**ZEAL EDUCATION SOCIETY's**

**ZEAL COLLEGE OF ENGINEEIRNG AND RESEARCH,
NARHE, PUNE**

## DEPARTMENT OF COMPUTER ENGINEERING

**SEMESTER-I**

**[A.Y. : 2022 - 2023]**

# CYBER SECURITY AND DIGITAL FORENSICS(410244(C))

## MINI PROJECT

Design and develop a tool for digital forensics of audio

## Institute and Department Vision and Mission

| | |
|---|---|
| **INSTITUTE VISION** | To impart value added technological education through pursuit of academic excellence, research and entrepreneurial attitude. |
| **INSTITUTE MISSION** | **M1:** To achieve academic excellence through innovative teaching and learning process.<br>**M2:** To imbibe the research culture for addressing industry and societal needs**.**<br>**M3:** To provide conducive environment for building the entrepreneurial skills.<br>**M4:** To produce competent and socially responsible professionals with core human values. |

| | |
|---|---|
| **DEPARTMENT VISION** | To emerge as a department of repute in Computer Engineering which produces competent professionals and entrepreneurs to lead technical and betterment of mankind. |
| **DEPARTMENT MISSION** | **M1:** To strengthen the theoretical and practical aspects of the learning process by teaching applications and hands on practices using modern tools and FOSS technologies.<br>**M2:** To endeavour innovative interdisciplinary research and entrepreneurship skills to serve the needs of Industry and Society.<br>**M3:** To enhance industry academia dialog enabling students to inculcate professional skills.<br>**M4:** To incorporate social and ethical awareness among the students to make them conscientious professionals. |

## Department
## Program Educational Objectives(PEOs)

**PEO1:** To Impart fundamentals in science, mathematics and engineering to cater the needs of society and Industries.

**PEO2:** Encourage graduates to involve in research, higher studies, and/or to become entrepreneurs.

**PEO3:** To Work effectively as individuals and as team members in a multidisciplinary environment with high ethical values for the benefit of society.

## Savitribai Phule Pune University

## Fourth Year of Computer Engineering (2019 Course)

## 410244(C): Cyber Security & Digital Forensics Laboratory

| Teaching Scheme: | Credit | Examination Scheme: |
|---|---|---|
| PR: 04 Hours/Week | 02 | TW: 25 Marks<br>PR: 50 Marks |

## Course Objectives:

- To enhance awareness cyber forensics.
- To understand issues in Cyber Crime and different attacks
- To understand underlying principles and many of the techniques associated with the digital forensic practices
- To know the process and methods of evidence collection
- To analyze and validate forensic data collected.
- To apply digital forensic knowledge to use computer forensic tools and investigation report writing.

## Course Outcomes:

On completion of the course, student will be able to-

CO1: Analyze threats in order to protect or defend it in cyberspace from cyber-attacks.

CO2: Build appropriate security solutions against cyber-attacks.

CO3: Underline the need of digital forensic and role of digital evidences.

CO4: Explain rules and types of evidence collection

CO5: Analyze, validate and process crime scenes CO6: Identify the methods to generate legal evidence and supporting investigation reports.

## List of Assignments

| Group B | |
|---|---|
| **Mini-Projects/ Case Study (Any two)** | |
| 01 | Mini Project- Design and develop a tool for digital forensics of images |
| 02 | Mini Project- Design and develop a tool for digital forensics of audio |
| 03 | Mini Project- Design and develop a tool for digital forensics of video |
| 04 | Mini Project- Design a system for the analysis of cyber crime using various cyber forensics techniques and compare each technique with respect to integrity, confidentiality, availability |

