Laboratorium Bezpieczeństwa Systemów Teleinformatycznych

Generatory liczb losowych

Podstawa opracowania:

- J. S. Teh, W. Teng and A. Samsudin, "A true random number generator based on hyperchaos and digital sound," 2016 3rd International Conference on Computer and Information Sciences (ICCOINS), Kuala Lumpur, Malaysia, 2016, pp. 264-269, doi: 10.1109/ICCOINS.2016.7783225.

Systematyczny przegląd literatury:

- Podczas wyboru publikacji do implementacji kierowaliśmy się jej przejrzystością. Sprawdzaliśmy czy dana publikacja zawiera pseudokod lub schemat blokowy.

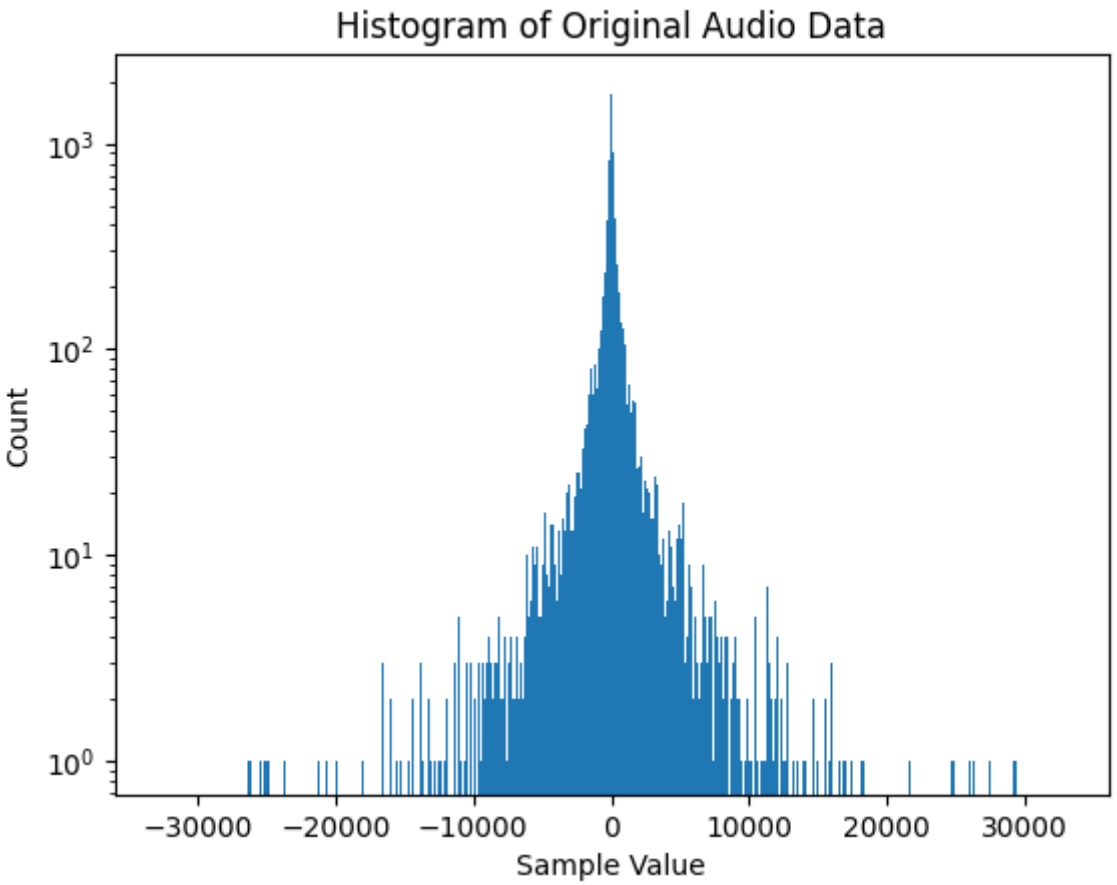
- W etapie wstępnym wytypowaliśmy trzy publikacje:
    - J. S. Teh, W. Teng and A. Samsudin, "A true random number generator based on hyperchaos and digital sound," 2016 3rd International Conference on Computer and Information Sciences (ICCOINS), Kuala Lumpur, Malaysia, 2016, pp. 264-269, doi: 10.1109/ICCOINS.2016.7783225.

    - Chen, I-Te. “Random Numbers Generated from Audio and Video Sources.” Mathematical Problems in Engineering 2013 (2013): 1-7.

