In [39]:
```
import numpy
import seaborn
from scipy import stats
import csv
import pandas

# Hamming weight calculator
def hw(int_no):
    count = 0
    while(int_no):
        int_no &= (int_no-1)
        count += 1
    return count

Sbox = (
    0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5, 0x30, 0x01, 0x67, 0x2B, 0xFE, 0xD7,
    0xCA, 0x82, 0xC9, 0x7D, 0xFA, 0x59, 0x47, 0xF0, 0xAD, 0xD4, 0xA2, 0xAF, 0x9C, 0xA4,
    0xB7, 0xFD, 0x93, 0x26, 0x36, 0x3F, 0xF7, 0xCC, 0x34, 0xA5, 0xE5, 0xF1, 0x71, 0xD8,
    0x04, 0xC7, 0x23, 0xC3, 0x18, 0x96, 0x05, 0x9A, 0x07, 0x12, 0x80, 0xE2, 0xEB, 0x27,
    0x09, 0x83, 0x2C, 0x1A, 0x1B, 0x6E, 0x5A, 0xA0, 0x52, 0x3B, 0xD6, 0xB3, 0x29, 0xE3,
    0x53, 0xD1, 0x00, 0xED, 0x20, 0xFC, 0xB1, 0x5B, 0x6A, 0xCB, 0xBE, 0x39, 0x4A, 0x4C,
    0xD0, 0xEF, 0xAA, 0xFB, 0x43, 0x4D, 0x33, 0x85, 0x45, 0xF9, 0x02, 0x7F, 0x50, 0x3C,
    0x51, 0xA3, 0x40, 0x8F, 0x92, 0x9D, 0x38, 0xF5, 0xBC, 0xB6, 0xDA, 0x21, 0x10, 0xFF,
    0xCD, 0x0C, 0x13, 0xEC, 0x5F, 0x97, 0x44, 0x17, 0xC4, 0xA7, 0x7E, 0x3D, 0x64, 0x5D,
    0x60, 0x81, 0x4F, 0xDC, 0x22, 0x2A, 0x90, 0x88, 0x46, 0xEE, 0xB8, 0x14, 0xDE, 0x5E,
    0xE0, 0x32, 0x3A, 0x0A, 0x49, 0x06, 0x24, 0x5C, 0xC2, 0xD3, 0xAC, 0x62, 0x91, 0x95,
    0xE7, 0xC8, 0x37, 0x6D, 0x8D, 0xD5, 0x4E, 0xA9, 0x6C, 0x56, 0xF4, 0xEA, 0x65, 0x7A,
    0xBA, 0x78, 0x25, 0x2E, 0x1C, 0xA6, 0xB4, 0xC6, 0xE8, 0xDD, 0x74, 0x1F, 0x4B, 0xBD,
    0x70, 0x3E, 0xB5, 0x66, 0x48, 0x03, 0xF6, 0x0E, 0x61, 0x35, 0x57, 0xB9, 0x86, 0xC1,
    0xE1, 0xF8, 0x98, 0x11, 0x69, 0xD9, 0x8E, 0x94, 0x9B, 0x1E, 0x87, 0xE9, 0xCE, 0x55,
    0x8C, 0xA1, 0x89, 0x0D, 0xBF, 0xE6, 0x42, 0x68, 0x41, 0x99, 0x2D, 0x0F, 0xB0, 0x54,
)
```

In [40]:
```
df = pandas.read_csv('anotherownwaveform.csv')
```

In [41]:
```
# arguments: key byte index (plaintext byte index) to build a power model for
# for this index, for each of the 100 plaintexts, xor each corresponding plaintext byte
# return a 256x100 power model M for that plaintext byte

# byteindex ranges from 0 to 15
def powermodelgenerator(byteindex):
    byteindex *= 2
    singleresultrow = []
    resultpowermodel = []