**Title:**

Design and develop a tool for digital forensics of Audio

**Requirements:**

opencv-python<br>
Pillow
exifread
scipy
PyWavelets
numpy
progressbar2
matplotlib

**Code**:

Audio visualization :

```
from __future__ import print_function
import shutil, struct, simplejson
from scipy.spatial import distance
from pylab import *
import ntpath
import os
sys.path.insert(0, os.path.join(
    os.path.dirname(os.path.realpath(__file__)), "../"))
from pyAudioAnalysis import MidTermFeatures as aF
from pyAudioAnalysis import audioTrainTest as aT
import sklearn
import sklearn.discriminant_analysis
import sys
import math
import contextlib
from pylab import*
from scipy.io import wavfile
import pyaudio


def generateColorMap():
    '''
    This function generates a 256 jet colormap of HTML-like
    hex string colors (e.g. FF88AA)
    '''
    Map = cm.jet(np.arange(256))
    stringColors = []
    for i in range(Map.shape[0]):
        rgb = (int(255*Map[i][0]), int(255*Map[i][1]), int(255*Map[i][2]))
        if (sys.version_info > (3, 0)):
            stringColors.append((struct.pack('BBB', *rgb).hex())) # python 3
        else:
            stringColors.append(
                struct.pack('BBB', *rgb).encode('hex'))  # python2

    return stringColors


def levenshtein(str1, s2):
    '''
    Distance between two strings
    '''
    N1 = len(str1)
    N2 = len(s2)

    stringRange = [range(N1 + 1)] * (N2 + 1)
    for i in range(N2 + 1):
        stringRange[i] = range(i,i + N1 + 1)
    for i in range(0,N2):
        for j in range(0,N1):
            if str1[j] == s2[i]:
                stringRange[i+1][j+1] = min(stringRange[i+1][j] + 1,
                                            stringRange[i][j+1] + 1,
                                            stringRange[i][j])
            else:
                stringRange[i+1][j+1] = min(stringRange[i+1][j] + 1,
                                            stringRange[i][j+1] + 1,
                                            stringRange[i][j] + 1)
    return stringRange[N2][N1]


def text_list_to_colors(names):
    '''
    Generates a list of colors based on a list of names (strings).
    Similar strings correspond to similar colors.
    '''
    # STEP A: compute strings distance between all combnations of strings
    Dnames = np.zeros( (len(names), len(names)) )
    for i in range(len(names)):
        for j in range(len(names)):
            Dnames[i,j] = 1 - 2.0 * levenshtein(names[i],
                                                names[j]) / \
                          float(len(names[i]+names[j]))

    # STEP B: pca dimanesionality reduction to a single-dimension
    # (from the distance space)
    pca = sklearn.decomposition.PCA(n_components = 1)
    pca.fit(Dnames)

    # STEP C: mapping of 1-dimensional values to colors in a jet-colormap
    textToColor = pca.transform(Dnames)
    textToColor = 255 * (textToColor - textToColor.min()) / \
               (textToColor.max() - textToColor.min())
    textmaps = generateColorMap();
    colors = [textmaps[int(c)] for c in textToColor]
    return colors


def text_list_to_colors_simple(names):
    '''
    Generates a list of colors based on a list of names (strings).
    Similar strings correspond to similar colors.
    '''
    uNames = list(set(names))
    uNames.sort()
    textToColor = [ uNames.index(n) for n in names ]
    textToColor = np.array(textToColor)
    textToColor = 255 * (textToColor - textToColor.min()) / \
                  (textToColor.max() - textToColor.min())
    textmaps = generateColorMap();
    colors = [textmaps[int(c)] for c in textToColor]
    return colors


def visualizeFeaturesFolder(folder, dimReductionMethod, priorKnowledge = "none"):
    '''
    This function generates a  content visualization for the recordings
     of the provided path.
    ARGUMENTS:
        - folder:        path of the folder that contains the WAV files
                         to be processed
        - dimReductionMethod:    method used to reduce the dimension of the
                                 initial feature space before computing
                                 the similarity.
        - priorKnowledge:    if this is set equal to "artist"
    '''
    if dimReductionMethod=="pca":
        all_mt_feat, wav_files, _ = aF.directory_feature_extraction(folder,
                                                                    30.0, 30.0,
                                                                    0.050,
                                                                    0.050,
