EE049017: Hardware Systems Security

Homework Assignment 1

Faculty of Electronic Engineering, Technion.

Submitted by:

#	Name	Id	email
Student 1	Kfir Girstein	316541218	kfirgirstein@campus.technion
Student 2	Alon Berkenstadt	205710205	alon.b@campus.technion.ac.il

Contents

The answers notebooks can be obtained by cloning the HW repo Hardware_Security_HW_049017 (https://github.com/kfirgirstein/Hardware_Security_HW_049017/tree/main/HW/HW or viewed in your browser by using nbviewer. and a notebook nbviewer (https://nbviewer.jupyter.org/github/kfirgirstein/Hardware_Security_HW_049017/

HW1 - part 1

In [16]:

```
import sys, random,binascii,AES,jupyter_utils
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from IPython.display import Markdown, display,HTML
%matplotlib inline
plt.rcParams['figure.dpi'] = 150
```

We would like to test these principles when operating AES in different situations Note that for the purpose of the exercise, we will allow ourselves a number of mitigating assumptions which will be detailed below

To do this, take the following text which was published today on the BBC website

In [17]:

```
with open("plaintext.txt","r") as f:
    plaintext = f.read()
print("Total Length:",len(plaintext))
display(Markdown(f'\n"**_{plaintext}_**"'))
```

```
Total Length: 1168
```

"***From the Pope to Greta Thunberg, there are growing calls for the crime of "ecocide" to be recognised in international criminal law - but could such a law ever work? In December 2019, at the International Criminal Court in the Hague, Vanuatu's ambassador to the European Union made a radical suggestion: make the destruction of the environment a crime. Vanuatu is a small island state in the South Pacific, a nation severely threatened by rising sea levels. Climate change is an imminent and existential crisis in the country, yet the actions that have caused rising temperatures - such as burning fossil fuels - have almost entirely taken place elsewhere, to serve other nations, with the blessing of state governments. Small island states like Vanuatu have long tried to persuade large powerful nations to voluntarily reduce their emissions, but change has been slow - so ambassador John Licht suggested that it might be time to change the law itself. An amendment to a treaty known as the Rome Statute, which established the International Criminal Court, could criminalise acts that amount to ecocide, he said, arguing "this radical idea merits serious discussion".***"

And we will try to encrypt the message, using an encryption key **"R is a Short key"**

In [18]:

```
key = "R is a Short key"
key = key.encode().hex()
print(f"'R is a Short key':\n 0x{key}")
aes_lib = AES.AES(int(key,base=16))
```

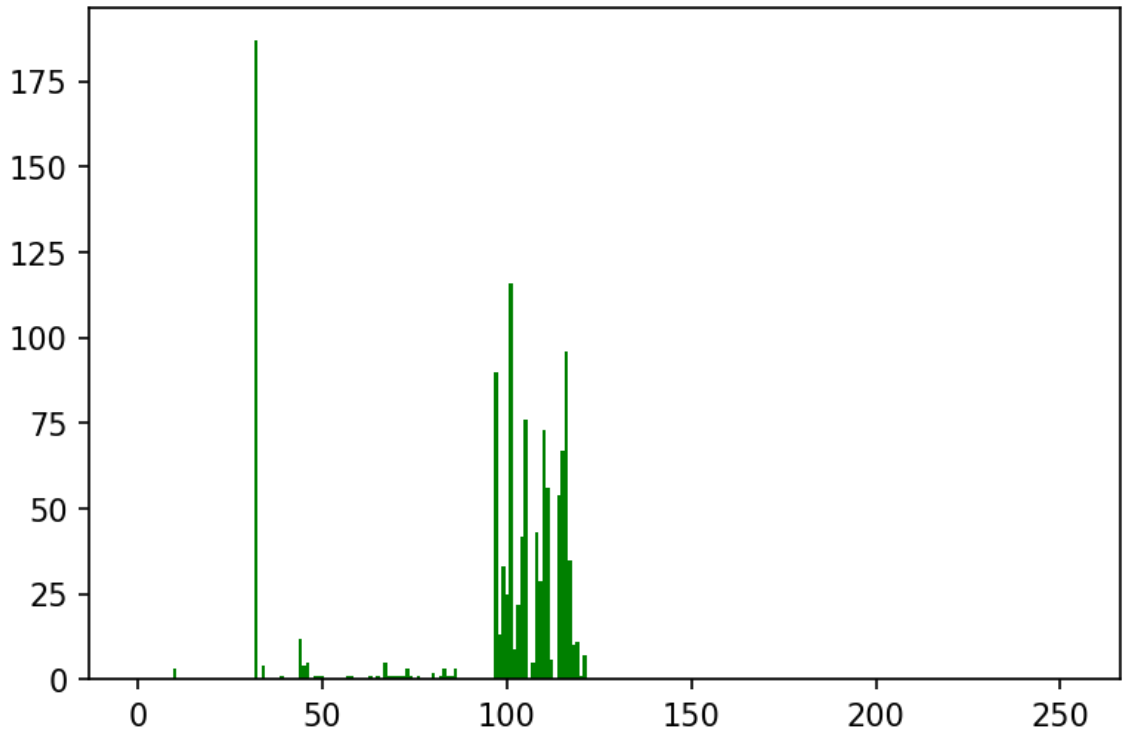
```
'R is a Short key':
 0x52206973206120536686f7274206b6579
```

Part One

1. Create a letter scatter histogram for each source message. Note that the message is divided into 16-letter blocks (aka STATE). The sotogram will collect the over the various statistics across the entire run (i.e. amount for the entire message)
2. We watched the article using AES, and after each round of AES and for the encrypted message at the end. Calculate the scatter plot of the letters throughout the entire message, some in the first section
3. Create one table that summarizes all the data and adds the average and standard deviation to each row

In [19]:

```
plaintext_binary = list(plaintext.encode())
plaintext_freq =  {i : plaintext_binary.count(i) for i in range(0,254)}
plt.bar( plaintext_freq.keys(), plaintext_freq.values(), 1.0, color='g')
plt.show()
```

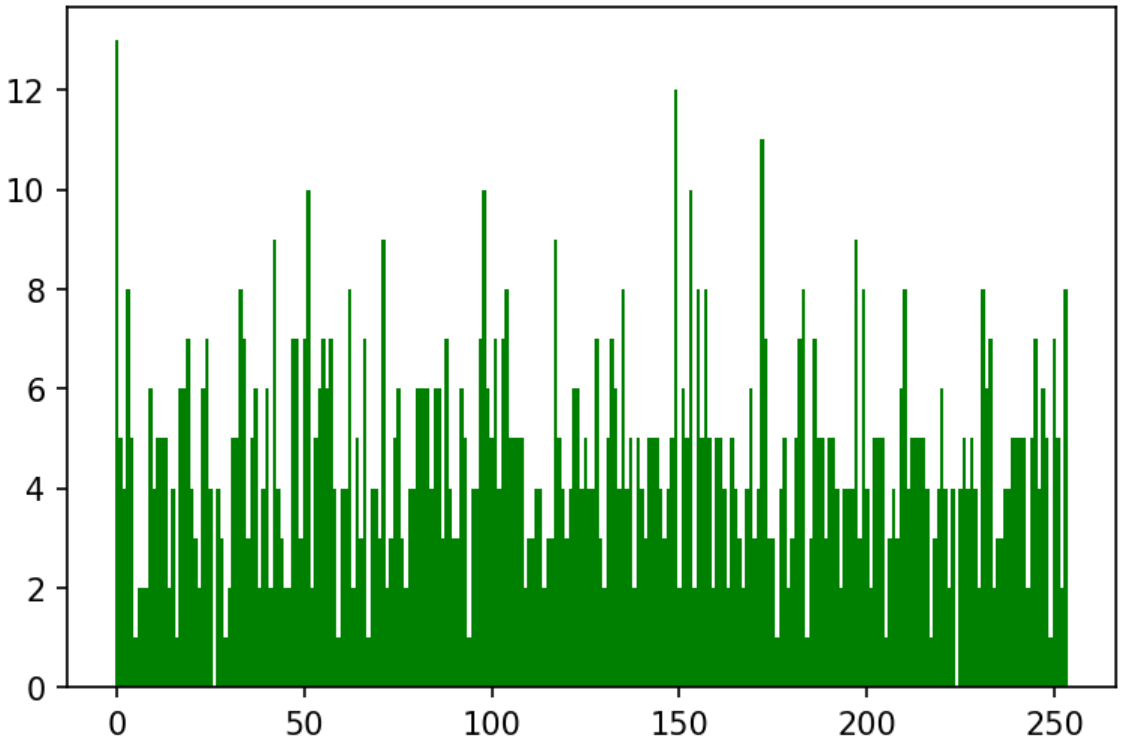


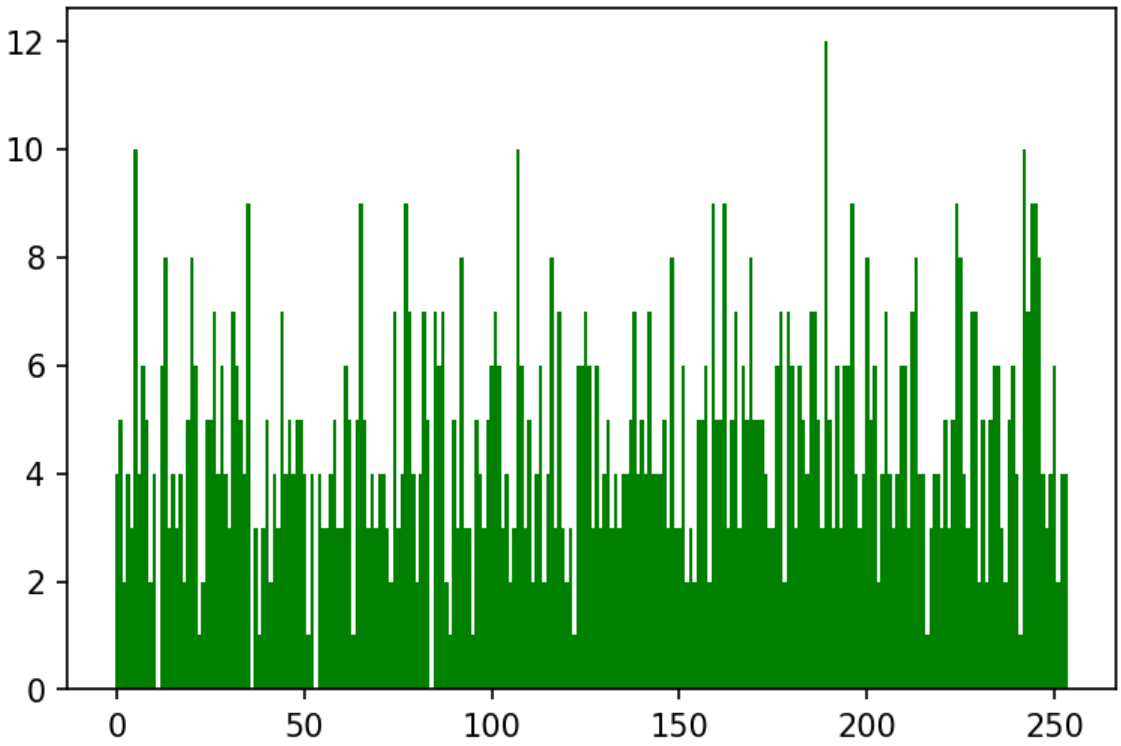
In [20]:

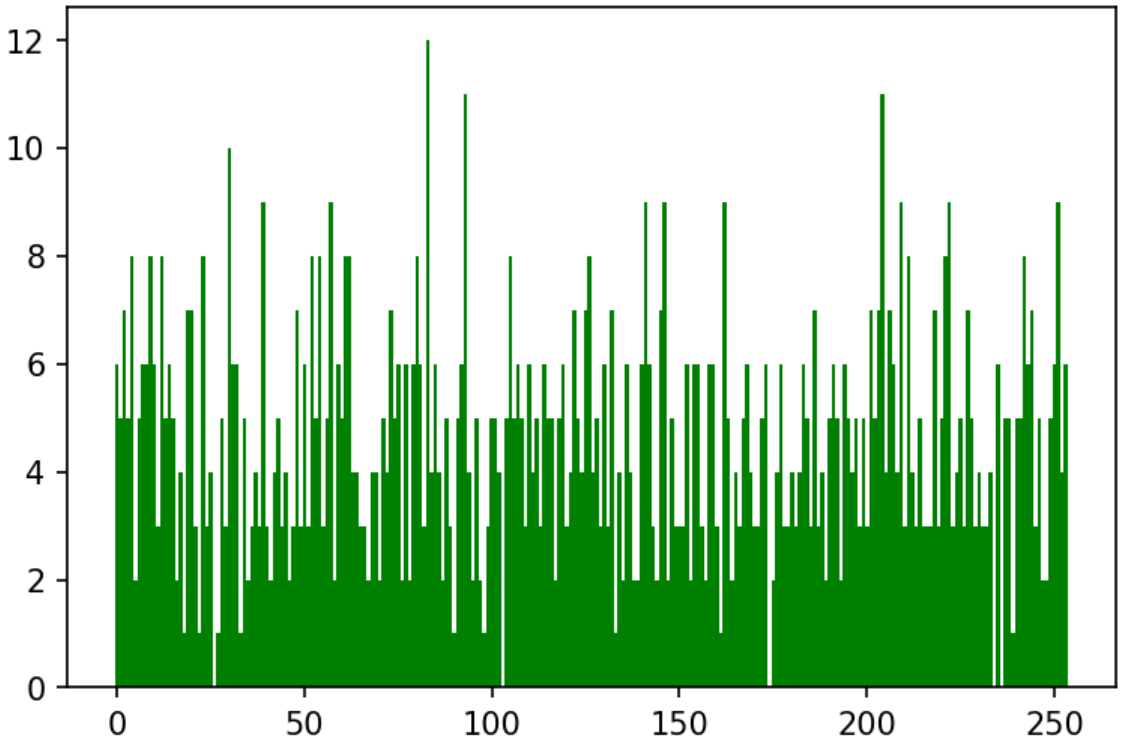
```
import binascii
spec_enc_text = [""]*10

for i in range(0, len(plaintext), 16):
    ciper = aes_lib.encrypt_by_stage(int(plaintext[i:i+16].encode().hex(),base=16))
    for j,c in enumerate(ciper):
        spec_enc_text[j] += ('{:x}'.format(c)).zfill(32)
```

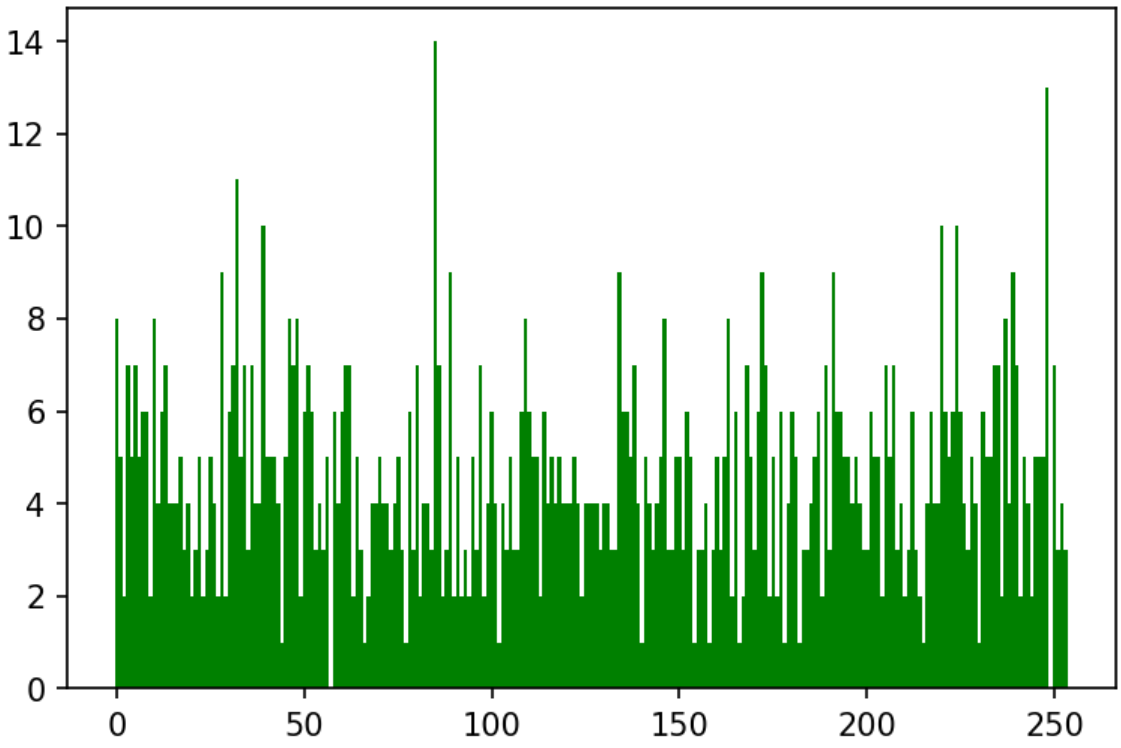
In [21]:

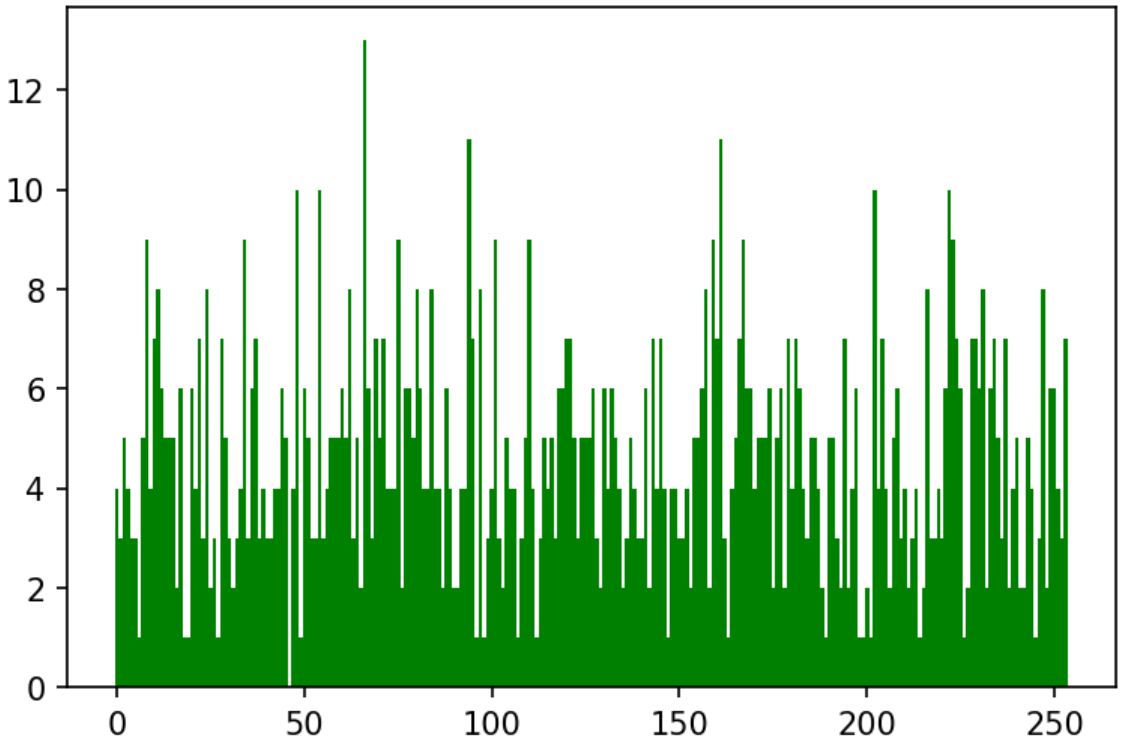
```
cipertext_freq = []
for s in spec_enc_text:
    val = {i :  list(bytes.fromhex(s)).count(i) for i in range(0,254)}
    cipertext_freq.append(val)
    plt.bar( val.keys(), val.values(), 1.0, color='g')
    plt.show()
```

