



NOISY

- Analyzing encrypted WAV files with Python
- Ajay Balguri
- Kanaka Durga Grandhi
- Anvitha Kanukuntla
- Naveen Kumar



OVERVIEW

Main agenda is to decrypt the hidden message from a WAV file.

Tools used

- `scipy`
- `Matplotlib`
- `numpy`.

TOOLS AND PYTHON LIBRARY

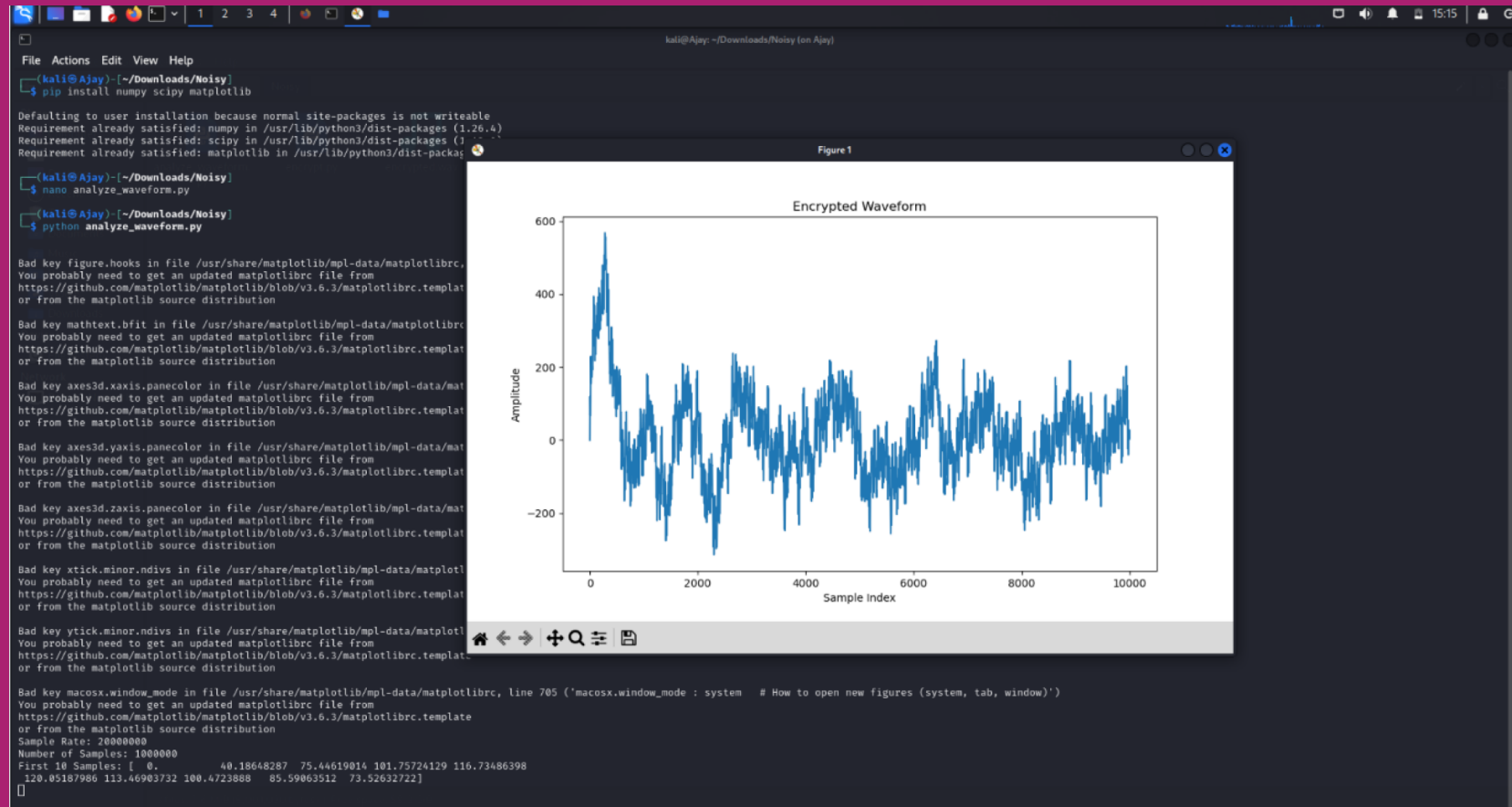
The **scipy** Python library is a comprehensive ecosystem for scientific and technical computing. It provides a variety of modules and functions that facilitate tasks in mathematics, science, and engineering.