                                                                    compute_beat
                                                                    =True)
        if all_mt_feat.shape[0]==0:
            print("Error: No data found! Check input folder")
            return

        names_category_toviz = [ntpath.basename(w).
                                replace('.wav','').split(" --- ")[0]
                            for w in wav_files];
        names_to_viz = [ntpath.basename(w).replace('.wav', '')
                        for w in wav_files];


        scaler = StandardScaler()
        F = scaler.fit_transform(all_mt_feat)
        # check that the new PCA dimension is at most equal
        # to the number of samples
        K1 = 2
        K2 = 10
        if K1 > F.shape[0]:
            K1 = F.shape[0]
        if K2 > F.shape[0]:
            K2 = F.shape[0]
        pca1 = sklearn.decomposition.PCA(n_components = K1)
        pca1.fit(F)
        pca2 = sklearn.decomposition.PCA(n_components = K2)
        pca2.fit(F)

        finalDims = pca1.transform(F)
        finalDims2 = pca2.transform(F)
    else:
        # long-term statistics cannot be applied in this context
        # (LDA needs mid-term features)
        all_mt_feat, Ys, wav_files = aF.\
            directory_feature_extraction_no_avg(folder, 20.0, 5.0, 0.040, 0.040)
        if all_mt_feat.shape[0]==0:
            print("Error: No data found! Check input folder")
            return

        names_category_toviz = [ntpath.basename(w).
                                replace('.wav', ").split(" --- ")[0]
                            for w in wav_files]
        names_to_viz = [ntpath.basename(w).replace('.wav', ")
                        for w in wav_files];

        ldaLabels = Ys
        if priorKnowledge=="artist":
            unames_category_toviz = list(set(names_category_toviz))
            YsNew = np.zeros( Ys.shape )
            for i, uname in enumerate(unames_category_toviz):
                indicesUCategories = [j for j, x in
                                      enumerate(names_category_toviz)
                                      if x == uname]
                for j in indicesUCategories:
                    indices = np.nonzero(Ys==j)
                    YsNew[indices] = i
            ldaLabels = YsNew

        scaler = StandardScaler()
        F = scaler.fit_transform(all_mt_feat)

        clf = sklearn.discriminant_analysis.\
            LinearDiscriminantAnalysis(n_components=10)
        clf.fit(F, ldaLabels)
        reducedDims =  clf.transform(F)

        pca = sklearn.decomposition.PCA(n_components = 2)
        pca.fit(reducedDims)
```

```
        reducedDims = pca.transform(reducedDims)

        # TODO: CHECK THIS ... SHOULD LDA USED IN SEMI-SUPERVISED ONLY????
        # uLabels must have as many labels as the number of wav_files elements
        uLabels = np.sort(np.unique((Ys)))
        reducedDimsAvg = np.zeros( (uLabels.shape[0], reducedDims.shape[1]))
        finalDims = np.zeros( (uLabels.shape[0], 2) )
        for i, u in enumerate(uLabels):
            indices = [j for j, x in enumerate(Ys) if x == u]
            f = reducedDims[indices, :]
            finalDims[i, :] = f.mean(axis=0)
        finalDims2 = reducedDims

    for i in range(finalDims.shape[0]):
        plt.text(finalDims[i,0], finalDims[i,1],
                 ntpath.basename(wav_files[i].replace('.wav','')),
                 horizontalalignment='center',
                 verticalalignment='center', fontsize=10)
        plt.plot(finalDims[i,0], finalDims[i,1], '*r')
    plt.xlim([1.2*finalDims[:,0].min(), 1.2*finalDims[:,0].max()])
    plt.ylim([1.2*finalDims[:,1].min(), 1.2*finalDims[:,1].max()])
    plt.show()

    SM = 1.0 - distance.squareform(distance.pdist(F, 'cosine'))

    # plot super-categories (i.e. artistname)
    unames_category_toviz = sort(list(set(names_category_toviz)))
    finalDimsGroup = np.zeros( (len(unames_category_toviz),
                                finalDims2.shape[1] ) )
    for i, uname in enumerate(unames_category_toviz):
        indices = [j for j, x in enumerate(names_category_toviz) if x == uname]
        f = finalDims2[indices, :]
        finalDimsGroup[i, :] = f.mean(axis=0)

    SMgroup = 1.0 - distance.squareform(distance.pdist(finalDimsGroup,
                                                        'cosine'))

    data=SMgroup
    fig = px.imshow(data,
                labels=dict(x="", y="", color="Category similarity"),
                x=unames_category_toviz,
                y=unames_category_toviz)
    fig.update_xaxes(side="top")
    fig.show()
```