    - Galajda, Pavol. “CHAOS-BASED TRUE RANDOM NUMBER GENERATOR EMBEDDED IN A MIXED-SIGNAL RECONFIGURABLE HARDWARE.” (2006).

Analiza źródła entropii:

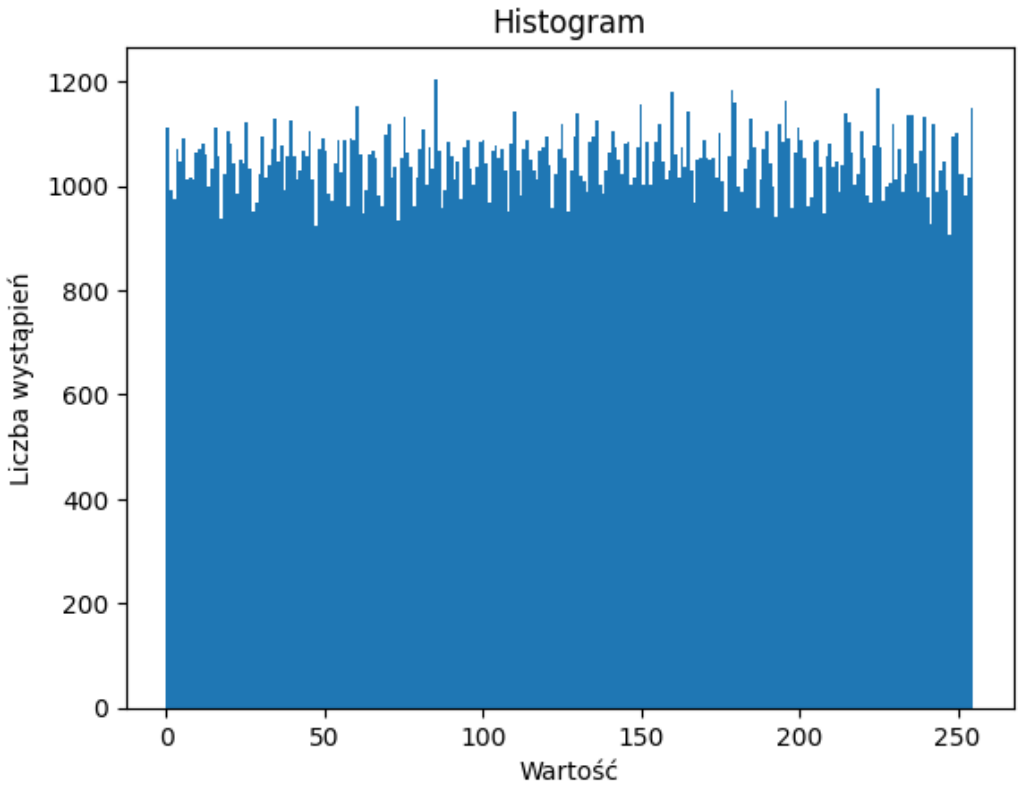
- Algorytm wykorzystuje ówcześnie nagrane pliki Waveform Audio Interface (WAV) zawierające ścieżkę dźwiękową z kanałami audio stereo. W pracy wykorzystano próbki dźwięku nagrane z częstotliwością próbkowania 44,1 kHz, rejestrując 44100 próbek na sekundę zapisywanych jako zmienne 16-bitowe.