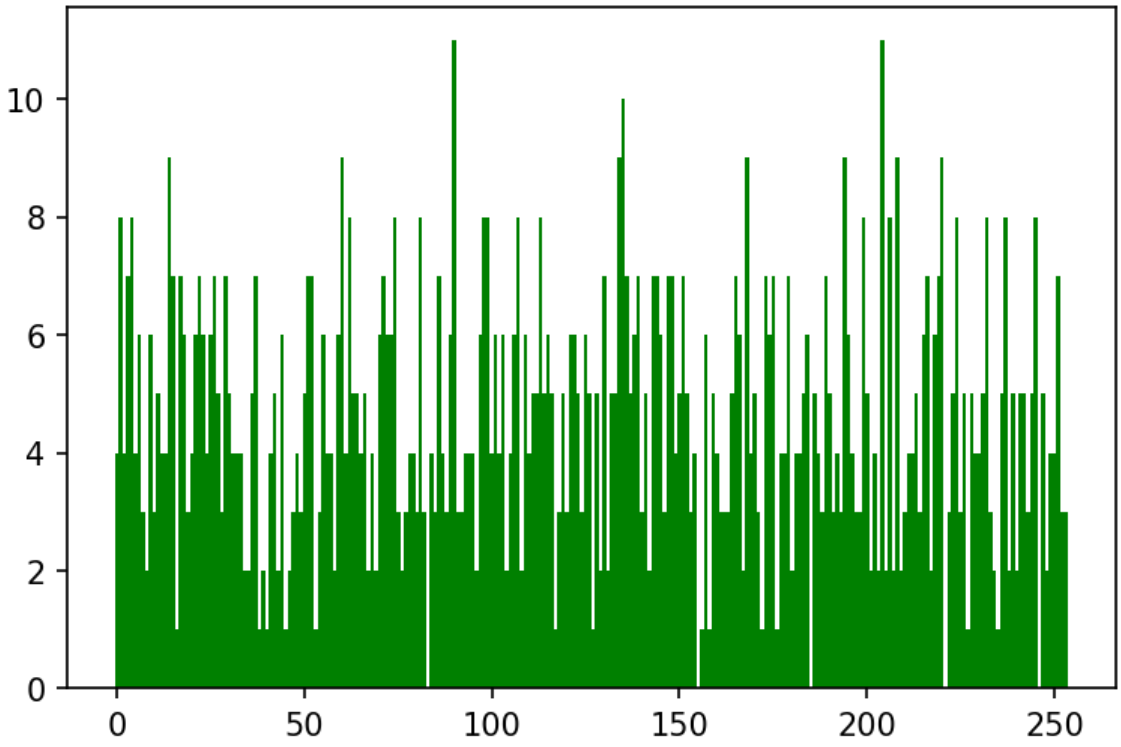

0
2
4
6
8
10
12
0
50
100
150
200
250

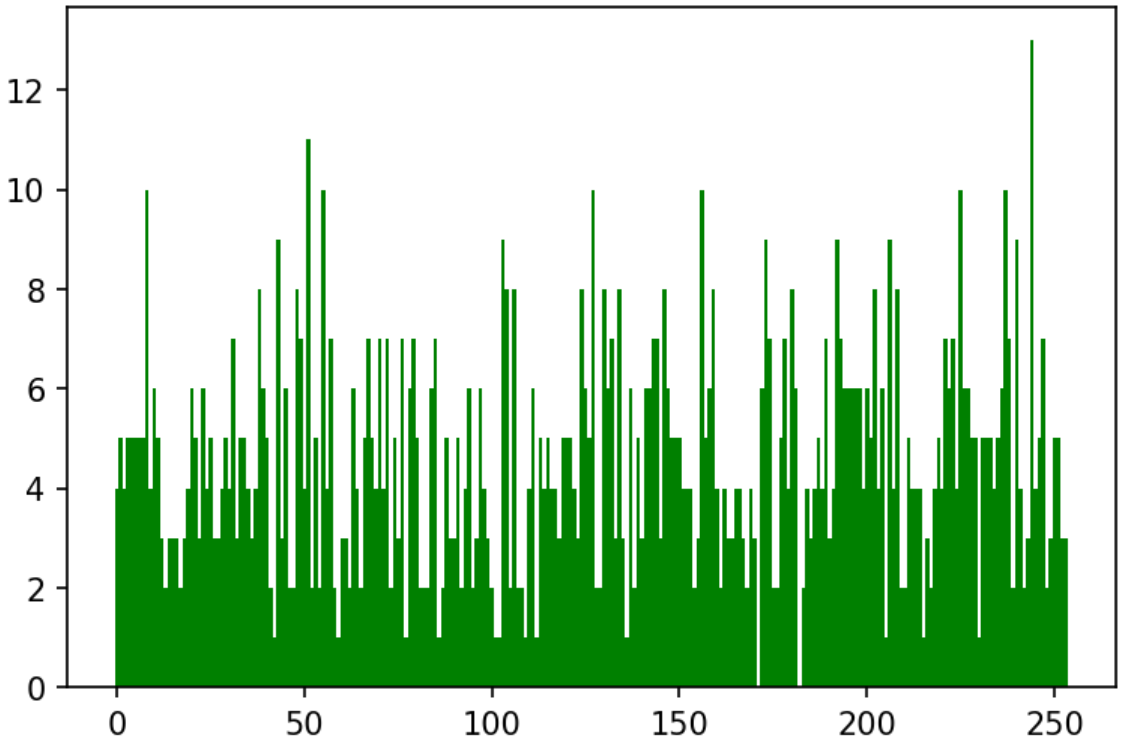

0
2
4
6
8
10
12
0
50
100
150
200
250

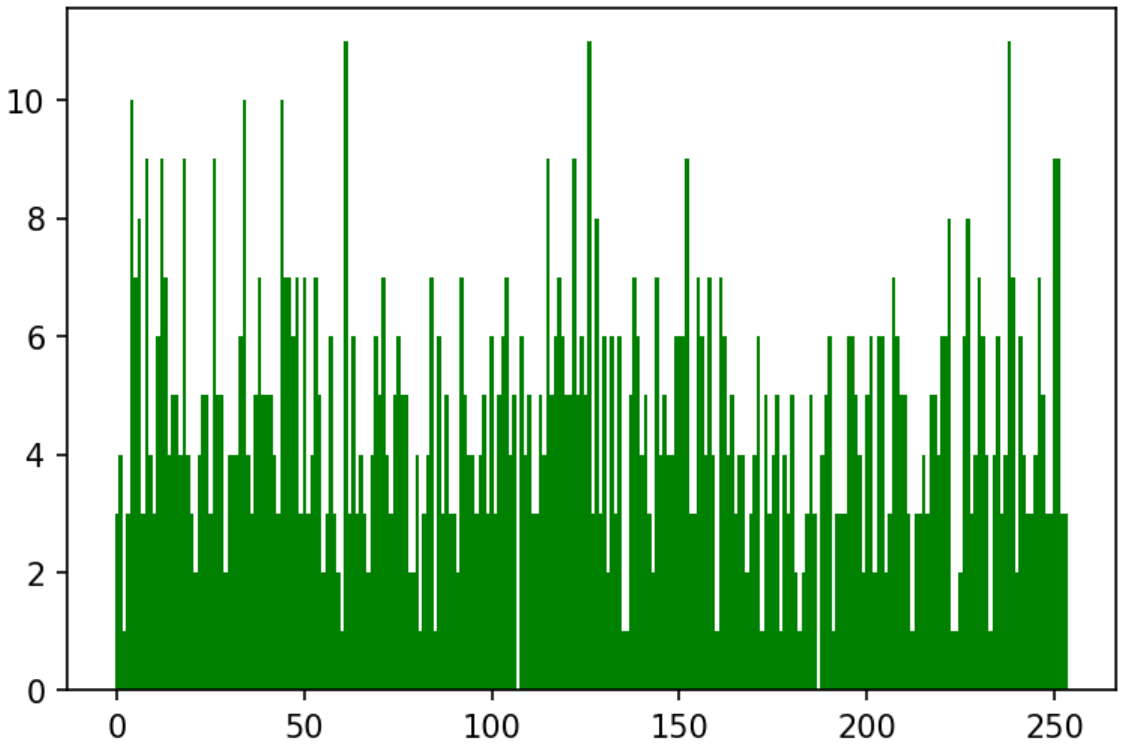

12
10
8
6
4
2
0
0
50
100
150
200
250

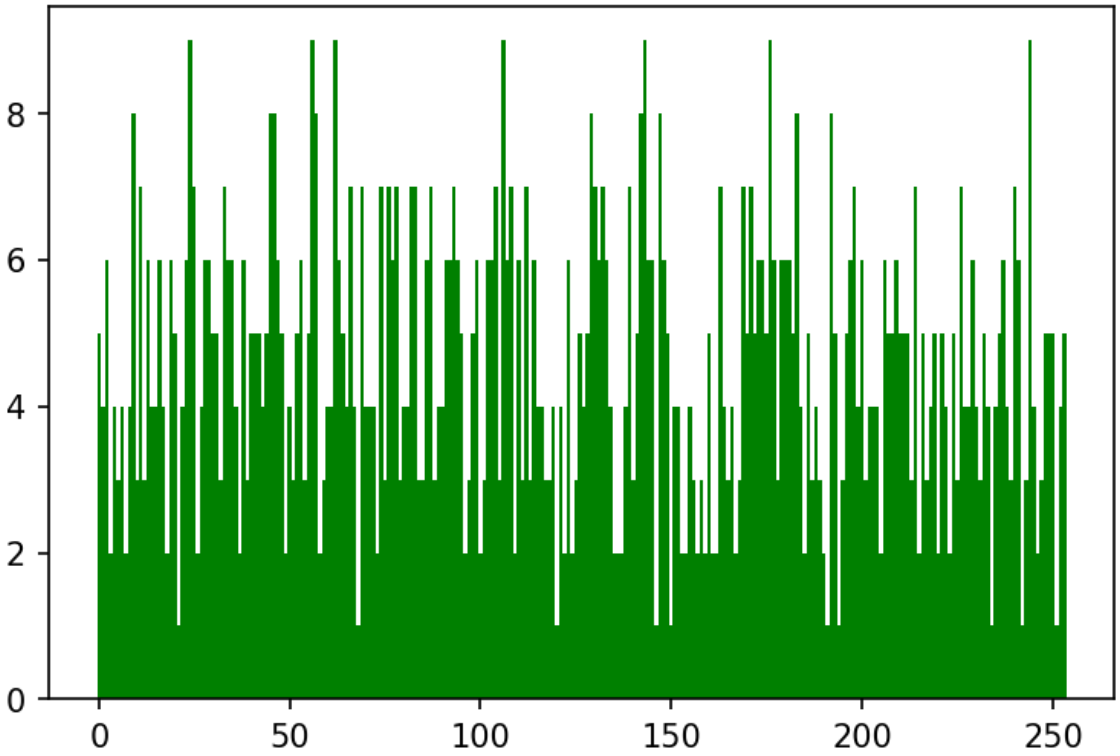

14
12
10
8
6
4
2
0
0
50
100
150
200
250

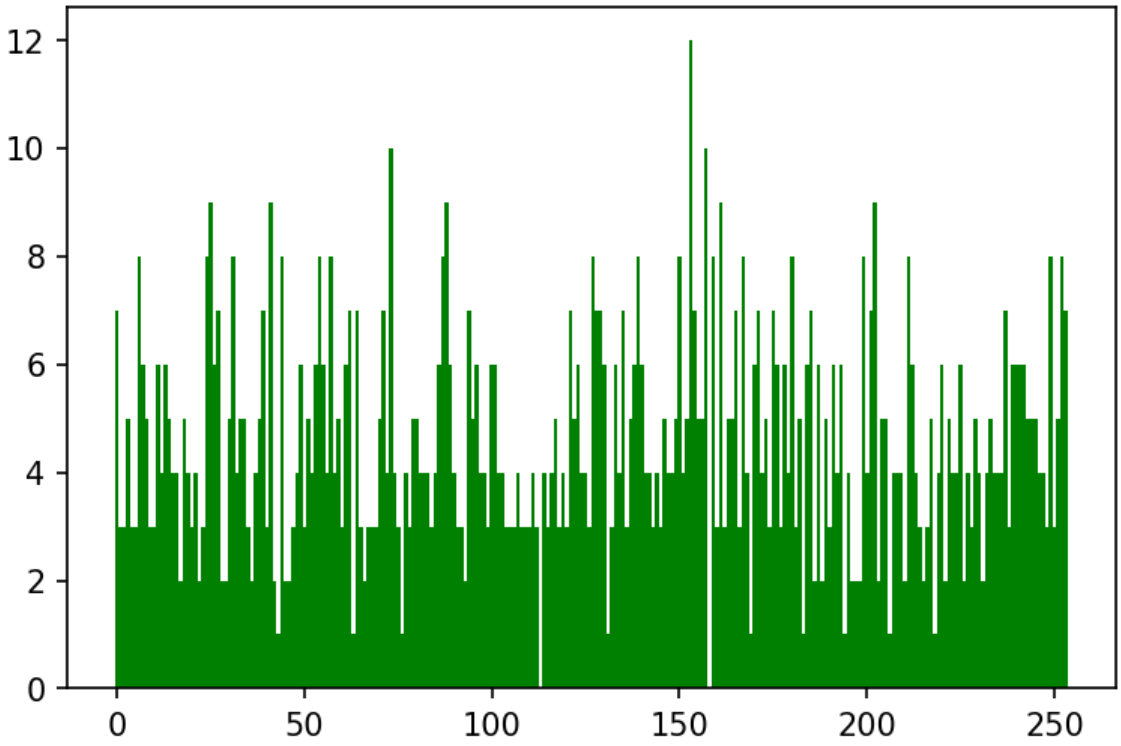

0
2
4
6
8
10
12
0
50
100
150
200
250


0
2
4
6
8
10
0
50
100
150
200
250


0
2
4
6
8
10
12
0
50
100
150
200
250


0
2
4
6
8
10
0
50
100
150
200
250


0
2
4
6
8
0
50
100
150
200
250


0
2
4
6
8
10
12
0
50
100
150
200
250

In [22]:

```
columns=[]
final_freq = [plaintext_freq]  + cipertext_freq
histo_result = np.zeros((len(final_freq),34))
for row,s in enumerate(final_freq):
    current_values = list(s.values())
    for col in range(0,254,8):
        histo_result[row][col//8] = sum(current_values[col:col+8])
    histo_result[row][-1] = np.std(current_values)
    histo_result[row][-2] = np.mean(current_values)

display(pd.DataFrame(histo_result, columns=[f"{col}-{col+7}" for col in range(0,254,8)]
+["STD","MEAN"]))
```

	0-7	8-15	16-23	24-31	32-39	40-47	48-55	56-63	64-71	72-79	...	192-199	200-207	208-215	216-223	22 2
0	0.0	3.0	0.0	0.0	192.0	21.0	3.0	3.0	10.0	6.0	...	0.0	0.0	0.0	0.0	(
1	40.0	33.0	35.0	26.0	40.0	35.0	47.0	36.0	36.0	29.0	...	38.0	29.0	41.0	28.0	32
2	38.0	32.0	31.0	41.0	31.0	34.0	26.0	30.0	37.0	39.0	...	41.0	39.0	42.0	28.0	45
3	44.0	47.0	33.0	32.0	33.0	26.0	43.0	47.0	27.0	38.0	...	35.0	50.0	39.0	41.0	34
4	45.0	41.0	28.0	38.0	51.0	40.0	39.0	37.0	28.0	29.0	...	38.0	40.0	24.0	45.0	39
5	28.0	49.0	30.0	31.0	39.0	29.0	41.0	41.0	48.0	40.0	...	26.0	35.0	25.0	46.0	44
6	44.0	40.0	40.0	41.0	27.0	24.0	36.0	42.0	35.0	36.0	...	40.0	36.0	36.0	39.0	35
7	38.0	36.0	32.0	35.0	38.0	30.0	49.0	28.0	38.0	38.0	...	50.0	43.0	30.0	38.0	42
8	39.0	47.0	36.0	37.0	44.0	47.0	38.0	35.0	34.0	32.0	...	32.0	37.0	30.0	38.0	37
9	30.0	39.0	34.0	44.0	37.0	46.0	33.0	45.0	36.0	39.0	...	39.0	34.0	38.0	30.0	36
10	38.0	36.0	27.0	47.0	35.0	30.0	42.0	38.0	33.0	34.0	...	29.0	37.0	33.0	30.0	30

11 rows × 34 columns

HW1 - part 2.A

In [9]:

```
import sys, random,binascii,AES,jupyter_utils
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from IPython.display import Markdown, display,HTML
%matplotlib inline
plt.rcParams['figure.dpi'] = 150
```

We would like to test these principles when operating AES in different situations Note that for the purpose of the exercise, we will allow ourselves a number of mitigating assumptions which will be detailed below

To do this, take the following text which was published today on the BBC website

In [10]:

```
with open("plaintext.txt","r") as f:
    plaintext = f.read()
print("Total Length:",len(plaintext))
display(Markdown(f'\n"**_{plaintext}_**"'))
```

```
Total Length: 1168
```

"***From the Pope to Greta Thunberg, there are growing calls for the crime of "ecocide" to be recognised in international criminal law - but could such a law ever work? In December 2019, at the International Criminal Court in the Hague, Vanuatu's ambassador to the European Union made a radical suggestion: make the destruction of the environment a crime. Vanuatu is a small island state in the South Pacific, a nation severely threatened by rising sea levels. Climate change is an imminent and existential crisis in the country, yet the actions that have caused rising temperatures - such as burning fossil fuels - have almost entirely taken place elsewhere, to serve other nations, with the blessing of state governments. Small island states like Vanuatu have long tried to persuade large powerful nations to voluntarily reduce their emissions, but change has been slow - so ambassador John Licht suggested that it might be time to change the law itself. An amendment to a treaty known as the Rome Statute, which established the International Criminal Court, could criminalise acts that amount to ecocide, he said, arguing "this radical idea merits serious discussion".***"

And we will try to encrypt the message, using an encryption key **"0000000000000000"**

In [11]:

```
key = "0000000000000000"
key = key.encode().hex()
print(f"'R is a Short key':\n 0x{key}")
aes_lib = AES.AES(int(key,base=16))
```

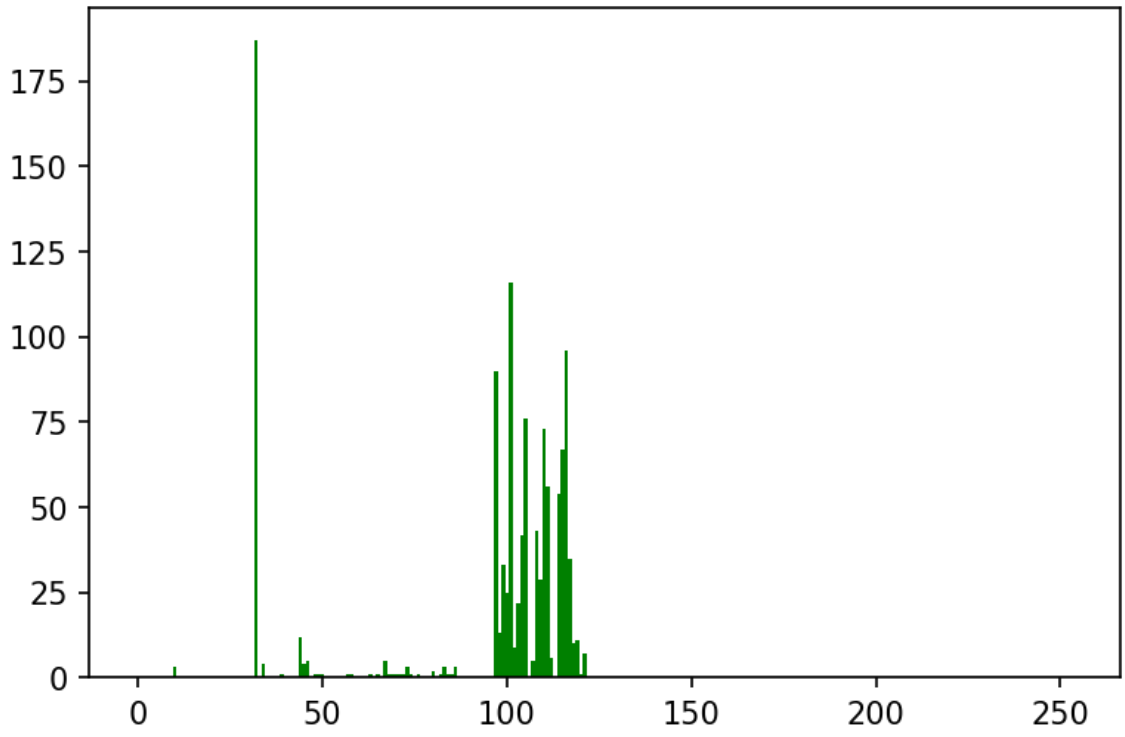
```
'R is a Short key':
 0x30303030303030303030303030303030
```

Part One

1. Create a letter scatter histogram for each source message. Note that the message is divided into 16-letter blocks (aka STATE). The sotogram will collect the over the various statistics across the entire run (i.e. amount for the entire message)
2. We watched the article using AES, and after each round of AES and for the encrypted message at the end. Calculate the scatter plot of the letters throughout the entire message, some in the first section
3. Create one table that summarizes all the data and adds the average and standard deviation to each row

In [12]:

```
plaintext_binary = list(plaintext.encode())
plaintext_freq =  {i : plaintext_binary.count(i) for i in range(0,254)}
plt.bar( plaintext_freq.keys(), plaintext_freq.values(), 1.0, color='g')
plt.show()
```

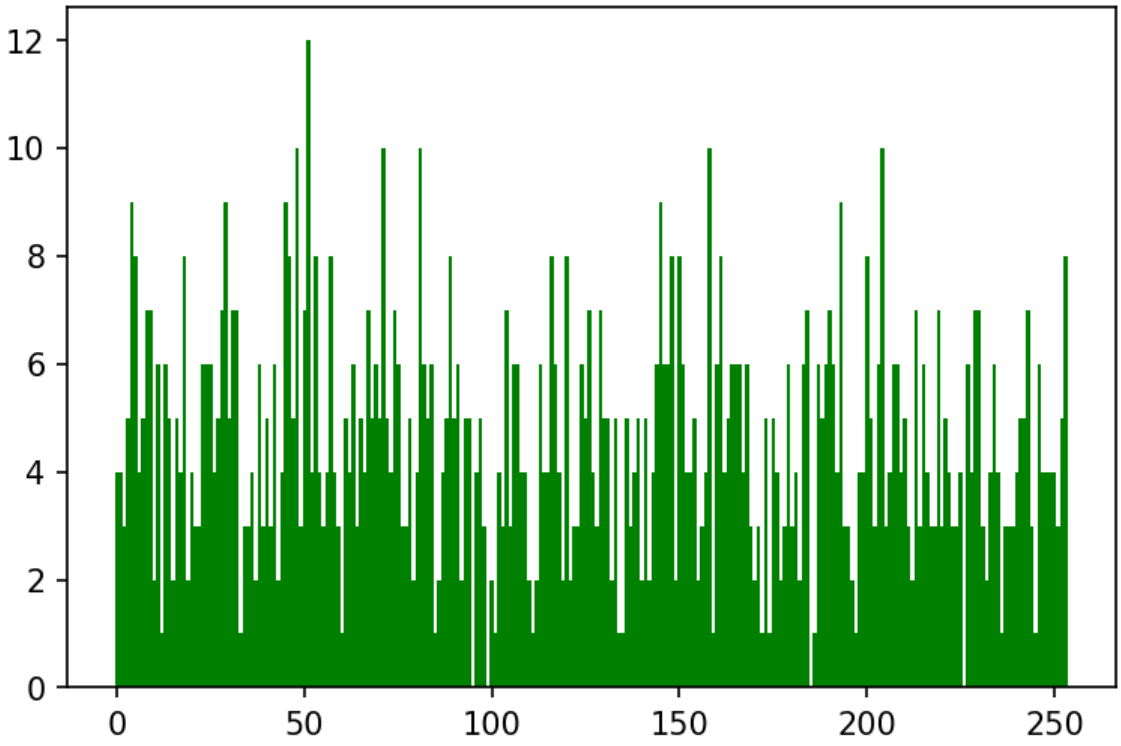


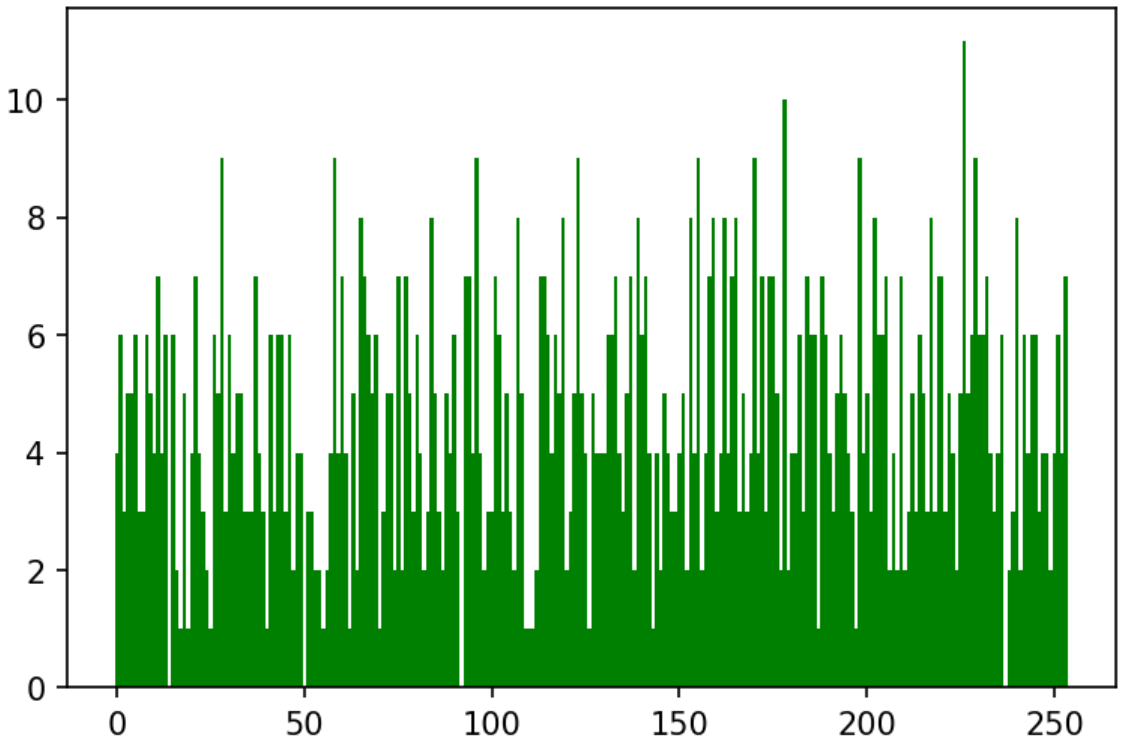
In [13]:

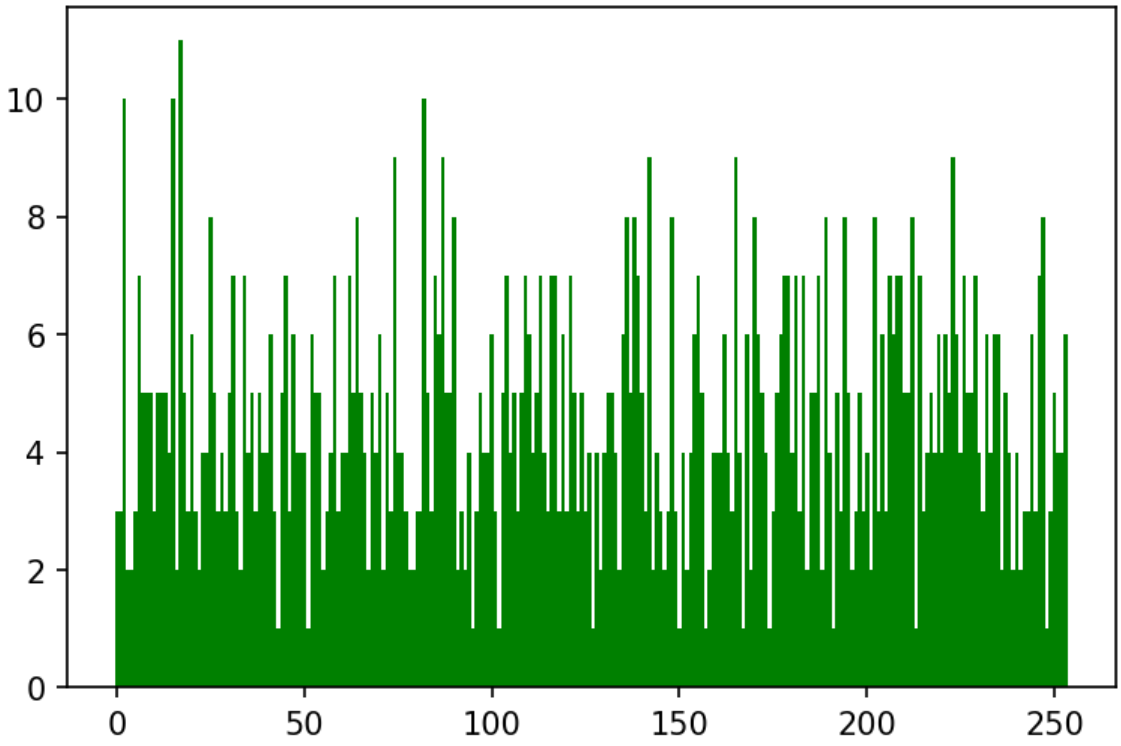
```
import binascii
spec_enc_text = [""]*10

for i in range(0, len(plaintext), 16):
    ciper = aes_lib.encrypt_by_stage(int(plaintext[i:i+16].encode().hex(),base=16))
    for j,c in enumerate(ciper):
        spec_enc_text[j] += ('{:x}'.format(c)).zfill(32)
```