    # key ranges from 0 to 255
    for key in range(0x00,0x100):
        for i in range(0,100):
            tmp = "0x" + df.iloc[i,0][byteindex:byteindex+2]
            tmpagain = int(tmp,16) ^ key
            tmpfinal = Sbox[tmpagain]
            singleresultrow.append(hw(tmpfinal))
        resultpowermodel.append(singleresultrow)
        singleresultrow = []

    return resultpowermodel
```

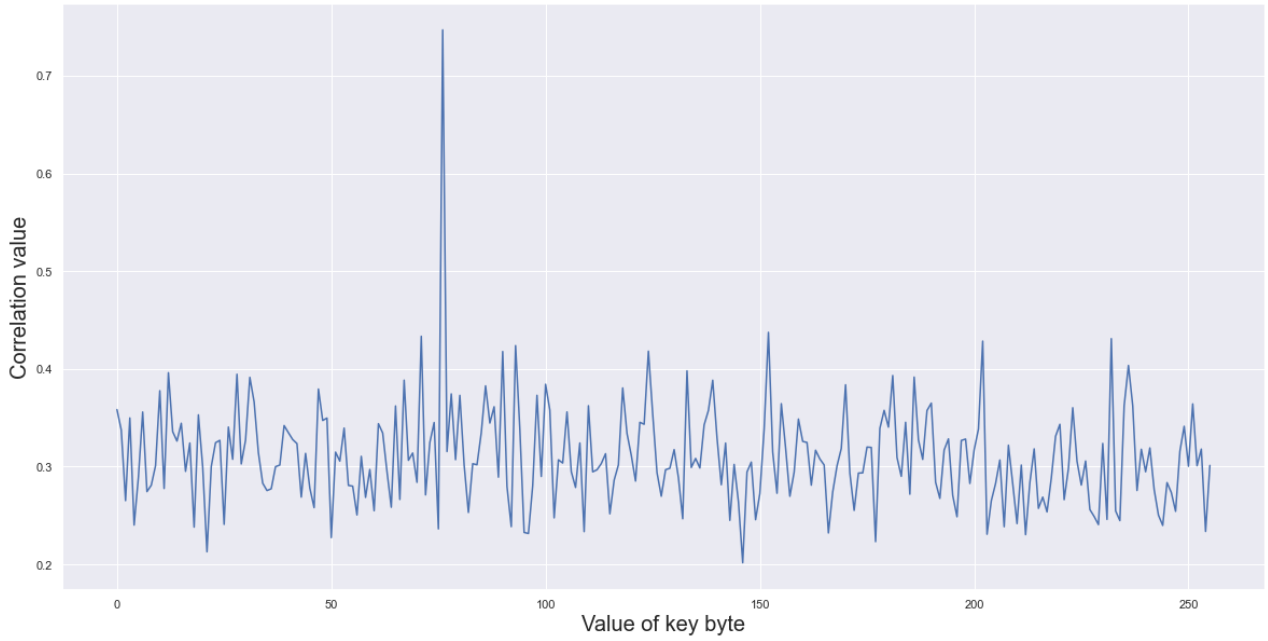
In [42]:
```
# m is a 256 times 100 power model: m[0 to 255][0 to 99]
# time to correlate each m[0] to m[255] with each df.iloc[:,2] to df.iloc[:,2501]
# power traces captured in lab ranges from df.iloc[:,2] to df.iloc[:,2501]

# arguments: powermodel built earlier
def correlationmatrixgenerator(powermodel):
    singleresultrow = []
    correlationmatrix = []

    for i in range(0,256):
        for j in range(2,2502):
            singleresultrow.append(abs(stats.pearsonr(powermodel[i],df.iloc[:,j])[0]))
        correlationmatrix.append(singleresultrow)
        singleresultrow = []

    return correlationmatrix
```

In [43]:
```
# plots the required graph using an obtained correlationmatrix
def plotgraph(correlationmatrix, byteindex):

    # for finding the highest correlation value in each row of correlationmatrix, so 25
    highestvaluearray = []
    for row in correlationmatrix:
        highestvaluearray.append(max(row))

    # for plotting a graph for a key byte: x-axis: value of key byte, y-axis: correlati
    print("Graph for byte number: " + str(byteindex))

    seaborn.set(rc={'figure.figsize':(20,10)})
    plot = seaborn.lineplot(data=highestvaluearray)
    plot.set_xlabel("Value of key byte", fontsize = 20)
    plot.set_ylabel("Correlation value", fontsize = 20)

    print("Highest correlation value: " + str(max(highestvaluearray)))
    for i in range(0,256):
        if highestvaluearray[i]==max(highestvaluearray):
            print("Recovered key byte: " + str(hex(i)))

            # return the recovered key byte
            return hex(i)
```

In [44]:
```
# for generating the graph for the first key byte
m = powermodelgenerator(0)

