```
## install speechrecognition module in python for converting speech to text and vice versa
!pip3 install SpeechRecognition pydub
```

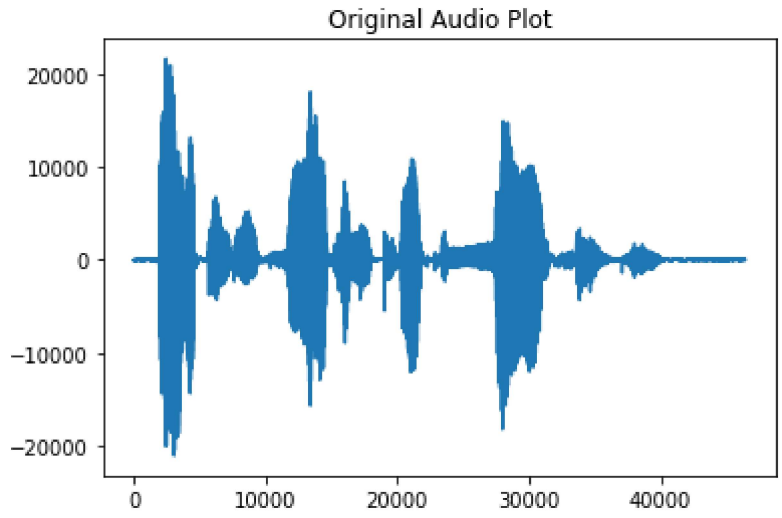
```
Collecting SpeechRecognition
  Downloading https://files.pythonhosted.org/packages/26/e1/7f5678cd94ec1234269d23756
     |████████████████████████████████| 32.8MB 114kB/s
Collecting pydub
  Downloading https://files.pythonhosted.org/packages/a6/53/d78dc063216e62fc55f6b2eeb
Installing collected packages: SpeechRecognition, pydub
Successfully installed SpeechRecognition-3.8.1 pydub-0.25.1
```

```
## import necessary modules
import speech_recognition as sr
import numpy as np
import binascii
import IPython
from scipy.io import wavfile
import matplotlib.pyplot as plt


#read the audio file into the code
filename = "machine-learning_speech-recognition_16-122828-0002.wav"
IPython.display.Audio(filename)
```