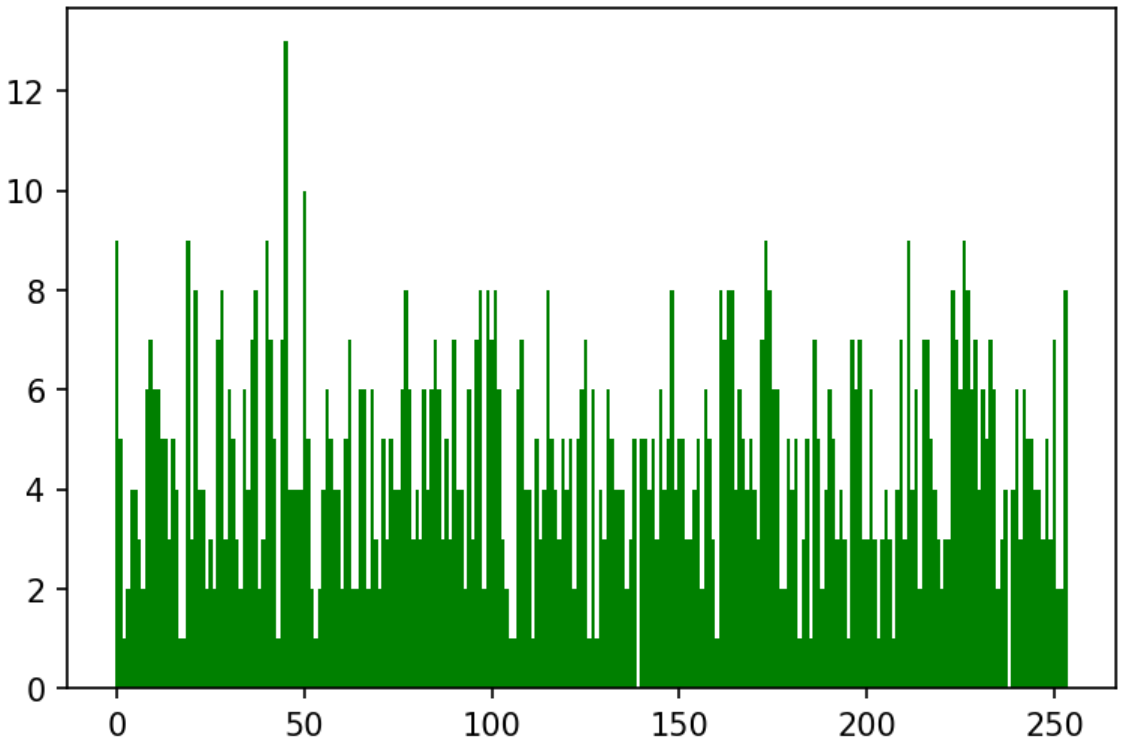
In [14]:

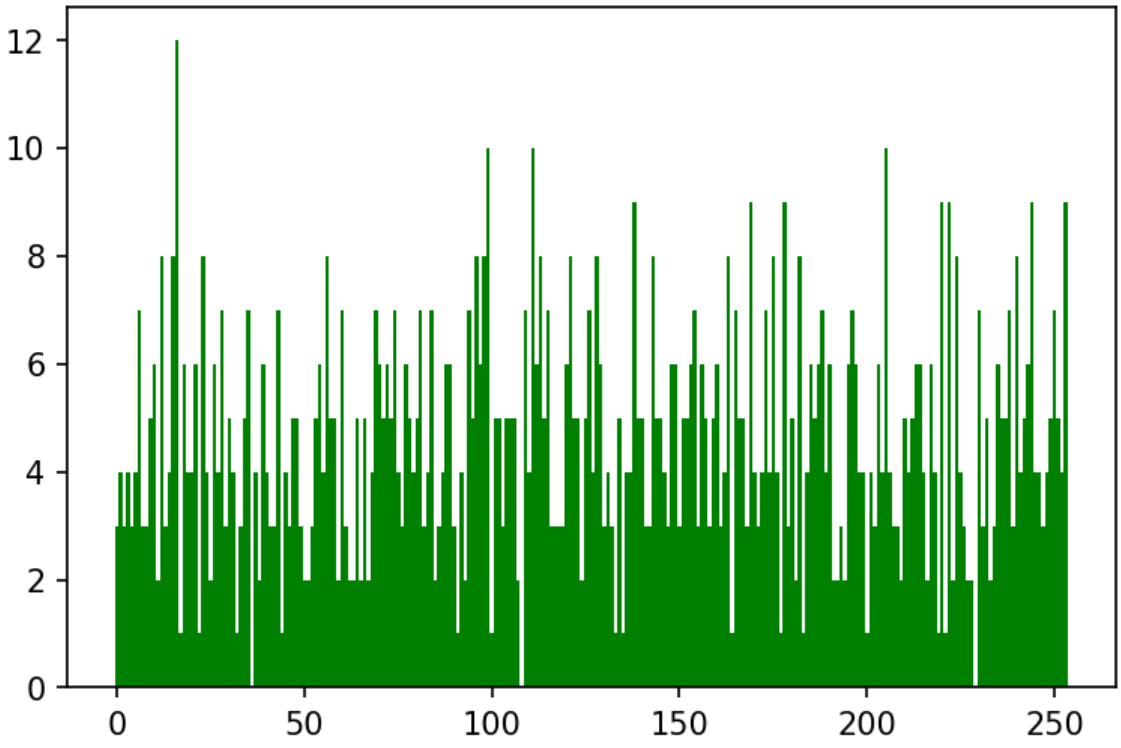
```
cipertext_freq = []
for s in spec_enc_text:
    val = {i :  list(bytes.fromhex(s)).count(i) for i in range(0,254)}
    cipertext_freq.append(val)
    plt.bar( val.keys(), val.values(), 1.0, color='g')
    plt.show()
```

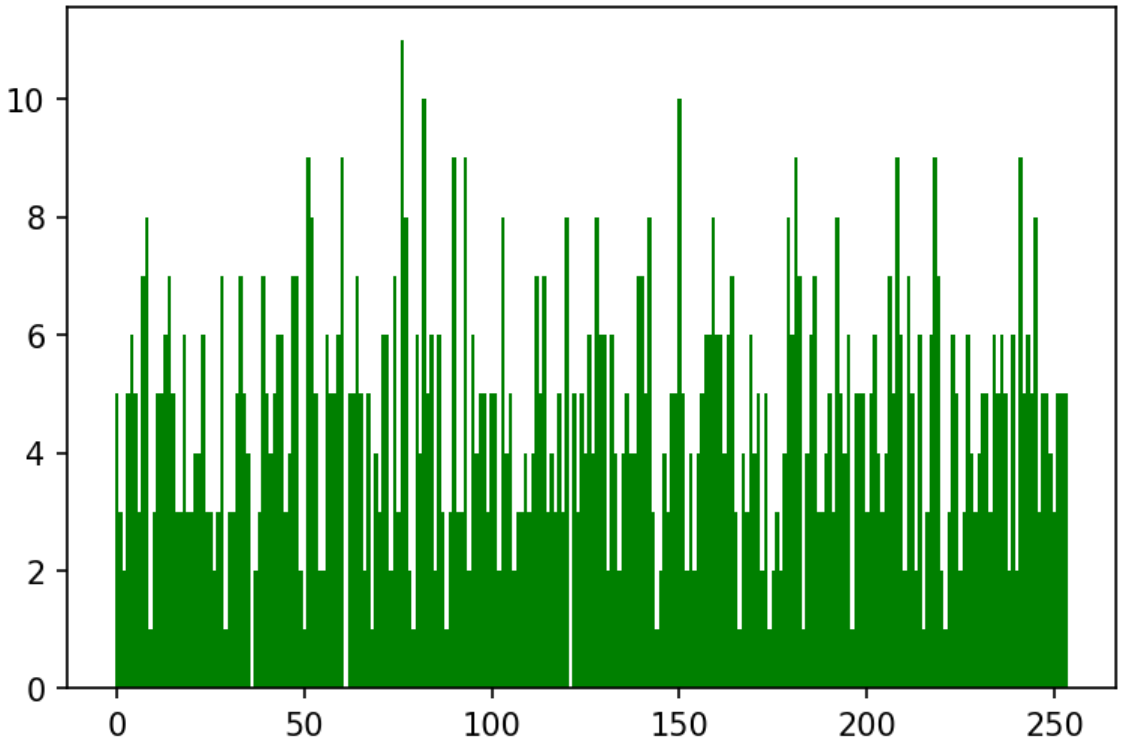

12
10
8
6
4
2
0
0
50
100
150
200
250

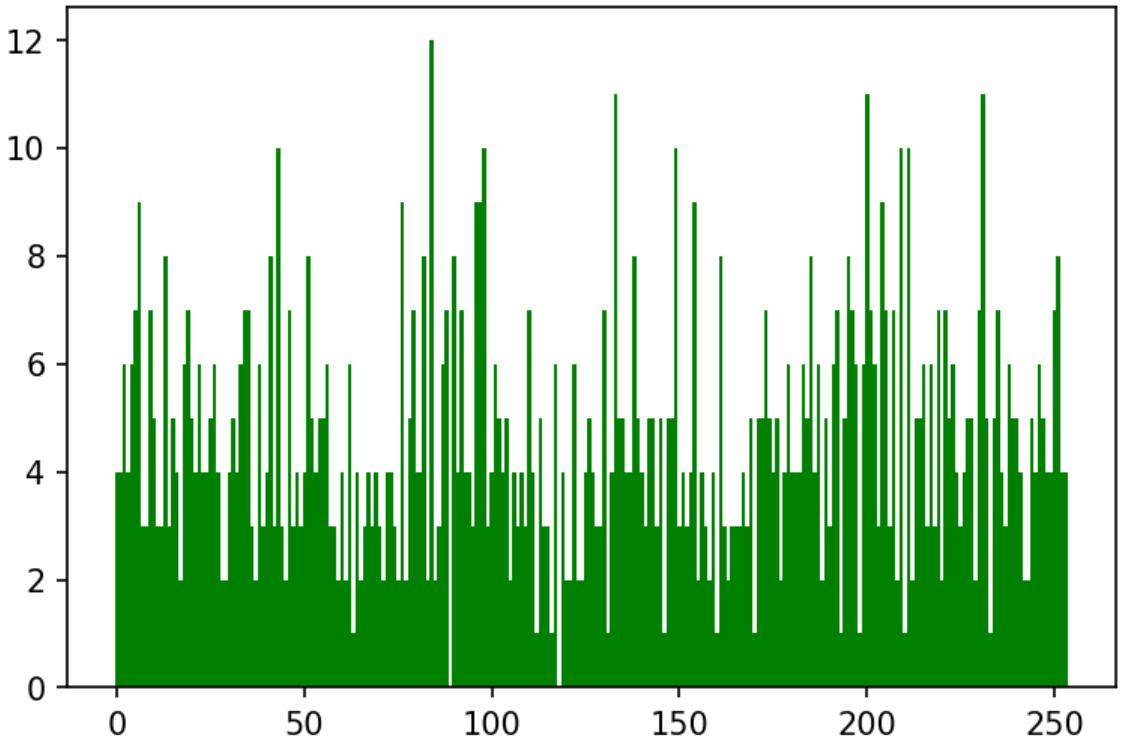

10
8
6
4
2
0
0
50
100
150
200
250

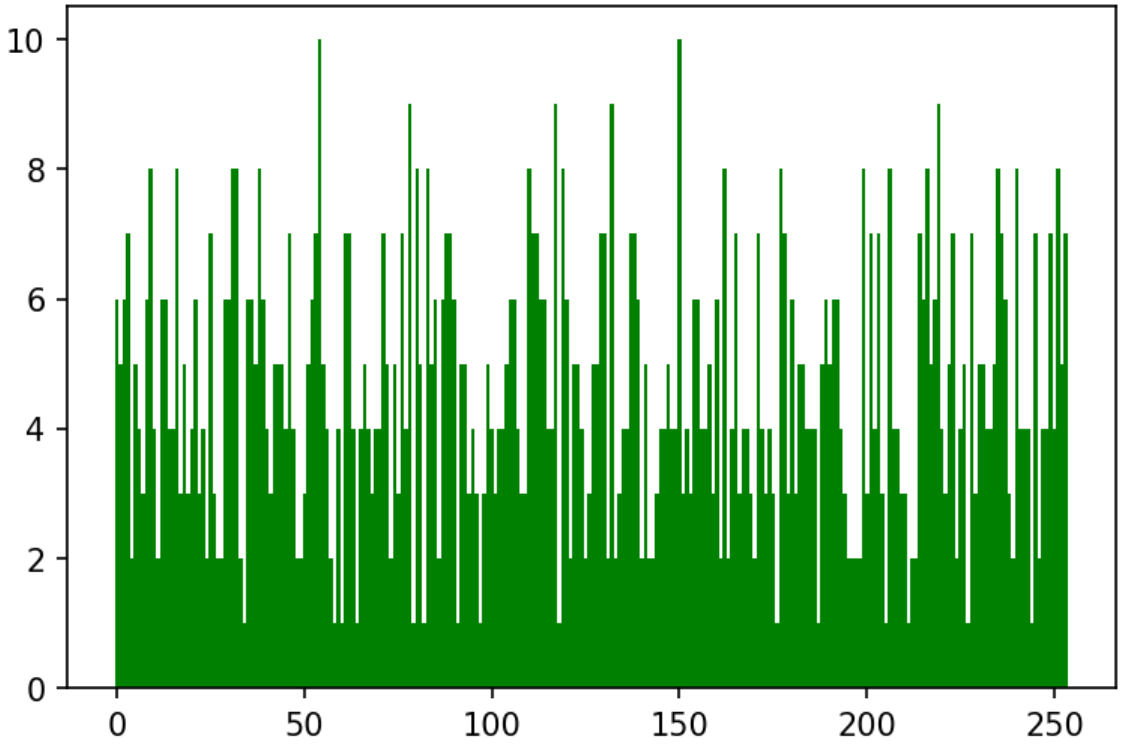

0
2
4
6
8
10
0
50
100
150
200
250

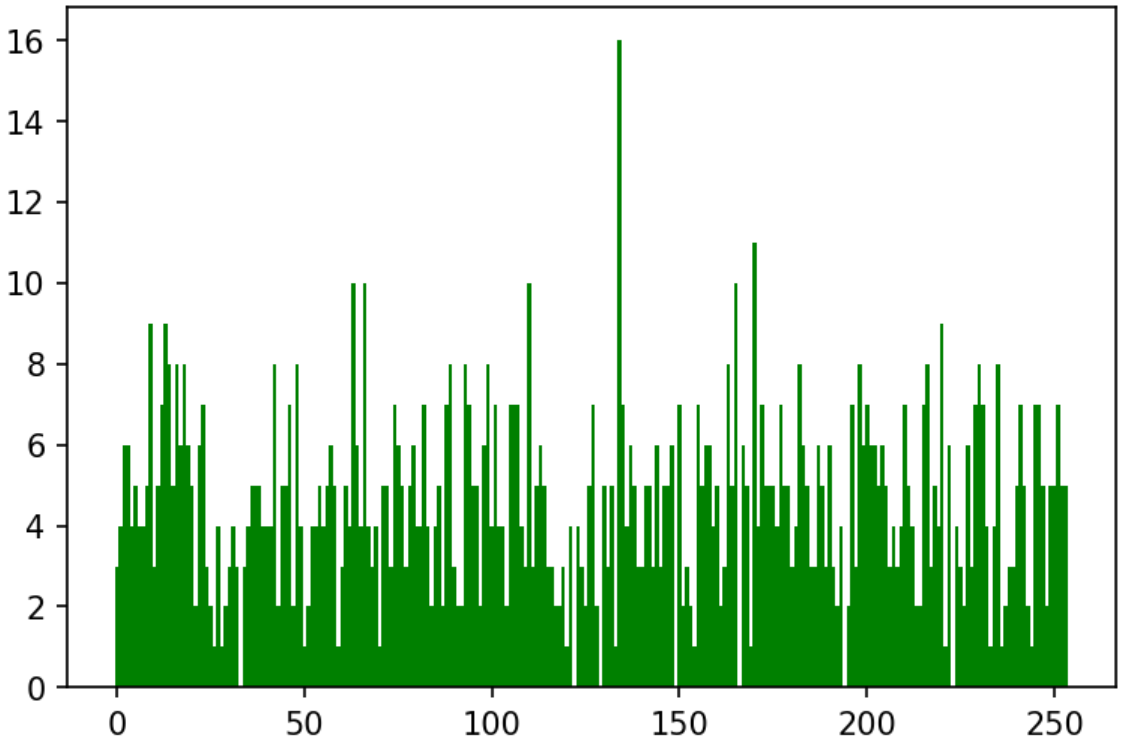

0
2
4
6
8
10
12
0
50
100
150
200
250

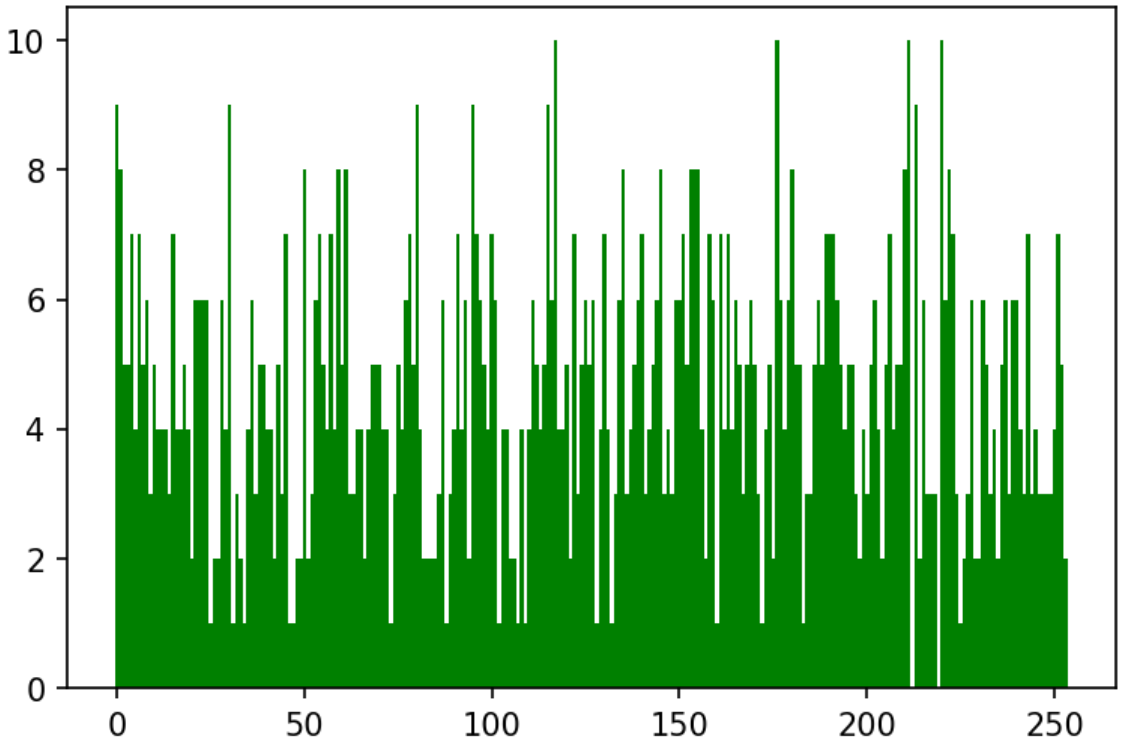

12
10
8
6
4
2
0
0
50
100
150
200
250


10
8
6
4
2
0
0
50
100
150
200
250


12
10
8
6
4
2
0
0
50
100
150
200
250


10
8
6
4
2
0
0
50
100
150
200
250


16
14
12
10
8
6
4
2
0
0
50
100
150
200
250


10
8
6
4
2
0
0
50
100
150
200
250

In [15]:

```
columns=[]
final_freq = [plaintext_freq]  + cipertext_freq
histo_result = np.zeros((len(final_freq),34))

for row,s in enumerate(final_freq):
    current_values = list(s.values())
    for col in range(0,254,8):
        histo_result[row][col//8] = sum(current_values[col:col+8])
    histo_result[row][-1] = np.std(current_values)
    histo_result[row][-2] = np.mean(current_values)

display(pd.DataFrame(histo_result, columns=[f"{col}-{col+7}" for col in range(0,254,8)]
+["STD","MEAN"]))
```

	0-7	8-15	16-23	24-31	32-39	40-47	48-55	56-63	64-71	72-79	...	192-199	200-207	208-215	216-223	22 2
0	0.0	3.0	0.0	0.0	192.0	21.0	3.0	3.0	10.0	6.0	...	0.0	0.0	0.0	0.0	(
1	42.0	36.0	35.0	49.0	29.0	42.0	51.0	35.0	45.0	35.0	...	30.0	45.0	36.0	32.0	34
2	35.0	38.0	27.0	36.0	33.0	33.0	19.0	36.0	38.0	36.0	...	37.0	41.0	33.0	40.0	5(
3	35.0	42.0	36.0	39.0	33.0	35.0	31.0	37.0	36.0	32.0	...	34.0	39.0	43.0	43.0	41
4	30.0	43.0	34.0	36.0	35.0	50.0	32.0	35.0	32.0	39.0	...	34.0	24.0	42.0	35.0	53
5	31.0	39.0	42.0	35.0	28.0	30.0	30.0	34.0	36.0	40.0	...	34.0	35.0	35.0	34.0	29
6	36.0	40.0	32.0	25.0	33.0	40.0	36.0	41.0	33.0	40.0	...	39.0	37.0	38.0	37.0	32
7	43.0	37.0	38.0	32.0	38.0	40.0	38.0	27.0	25.0	36.0	...	41.0	53.0	41.0	39.0	41
8	38.0	40.0	36.0	36.0	42.0	37.0	40.0	30.0	32.0	36.0	...	29.0	37.0	28.0	47.0	32
9	36.0	51.0	48.0	20.0	29.0	37.0	32.0	39.0	37.0	40.0	...	32.0	42.0	34.0	36.0	4(
10	50.0	36.0	37.0	31.0	29.0	27.0	35.0	42.0	33.0	35.0	...	34.0	36.0	45.0	40.0	25

11 rows × 34 columns

HW1 - part 2.B

In [8]:

```
import sys, random,binascii,AES,jupyter_utils
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from IPython.display import Markdown, display,HTML
%matplotlib inline
plt.rcParams['figure.dpi'] = 150
```

We would like to test these principles when operating AES in different situations Note that for the purpose of the exercise, we will allow ourselves a number of mitigating assumptions which will be detailed below

To do this, take the following text which was published today on the BBC website

In [9]:

```
with open("plaintext.txt","r") as f:
    plaintext = f.read()
print("Total Length:",len(plaintext))
display(Markdown(f'\n"**_{plaintext}_**"'))
```

```
Total Length: 1168
```

