the frame. Since the Audio Capture block outputs a matrix of samples, the Selector block helps isolate the required data, making it easier to process, transmit, or analyze. In our setup, it was mainly used to select one audio channel from the captured stereo signal to simplify the data before further processing.

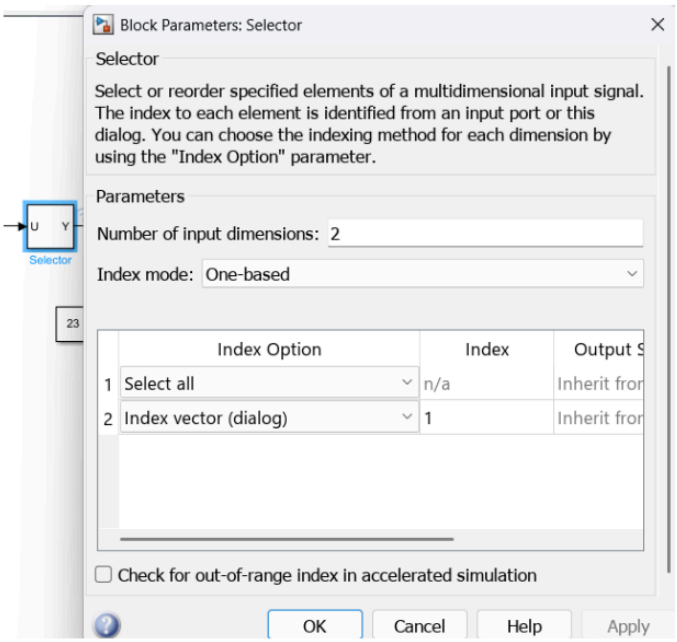

Figure 3

- After the Selector block, a Hanning window is used to smooth the edges of each audio frame before further processing. Applying a window like Hanning reduces spectral leakage that occurs when performing operations such as Fourier transforms on finite-length frames. Without windowing, the abrupt start and end of each frame can introduce artificial high-frequency components. The Hanning window tapers the signal gradually to zero at the edges, minimizing discontinuities between frames and improving the accuracy of frequency-domain analysis or transmission quality of the speech signal.

  A MATLAB function was made, which contained the formula for hanning window(raised cosine)

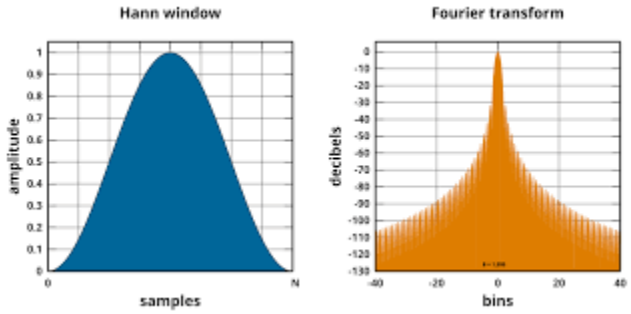

Figure 4

- Now that our time domain continuous signal was ready, it was time to convert it to the frequency domain by taking the FFT

  After applying the Hanning window, a custom performFFT function is used instead of the standard FFT block to provide better control over how the Fourier transform is applied to the audio data. This function first converts the input to double and complex format to avoid data type issues, then automatically determines the FFT length based on the frame size, and finally performs the FFT along the correct dimension of the data. Using this custom function ensures consistent handling of different data types and frame sizes, which was not managed using the built-in FFT block.

```
function X = performFFT(u)


x = complex(double(u));          % avoid complex-integer issues
N = size(x,1);                   % FFT length = frame length
X = fft(x, N, 1);                % FFT along dim-1 (rows)
end
```