0:00 / 0:02

```
fs, data = wavfile.read('machine-learning_speech-recognition_16-122828-0002.wav')
plt.plot(data)            # fs = sampling frequency = 44.1kHz
plt.title("Original Audio Plot")
```

```
Text(0.5, 1.0, 'Original Audio Plot')
```




```
r = sr.Recognizer()
texxt = "";
```

```
## Convert audio to text using google voice recognition
with sr.AudioFile(filename) as source:
    # listen for the data (load audio to memory)
    audio_data = r.record(source)
    # recognize (convert from speech to text)
    text = r.recognize_google(audio_data)
    texxt += text;
    print(text)
```

```
 I believe you're just talking nonsense
```

```
## store the converted text as a string
txt = str(texxt)
print(txt)
```

```
 I believe you're just talking nonsense
```

```
## Hepler functions for interconversion of bits and strings
## It converts bits to strings and vice versa
def text_to_bits(text, encoding='utf-8', errors='surrogatepass'):
    bits = bin(int(binascii.hexlify(text.encode(encoding, errors)), 16))[2:]
    return bits.zfill(8 * ((len(bits) + 7) // 8))

def text_from_bits(bits, encoding='utf-8', errors='surrogatepass'):
    n = int(bits, 2)
    return int2bytes(n).decode(encoding, errors)

def int2bytes(i):
    hex_string = '%x' % i
    n = len(hex_string)
    return binascii.unhexlify(hex_string.zfill(n + (n & 1)))
```

```
## Main building blocks of Grain Cipher, namely a Linear Feedback Shift Register (LFSR)
## and a Non-linear Feedback Shift Register (NFSR)

lfsr = np.zeros(80,dtype=bool)
nfsr = np.zeros(80,dtype=bool)
```

```
## Initialise LFSR and NFSR using IV and Secret Key
## First load the NFSR with the key bits, bi = ki, 0 ≤ i ≤ 79,
## then load the first 64 bits of the LFSR with the IV, si = IVi, 0 ≤ i ≤ 63.
## The remaining bits of the LFSR are filled with ones, si = 1, 64 ≤ i ≤ 79.
## Because of this the LFSR cannot be initialized to the all zero state.

def init(iv,key):
    iv_bin = text_to_bits(iv)
    iv_bin = ''.join(iv_bin)
    lfsr[:64] = [bool(int(iv_bin[ix])) for ix in range(len(iv_bin))]
    lfsr[64:] = 1
    key_bin = text_to_bits(key)
    key_bin = ''.join(key_bin)
    nfsr[:] = [bool(int(key_bin[ix])) for ix in range(len(key_bin))]
```

```
init('absolute','california')


## The cipher is clocked 160 times without producing any running key
## The output of the filter function, h(x), is fed back and xored with the input, both to

def clock():
    hx=0
    fx=0
    gx=0
    global lfsr
    global nfsr
    for ix in range(160):
        fx = lfsr[62] ^ lfsr[51] ^ lfsr[38] ^ lfsr[23] ^ lfsr[13] ^ lfsr[0] ^ hx
        gx = hx ^ nfsr[0] ^ nfsr[63] ^ nfsr[60] ^ nfsr[52] ^ nfsr[45] ^ nfsr[37] ^ nfsr[33
        x0 = lfsr[0]
        x1 = lfsr[25]
        x2 = lfsr[46]
        x3 = lfsr[64]
        x4 = nfsr[63]
        hx = x1 ^ x4 ^ x0 & x3 ^ x2 & x3 ^ x3 & x3 ^ x0 & x1 & x2 ^ x0 & x2 & x3 ^ x0 & x2
        lfsr[:-1] = lfsr[1:]
        lfsr[-1] = fx
        nfsr[:-1] = nfsr[1:]
        nfsr[-1] = gx


clock()


## Return a stream generator which implements the filter function

def gen_key_stream():
    hx = 0
    while True:
        fx = lfsr[62] ^ lfsr[51] ^ lfsr[38] ^ lfsr[23] ^ lfsr[13] ^ lfsr[0]
        gx = nfsr[0] ^ nfsr[63] ^ nfsr[60] ^ nfsr[52] ^ nfsr[45] ^ nfsr[37] ^ nfsr[33] ^ n
        x0 = lfsr[0]
        x1 = lfsr[25]
        x2 = lfsr[46]
        x3 = lfsr[64]
        x4 = nfsr[63]
        hx = x1 ^ x4 ^ x0 & x3 ^ x2 & x3 ^ x3 & x3 ^ x0 & x1 & x2 ^ x0 & x2 & x3 ^ x0 & x2
        lfsr[:-1] = lfsr[1:]
        lfsr[-1] = fx
        nfsr[:-1] = nfsr[1:]
        nfsr[-1] = gx
        yield hx


## define a function which will take iv, key and plain text as input and convert excrpt it
def encrypt(iv,key,plain):
    init(iv,key)
    clock()
    plain = text_to_bits(plain)
    stream = gen_key_stream()
```

```
    stream = gen_key_stream()
    cipher = [str(int(bool(int(plain[ix]))^next(stream))) for ix in range(len(plain))]
    cipher = ''.join(cipher)
    return cipher


## storing the ciphertext in cipher variable
cipher = encrypt('absolute','california',txt)


print (cipher)
bin_exc_string = str(cipher)
```

```
1010100000101110011000100000011100110011010000010101010000110000101111010001001101111
```

```
## converting ciphertext into excrypted string
ascii_string = ""
for binary_value in bin_exc_string:
    an_integer = int(binary_value, 2)
    ascii_character = chr(an_integer)
    ascii_string += ascii_character
print(ascii_string)
```

```
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
```

```
## a function to decrpyt the ciphertext back into the orignial string
def decrypt(iv,key,cipher):
    init(iv,key)
    clock()
    stream = gen_key_stream()
    plain = [str(int(bool(int(cipher[ix]))^next(stream))) for ix in range(len(cipher))]
    plain = ''.join(plain)
    plain = text_from_bits(plain)
    return plain


plain = decrypt('absolute','california',cipher)


print (plain)
```

```
I believe you're just talking nonsense
```

```
excrypted_string = ascii_string
decrypted_string = plain


!pip3 install gTTS pyttsx3 playsound
```

```
Collecting gTTS
  Downloading https://files.pythonhosted.org/packages/5f/b9/94e59337107be134b21ce395a
Collecting pyttsx3
  Downloading https://files.pythonhosted.org/packages/33/9a/de4781245f5ad966646fd2762
Collecting playsound
```