# These are debugging statements for development use.
# print(bin(Sbox[0xFD ^ 0x56]))
# print(m[253][99])
# print(Sbox[0xFE ^ 0x56])
# print(m[254][99])
# print(bin(Sbox[0xFF ^ 0x56]))
# print(m[255][99])
# print(Sbox[0xAA ^ 0xF8])
# print(m[170][5])

# This will collect the recovered key bytes:
recoveredkeybytes = []
```

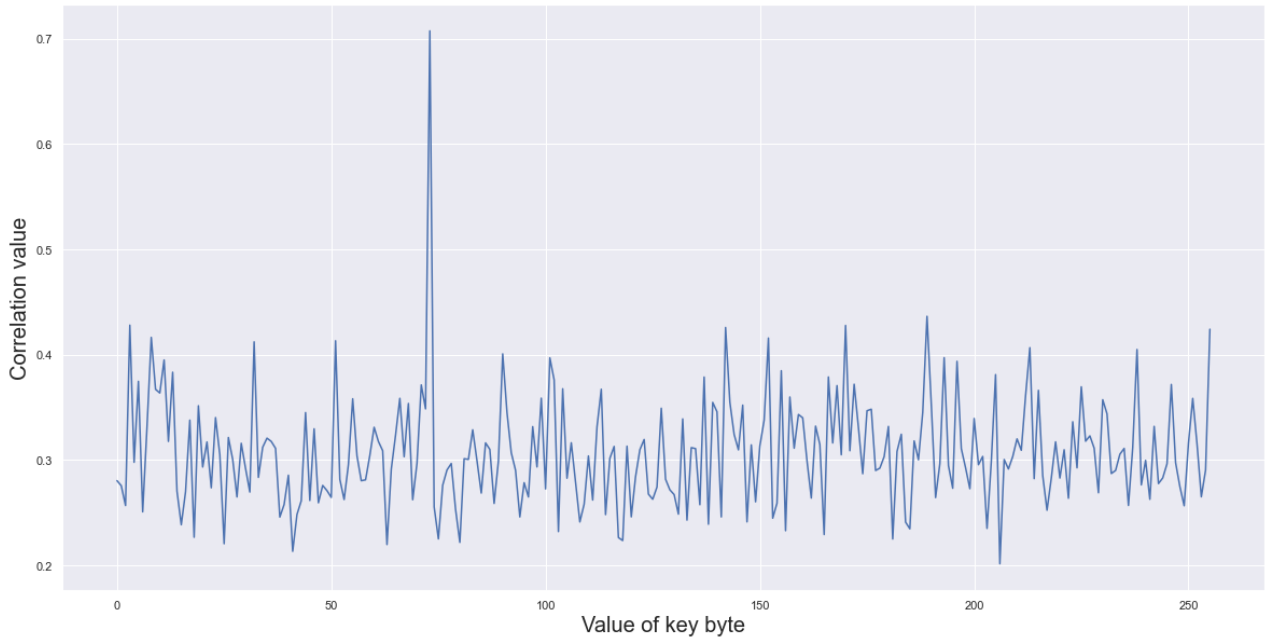
```
correlationmatrix = correlationmatrixgenerator(m)
recoveredkeybytes.append(plotgraph(correlationmatrix,0))
```

```
Graph for byte number: 0
Highest correlation value: 0.7469190510872585