Matplotlib is a widely used Python library for creating static, interactive, and animated visualizations. It provides tools to generate plots, graphs, and charts to help visualize data effectively.

NumPy is a powerful Python library primarily used for numerical computing. It provides support for working with large, multi-dimensional arrays and matrices, along with a vast collection of mathematical functions to operate on these arrays efficiently.

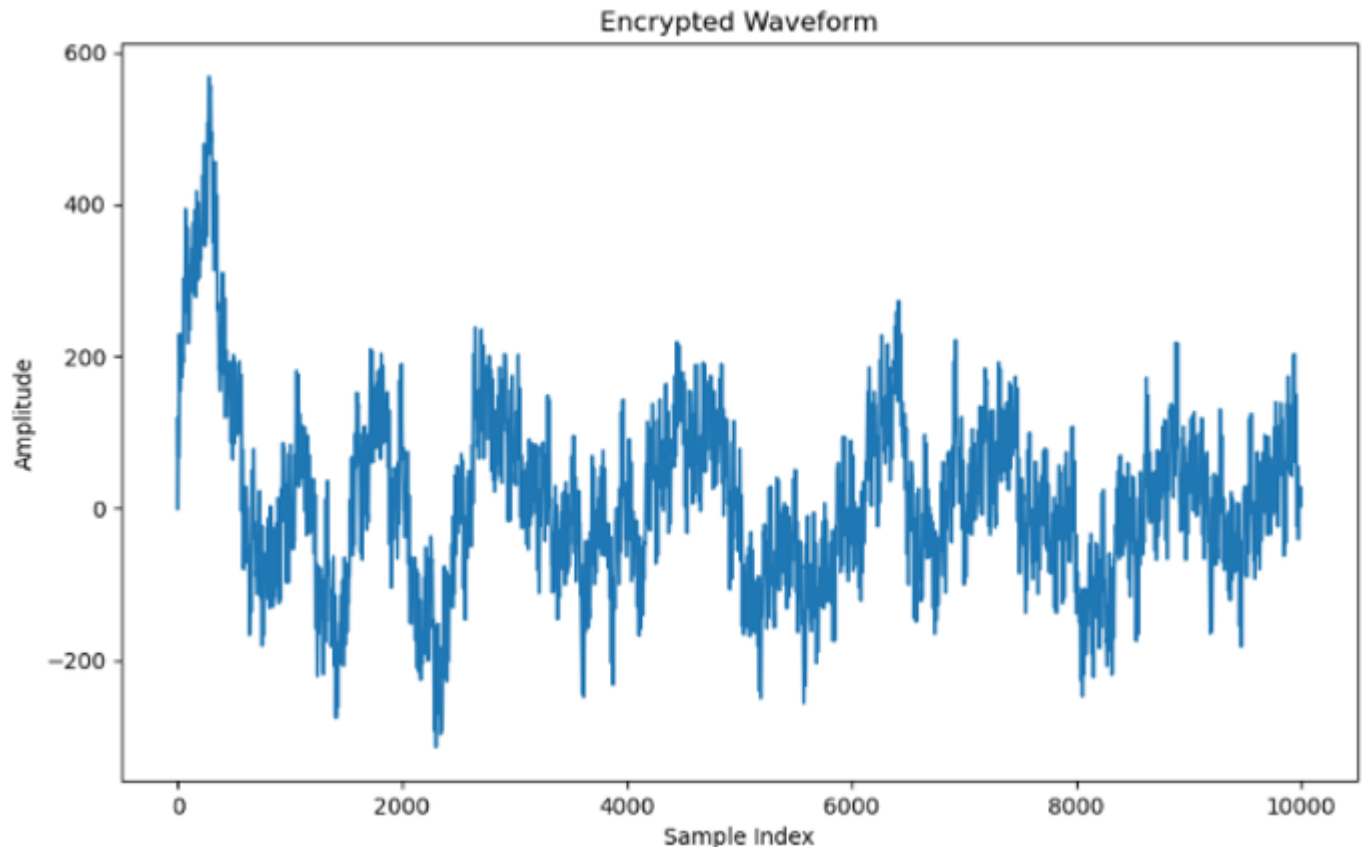


ANALYZING THE WAV FILE



VISUALIZING THE WAVEFORM

```
plt.figure(figsize=(10, 6))  
plt.plot(data[:10000]) # Plot the first 10,000 samples for visualization  
plt.title("Encrypted Waveform")  
plt.xlabel("Sample Index")  
plt.ylabel("Amplitude")  
plt.show()
```