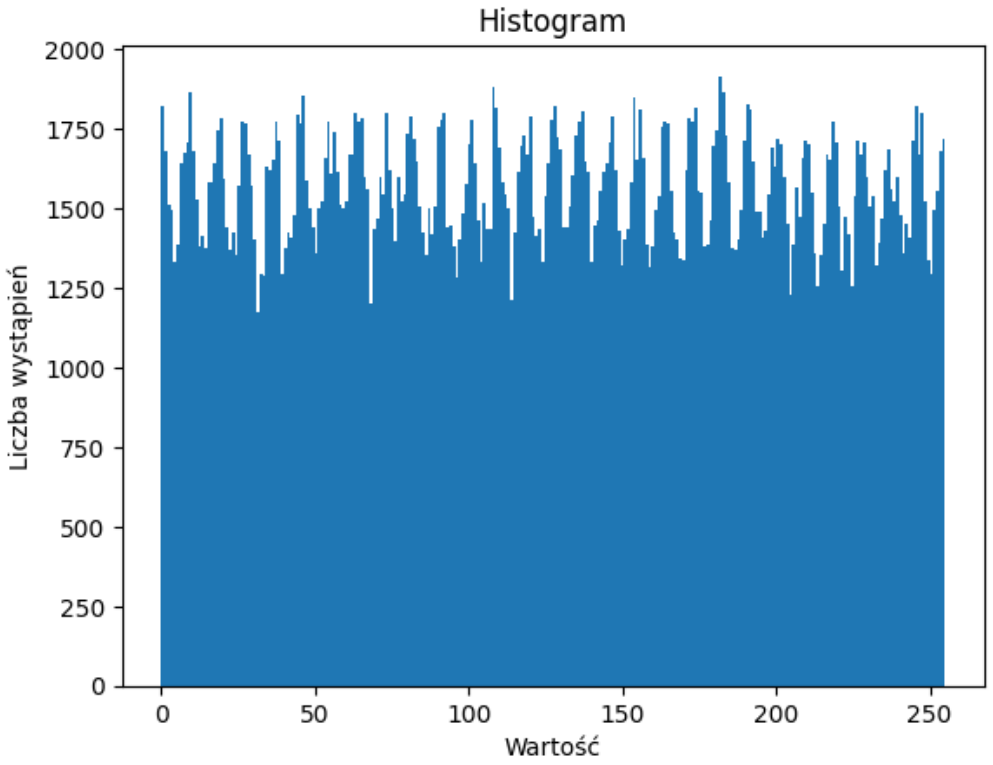
Metoda poprawy właściwości statystycznych:

- Główne źródło szumu ograniczono do trzech najmniej znaczących bitów każdej próbki. W raporcie dodatkowo zamieszczone są wyniki dla dwóch najmniej znaczących bitów pozostawiając jedno zero wiodące.

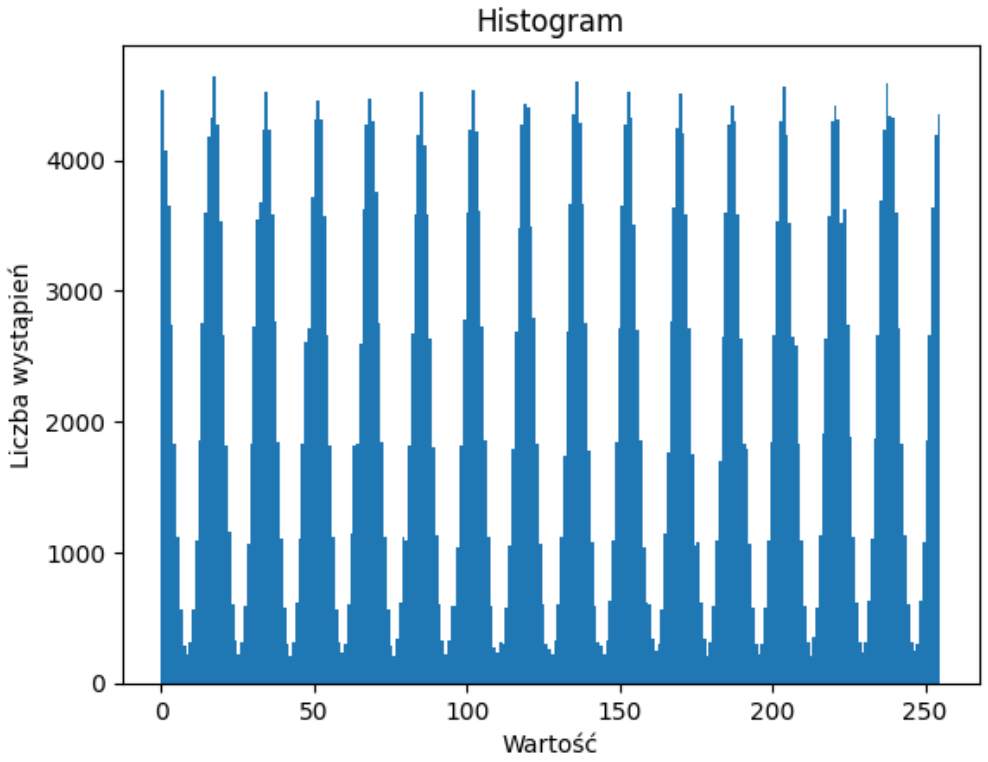
- Poniżej przedstawiamy także rozkład danych wejściowych dla zlepionych w 8 bitowe paczki trzech konfiguracji (najmłodszych bitów): dwa, trzy i cztery.