Recovered key byte: 0x4c
```



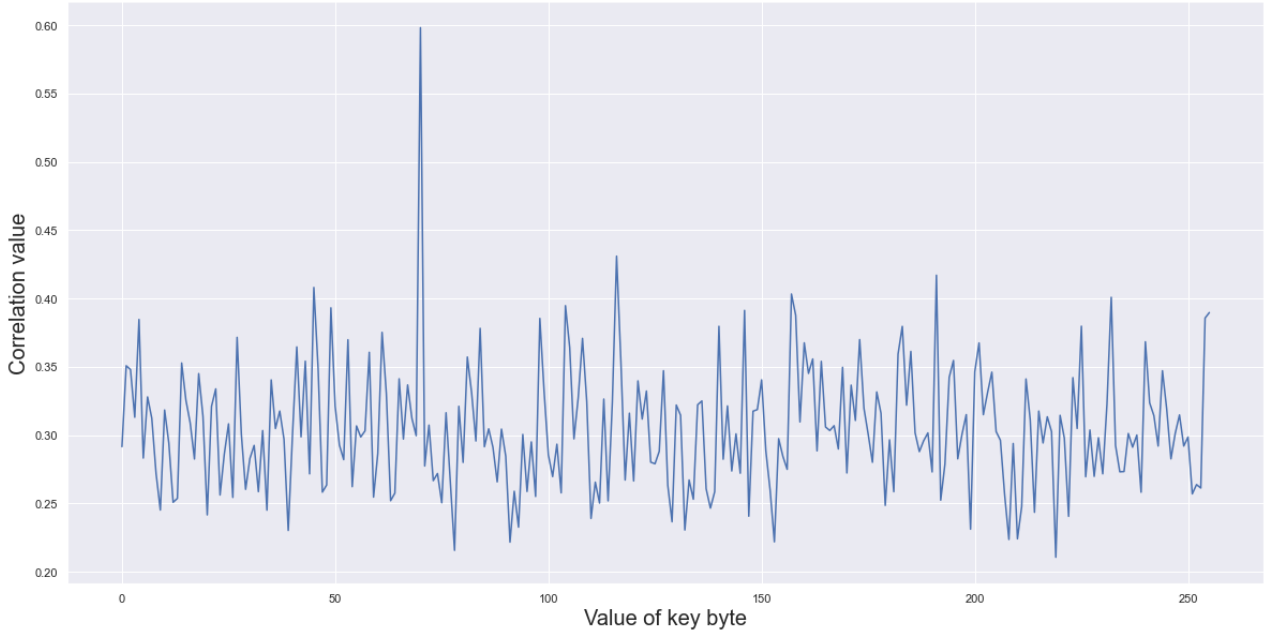
In [45]:
```
# Time to generate the rest of the graphs and find out our recovered key:
m = powermodelgenerator(1)
correlationmatrix = correlationmatrixgenerator(m)
recoveredkeybytes.append(plotgraph(correlationmatrix,1))
```

```
Graph for byte number: 1
Highest correlation value: 0.7074481672016952
Recovered key byte: 0x49
```



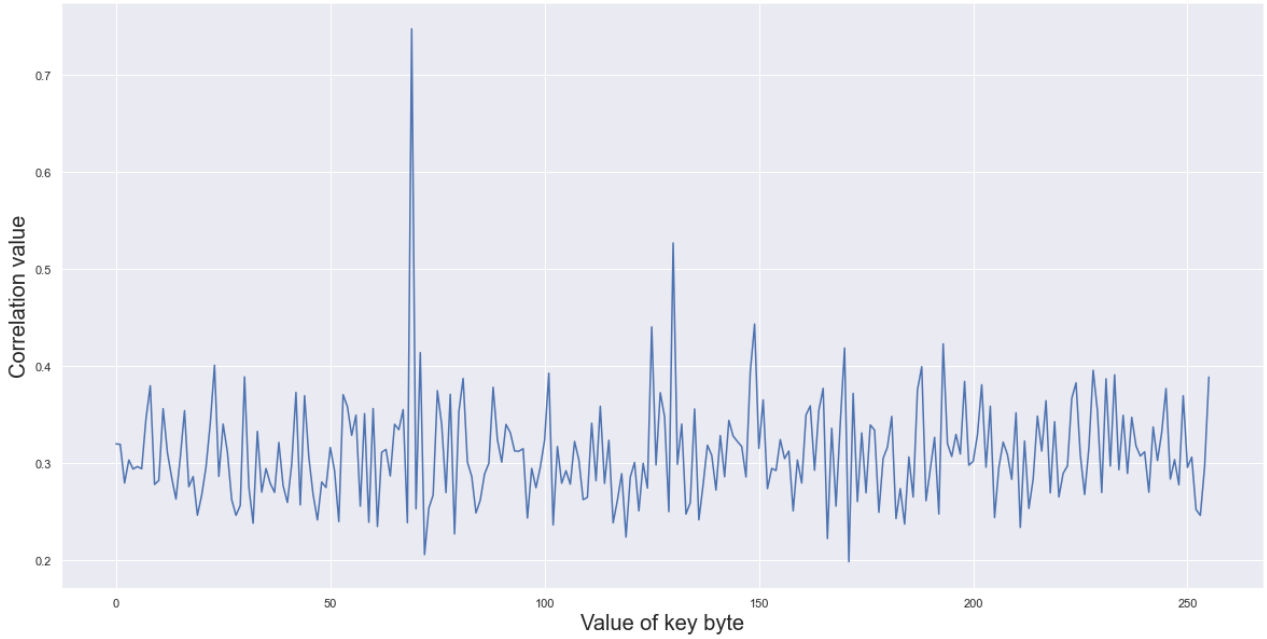
In [46]:
```
# Time to generate the rest of the graphs and find out our recovered key:
m = powermodelgenerator(2)