"***From the Pope to Greta Thunberg, there are growing calls for the crime of "ecocide" to be recognised in international criminal law - but could such a law ever work? In December 2019, at the International Criminal Court in the Hague, Vanuatu's ambassador to the European Union made a radical suggestion: make the destruction of the environment a crime. Vanuatu is a small island state in the South Pacific, a nation severely threatened by rising sea levels. Climate change is an imminent and existential crisis in the country, yet the actions that have caused rising temperatures - such as burning fossil fuels - have almost entirely taken place elsewhere, to serve other nations, with the blessing of state governments. Small island states like Vanuatu have long tried to persuade large powerful nations to voluntarily reduce their emissions, but change has been slow - so ambassador John Licht suggested that it might be time to change the law itself. An amendment to a treaty known as the Rome Statute, which established the International Criminal Court, could criminalise acts that amount to ecocide, he said, arguing "this radical idea merits serious discussion".***"

And we will try to encrypt the message, using an encryption key **"1111111111111111"**

In [10]:

```
key = "1111111111111111"
key = key.encode().hex()
print(f"'R is a Short key':\n 0x{key}")
aes_lib = AES.AES(int(key,base=16))
```

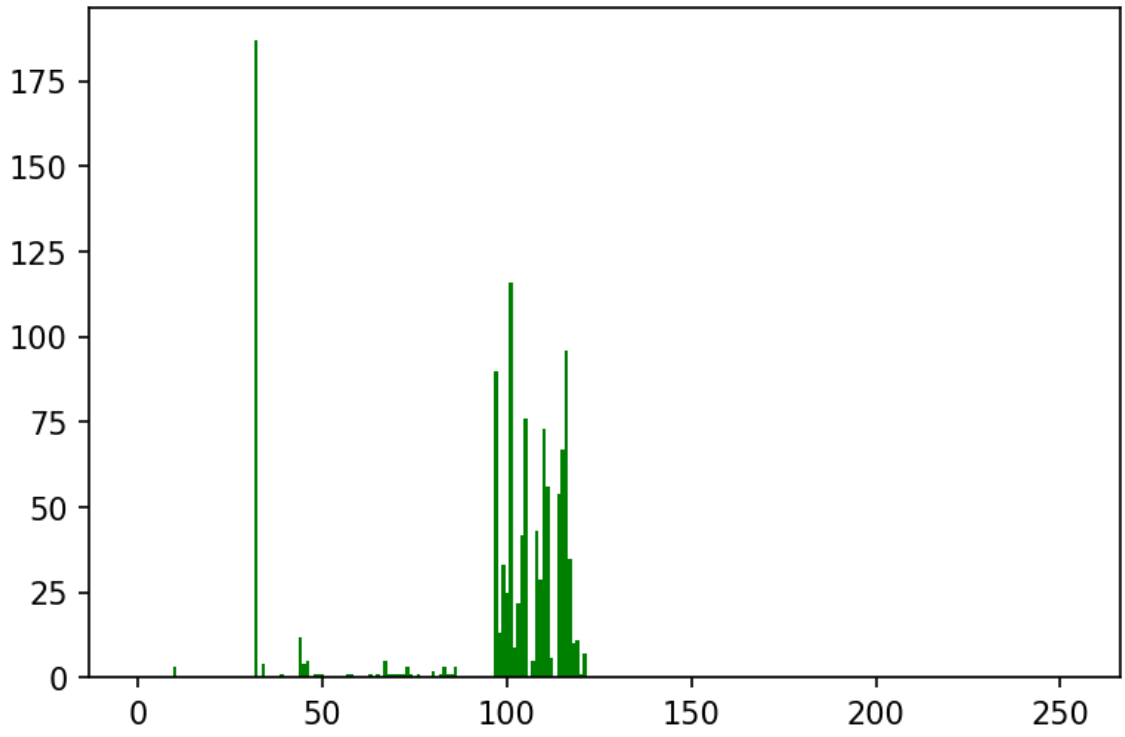
```
'R is a Short key':
 0x31313131313131313131313131313131
```

Part One

1. Create a letter scatter histogram for each source message. Note that the message is divided into 16-letter blocks (aka STATE). The sotogram will collect the over the various statistics across the entire run (i.e. amount for the entire message)
2. We watched the article using AES, and after each round of AES and for the encrypted message at the end. Calculate the scatter plot of the letters throughout the entire message, some in the first section
3. Create one table that summarizes all the data and adds the average and standard deviation to each row

In [11]:

```
plaintext_binary = list(plaintext.encode())
plaintext_freq =  {i : plaintext_binary.count(i) for i in range(0,254)}
plt.bar( plaintext_freq.keys(), plaintext_freq.values(), 1.0, color='g')
plt.show()
```

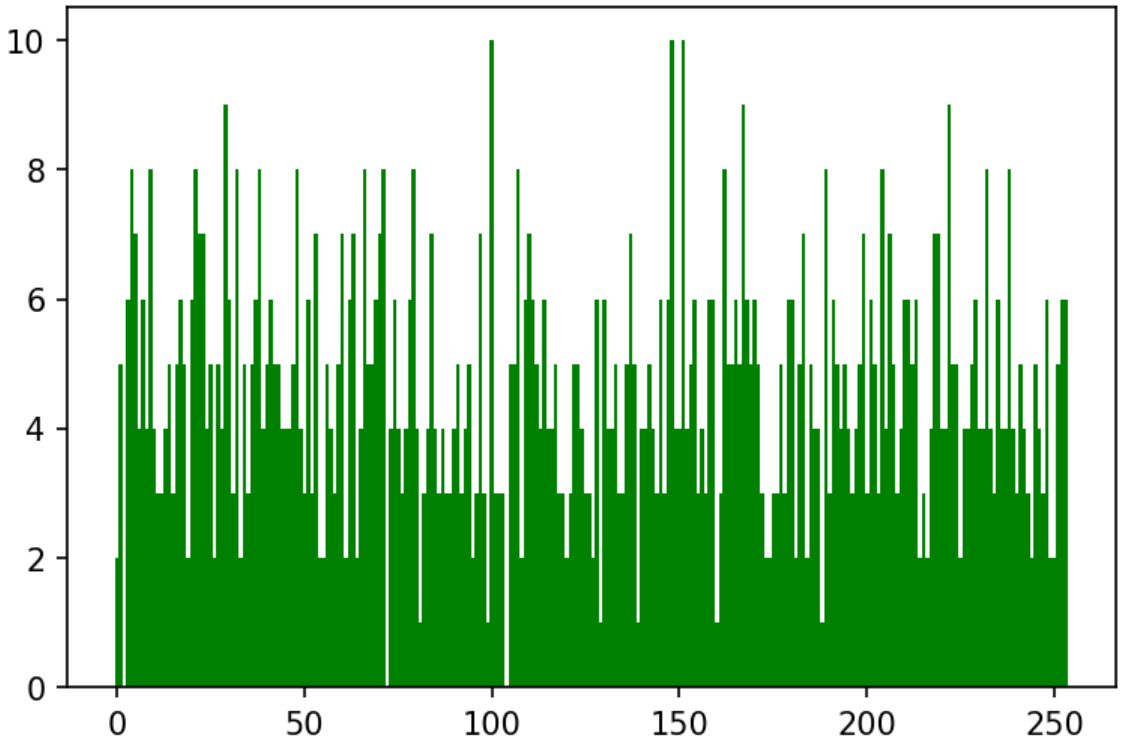


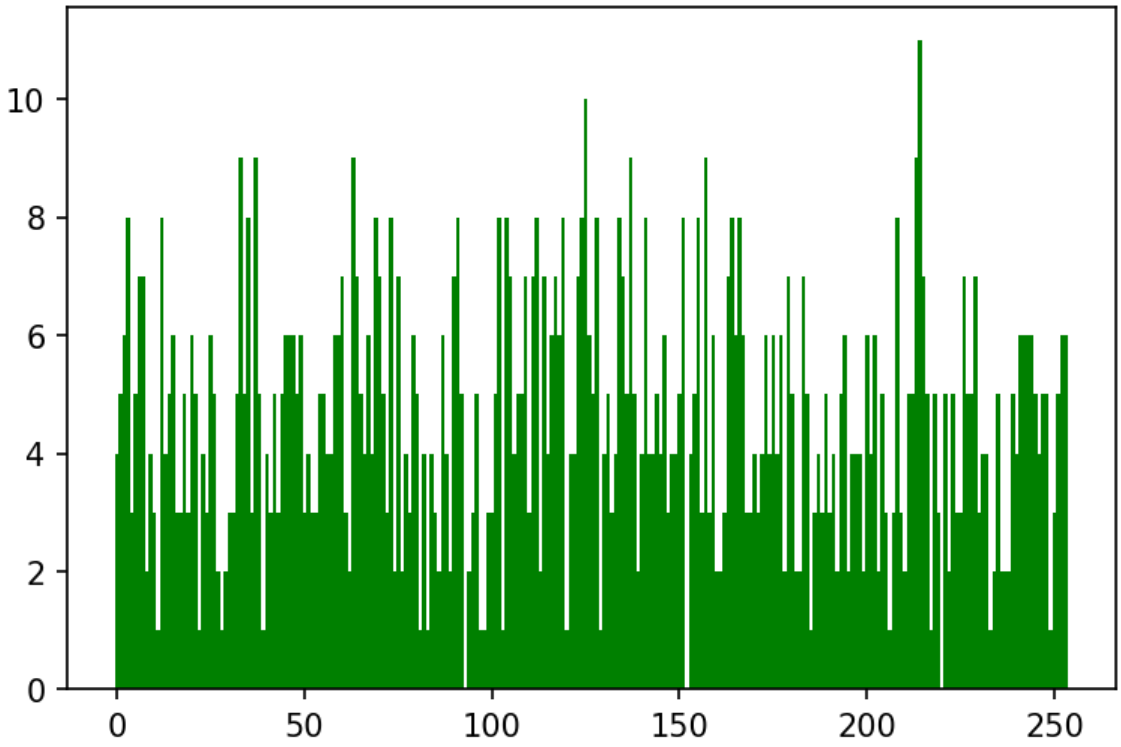
In [12]:

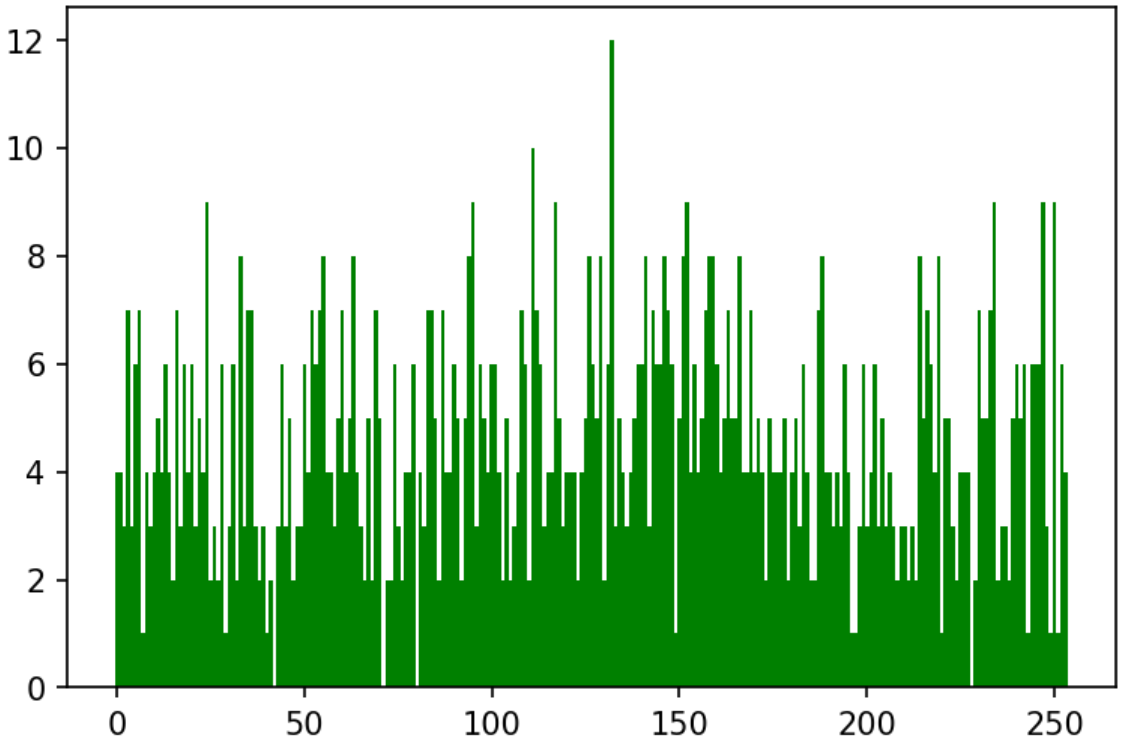
```
import binascii
spec_enc_text = [""]*10

for i in range(0, len(plaintext), 16):
    ciper = aes_lib.encrypt_by_stage(int(plaintext[i:i+16].encode().hex(),base=16))
    for j,c in enumerate(ciper):
        spec_enc_text[j] += ('{:x}'.format(c)).zfill(32)
```

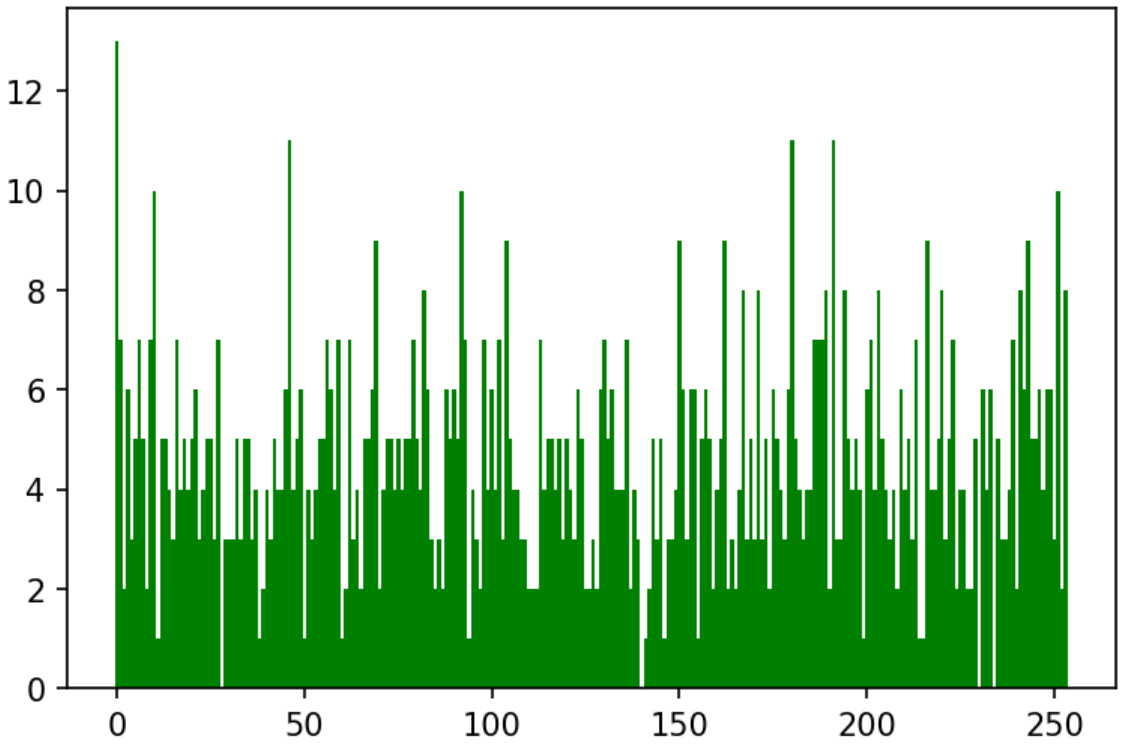
In [13]:

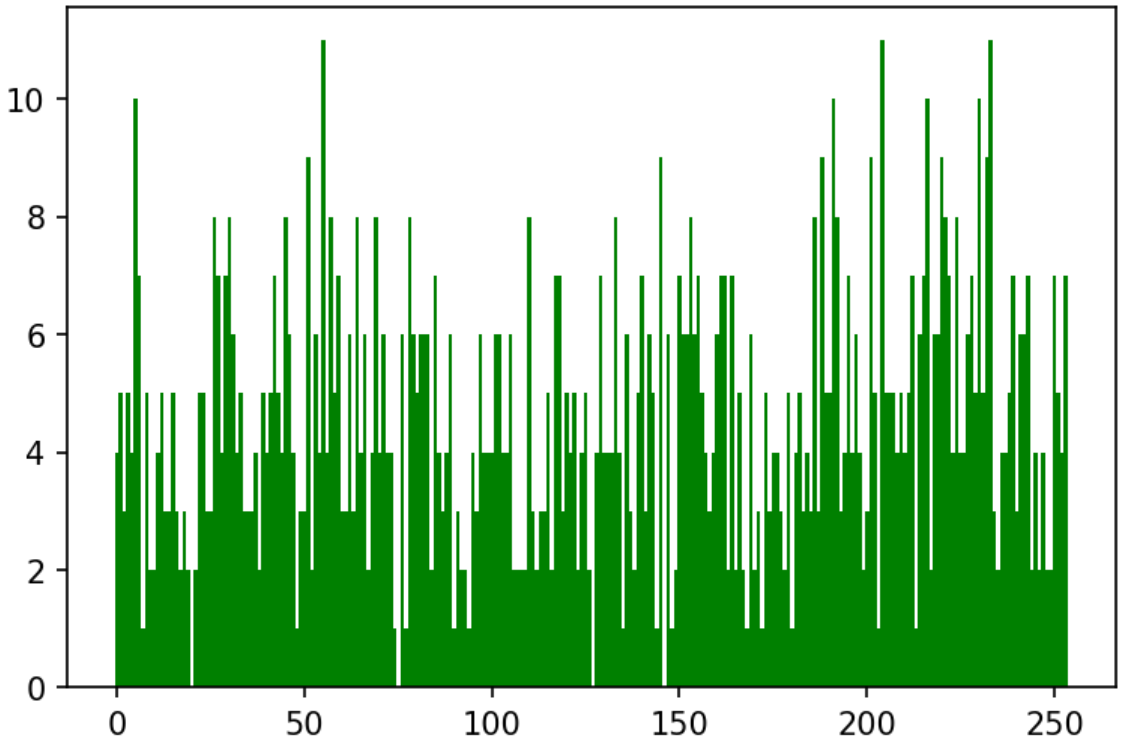
```
cipertext_freq = []
for s in spec_enc_text:
    val = {i :  list(bytes.fromhex(s)).count(i) for i in range(0,254)}
    cipertext_freq.append(val)
    plt.bar( val.keys(), val.values(), 1.0, color='g')
    plt.show()
```