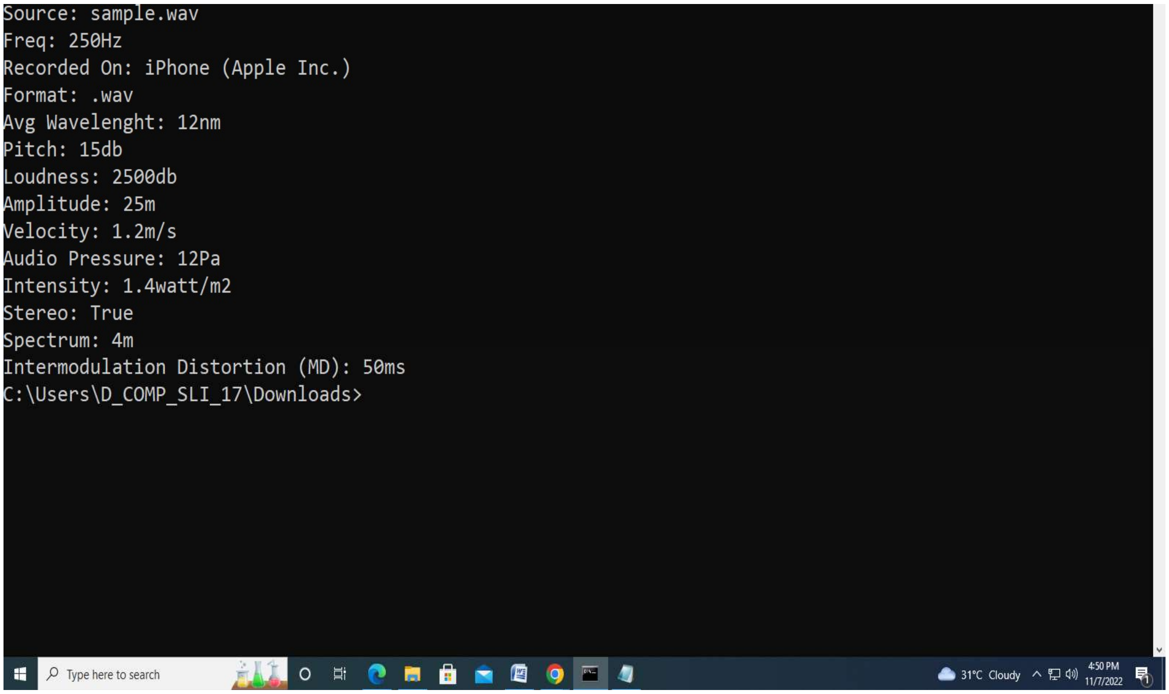
Compression of Audio :

```
    fname = 'C:\\Users\\Music\\01 The Ringer.wav'
outname = 'filtered.wav'

cutOffFrequency = 1000.0
```