correlationmatrix = correlationmatrixgenerator(m)
recoveredkeybytes.append(plotgraph(correlationmatrix,2))
```

```
Graph for byte number: 2
Highest correlation value: 0.5981468222883599
Recovered key byte: 0x46
```



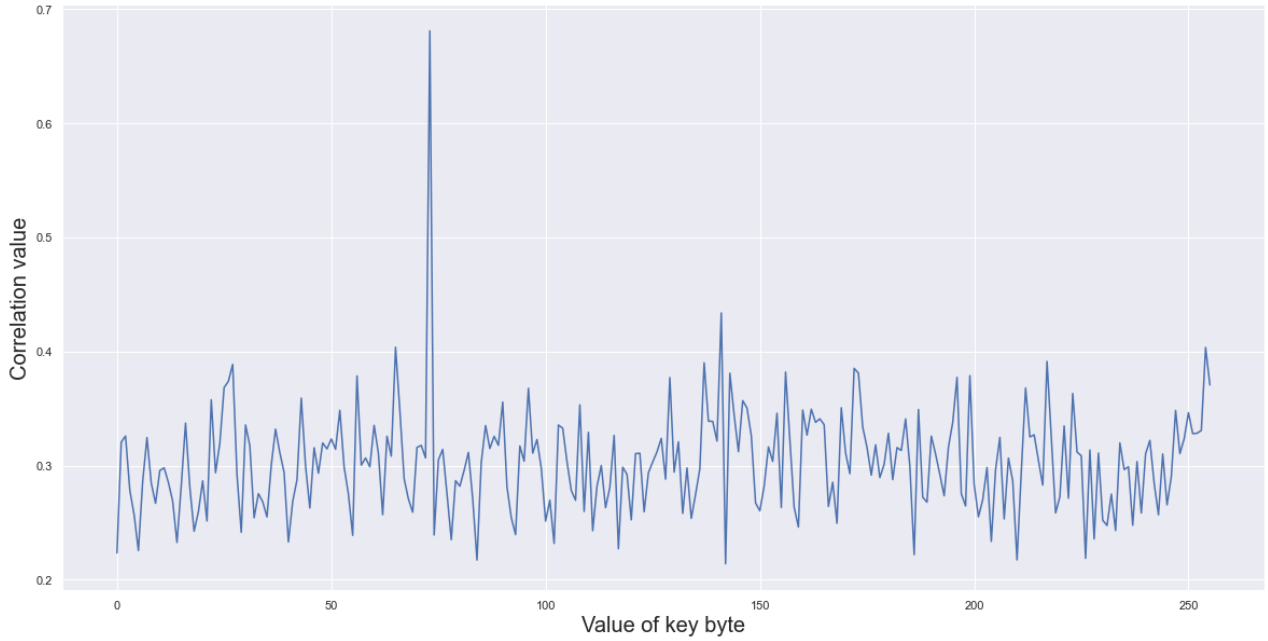
In [47]:
```
# Time to generate the rest of the graphs and find out our recovered key:
m = powermodelgenerator(3)
correlationmatrix = correlationmatrixgenerator(m)
recoveredkeybytes.append(plotgraph(correlationmatrix,3))
```