PERFORMING FFT ON DATA

```
import numpy as np
```

```
from scipy.fftpack import fft
```

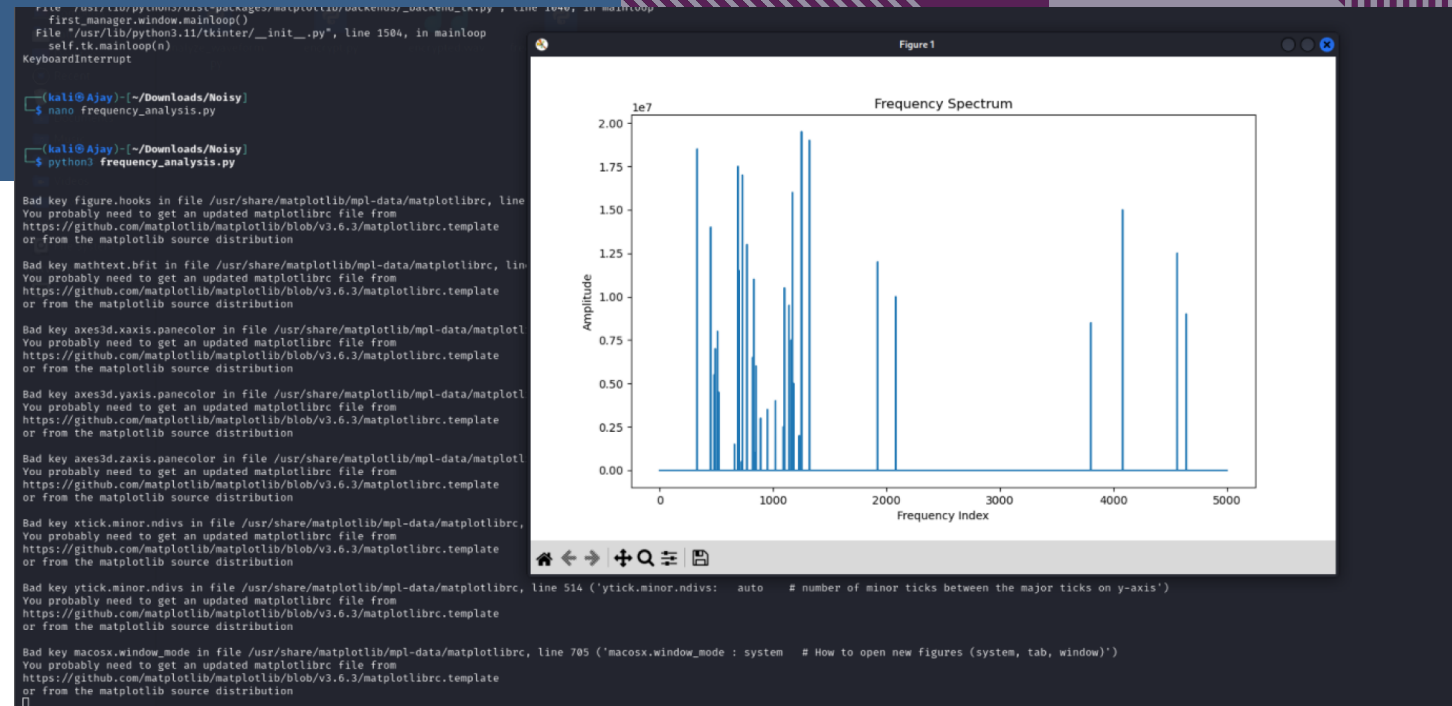
```
from scipy.io.wavfile import read
```

```
# Step 1: Load the encrypted WAV file
```

```
rate, data = read("encrypted.wav")
```

```
# Step 2: Perform FFT on the data
```

```
frequencies = np.abs(fft(data))
```