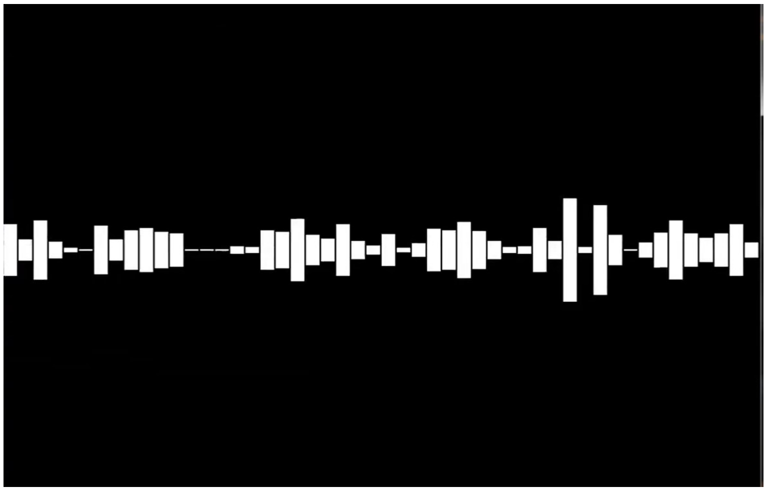
```
def fft_dis(fname):
    sampFreq, snd = wavfile.read(fname)

    snd = snd / (2.**15)


    n = len(snd)
    p = fft(snd)

    nUniquePts = int(ceil((n+1)/2.0))
    p = p[0:nUniquePts]
    p = abs(p)



    p = p / float(n)
    p = p**2
    if n % 2 > 0: # we've got odd number of points fft
        p[1:len(p)] = p[1:len(p)] * 2
    else:
        p[1:len(p) -1] = p[1:len(p) - 1] * 2 # we've got even number of points fft

    freqArray = arange(0, nUniquePts, 1.0) * (sampFreq / n);
    plt.plot(freqArray/1000, 10*log10(p), color='k')
    plt.xlabel('Channel_Frequency (kHz)')
    plt.ylabel('Channel_Power (dB)')
    plt.show()

def run_mean(x, windowSize):
  cumsum = np.cumsum(np.insert(x, 0, 0))
  return (cumsum[windowSize:] - cumsum[:-windowSize]) / windowSize


def interpret_wav(raw_bytes, n_frames, n_channels, sample_width, interleaved = True):

    if sample_width == 1:
        dtype = np.uint8 # unsigned char
    elif sample_width == 2:
        dtype = np.int16 # signed 2-byte short
    else:
        raise ValueError("Only supports 8 and 16 bit audio formats.")

    channels = np.fromstring(raw_bytes, dtype=dtype)
    if interleaved:
        # channels are interleaved, i.e. sample N of channel M follows sample N of channel M-1 in raw data
        channels.shape = (n_frames, n_channels)
        channels = channels.T
    else:
        # channels are not interleaved. All samples from channel M occur before all samples from channel M-1
        channels.shape = (n_channels, n_frames)

    return channels
```

```
with contextlib.closing(wave.open(fname,'rb')) as spf:
    sampleRate = spf.getframerate()
    ampWidth = spf.getsampwidth()
    nChannels = spf.getnchannels()
    nFrames = spf.getnframes()

    # Extract Raw Audio from multi-channel Wav File
    signal = spf.readframes(nFrames*nChannels)
    spf.close()
    channels = interpret_wav(signal, nFrames, nChannels, ampWidth, True)

    # get window size
    fqRatio = (cutOffFrequency/sampleRate)
    N = int(math.sqrt(0.196196 + fqRatio**2)/fqRatio)

    # Use moviung average (only on first channel)
    filt = run_mean(channels[0], N).astype(channels.dtype)

    wav_file = wave.open(outname, "w")
    wav_file.setparams((1, ampWidth, sampleRate, nFrames, spf.getcomptype(), spf.getcompname()))
    wav_file.writeframes(filt.tobytes('C'))
    wav_file.close()
n = 0
for n in range (0,2):
    if n==0:
        fft_dis(fname)
    elif n==1:
        fft_dis(outname)
```

**Output:**



```
Source: sample.wav
Freq: 250Hz
Recorded On: iPhone (Apple Inc.)
Format: .wav
Avg Wavelenght: 12nm
Pitch: 15db
Loudness: 2500db
Amplitude: 25m
Velocity: 1.2m/s
Audio Pressure: 12Pa
Intensity: 1.4watt/m2
Stereo: True
Spectrum: 4m
Intermodulation Distortion (MD): 50ms
C:\Users\D_COMP_SLI_17\Downloads>
```



| | |
|---|---|
| **Date:** | |
| **Marks obtained:** | |
| **Sign of course coordinator:** | |
| **Name of course Coordinator :** | |