```
Graph for byte number: 3
Highest correlation value: 0.7474320782405013
Recovered key byte: 0x45
```



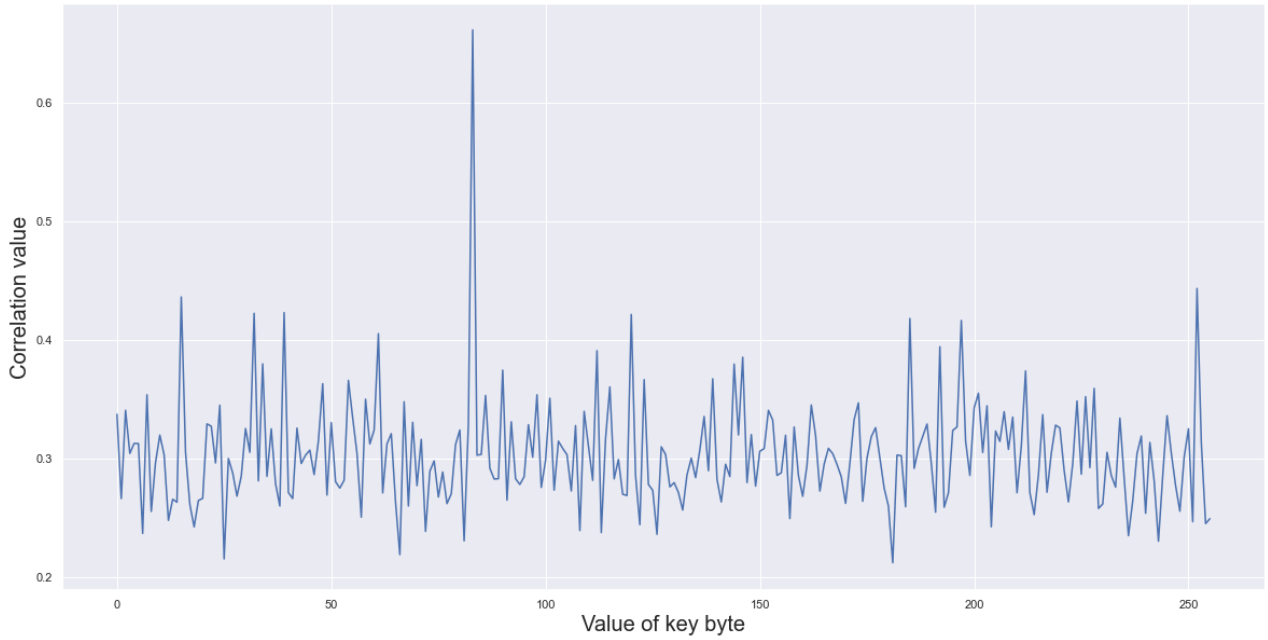
In [48]:
```
# Time to generate the rest of the graphs and find out our recovered key:
m = powermodelgenerator(4)
correlationmatrix = correlationmatrixgenerator(m)
recoveredkeybytes.append(plotgraph(correlationmatrix,4))
```

```
Graph for byte number: 4
Highest correlation value: 0.6810405567669411
Recovered key byte: 0x49
```



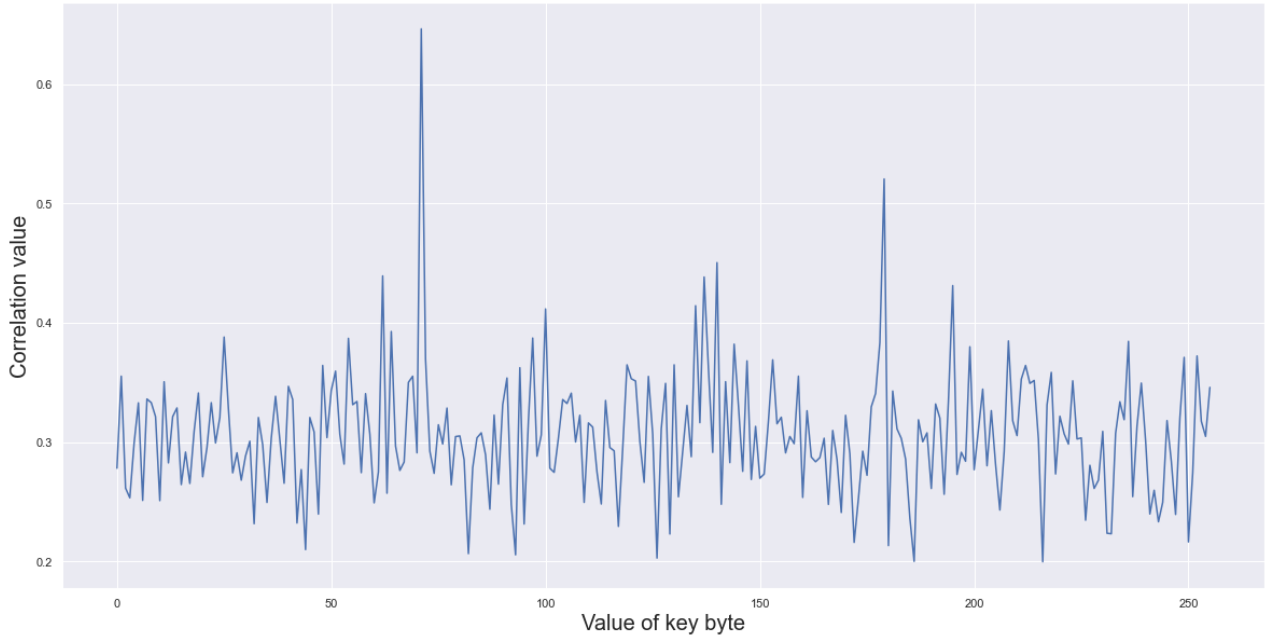
In [49]:
```