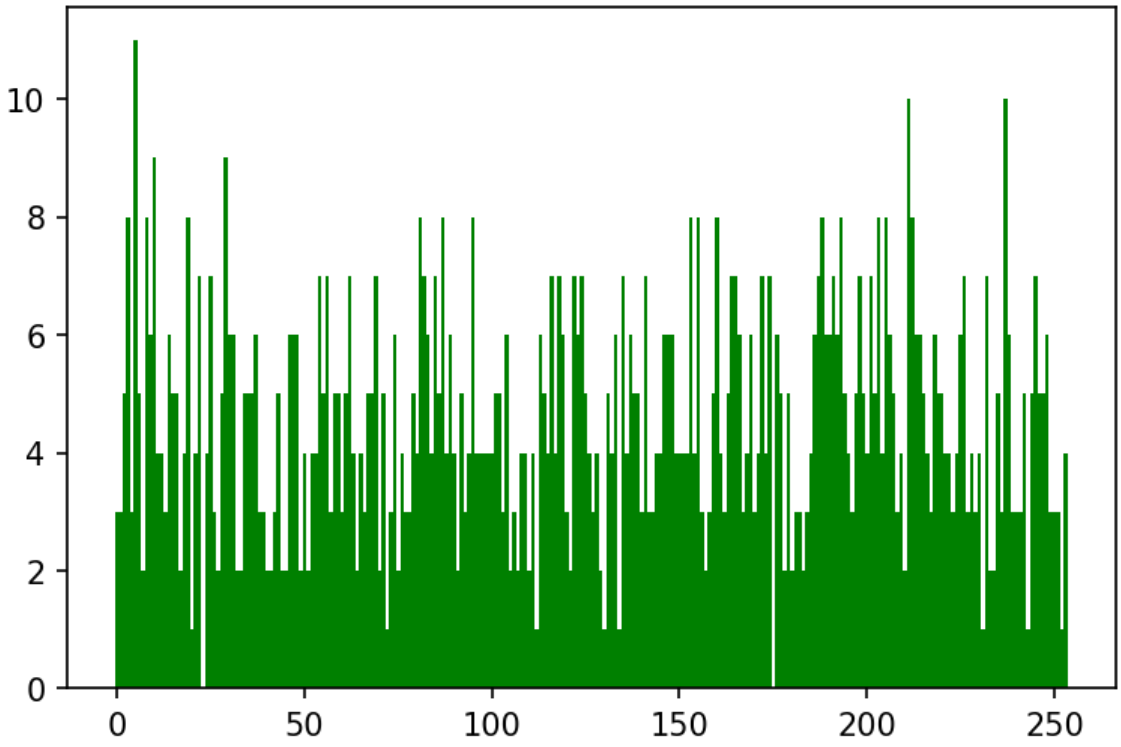

10
8
6
4
2
0
0
50
100
150
200
250

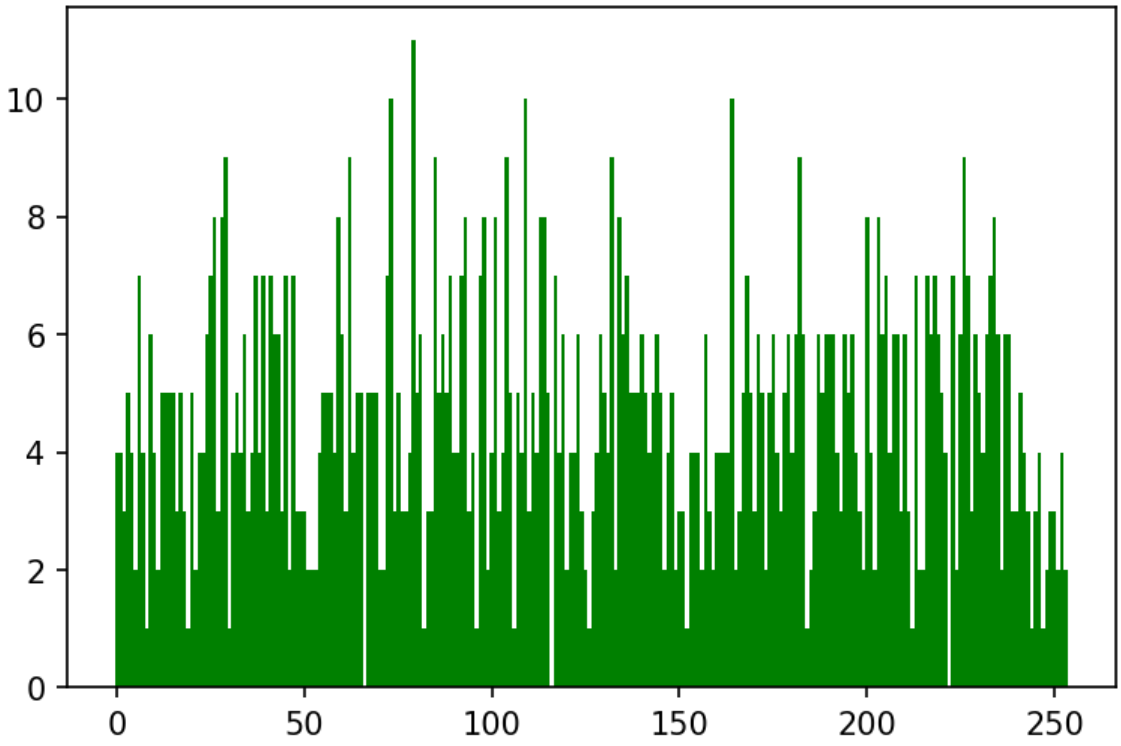

10
8
6
4
2
0
0
50
100
150
200
250

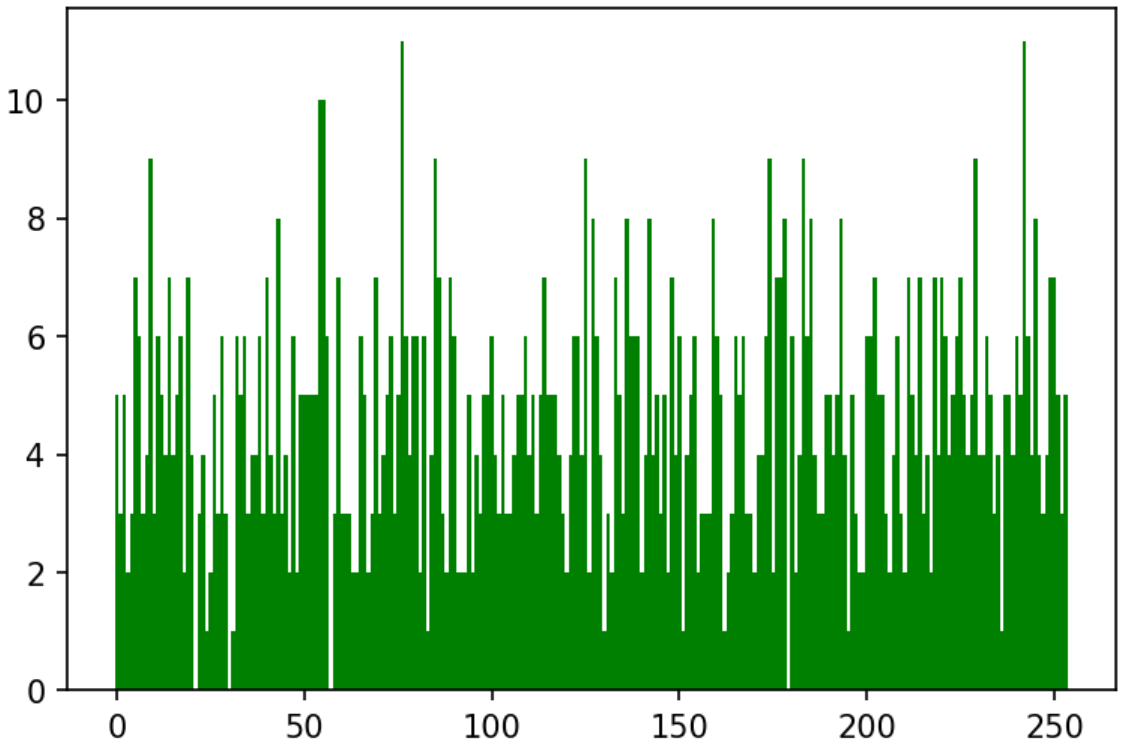

12
10
8
6
4
2
0
0
50
100
150
200
250

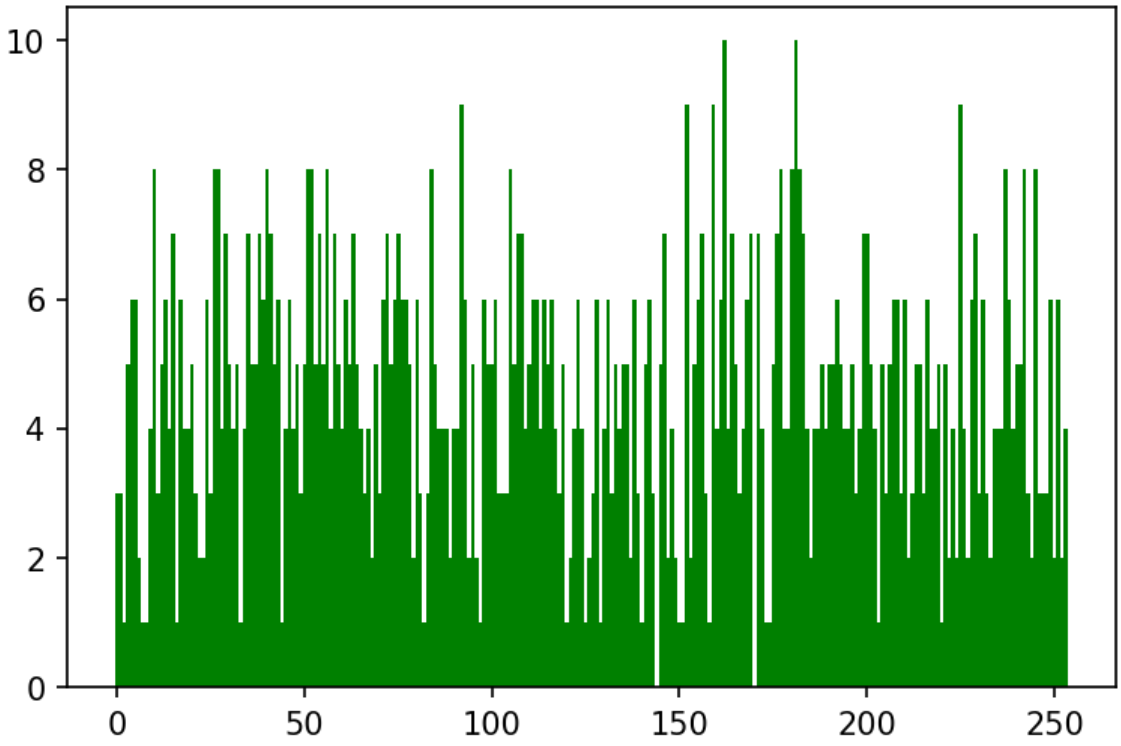

12
10
8
6
4
2
0
0
50
100
150
200
250

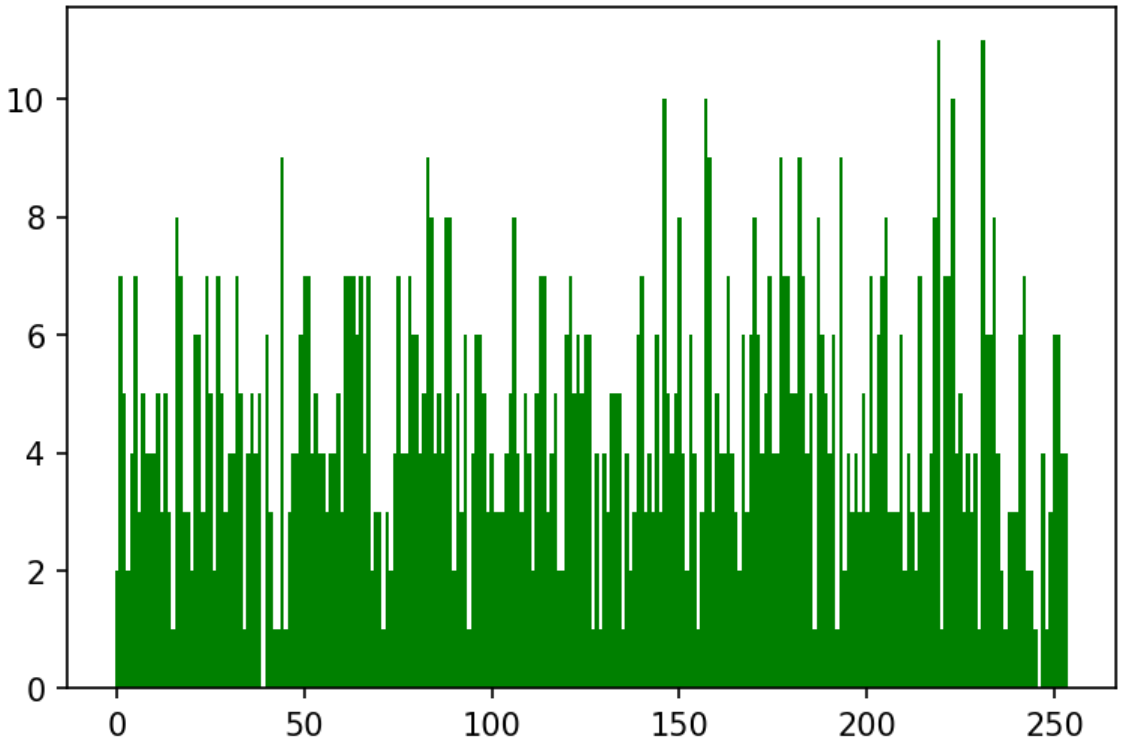

0
2
4
6
8
10
0
50
100
150
200
250


0
2
4
6
8
10
0
50
100
150
200
250


0
2
4
6
8
10
0
50
100
150
200
250


0
2
4
6
8
10
0
50
100
150
200
250


10
8
6
4
2
0
0
50
100
150
200
250


10
8
6
4
2
0
0
50
100
150
200
250

In [14]:

```
columns=[]
final_freq = [plaintext_freq]  + cipertext_freq
histo_result = np.zeros((len(final_freq)+1,34))

for row,s in enumerate(final_freq):
    current_values = list(s.values())
    for col in range(0,254,8):
        histo_result[row][col//8] = sum(current_values[col:col+8])
    histo_result[row][-1] = np.std(current_values)
    histo_result[row][-2] = np.mean(current_values)

pd.DataFrame(histo_result, columns=[f"{col}-{col+7}" for col in range(0,254,8)]+["STD",
"MEAN"])
```

Out[14]:

	0-7	8-15	16-23	24-31	32-39	40-47	48-55	56-63	64-71	72-79	...	192-199	200-207	208-215	216-223	22 2
0	0.0	3.0	0.0	0.0	192.0	21.0	3.0	3.0	10.0	6.0	...	0.0	0.0	0.0	0.0	(
1	38.0	34.0	46.0	38.0	41.0	38.0	35.0	39.0	45.0	35.0	...	37.0	41.0	35.0	42.0	34
2	45.0	33.0	30.0	25.0	45.0	38.0	34.0	41.0	46.0	35.0	...	29.0	30.0	50.0	26.0	37
3	35.0	32.0	38.0	32.0	35.0	22.0	44.0	40.0	28.0	29.0	...	28.0	31.0	28.0	39.0	28
4	48.0	37.0	38.0	29.0	28.0	41.0	33.0	37.0	37.0	40.0	...	33.0	41.0	29.0	45.0	25
5	39.0	29.0	22.0	46.0	29.0	43.0	39.0	39.0	42.0	30.0	...	38.0	44.0	39.0	52.0	49
6	40.0	45.0	31.0	42.0	31.0	28.0	34.0	39.0	33.0	27.0	...	43.0	47.0	44.0	34.0	32
7	33.0	33.0	27.0	46.0	40.0	41.0	24.0	44.0	29.0	46.0	...	33.0	45.0	30.0	42.0	42
8	34.0	42.0	31.0	21.0	37.0	37.0	47.0	27.0	32.0	46.0	...	30.0	38.0	37.0	39.0	44
9	27.0	38.0	27.0	45.0	40.0	41.0	46.0	46.0	32.0	44.0	...	38.0	36.0	33.0	31.0	39
10	35.0	29.0	38.0	37.0	31.0	28.0	41.0	40.0	33.0	37.0	...	31.0	41.0	30.0	51.0	35
11	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	(

12 rows × 34 columns

HW1 - part 3 - Rand

In [8]:

```
import sys, random,binascii,AES_random,jupyter_utils
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from IPython.display import Markdown, display,HTML
%matplotlib inline
plt.rcParams['figure.dpi'] = 150
```

We would like to test these principles when operating AES in different situations Note that for the purpose of the exercise, we will allow ourselves a number of mitigating assumptions which will be detailed below

To do this, take the following text which was published today on the BBC website

In [9]:

```
with open("plaintext.txt","r") as f:
    plaintext = f.read()
print("Total Length:",len(plaintext))
display(Markdown(f'\n"**_{plaintext}_**"'))
```

```
Total Length: 1168
```

"***From the Pope to Greta Thunberg, there are growing calls for the crime of "ecocide" to be recognised in international criminal law - but could such a law ever work? In December 2019, at the International Criminal Court in the Hague, Vanuatu's ambassador to the European Union made a radical suggestion: make the destruction of the environment a crime. Vanuatu is a small island state in the South Pacific, a nation severely threatened by rising sea levels. Climate change is an imminent and existential crisis in the country, yet the actions that have caused rising temperatures - such as burning fossil fuels - have almost entirely taken place elsewhere, to serve other nations, with the blessing of state governments. Small island states like Vanuatu have long tried to persuade large powerful nations to voluntarily reduce their emissions, but change has been slow - so ambassador John Licht suggested that it might be time to change the law itself. An amendment to a treaty known as the Rome Statute, which established the International Criminal Court, could criminalise acts that amount to ecocide, he said, arguing "this radical idea merits serious discussion".***"

And we will try to encrypt the message, using an encryption key **"R is a Short key"**

In [10]:

```
key = "R is a Short key"
key = key.encode().hex()
print(f"'R is a Short key':\n 0x{key}")
aes_lib_rand = AES_random.AES(int(key,base=16))
```

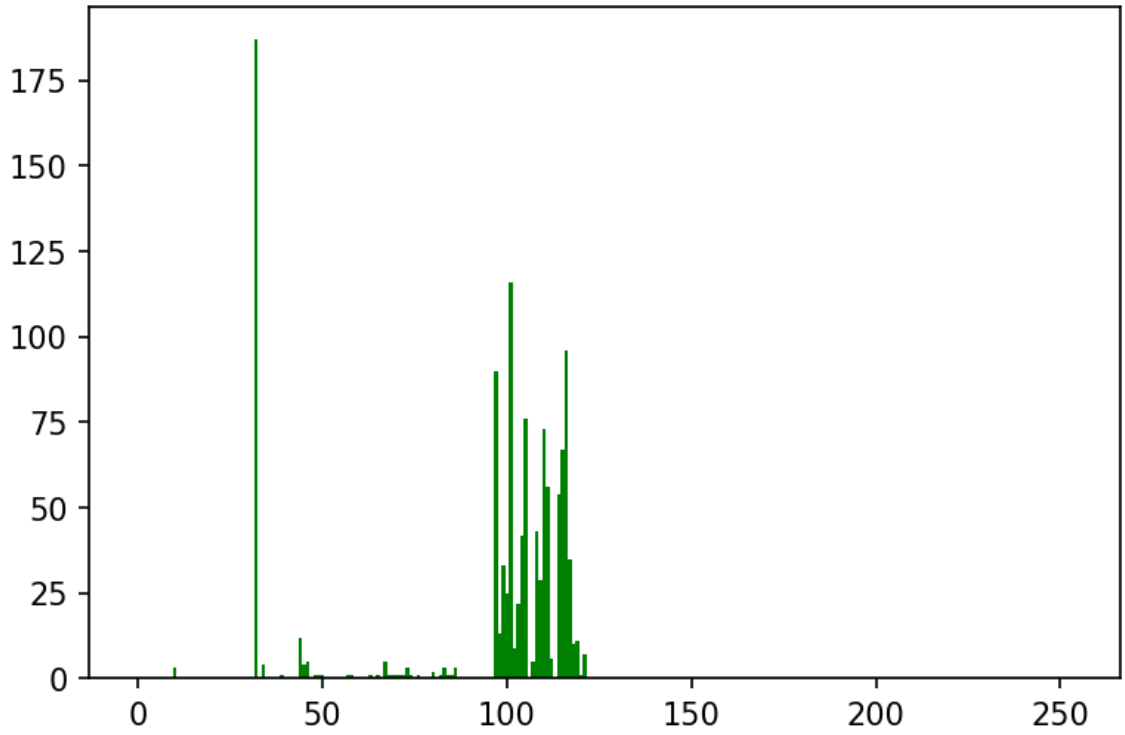
```
'R is a Short key':
 0x52206973206120536686f7274206b6579
```

Part One

1. Create a letter scatter histogram for each source message. Note that the message is divided into 16-letter blocks (aka STATE). The sotogram will collect the over the various statistics across the entire run (i.e. amount for the entire message)
2. We watched the article using AES, and after each round of AES and for the encrypted message at the end. Calculate the scatter plot of the letters throughout the entire message, some in the first section
3. Create one table that summarizes all the data and adds the average and standard deviation to each row

In [11]:

```
plaintext_binary = list(plaintext.encode())
plaintext_freq =  {i : plaintext_binary.count(i) for i in range(0,254)}
plt.bar( plaintext_freq.keys(), plaintext_freq.values(), 1.0, color='g')
plt.show()
```

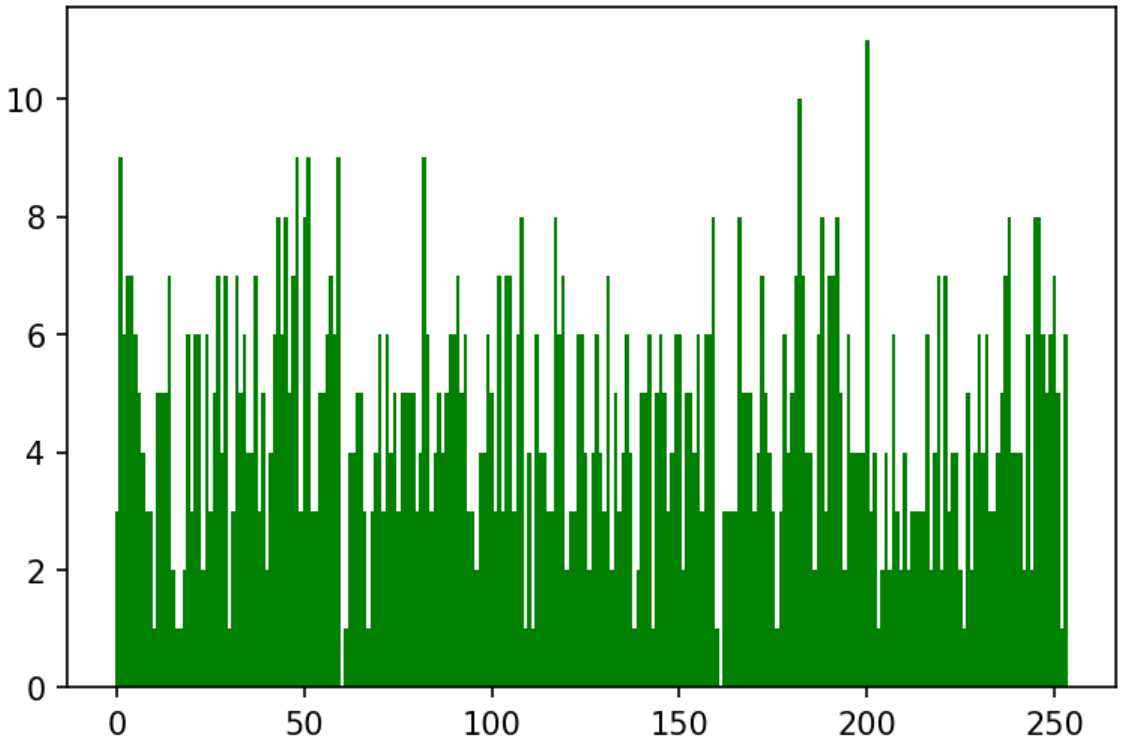


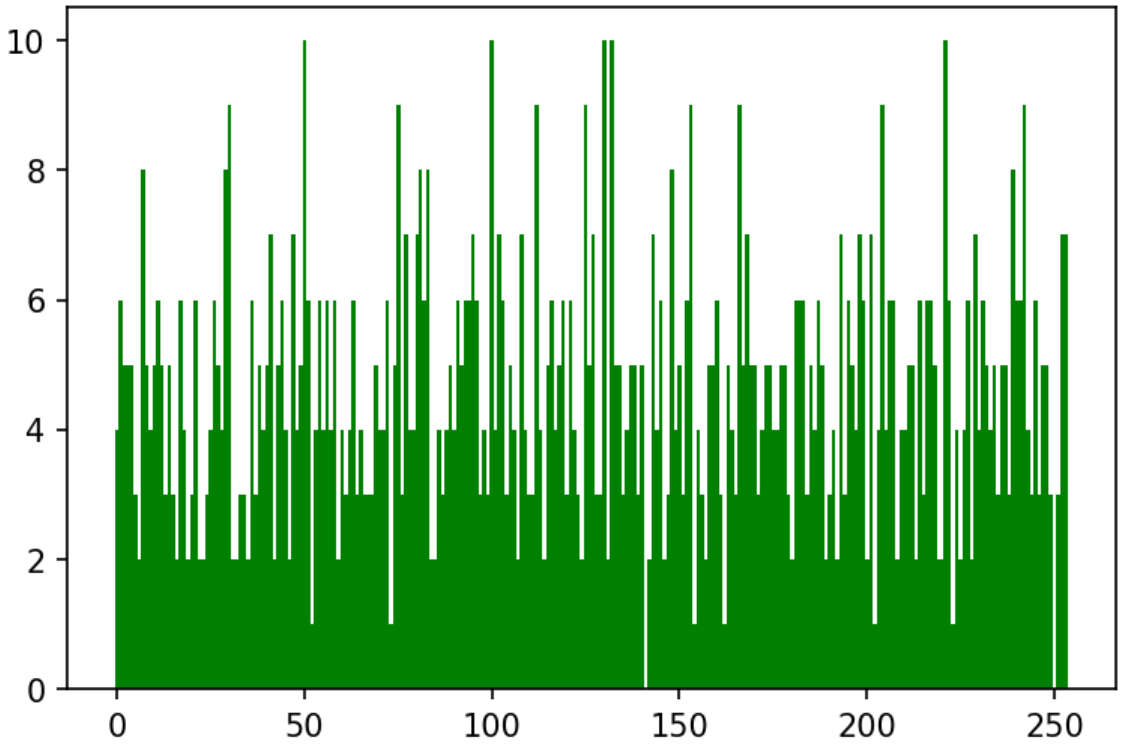
In [12]:

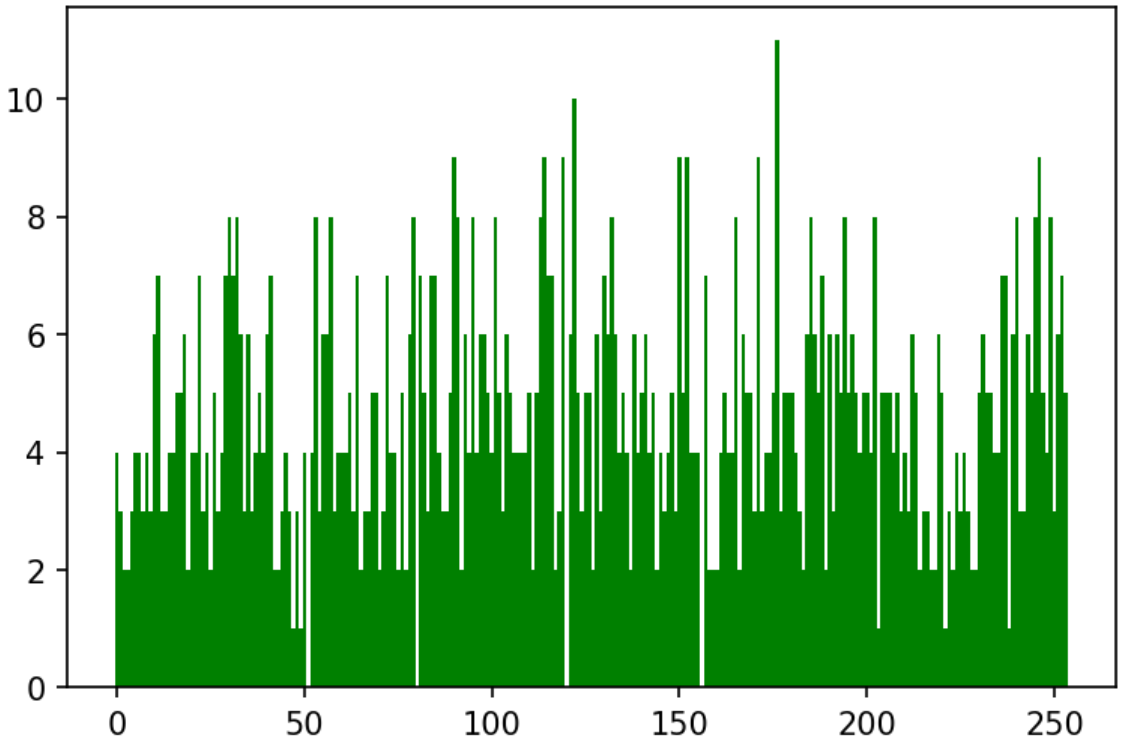
```
import binascii
spec_enc_text = [""]*10

for i in range(0, len(plaintext), 16):
    ciper = aes_lib_rand.encrypt_by_stage(int(plaintext[i:i+16].encode().hex(),base=16
))
    for j,c in enumerate(ciper):
        spec_enc_text[j] += ('{:x}'.format(c)).zfill(32)
```