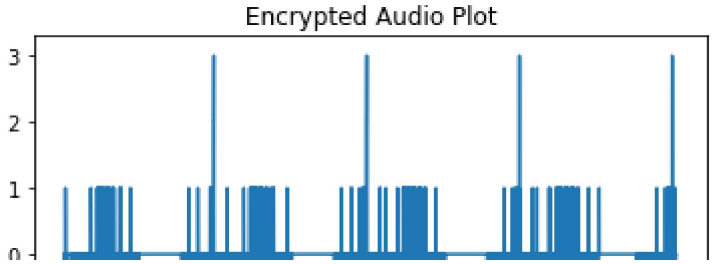
```
  Downloading https://files.pythonhosted.org/packages/f5/16/10d897b0a83fb4b05b03a63d7
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from gT
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: click in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/
Installing collected packages: gTTS, pyttsx3, playsound
Successfully installed gTTS-2.2.2 playsound-1.2.2 pyttsx3-2.90
```

```
import gtts
from playsound import playsound


## encrypted_audio stores the encrypted audio
## original_audio stores the decrypted audio
encrypted_audio = gtts.gTTS(excrypted_string)
original_audio = gtts.gTTS(decrypted_string)


## saves the audio files to the current directory
encrypted_audio.save("encrypted.mp3")
original_audio.save("output.mp3")

enc_aud = "encrypted.mp3"
out_aud = "output.mp3"


from os import path
from pydub import AudioSegment


# for encrypted
src = "encrypted.mp3"
dst = "encrypted.wav"


sound = AudioSegment.from_mp3(src)
sound.export(dst, format="wav")
```

```
<_io.BufferedRandom name='encrypted.wav'>
```

```
## Plotting graph of encrypted voice audio
fs, data = wavfile.read('encrypted.wav')
plt.plot(data)            # fs = sampling frequency = 44.1kHz
plt.title("Encrypted Audio Plot")
```

```
Text(0.5, 1.0, 'Encrypted Audio Plot')
```




```
## excrypted audio
IPython.display.Audio('encrypted.wav')
```

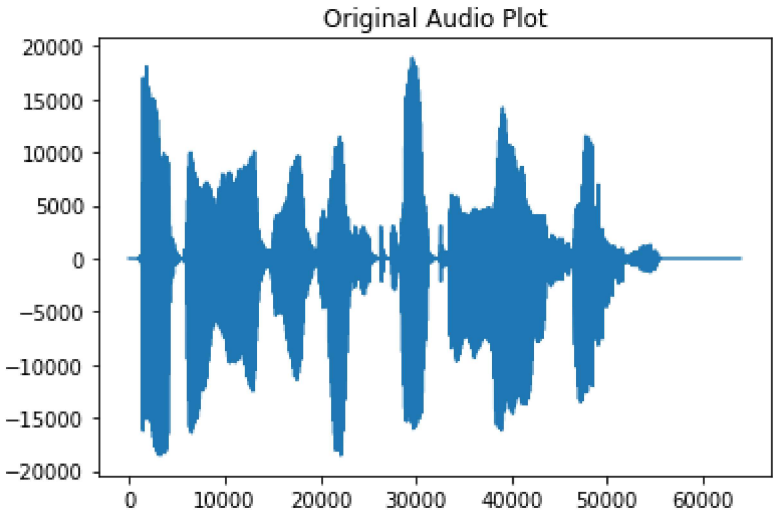
0:00 / 0:01

```
# for output
src = "output.mp3"
dst = "output.wav"


sound = AudioSegment.from_mp3(src)
sound.export(dst, format="wav")
```

```
<_io.BufferedRandom name='output.wav'>
```

```
fs, data = wavfile.read('output.wav')
plt.plot(data)            # fs = sampling frequency = 44.1kHz
plt.title("Original Audio Plot")
```

```
Text(0.5, 1.0, 'Original Audio Plot')
```




```
## output audio
IPython.display.Audio(out_aud)
```

0:00 / 0:02

✓ 0s completed at 9:30 AM