- For key-based scrambling: After the FFT is done, the custom MATLAB function is applied to scramble the frequency components of the speech signal, adding a layer of security during transmission. The scrambling process begins by converting the user-provided key into a 32-bit integer seed using a simple hash-like method, where the first four bytes of the key are combined in little-endian order and mixed with the frame size. This seed is then used to initialize a pseudo-random number generator (PRNG) based on the xorshift32 algorithm, which creates a reproducible sequence of random numbers tied to the key. Using these random numbers, the Fisher–Yates shuffle algorithm generates a random permutation of the positive frequency indices in the FFT spectrum. These selected frequency components are rearranged according to this permutation, and their conjugate negative frequency counterparts are also reordered to maintain Hermitian symmetry, ensuring that the inverse FFT still produces a real-valued signal. This way, the resulting spectrum appears random and cannot be reconstructed correctly without the original key.[2]

- Now, after scrambling, the signal is converted back to the time domain using the performIFFT function. This function takes the scrambled frequency-domain data and applies the Inverse Fast Fourier Transform (IFFT) along the frame dimension to reconstruct the time-domain audio signal. Before performing IFFT, the data is converted to complex double format to avoid any type-related issues, and after the transformation, only the real part is taken because the original audio signal is real-valued. This step completes the scrambling process by producing a time-domain signal that sounds unintelligible but can be later recovered using the correct key during the descrambling stage.

- Now pack this scrambled audio with motion data (accelerometer and gyroscope) by a concatenation block and send this by TCP/IP send block to the receiver

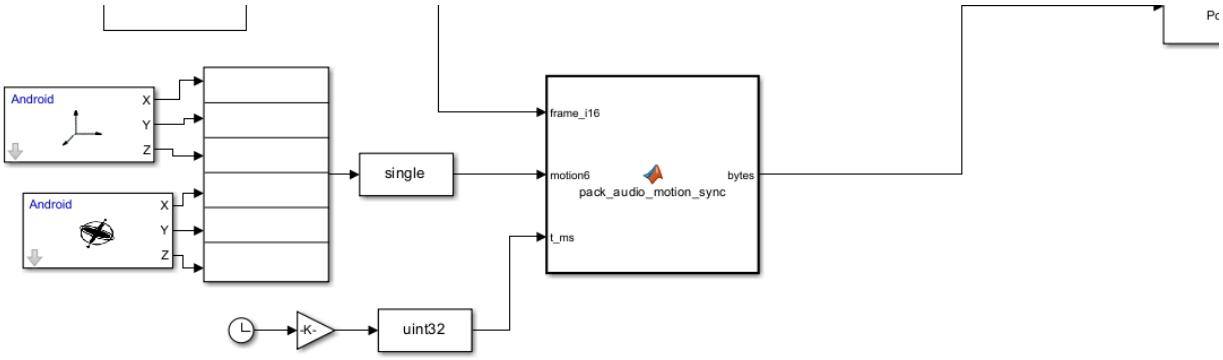

Figure 5

Here is the complete Sender model pipeline

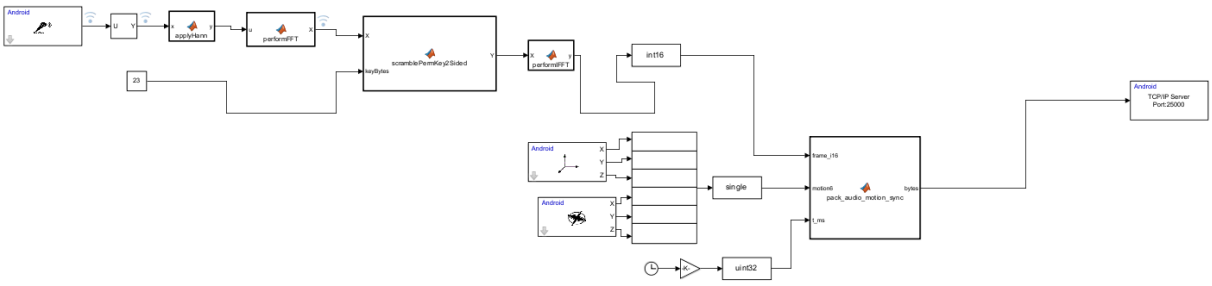

Figure 6

4. **Receiver Simulink Model**
    - At the receiver side, the Simulink model uses a **TCP/IP Client block** to receive the transmitted data stream, which contains both the **speech signal (audio samples)** and the **motion data** packed together. This combined data is first received as a sequence of bytes. The provided unpack_audio_motion_sync function is then used to **separate and reconstruct** these two parts from the incoming byte stream.