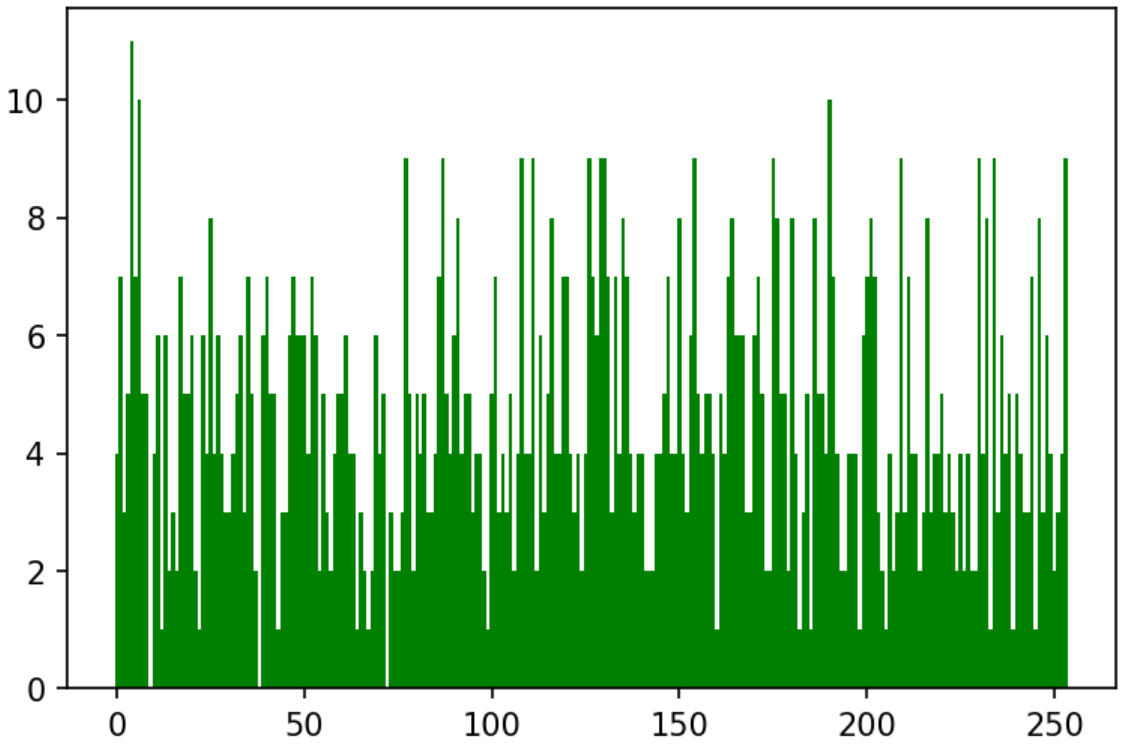
In [13]:

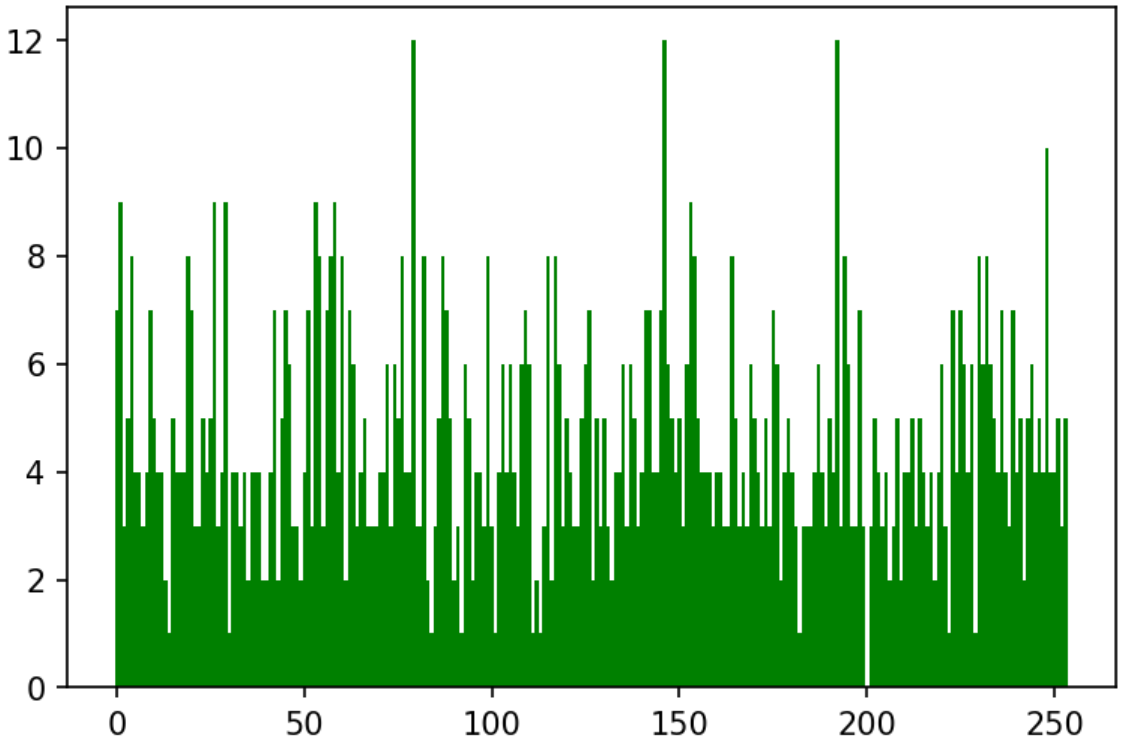
```
cipertext_freq = []
for s in spec_enc_text:
    val = {i :  list(bytes.fromhex(s)).count(i) for i in range(0,254)}
    cipertext_freq.append(val)
    plt.bar( val.keys(), val.values(), 1.0, color='g')
    plt.show()
```

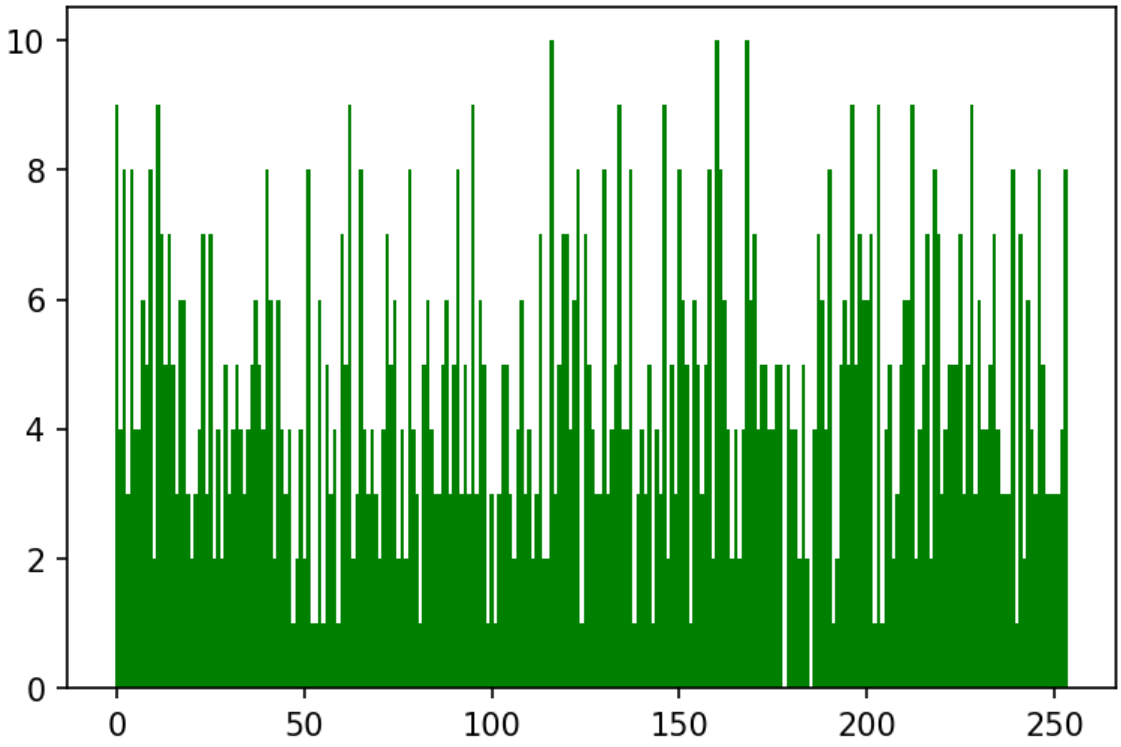

10
8
6
4
2
0
0
50
100
150
200
250

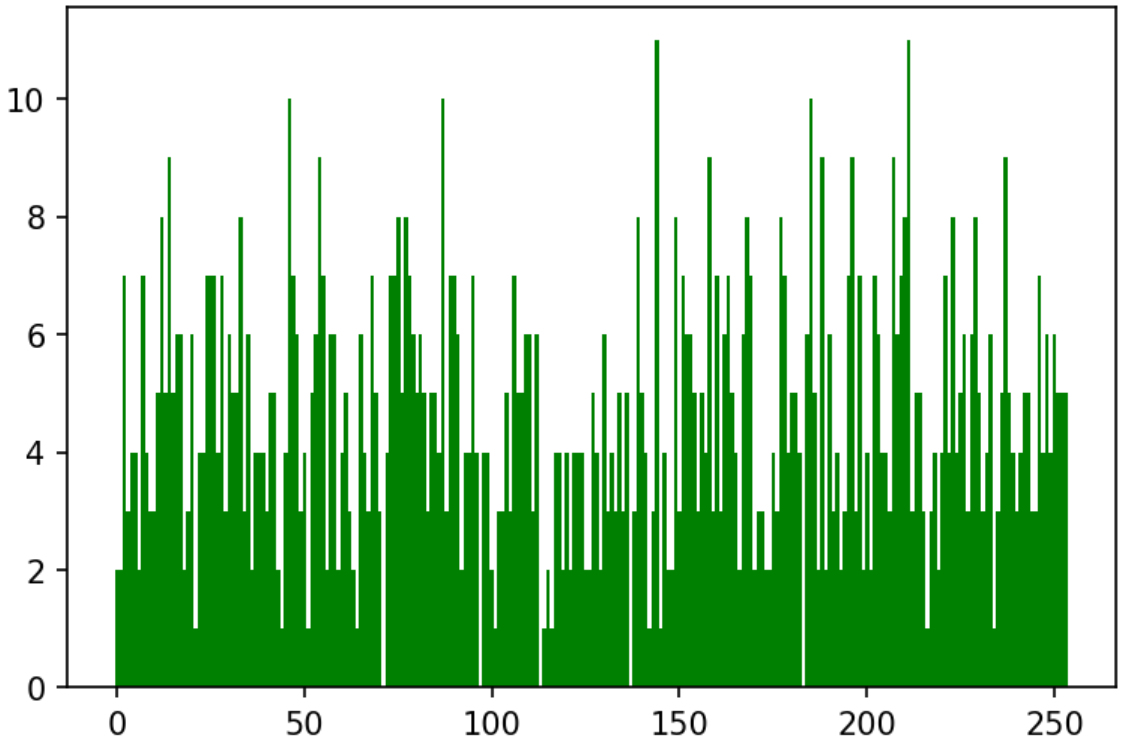

10
8
6
4
2
0
0
50
100
150
200
250

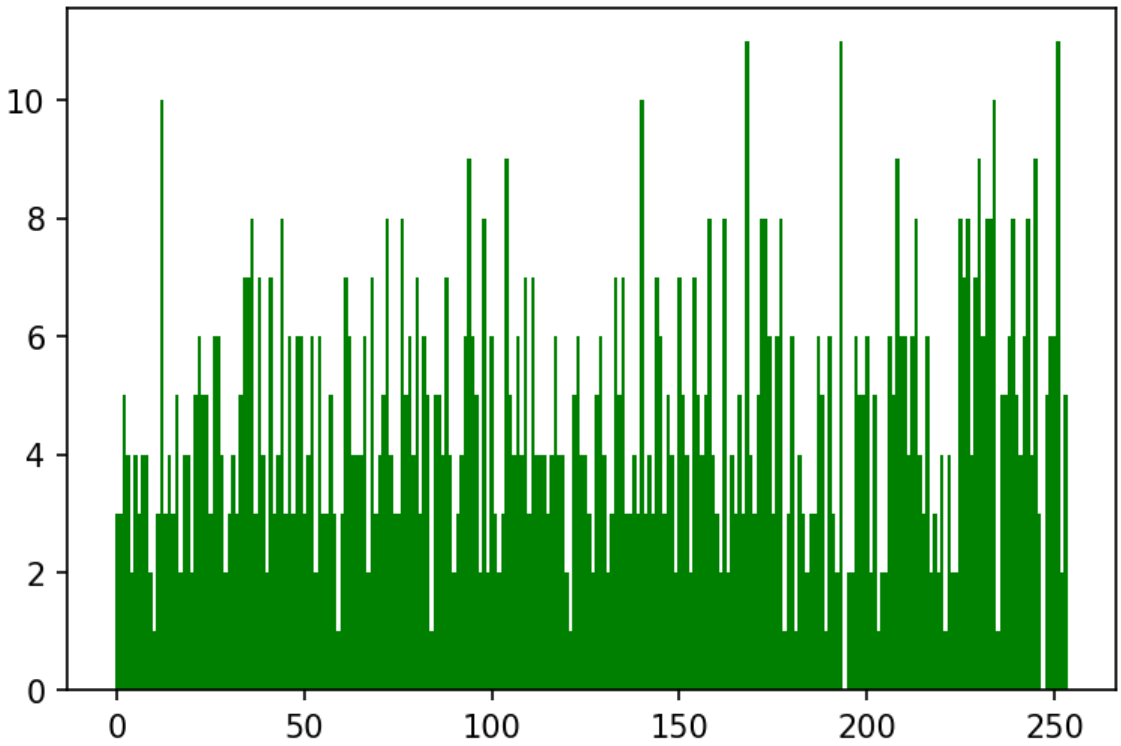

0
2
4
6
8
10
0
50
100
150
200
250

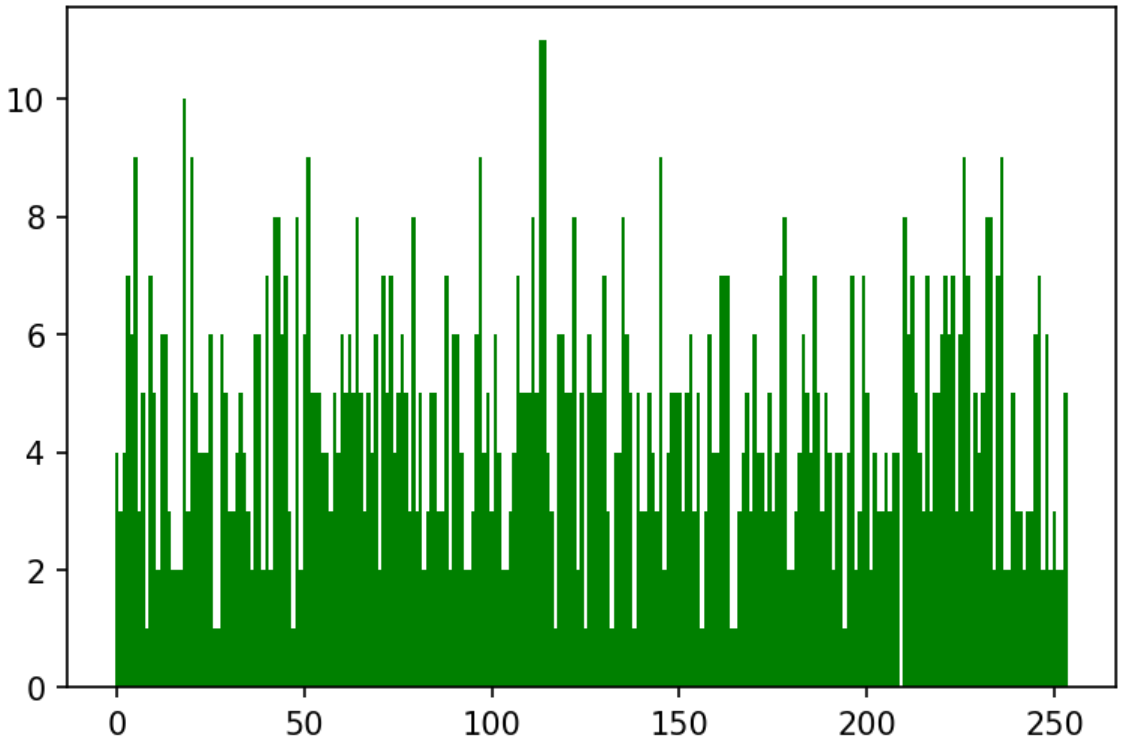

0
2
4
6
8
10
0
50
100
150
200
250

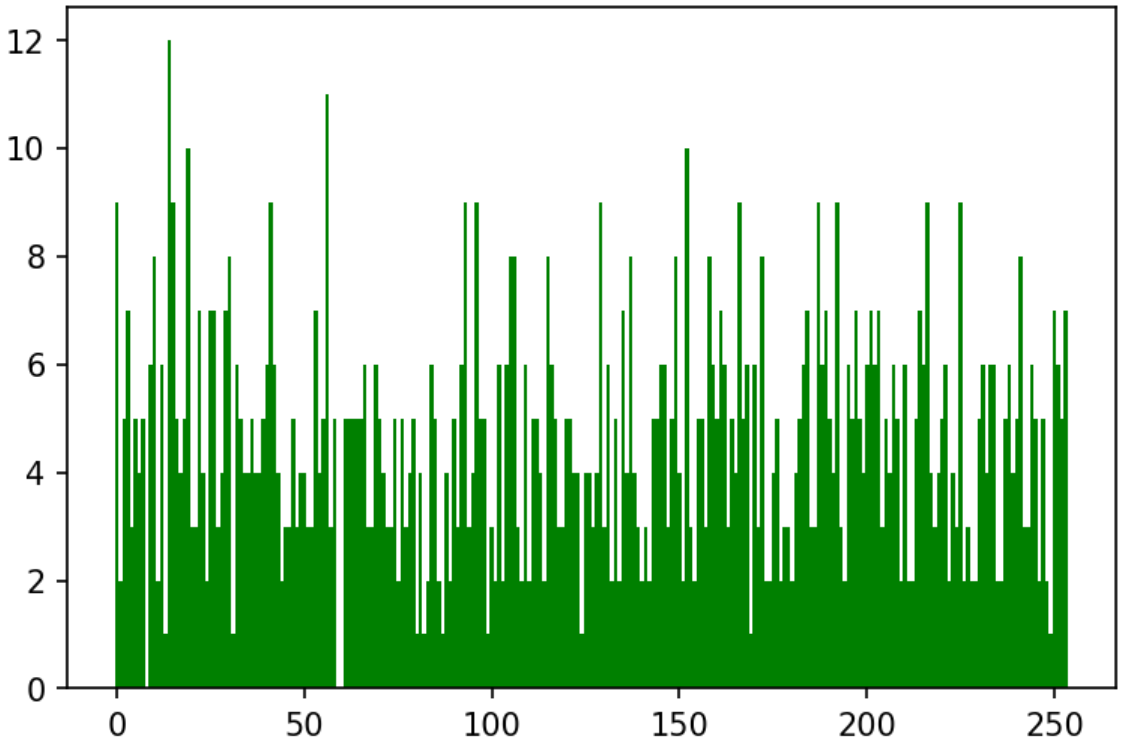

12
10
8
6
4
2
0
0
50
100
150
200
250


10
8
6
4
2
0
0
50
100
150
200
250


0
2
4
6
8
10
0
50
100
150
200
250


0
2
4
6
8
10
0
50
100
150
200
250


0
2
4
6
8
10
0
50
100
150
200
250


0
2
4
6
8
10
12
0
50
100
150
200
250

In [14]:

```
columns=[]
final_freq = [plaintext_freq]  + cipertext_freq
histo_result = np.zeros((len(final_freq),34))

for row,s in enumerate(final_freq):
    current_values = list(s.values())
    for col in range(0,254,8):
        histo_result[row][col//8] = sum(current_values[col:col+8])
    histo_result[row][-1] = np.std(current_values)
    histo_result[row][-2] = np.mean(current_values)

display(pd.DataFrame(histo_result, columns=[f"{col}-{col+7}" for col in range(0,254,8)]
+["STD","MEAN"]))
```

	0-7	8-15	16-23	24-31	32-39	40-47	48-55	56-63	64-71	72-79	...	192-199	200-207	208-215	216-223	22 2
0	0.0	3.0	0.0	0.0	192.0	21.0	3.0	3.0	10.0	6.0	...	0.0	0.0	0.0	0.0	(
1	47.0	31.0	27.0	36.0	41.0	46.0	45.0	37.0	30.0	38.0	...	37.0	33.0	23.0	35.0	28
2	38.0	36.0	27.0	41.0	28.0	38.0	40.0	35.0	29.0	39.0	...	40.0	39.0	31.0	38.0	35
3	25.0	34.0	36.0	40.0	39.0	28.0	29.0	37.0	30.0	38.0	...	44.0	37.0	31.0	24.0	29
4	52.0	27.0	34.0	36.0	34.0	37.0	42.0	33.0	24.0	26.0	...	27.0	34.0	35.0	34.0	29
5	43.0	32.0	38.0	39.0	27.0	36.0	39.0	51.0	29.0	48.0	...	45.0	24.0	32.0	30.0	42
6	46.0	48.0	34.0	30.0	36.0	34.0	25.0	36.0	31.0	38.0	...	45.0	35.0	40.0	41.0	42
7	31.0	42.0	32.0	46.0	36.0	37.0	41.0	30.0	29.0	52.0	...	37.0	39.0	48.0	33.0	40
8	28.0	30.0	33.0	33.0	44.0	36.0	36.0	32.0	35.0	41.0	...	33.0	29.0	46.0	24.0	51
9	41.0	32.0	39.0	29.0	32.0	42.0	44.0	38.0	40.0	43.0	...	32.0	28.0	37.0	46.0	42
10	40.0	44.0	41.0	39.0	37.0	38.0	33.0	34.0	37.0	30.0	...	41.0	44.0	35.0	37.0	32

11 rows × 34 columns

HW1 - part 3 - Seq

In [8]:

```
import sys, random,binascii,AES_Continuous,jupyter_utils
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from IPython.display import Markdown, display,HTML
%matplotlib inline
plt.rcParams['figure.dpi'] = 150
```

We would like to test these principles when operating AES in different situations Note that for the purpose of the exercise, we will allow ourselves a number of mitigating assumptions which will be detailed below

To do this, take the following text which was published today on the BBC website

In [9]:

```
with open("plaintext.txt","r") as f:
    plaintext = f.read()
print("Total Length:",len(plaintext))
display(Markdown(f'\n"**_{plaintext}_**"'))
```

```
Total Length: 1168
```

"***From the Pope to Greta Thunberg, there are growing calls for the crime of "ecocide" to be recognised in international criminal law - but could such a law ever work? In December 2019, at the International Criminal Court in the Hague, Vanuatu's ambassador to the European Union made a radical suggestion: make the destruction of the environment a crime. Vanuatu is a small island state in the South Pacific, a nation severely threatened by rising sea levels. Climate change is an imminent and existential crisis in the country, yet the actions that have caused rising temperatures - such as burning fossil fuels - have almost entirely taken place elsewhere, to serve other nations, with the blessing of state governments. Small island states like Vanuatu have long tried to persuade large powerful nations to voluntarily reduce their emissions, but change has been slow - so ambassador John Licht suggested that it might be time to change the law itself. An amendment to a treaty known as the Rome Statute, which established the International Criminal Court, could criminalise acts that amount to ecocide, he said, arguing "this radical idea merits serious discussion".***"

And we will try to encrypt the message, using an encryption key **"R is a Short key"**

In [10]:

```
key = "R is a Short key"
key = key.encode().hex()
print(f"'R is a Short key':\n 0x{key}")
aes_lib_seq = AES_Continuous.AES(int(key,base=16))
```