Dla 2 bitów, entropy = 7.998027813825241



Dla 3 bitów, entropy = 7.992313230925775

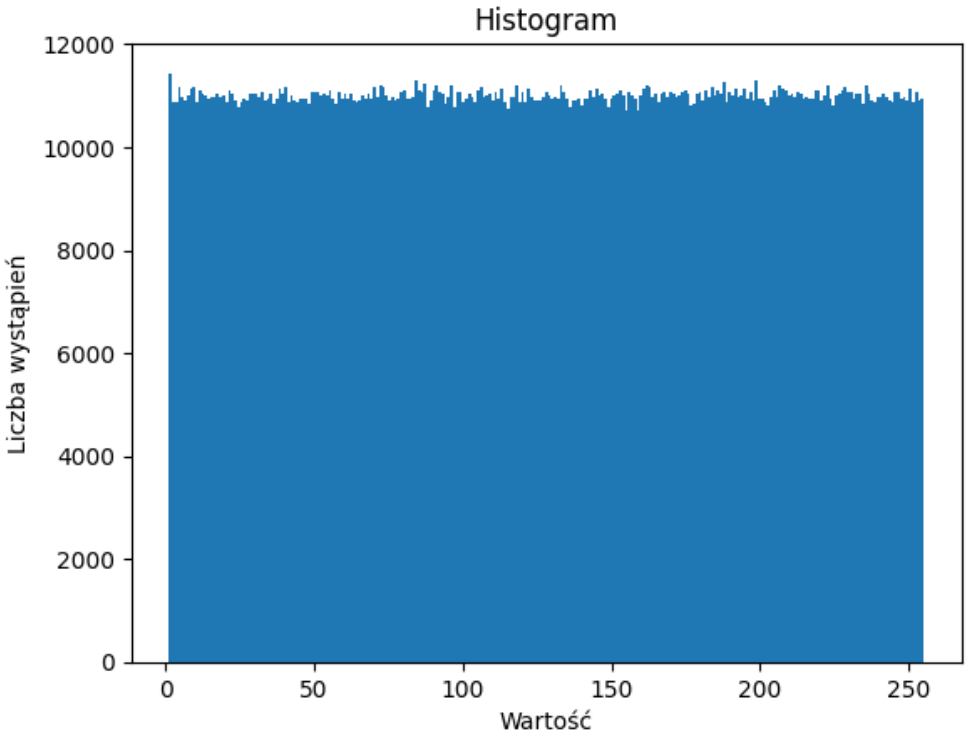


Dla 4 bitów, entropy = 7.5832553283376605

Jako drugi krok wykorzystywaną metodą prowadzącą do lepszej entropii jest wyeliminowanie efektu transjentu na początku nagrania poprzez nie wykorzystywanie pierwszych 10 000 próbek.

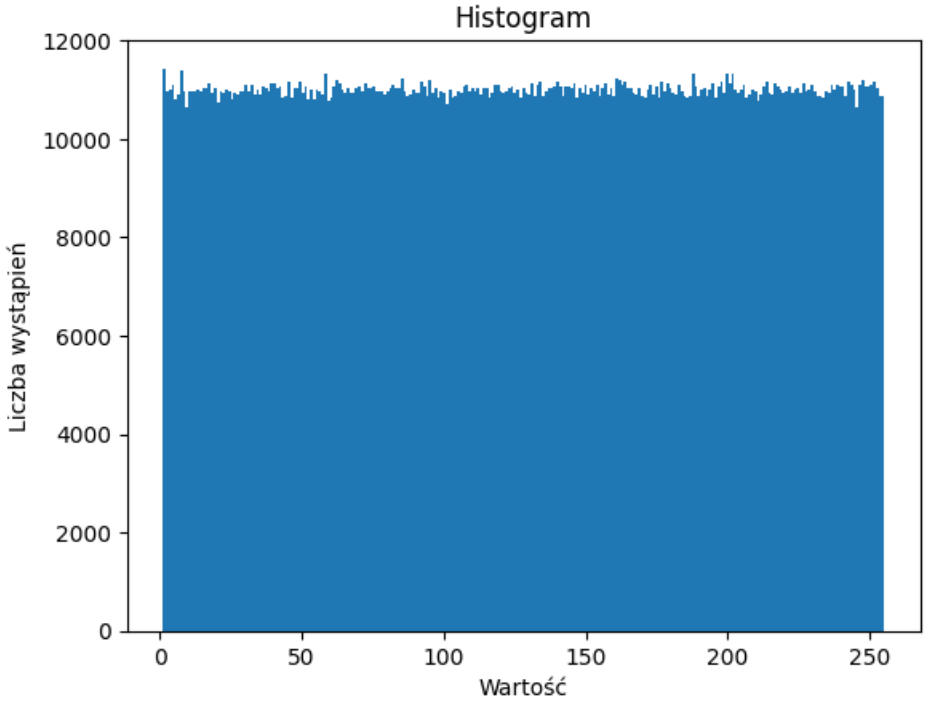
Następnie wykorzystywana jest mapa chaotyczna na pozostałych próbkach aż do uzyskania oczekiwanej liczby losowych elementów:

$$f_T(x) = \begin{cases} \alpha x & 0 \leq x < 0.5 \\ \alpha(1-x), & 0.5 \leq x \leq 1, \end{cases}$$



Empiryczny rozkład zmiennych losowych po post-processingu dla 3 bitów

Entropia wyliczona zgodnie ze wzorem: $e = -\sum i\ pi\ log2\ (pi\ )$, dla powyższego rozkładu wynosi 7.99428189705125 bita przy 256000 próbek. Entropia różni się w zależności od ilości próbek (przy wzroście ilości próbek entropia rośnie).



Empiryczny rozkład zmiennych losowych po post-processingu dla 2 bitów wejściowych z 0 wiodącym

Entropia wyliczona zgodnie ze wzorem: $e = -\sum i\ pi\ log2\ (pi\ )$, dla powyższego rozkładu wynosi 7.994272711757638 bita przy 256000 próbek. Entropia różni się w zależności od ilości próbek (przy wzroście ilości próbek entropia rośnie).

Uwagi:

- Po post-processingu w tablicy uzyskanych danych występuje bardzo duża ilość wartości 0, obcinamy ją żeby uzyskać jak najwyższą entropię. Mocną stroną kodu jest stosunkowo dobre i równomierne rozmieszczenie wartości, jednakże możemy to zauważyć dopiero przy N > 1000. Poniżej tej wartości oznaczającej przebieg nie uzyskujemy tak “równego” histogramu
- Dla zadanych wyżej wartości (N=256000 i wchodzących 3 najmniej znaczących bitów) z pliku o wielkości 2 144 668 B uzyskujemy długość tablicy z próbkami na poziomie 2 805 431, każda wartość z tablicy to 8 bitów czyli jeden bajt co wskazuje na to, że wielkość tej tablicy to 2 805 431 B
- Najlepszą entropię danych wejściowych otrzymujemy po utworzeniu próbek 8-bitowych z 2 najmniej znaczących bitów, a najgorszą dla 4 bitów.