      When the data was sent from the transmitter, it was organized in a specific format using the pack_audio_motion_sync function — the first 4 bytes stored a timestamp (t_ms as uint32), the next 24 bytes stored 6 motion values (motion6 as single), and the remaining 8820 bytes stored 4410 audio samples (frame_i16 as int16). At the receiver, the unpack_audio_motion_sync function **typecasts** these respective sections of the byte array back into their original data types and shapes, effectively **recovering the audio and motion signals** for further processing (like descrambling and playback) inside the Simulink pipeline.

- After unpacking the data, the extracted motion data is passed to a **1x6 DeMUX block** to separate it into six individual signals, each corresponding to one motion sensor axis — three from the accelerometer (X, Y, Z) and three from the gyroscope (X, Y, Z). These six outputs are then fed into a **Scope block** in Simulink, which plots their values against time in real time. This allows the user to visually monitor and analyze the motion patterns of the device as the data is being received, enabling immediate observation of changes in orientation, movement, or vibrations.

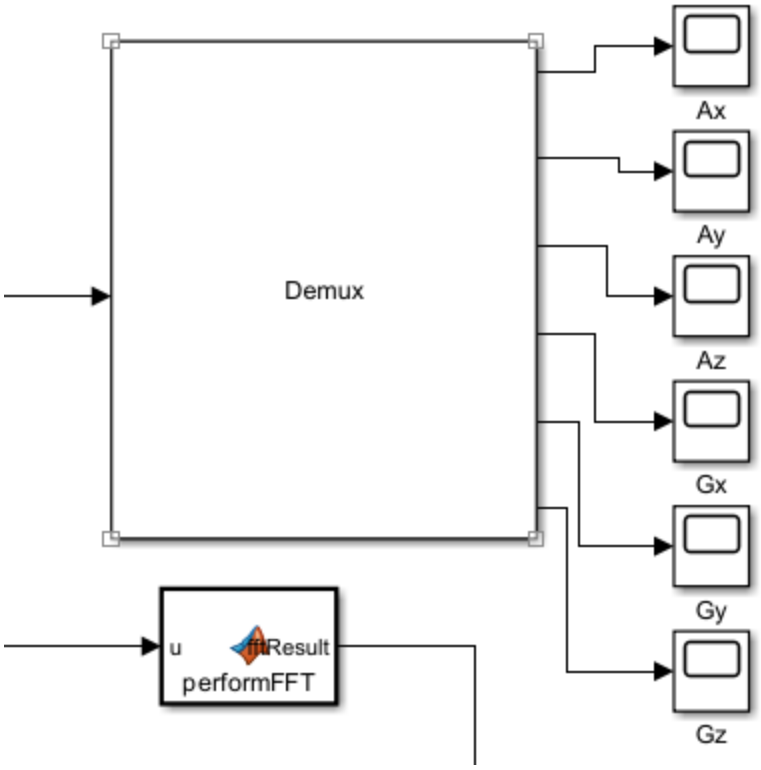

Figure 7

- After the received audio signal is converted back to the frequency domain using FFT, the unscramblePerm_simple function is applied to reverse the scrambling that was done on the sender side. This function takes the same key (entered on the receiver's phone) and regenerates the exact same pseudo-random permutation that was used for scrambling. It does this by first converting the key bytes into a 32-bit seed (just like in the scrambling step) and using that seed to initialize the xorshift32 PRNG. This PRNG output is used in the Fisher–Yates shuffle algorithm to reconstruct the same permutation of frequency indices. The function then builds an inverse permutation map to reorder the scrambled positive frequency components back to their original positions, while also restoring the corresponding negative frequencies to maintain Hermitian symmetry. The DC and Nyquist components remain unchanged. As a result, the original frequency spectrum is successfully recovered, making it possible to reconstruct the original time-domain audio signal through IFFT.