VISUALIZING FREQUENCY SPECTRUM

```
plt.figure(figsize=(10, 6))

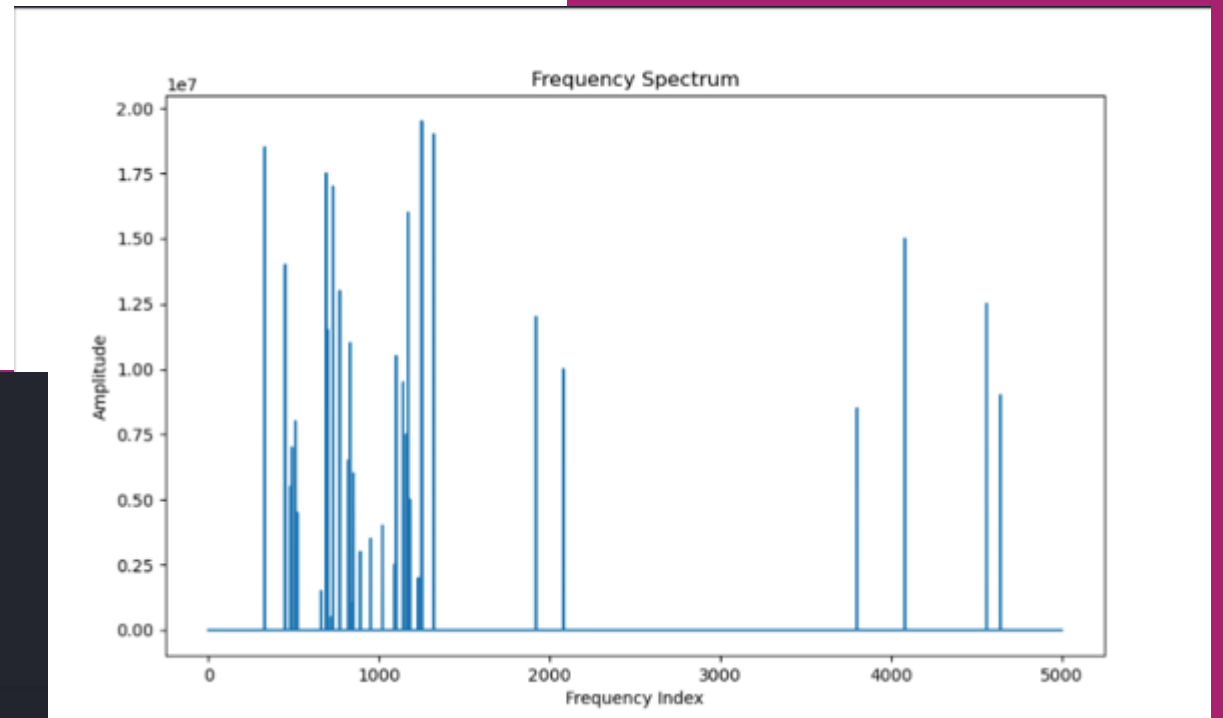
plt.plot(frequencies[:5000]) # Plot the first 5000 frequency components

plt.title("Frequency Spectrum")

plt.xlabel("Frequency Index")

plt.ylabel("Amplitude")

plt.show()
```



```
(kali@Ajay) ~/Downloads/Noisy
$ nano frequency_analysis2.py

(kali@Ajay) ~/Downloads/Noisy
$ python3 frequency_analysis2.py
Dominant Frequencies (Hz): [1.99750e+07 2.50000e+04 1.99736e+07 2.64000e+04 6.60000e+03 1.99934e+07
1.45920e+06 1.85408e+07 1.38000e+04 1.99862e+07]
Corresponding Amplitudes: [19500000.0000001 19500000.0000001 18999999.9999999 18999999.9999999
18499999.9999999 18499999.9999999 17999999.9999999 17999999.9999999
17500000.00000085 17500000.00000085]

(kali@Ajay) ~/Downloads/Noisy
$
```

EXTRACTING DOMINANT FREQUENCIES

```
# Find the indices of the 10 most significant frequencies

dominant_indices = np.argsort(-frequencies)[:10] # Top 10 frequencies
dominant_amplitudes = frequencies[dominant_indices] # Get their amplitudes


# Convert indices to frequencies

dominant_frequencies = dominant_indices * rate / len(data) # Frequency in Hz


# Print results

print("Dominant Frequencies (Hz):", dominant_frequencies)
print("Corresponding Amplitudes:", dominant_amplitudes)
```




DECRYPTING THE MESSAGE

Character Mapping Logic:

1. Frequency Formula:

$$f = (i + 1) \cdot \sin(2\pi x \cdot \text{multiplier})$$

- $i + 1$: Index of the character.
- **multiplier**: Proportional to the character count.