Kod programu:

```
import wave
import matplotlib.pyplot as plt
import struct
import numpy as np



def entropy1(labels, base=None):
 value,counts = np.unique(labels, return_counts=True)
 norm_counts = counts / counts.sum()
 base = 2 if base is None else base
 return -(norm_counts * np.log(norm_counts)/np.log(base)).sum()

def swap_bits(num):
   # Swap the 32 most significant bits with the 32 least significant bits
   msb = num >> 32
   lsb = num & ((1 << 32) - 1)
   swapped_num = (lsb << 32) | msb
   # Perform XOR operation to increase randomness
   result = swapped_num ^ num
   return result

def hist_input(A, n = 3):
   if n == 2:
       r = []
       mask = 0b00000011
```

```
        for v in A:
            r.append(v & mask)

        r_new = []
        for i in range(0, len(r), 4):
            byteFromBts = 0
            byteFromBts = r[i] << 6 | r[i+1] << 4 | r[i+2] << 2 | r[i+3]
            r_new.append(byteFromBts)

    elif n == 3:
        r = []
        mask = 0b00000111
        for v in A:
            r.append(v & mask)

        r24 = []
        for i in range(0, len(r), 8):
            temp = r[i] << 21 | r[i+1] << 18 | r[i+2] << 15 | r[i+3] << 12 |
r[i+4] << 9 | r[i+5] << 6 | r[i+6] << 3 | r[i+7]
            r24.append(temp)

        r_new = []
        for i in range(len(r24)):
            r_new.append(r24[i] & 0b111111110000000000000000 >> 16)
            r_new.append(r24[i] & 0b000000001111111100000000 >> 8)
            r_new.append(r24[i] & 0b000000000000000011111111)

        for x in r_new:
            if x == 0:
                r_new.pop(x)


    elif n==4:
        r = []
        mask = 0b00001111
        for v in A:
            r.append(v & mask)

        r_new = []
        for i in range(0, len(r), 2):
            byteFromBts = 0
            byteFromBts = r[i] << 4
            byteFromBts = byteFromBts | r[i+1]
            r_new.append(byteFromBts)
    else:
```

```
        raise ValueError("n must be in range of [2, 4]")

    # Tworzenie histogramu
    plt.hist(r_new, 256)

    # Tytuł wykresu i etykiety osi
    plt.title("Histogram")
    plt.xlabel("Wartość")
    plt.ylabel("Liczba wystąpień")

    # Wyświetlanie wykresu
    plt.show()

    return r



# Open the wave file and read the binary data
w = wave.open("sound-samples/test.wav", "rb")
if w is None:
    print("Error: Unable to open file")
    exit()


nchannels = w.getnchannels()
sample_width = w.getsampwidth()
framerate = w.getframerate()
nframes = w.getnframes()

# Read the audio data as a string of bytes
audio_data = w.readframes(nframes)

# Convert the audio data to a numpy array
audio_data = np.frombuffer(audio_data, dtype=np.uint16)

w.close()

#podajemy parametry
L = 8 #edytujemy L
N = 256000/2 #edytujemy N
gamma = 2
epsilon = 0.1
alpha = 1
n = int(N / 256 * 8)
```

```
A = []
for byte in audio_data:
   A.append(byte)

# check if audio file is long enough
if len(A) < n:
   raise ValueError("Audio file is too short")

hist_input(A, 2)
hist_input(A, 3)
hist_input(A, 4)

r = []
mask = 0b00000111
for v in A:
   r.append(v & mask)

x = [[0.141592, 0.653589, 0.793238, 0.462643, 0.383279, 0.502884, 0.197169,
0.399375]]
c = 0

def fT(x, alpha):
   if 0 <= x < 0.5:
       return alpha * x
   elif 0.5 <= x <= 1:
       return alpha * (1 - x)
   else:
       raise ValueError("x must be in range of [0, 1]")

z = [0,0,0,0,0,0,0,0]
O = []
y = 0

while len(O) <= N:
   for i in range(L):
       t = len(x)-1
       x[t][i] = ((0.071428571 * r[y]) + x[t][i]) * 0.666666667
       c += 1
   for t in range(gamma):
       for i in range(L):
           try:
               x[t+1][i] = (1 - epsilon) * fT(x[t][i], alpha) + epsilon/2 *
(fT(x[t][(i+1)%L], alpha)) + fT(x[t][(i-1)%L], alpha)
           except:
               x.append([0,0,0,0,0,0,0,0])
```

```
                x[t+1][i] = (1 - epsilon) * fT(x[t][i], alpha) + epsilon/2 *
(fT(x[t][(i+1)%L], alpha)) + fT(x[t][(i-1)%L], alpha)
    for i in range(L):
        word = struct.pack('d', x[2][i])
        # Konwersja ciągu bajtów na wartość uint64
        int_value = int.from_bytes(word, byteorder='big', signed=False)
        z[i] = int_value
        x[0][i] = x[2][i]
    for i in range(int(L/2)):
        z[i] = int(z[i]) ^ swap_bits(int(z[i+int(L/2)]))
    O.append(z[0] + z[1]*256 + z[2]*pow(2,16) + z[3]*pow(2,24))
    y+=1

# Pobieranie po 8 bitów z kazdej próbki wyjściowej
bajty_z_O = []
for j in range(len(O)-1):
    for i in range(0, 256, 8):
        bajt = (O[j] >> (256 - (i + 8))) & 0xFF
        if(bajt != 0 ):
            bajty_z_O.append(bajt)

# Tworzenie histogramu
plt.hist(bajty_z_O, 256)

# Tytuł wykresu i etykiety osi
plt.title("Histogram")
plt.xlabel("Wartość")
plt.ylabel("Liczba wystąpień")

# Wyświetlanie wykresu
plt.show()

# Liczenie entropi
e = entropy1(bajty_z_O, base=2)
print(e)
```