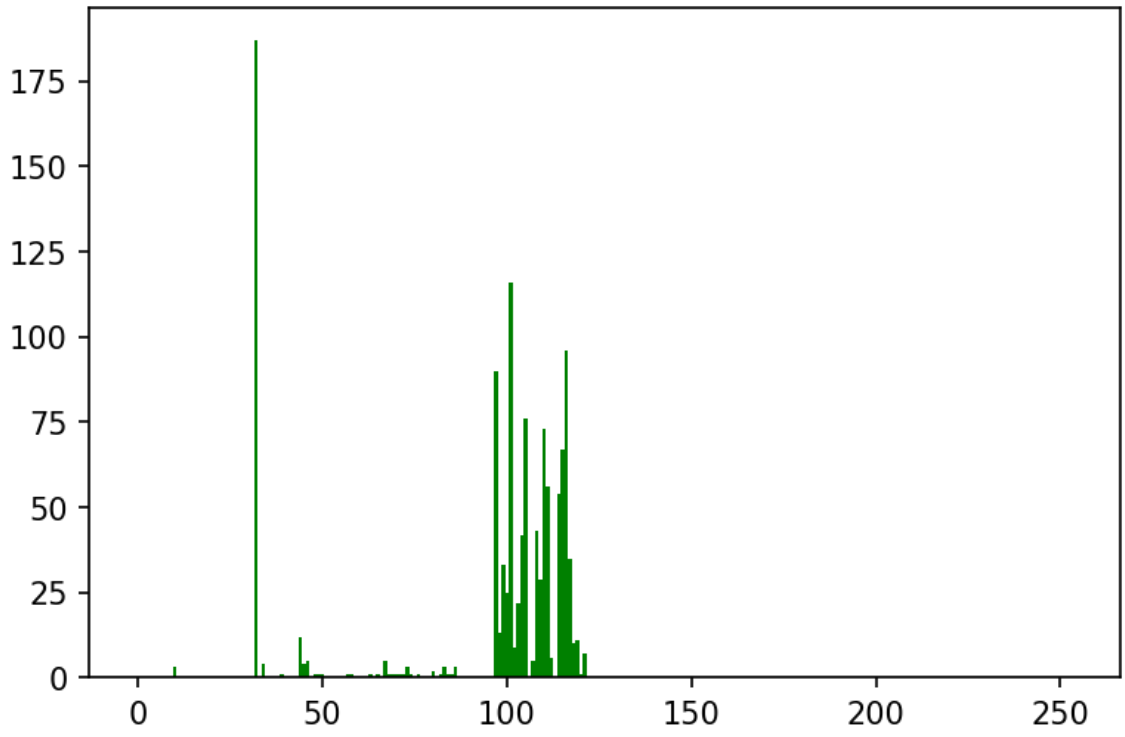
```
'R is a Short key':
 0x52206973206120536686f7274206b6579
```

Part One

1. Create a letter scatter histogram for each source message. Note that the message is divided into 16-letter blocks (aka STATE). The sotogram will collect the over the various statistics across the entire run (i.e. amount for the entire message)
2. We watched the article using AES, and after each round of AES and for the encrypted message at the end. Calculate the scatter plot of the letters throughout the entire message, some in the first section
3. Create one table that summarizes all the data and adds the average and standard deviation to each row

In [11]:

```
plaintext_binary = list(plaintext.encode())
plaintext_freq =  {i : plaintext_binary.count(i) for i in range(0,254)}
plt.bar( plaintext_freq.keys(), plaintext_freq.values(), 1.0, color='g')
plt.show()
```

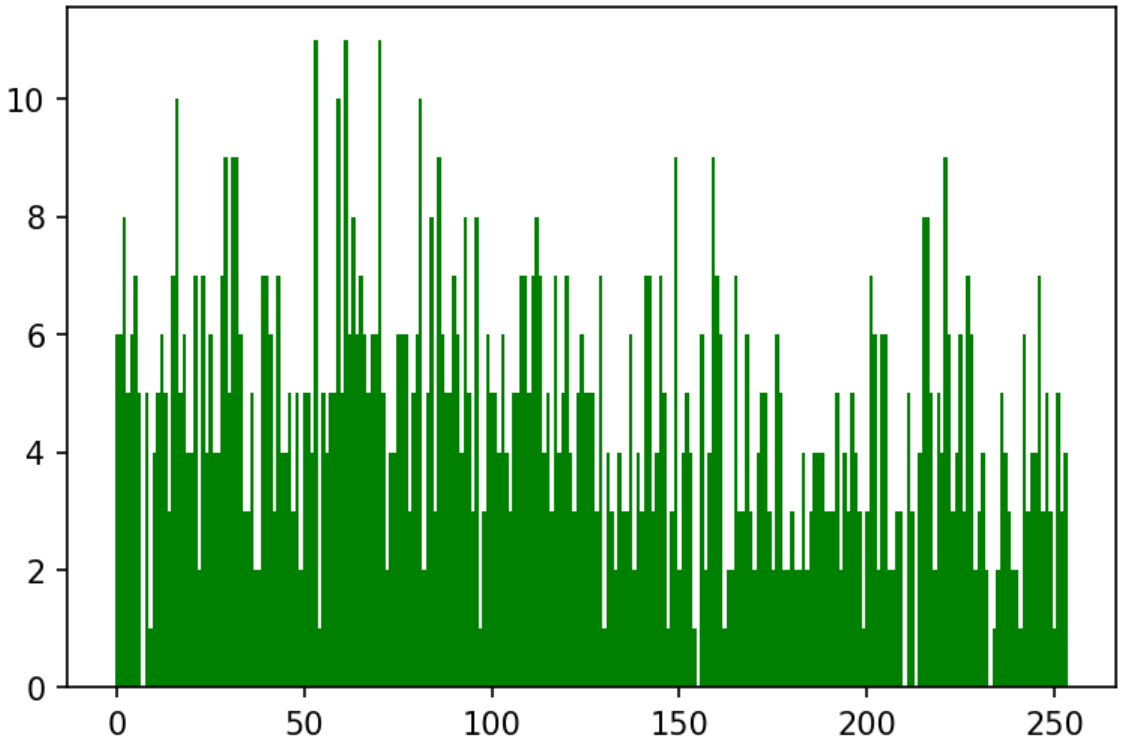


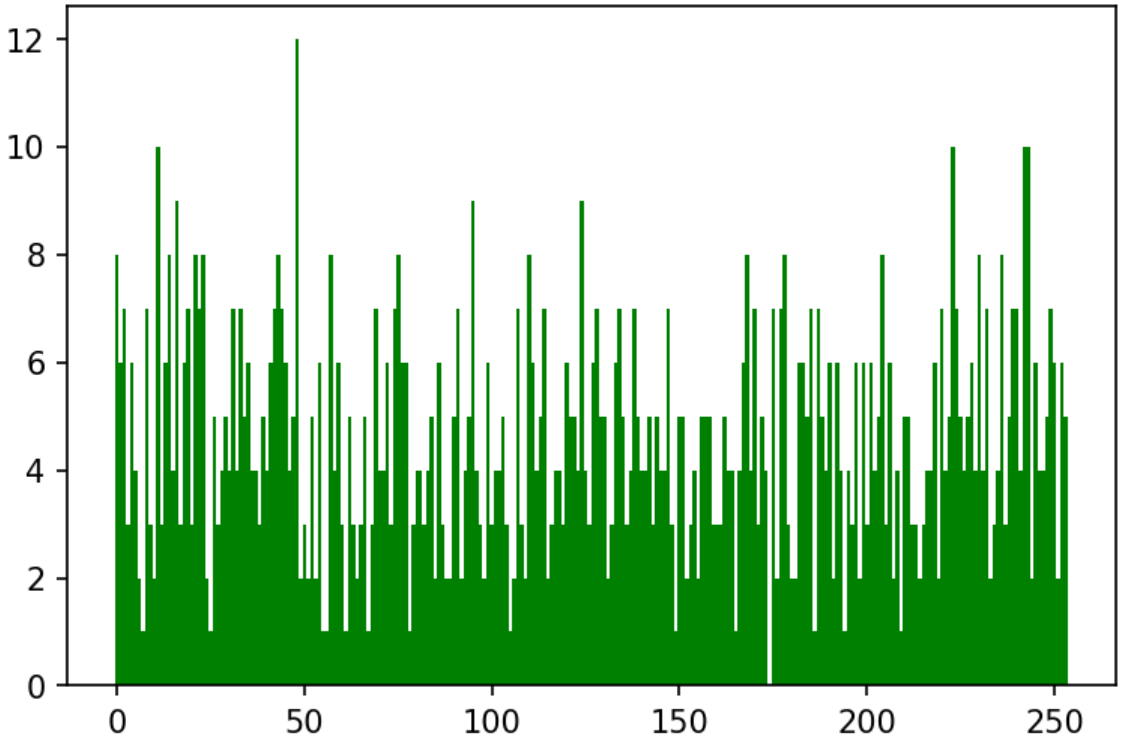
In [12]:

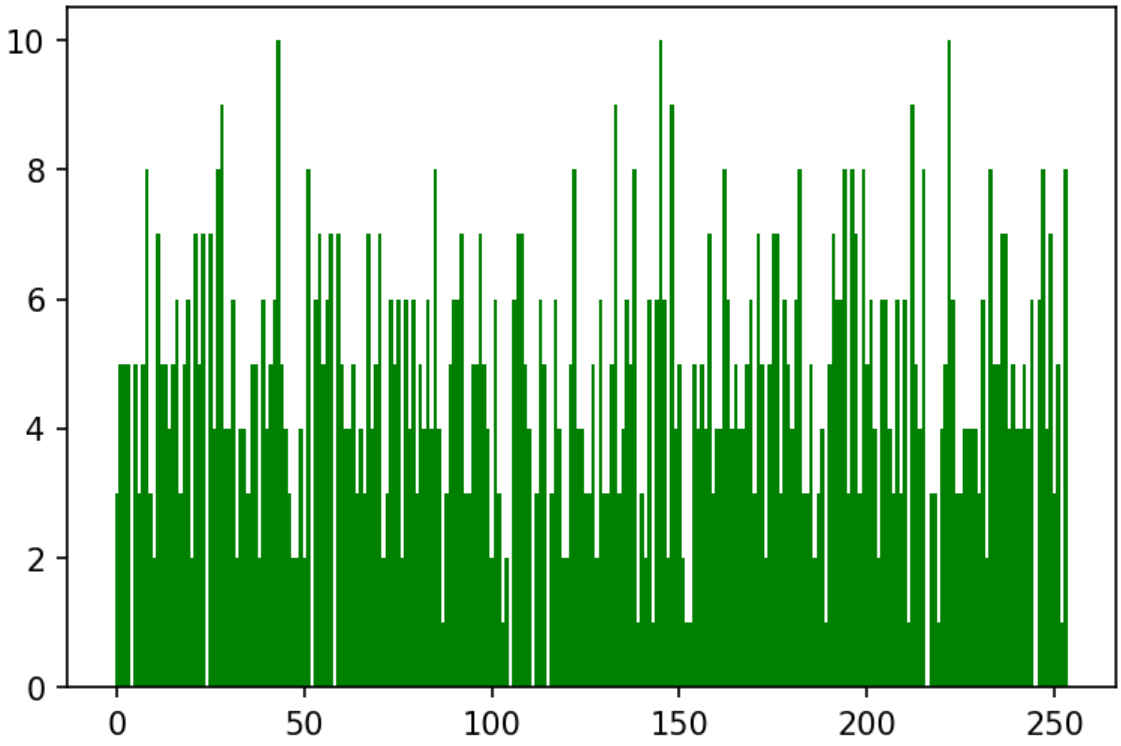
```
import binascii
spec_enc_text = [""]*10

for i in range(0, len(plaintext), 16):
    ciper = aes_lib_seq.encrypt_by_stage(int(plaintext[i:i+16].encode().hex(),base=16))
    for j,c in enumerate(ciper):
        spec_enc_text[j] += ('{:x}'.format(c)).zfill(32)
```

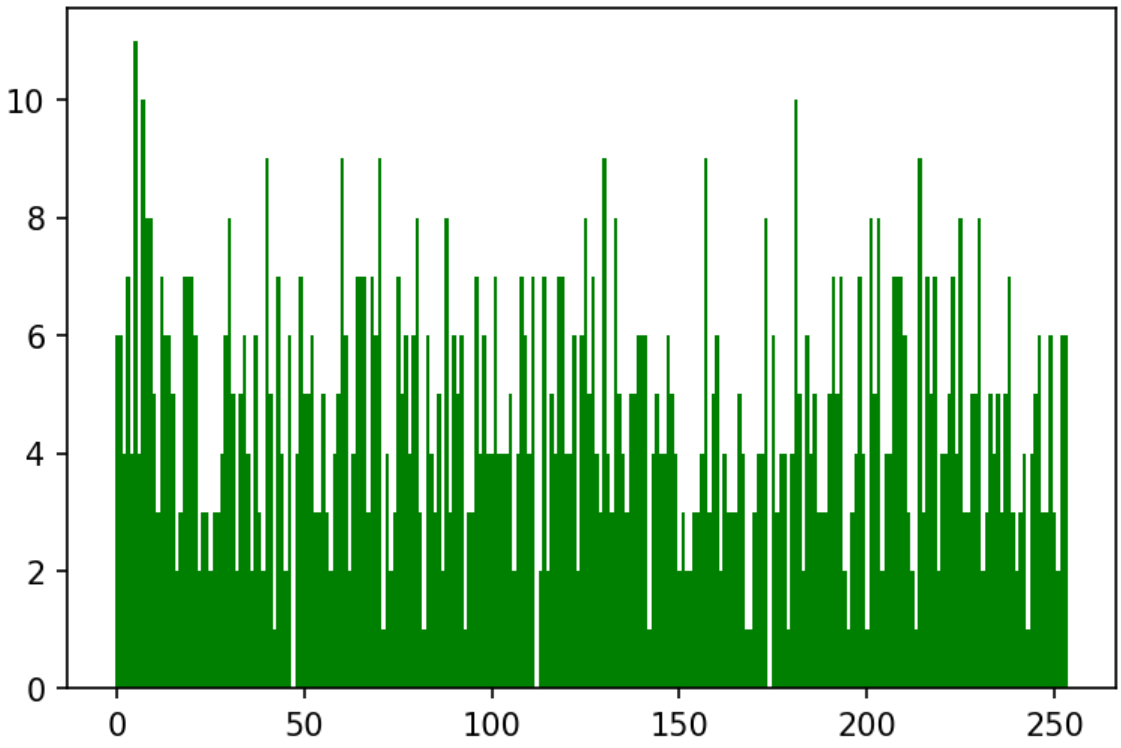
In [13]:

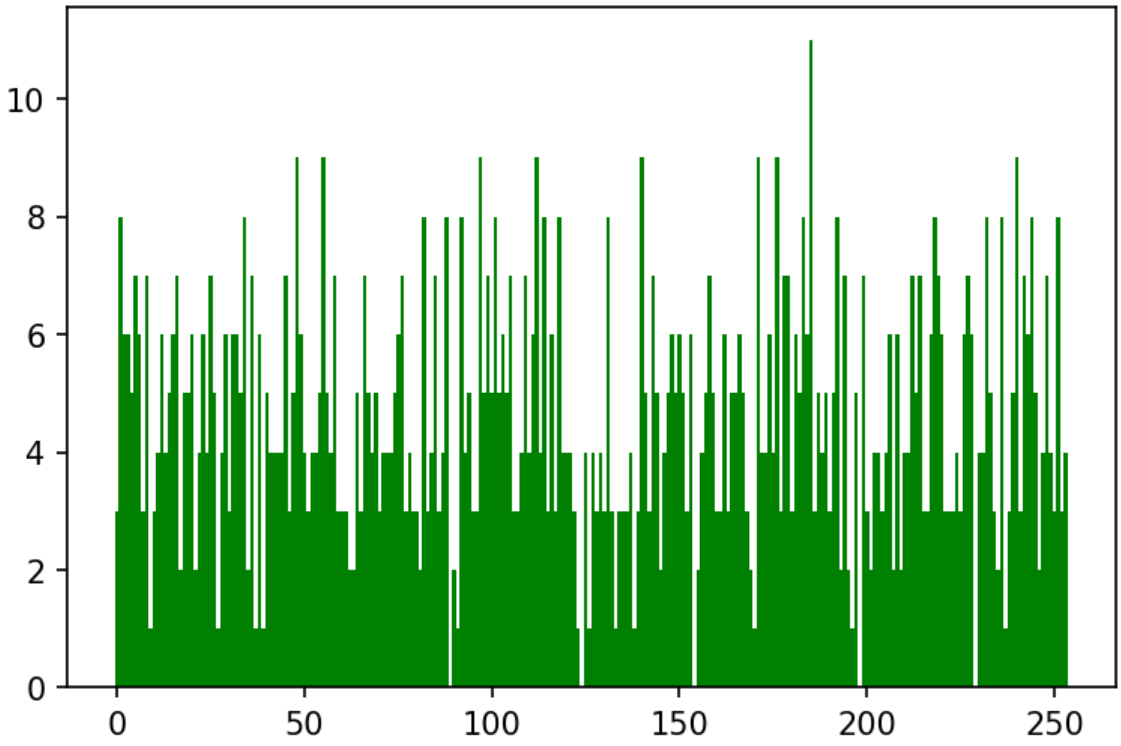
```
cipertext_freq = []
for s in spec_enc_text:
    val = {i :  list(bytes.fromhex(s)).count(i) for i in range(0,254)}
    cipertext_freq.append(val)
    plt.bar( val.keys(), val.values(), 1.0, color='g')
    plt.show()
```