2. Decode Each Character:

- Divide the frequency by $\text{rate}/\text{len}(\text{data})$.
- Map the result to characters.

DISCRETE SINE TRANSFORM (DST):

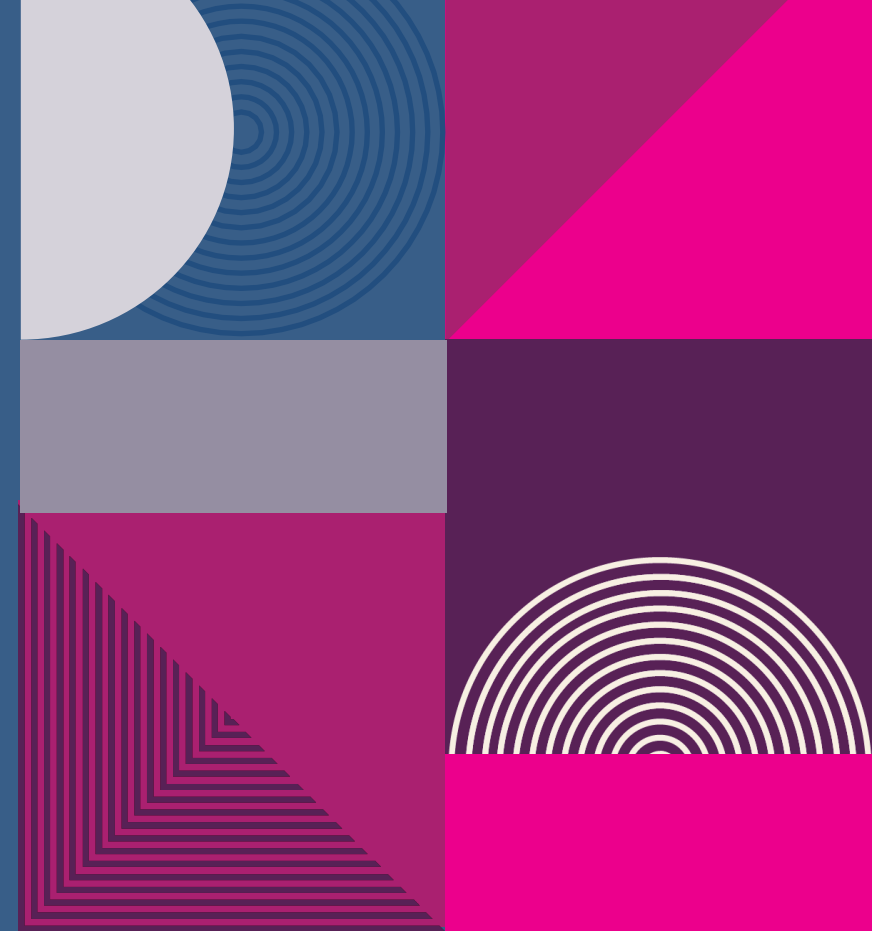
- `dst(final_waveform[1])` performs a Discrete Sine Transform on the waveform data
- The DST is similar to the Discrete Fourier Transform (DFT) but uses sine functions instead of exponential functions. It converts the time-domain signal into its frequency-domain representation.
- The result, `result`, is a sequence of frequency-domain coefficients.

The mathematical formula for DST of a sequence x_n is:

$$X_k = \sum_{n=1}^N x_n \sin \left(\frac{\pi k(n+1)}{N+1} \right), \quad k = 1, 2, \dots, N$$

Here:

- x_n is the input time-domain signal (amplitude values).
- X_k is the frequency-domain representation.
- N is the length of the input signal.