- The unscrambled time domain signal is sent to the audioplayback block, and the receiver gets the speech signal

- Following is the complete receiver pipeline

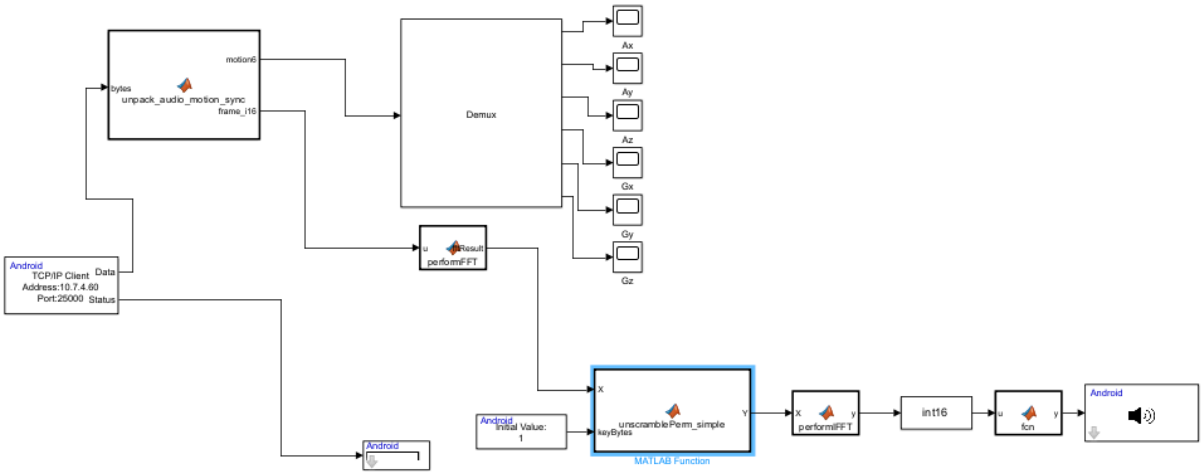

Figure 8

During the observation, it was noted that the receiver's voice appeared jittery, exhibiting slight fluctuations or instability, while the overall speech content remained clear and intelligible. However, there was a noticeable presence of background noise accompanying the speech, which slightly affected the listening experience despite the clarity of the spoken words.

The jittery nature of the receiver's voice, despite the speech being clear, could be attributed to network-related issues such as packet loss, jitter (variation in packet arrival times), or latency fluctuations during transmission. These factors can cause small gaps or rapid changes in the timing of audio packets, resulting in a shaky or trembling effect in the voice. Additionally, background noise may be introduced due to a low signal-to-noise ratio, microphone sensitivity, or interference from the surrounding environment. While the speech content remains understandable because most audio packets are still being delivered, the combination of timing irregularities and background noise can create a perception of jitter in the voice.

A demonstration video is also made, demonstrating the working of the app on both devices; it also explains the scrambling descrambling logic and pipelining of simulink model

Click this YouTube link

Or scan this QR code:

References:

[1]R. M. Milton, *"A time and frequency-domain speech scrambler,"* pp. 125–130, Jan. 2003, doi: https://doi.org/10.1109/comsig.1989.129030

[2] "Fisher–Yates shuffle," *Isa-afp.org*, 2016. https://www.isa-afp.org/entries/Fisher_Yates.html#cite-popup (accessed Sep. 15, 2025).