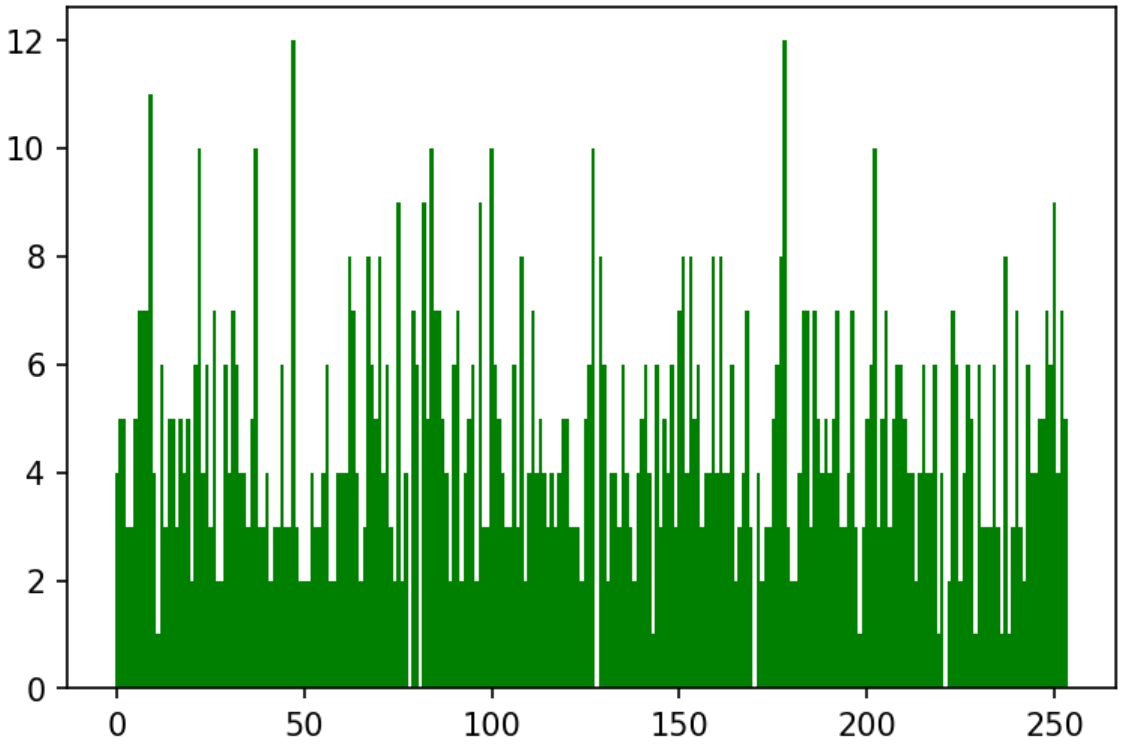

0
2
4
6
8
10
0
50
100
150
200
250

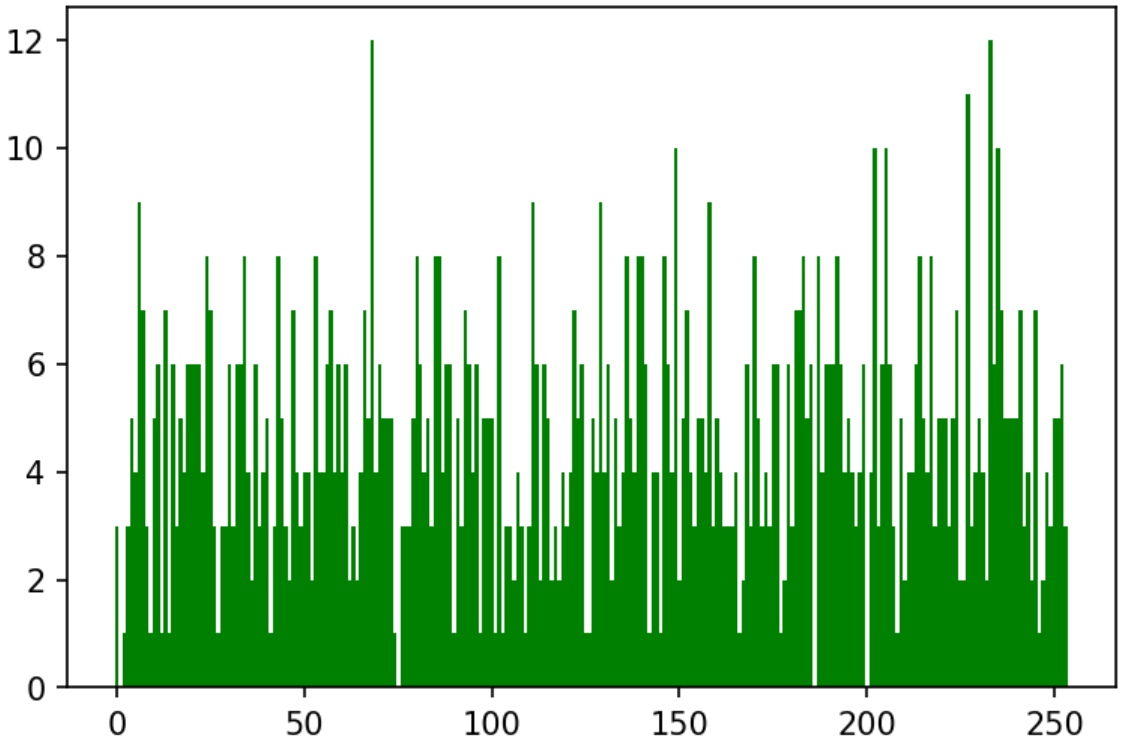

0
2
4
6
8
10
12
0
50
100
150
200
250

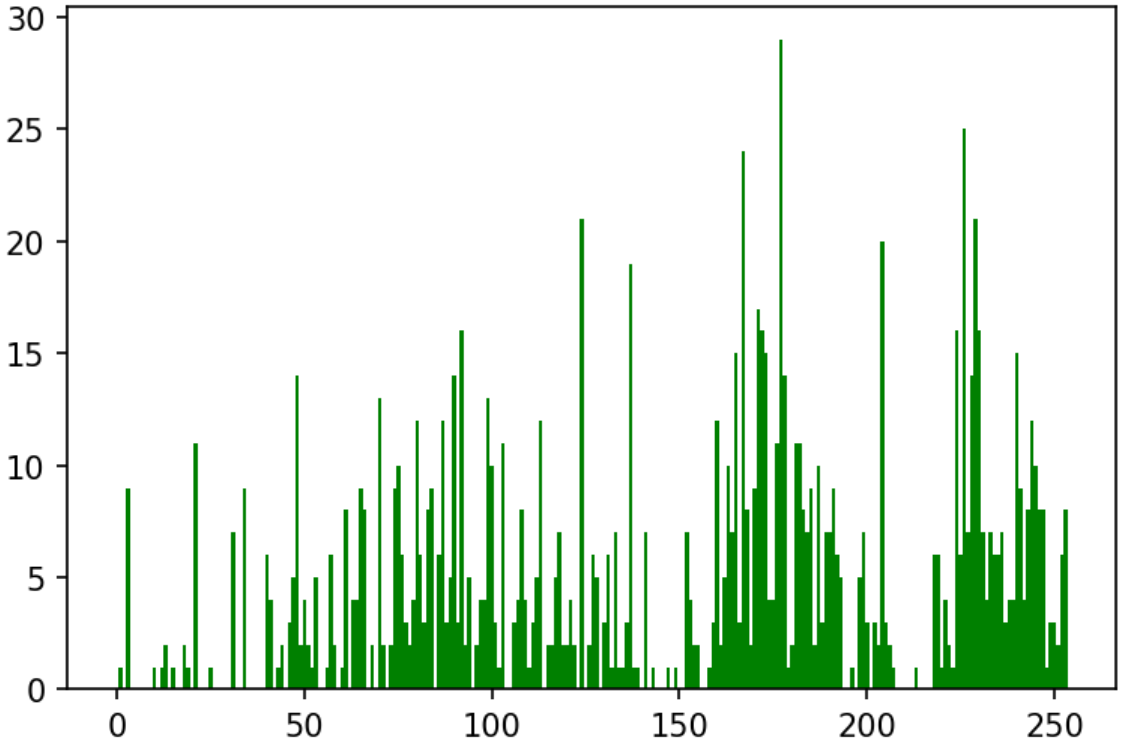

10
8
6
4
2
0
0
50
100
150
200
250

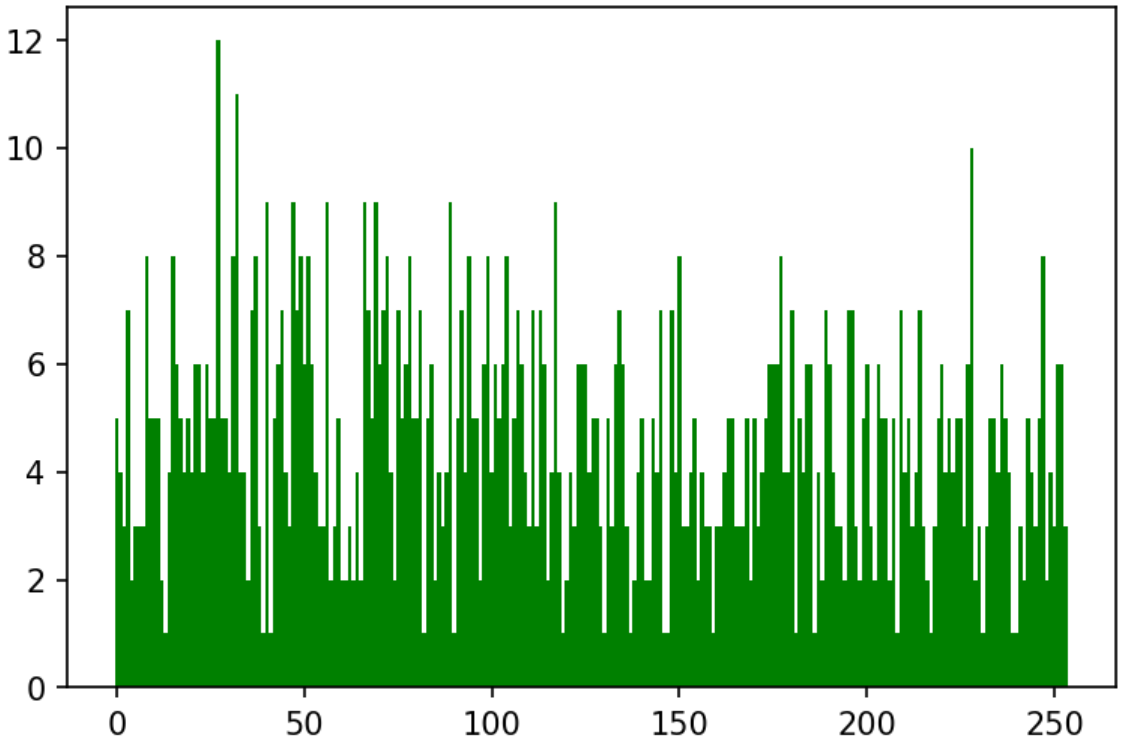

10
8
6
4
2
0
0
50
100
150
200
250

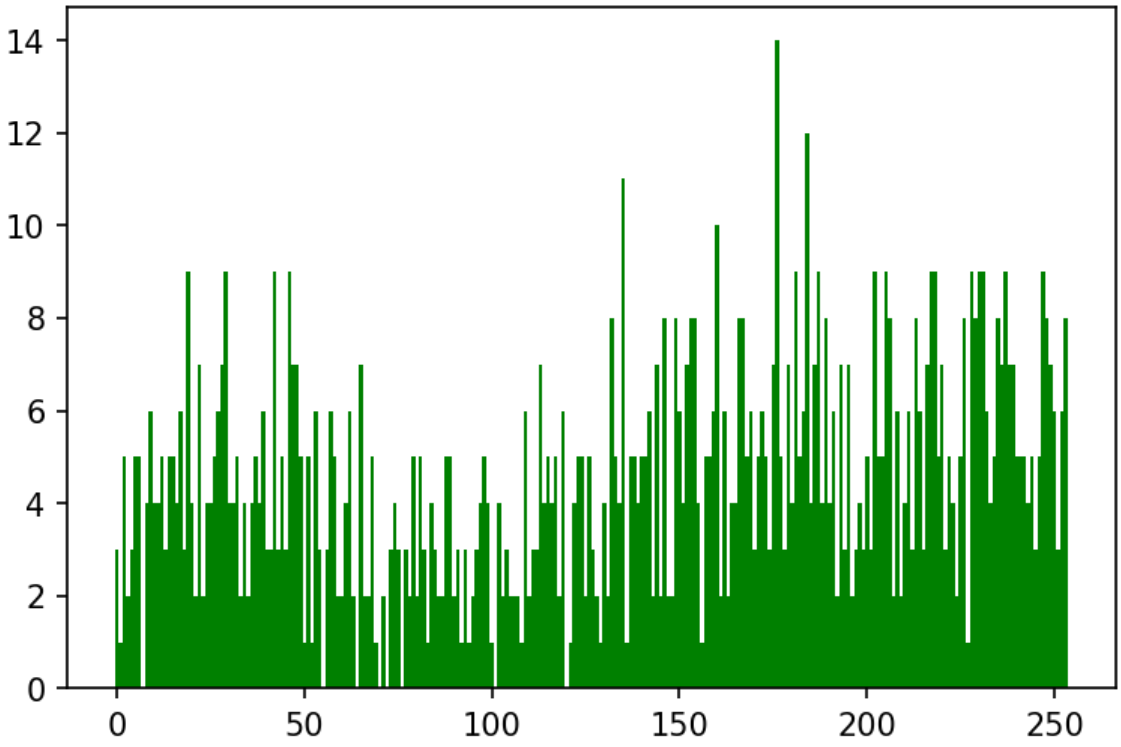

0
2
4
6
8
10
0
50
100
150
200
250


0
2
4
6
8
10
12
0
50
100
150
200
250


12
10
8
6
4
2
0
0
50
100
150
200
250


30
25
20
15
10
5
0
0
50
100
150
200
250


12
10
8
6
4
2
0
0
50
100
150
200
250


14
12
10
8
6
4
2
0
0
50
100
150
200
250

In [14]:

```
columns=[]
final_freq = [plaintext_freq]  + cipertext_freq
histo_result = np.zeros((len(final_freq),34))

for row,s in enumerate(final_freq):
    current_values = list(s.values())
    for col in range(0,254,8):
        histo_result[row][col//8] = sum(current_values[col:col+8])
    histo_result[row][-1] = np.std(current_values)
    histo_result[row][-2] = np.mean(current_values)

pd.DataFrame(histo_result, columns=[f"{col}-{col+7}" for col in range(0,254,8)]+["STD",
"MEAN"])
```

Out[14]:

	0-7	8-15	16-23	24-31	32-39	40-47	48-55	56-63	64-71	72-79	...	192-199	200-207	208-215	216-223	2
0	0.0	3.0	0.0	0.0	192.0	21.0	3.0	3.0	10.0	6.0	...	0.0	0.0	0.0	0.0	
1	43.0	36.0	45.0	48.0	37.0	39.0	38.0	54.0	52.0	36.0	...	27.0	34.0	26.0	42.0	3
2	37.0	43.0	51.0	31.0	38.0	47.0	33.0	31.0	29.0	40.0	...	32.0	37.0	26.0	42.0	4
3	31.0	39.0	41.0	42.0	31.0	39.0	34.0	38.0	35.0	38.0	...	49.0	36.0	42.0	32.0	3
4	52.0	48.0	37.0	34.0	30.0	34.0	38.0	35.0	47.0	37.0	...	33.0	39.0	38.0	41.0	3
5	44.0	36.0	37.0	36.0	36.0	36.0	44.0	29.0	36.0	36.0	...	32.0	28.0	38.0	39.0	3
6	39.0	42.0	39.0	37.0	38.0	36.0	23.0	37.0	40.0	33.0	...	31.0	44.0	37.0	28.0	3
7	32.0	30.0	40.0	34.0	39.0	34.0	33.0	38.0	45.0	25.0	...	40.0	42.0	35.0	38.0	3
8	10.0	5.0	14.0	8.0	9.0	21.0	28.0	22.0	38.0	36.0	...	24.0	34.0	1.0	20.0	11
9	30.0	38.0	40.0	50.0	40.0	44.0	45.0	28.0	49.0	45.0	...	32.0	34.0	34.0	30.0	3
10	24.0	36.0	37.0	43.0	32.0	42.0	28.0	30.0	19.0	20.0	...	31.0	46.0	38.0	49.0	5

11 rows × 34 columns

HW1 - part 4

In [31]:

```
import sys, random,binascii,AES,jupyter_utils,random
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from IPython.display import Markdown, display,HTML
%matplotlib inline
plt.rcParams['figure.dpi'] = 150
```

This section will focus on the Diffusion principle and examine the extent to which a change in input or internal states affects the end result. The runs in the strain part will be performed on a single input block (16 characters)

In [32]:

```
plaintext = "From the Pope to"
print("Total Length:",len(plaintext))
display(Markdown(f'\n"**_{plaintext}_**"'))
```

```
Total Length: 16
```

"***From the Pope to***"

And we will try to encrypt the message, using an encryption key **"R is a Short key"**

In [33]:

```
key = "R is a Short key"
key = key.encode().hex()
print(f"'R is a Short key':\n 0x{key}")
aes_lib = AES.AES(int(key,base=16))
```

```
'R is a Short key':
 0x52206973206120536686f7274206b6579
```

Question 4:

1. Run the AES algorithm, and save the intermediate results after each round (including the final result)
2. Randomly select a bit in the login message and change it from 0 -> 1 or 1-> 0 accordingly Repeat (1) and calculate how many bits changed their value as a result of the change of the single bit in each round, and the encrypted message (relative to the same situation when no errors were injected)
3. Repeat the experiment 5 times when each time you change one bit in relation to the message in (1) at the end of the runs, present a table

In [34]:

```
def run_aes_stages(plaintext):
    spec_enc_text = [""]*10
    for i in range(0, len(plaintext), 16):
        ciper = aes_lib.encrypt_by_stage(int(plaintext[i:i+16].encode().hex(),base=16))
        for j,c in enumerate(ciper):
            spec_enc_text[j] += ('{:x}'.format(c)).zfill(32)
    return spec_enc_text
spec_enc_text = run_aes_stages(plaintext)
display(spec_enc_text)
```

```
['cb95e6a2a56a602164f8bdc338622f85',
 'cbf42cb3f27bbb6266e1e7abbc87199e',
 '5953b07e2a304dc27be50e6355176310',
 '203d5627e023761e3d86efd4ace38e38',
 'b874b055f6f6fc733e8c7952dbf7ea9c',
 '1dba47880ed16f3c72d966c2526347cc',
 '737bf4f4317604824733b3b13695b2cf',
 'd0d5082d7e23c65c2e2cd9304833f9f6',
 '54d92855bb1e7c8075c0eb813950e85d',
 '793ec12a27911757630e1fd3c4affd06']
```

For stage II and on, we wants' to create a convertion function

In [35]:

```
def tobits(s):
    result = []
    for c in s:
        bits = bin(ord(c))[2:]
        bits = '00000000'[len(bits):] + bits
        result.extend([int(b) for b in bits])
    return result

def frombits(bits):
    chars = []
    for b in range(len(bits) // 8 ):
        byte = bits[b*8:(b+1)*8]
        chars.append(chr(int(''.join([str(bit) for bit in byte]), 2)))
    return ''.join(chars)

def count_flips_to_convert(a, b):

    diff = a ^ b

    # count number of ones in diff
    count = 0
    while diff:
        diff &= (diff - 1)
        count += 1
    return count
```

In [36]:

```
def flip_plaintext(plaintext):
    bitflip_p = tobits(plaintext)
    place = random.choice(range(len(bitflip_p)))
    bitflip_p[place] = int(1 - bitflip_p[place])
    bitflip_plaintext = frombits(bitflip_p)
    return bitflip_plaintext,place
bitflip_plaintext,place = flip_plaintext(plaintext)
print("The bit which was chosen to be replaced: ", place)
print("Result:",bitflip_plaintext)
```

```
The bit which was chosen to be replaced:  84
Result: From the Pgpe to
```

In [37]:

```
bitflip_spec_enc_text = run_aes_stages(bitflip_plaintext)
display(bitflip_spec_enc_text)
```

```
['3689075fa56a602164f8bdc338622f85',
 'ffee369d078ebf932d3c71e030869413',
 '420575864268b0954142c48e54fdd4db',
 'f2e910056962f56de0f60ce978c6e11a',
 'cf8fb4eb8a7dc9e968d83e414287a3a0',
 '2c0f567ce17e68dafdff92ecb51b77f9',
 '2079799e7348b9b23e000eab1c980209',
 '53e34bc4e144691793d9d92bcdc2413a',
 'f0fcd54bbf775050bcde28a772b98bd9',
 'd5b91c53c536b1189b0b284d962abe56']
```

In [38]:

```
def calulate_diffrence(textlist,flip_textlist):
    diff = []
    for i in range(len(flip_textlist)):
        p =  int(textlist[i],base=16)
        c =  int(flip_textlist[i],base=16)
        diff.append(count_flips_to_convert(p,c))
    return diff
diff = calulate_diffrence(bitflip_spec_enc_text,spec_enc_text)
display(diff)
```

```
[21, 61, 71, 60, 65, 69, 59, 68, 58, 65]
```

In [39]:

```
table_result = np.zeros((6,len(bitflip_spec_enc_text)))

for i in range(5):
    bfp,place = flip_plaintext(plaintext)
    bfp_stage = run_aes_stages(bfp)
    table_result[i] = calulate_diffrence(bfp_stage,spec_enc_text)
```

In [40]:

```
table_result[5] = np.mean(table_result[:5], axis = 0)
display(pd.DataFrame(table_result, columns=[f"Round {col+1}" for col in range(len(bitfl
ip_spec_enc_text))]))
```

	Round 1	Round 2	Round 3	Round 4	Round 5	Round 6	Round 7	Round 8	Round 9	Round 10
0	20.0	62.0	67.0	64.0	57.0	68.0	71.0	67.0	53.0	63.0
1	21.0	66.0	56.0	67.0	57.0	73.0	58.0	70.0	64.0	65.0
2	15.0	65.0	48.0	60.0	59.0	63.0	73.0	73.0	56.0	71.0
3	20.0	67.0	62.0	58.0	56.0	65.0	64.0	67.0	64.0	54.0
4	68.0	61.0	64.0	71.0	53.0	67.0	60.0	61.0	69.0	62.0
5	28.8	64.2	59.4	64.0	56.4	67.2	65.2	67.6	61.2	63.0

HW1 - part 5

In [5]:

```
import sys, random,binascii,AES,jupyter_utils,random
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from IPython.display import Markdown, display,HTML
%matplotlib inline
plt.rcParams['figure.dpi'] = 150
```

This section will focus on the Diffusion principle and examine the extent to which a change in input or internal states affects the end result. The runs in the strain part will be performed on a single input block (16 characters)

In [6]:

```
plaintext = "From the Pope to"
print("Total Length:",len(plaintext))
display(Markdown(f'\n"**_{plaintext}_**"'))
```

```
Total Length: 16
```

"***From the Pope to***"

And we will try to encrypt the message, using an encryption key **"R is a Short key"**

In [7]:

```
key = "R is a Short key"
key = key.encode().hex()
print(f"'R is a Short key':\n 0x{key}")
aes_lib = AES.AES(int(key,base=16))
```

```
'R is a Short key':
 0x52206973206120536f7274206b6579
```

Question 5:

Repeat question 4, but this time change only the first bit in the 9th house Make the change 11 times, the first time change the country in plain text, the next 10 times run all the AES algorithms, but the first time change the confidence in the first round, the second time, the second round and so on. In each cell indicate how many bits changed their values, in each layer, together to the normal state

In [8]:

```
spec_enc_text = [""]*11
for i in range(-1,10):
    ciper = aes_lib.encrypt_by_stage(int(plaintext.encode().hex(),base=16), injection_i
tr=i)
    spec_enc_text[i+1] = ciper

spec_enc_diff = [""]*10

for i in range(0,10):
    spec_enc_diff[i] = [bin(spec_enc_text[i+1][j] ^ spec_enc_text[0][j]).count("1") for
j in range(len(spec_enc_text[0]))]

display(pd.DataFrame(spec_enc_diff))

# table shows diff from non injected text
# row = injection round [0,...,10]
# col = intermediate stage [1,..,10]
```

	0	1	2	3	4	5	6	7	8	9
0	18	57	57	61	61	58	67	65	71	59
1	1	11	57	66	69	64	63	73	57	67
2	0	1	19	66	59	68	60	69	66	65
3	0	0	1	23	79	69	66	64	75	65
4	0	0	0	1	16	67	68	66	65	57
5	0	0	0	0	1	17	54	60	64	76
6	0	0	0	0	0	1	18	69	70	62
7	0	0	0	0	0	0	1	22	61	66
8	0	0	0	0	0	0	0	1	10	18
9	0	0	0	0	0	0	0	0	1	4

HW1 - part 6

In [10]:

```
import sys, random,binascii,AES,jupyter_utils,random
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from IPython.display import Markdown, display,HTML
%matplotlib inline
plt.rcParams['figure.dpi'] = 150
```

This section will focus on the Diffusion principle and examine the extent to which a change in input or internal states affects the end result. The runs in the strain part will be performed on a single input block (16 characters)

In [11]:

```
plaintext = "From the Pope to"
print("Total Length:",len(plaintext))
display(Markdown(f'\n"**_{plaintext}_**"'))
```

```
Total Length: 16
```

"***From the Pope to***"

In [12]:

```
def tobits(s):
    result = []
    for c in s:
        bits = bin(ord(c))[2:]
        bits = '00000000'[len(bits):] + bits
        result.extend([int(b) for b in bits])
    return result

def frombits(bits):
    chars = []
    for b in range(len(bits) // 8 ):
        byte = bits[b*8:(b+1)*8]
        chars.append(chr(int(''.join([str(bit) for bit in byte]), 2)))
    return ''.join(chars)

def count_flips_to_convert(a, b):

    diff = a ^ b

    # count number of ones in diff
    count = 0
    while diff:
        diff &= (diff - 1)
        count += 1
    return count

def flip_plaintext(plaintext):
    bitflip_p = tobits(plaintext)
    place = random.choice(range(len(bitflip_p)))
    bitflip_p[place] = int(1 - bitflip_p[place])
    bitflip_plaintext = frombits(bitflip_p)
    return bitflip_plaintext,place
bitflip_plaintext,place = flip_plaintext(plaintext)
print("The bit which was chosen to be replaced: ", place)
print("Result:",bitflip_plaintext)
```

```
The bit which was chosen to be replaced:  110
Result: From the Pope"to
```

And we will try to encrypt the message, using an encryption key **"R is a Short key"**

In [13]:

```
key = "R is a Short key"
print(f"'R is a Short key':\n 0x{key.encode().hex()}")

keys = 7*[key]
place = 7*[""]
aes_lib = 7*[""]

aes_lib[0] = AES.AES(int(keys[0].encode().hex(),base=16))
print("0st key wasn't flipped")
for i in range(6):
    keys[i+1],place[i+1] = flip_plaintext(key)
    print(str(i+1) + "st key was flipped at place " + str(place[i+1]))
    aes_lib[i+1] = AES.AES(int(keys[i+1].encode().hex(),base=16))
```

```
'R is a Short key':
 0x52206973206120536f7274206b6579
0st key wasn't flipped
1st key was flipped at place 76
2st key was flipped at place 58
3st key was flipped at place 31
4st key was flipped at place 55
5st key was flipped at place 34
6st key was flipped at place 101
```

Question 6:

1. Run the AES algorithm, and save the intermediate results after each round (including the final result)
2. Randomly select a bit in the key and change it from 0 -> 1 or 1-> 0 accordingly Repeat (1) and calculate how many bits changed their value as a result of the change of the single bit in each round, and the encrypted message (relative to the same situation when no errors were injected)
3. Repeat the experiment 5 or 6 times when each time you change one bit in relation to the message in (1) at the end of the runs, present a table

In [14]:

```
def run_aes_stages(plaintext, key_num):
    spec_enc_text = [""]*10
    for i in range(0, len(plaintext), 16):
        ciper = aes_lib[key_num].encrypt_by_stage(int(plaintext[i:i+16].encode().hex(),
base=16))
        for j,c in enumerate(ciper):
            spec_enc_text[j] += ('{:x}'.format(c)).zfill(32)
    return spec_enc_text
spec_enc_text = run_aes_stages(plaintext, 0)
display(spec_enc_text)
```

```
['cb95e6a2a56a602164f8bdc338622f85',
 'cbf42cb3f27bbb6266e1e7abbc87199e',
 '5953b07e2a304dc27be50e6355176310',
 '203d5627e023761e3d86efd4ace38e38',
 'b874b055f6f6fc733e8c7952dbf7ea9c',
 '1dba47880ed16f3c72d966c2526347cc',
 '737bf4f4317604824733b3b13695b2cf',
 'd0d5082d7e23c65c2e2cd9304833f9f6',
 '54d92855bb1e7c8075c0eb813950e85d',
 '793ec12a27911757630e1fd3c4affd06']
```

In [15]:

```
print("Displaying output with another key")
bitflip_spec_enc_text = run_aes_stages(plaintext, 1)
display(bitflip_spec_enc_text)
```

```
Displaying output with another key
```

```
['cb95e6a28faf8fce64f0bdc3386a2f85',
 '150257c8078f0330a0d4d5ef4c79443d',
 'cb3fe5e533c8a0b4e9b7c10da2cd0230',
 'cb8a3334f040f4dc3bc1ab619e270baa',
 'fe8119a88e15aeb86cf62b573f96ceaf',
 '1b41fa229eec4587291a80e4202ff4ff',
 'b3260284807cf48689f402d1504431f0',
 '63a613f7f63a9f15a3c8d6ba71f3a960',
 'f12adadf880b0bb65bcefaeb36d8eb64',
 'fe8480730fa0e0358e4ab2d29a7f08e3']
```

In [16]:

```
def calulate_diffrence(textlist,flip_textlist):
    diff = []
    for i in range(len(flip_textlist)):
        p =  int(textlist[i],base=16)
        c =  int(flip_textlist[i],base=16)
        diff.append(count_flips_to_convert(p,c))
    return diff
diff = calulate_diffrence(bitflip_spec_enc_text,spec_enc_text)
display(diff)
```

```
[23, 75, 68, 54, 63, 68, 60, 58, 60, 63]
```

In [17]:

```
table_result = np.zeros((7,len(bitflip_spec_enc_text)))

for i in range(6):
    bfp_stage = run_aes_stages(plaintext, key_num=1+i)
    table_result[i] = calulate_diffrence(bfp_stage,spec_enc_text)
```

In [18]:

```
table_result[6] = np.mean(table_result[:6], axis = 0)
display(pd.DataFrame(table_result, columns=[f"Round {col+1}" for col in range(len(bitfl
ip_spec_enc_text))]))
```

	Round 1	Round 2	Round 3	Round 4	Round 5	Round 6	Round 7	Round 8	Round 9
0	23.000000	75.000000	68.000000	54.000000	63.0	68.0	60.000000	58.0	60.000000
1	13.000000	68.000000	58.000000	58.000000	65.0	54.0	63.000000	61.0	69.000000
2	22.000000	62.000000	62.000000	67.000000	66.0	63.0	69.000000	61.0	61.000000
3	21.000000	66.000000	65.000000	63.000000	69.0	67.0	66.000000	60.0	65.000000
4	19.000000	63.000000	58.000000	71.000000	68.0	60.0	57.000000	64.0	68.000000
5	27.000000	60.000000	54.000000	67.000000	56.0	69.0	50.000000	59.0	66.000000
6	20.833333	65.666667	60.833333	63.333333	64.5	63.5	60.833333	60.5	64.833333

Summary and Conclusions

In the first three questions, we have seen that all the histograms look entirely different from the original since the first iteration. Thus, we can say that these findings support the property of confusion. In the last three questions, we can see that the flipping of one bit in the plaintext or the key changes the result dramatically. However, depending on the flipping location and the iteration number, we have seen that there is a large variance in the number of flipped bits in the following text state.