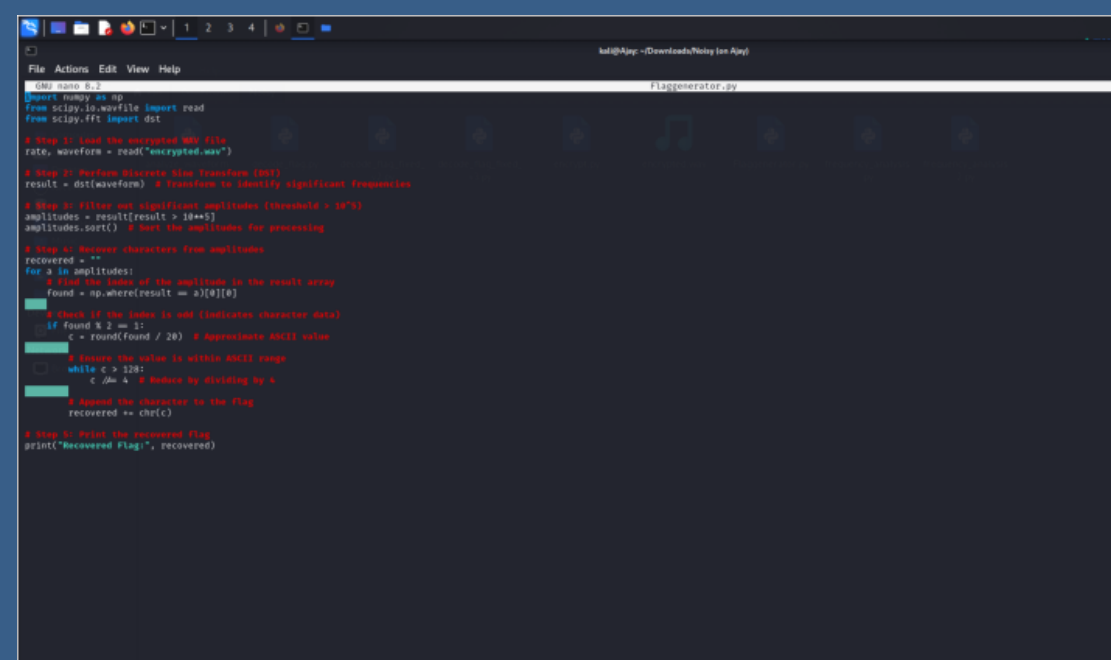


CONCLUSION

- First we have loaded the waveform and then visualized it.
- We then have used fast fourier transform on the data.
- Now visualize the frequency spectrum and extract dominant frequency.
- Now map the result to characters and we finally will have the key.

FLAG

HTB{mY_f4v0UR1t3_tr4nSF0rM_-_f0urIEr!!}



```
kali@Ajay: ~/Downloads/Noisy (on Ajay)
File Actions Edit View Help
Flaggenerator.py
import numpy as np
from scipy.io.wavfile import read
from scipy.fft import fft

# Step 1: Load the encrypted WAV file
rate, waveform = read("encrypted.wav")

# Step 2: Perform Discrete Sine Transform (DST)
result = dst(waveform) # Transform to identify significant frequencies

# Step 3: Filter out significant amplitudes (threshold > 10%)
amplitudes = result[result > 10**5]
amplitudes.sort() # Sort the amplitudes for processing

# Step 4: Recover characters from amplitudes
recovered = ""
for a in amplitudes:
    # Find the index of the amplitude in the result array
    found = np.where(result == a)[0][0]

    # Check if the index is odd (indicates character data)
    if found % 2 == 1:
        c = round(found / 20) # Approximate ASCII value

        # Ensure the value is within ASCII range
        while c > 128:
            c -= 1 # Reduce by dividing by 4

        # Append the character to the flag
        recovered += chr(c)

# Step 5: Print the recovered flag
print("Recovered Flag:", recovered)
```

```
(kali@Ajay)-[~/Downloads/Noisy]
$ nano Flaggenerator.py
```

```
(kali@Ajay)-[~/Downloads/Noisy]
$ python3 Flaggenerator.py
```

Recovered Flag: HTB{mY_f4v0UR1t3_tr4nSF0rM_-_f0urIEr !! }

```
(kali@Ajay)-[~/Downloads/Noisy]
$
```



THANK YOU

Group 7