# Time to generate the rest of the graphs and find out our recovered key:
m = powermodelgenerator(5)
correlationmatrix = correlationmatrixgenerator(m)
recoveredkeybytes.append(plotgraph(correlationmatrix,5))
```

```
Graph for byte number: 5
Highest correlation value: 0.6610132529375801
Recovered key byte: 0x53
```



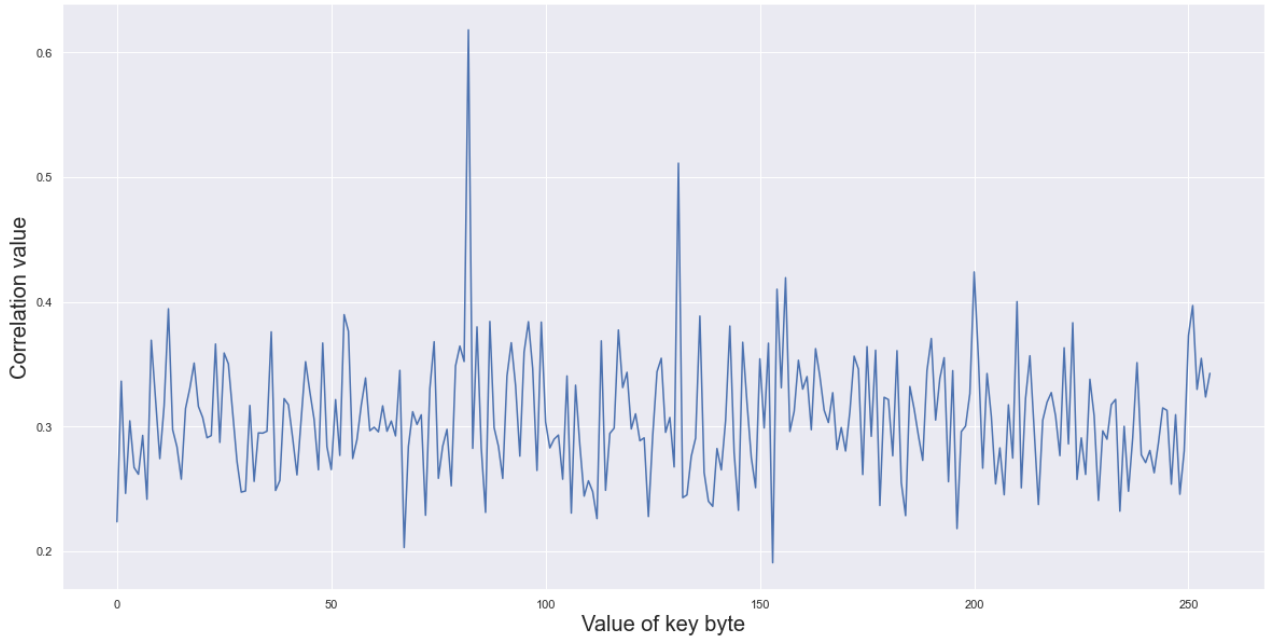
In [50]:
```
# Time to generate the rest of the graphs and find out our recovered key:
m = powermodelgenerator(6)
correlationmatrix = correlationmatrixgenerator(m)
recoveredkeybytes.append(plotgraph(correlationmatrix,6))
```

```
Graph for byte number: 6
Highest correlation value: 0.6464015183106125
Recovered key byte: 0x47
```



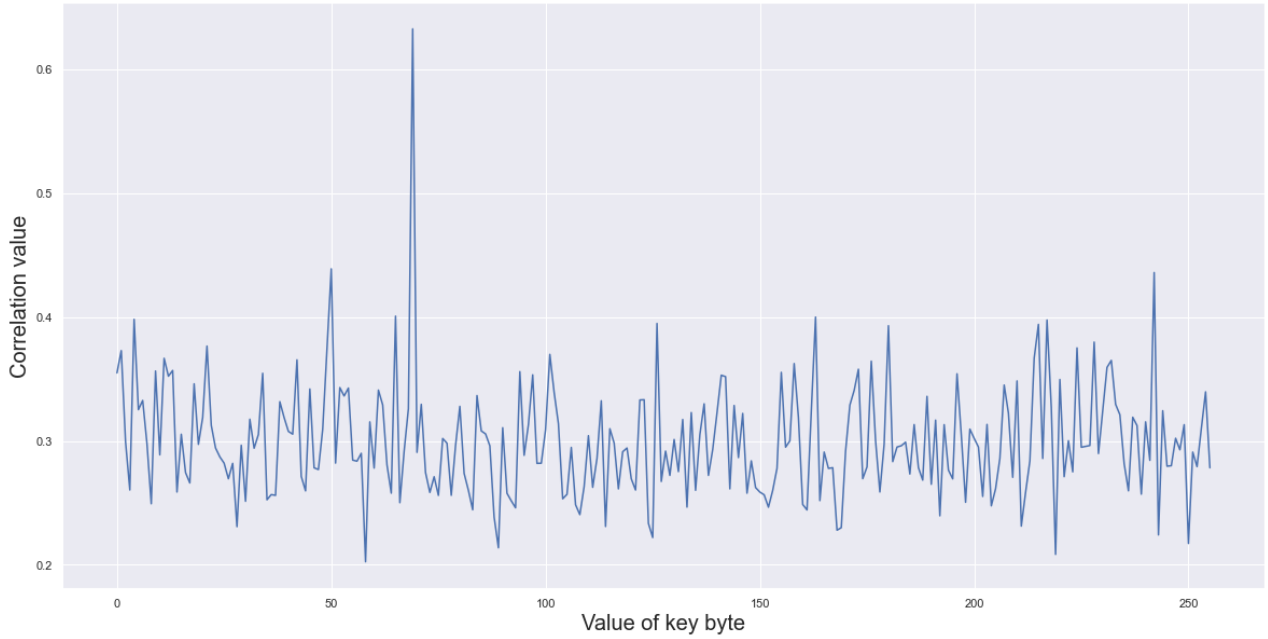
In [51]:
```
# Time to generate the rest of the graphs and find out our recovered key:
m = powermodelgenerator(7)
correlationmatrix = correlationmatrixgenerator(m)
recoveredkeybytes.append(plotgraph(correlationmatrix,7))
```

```
Graph for byte number: 7
Highest correlation value: 0.6180139097336745
Recovered key byte: 0x52
```



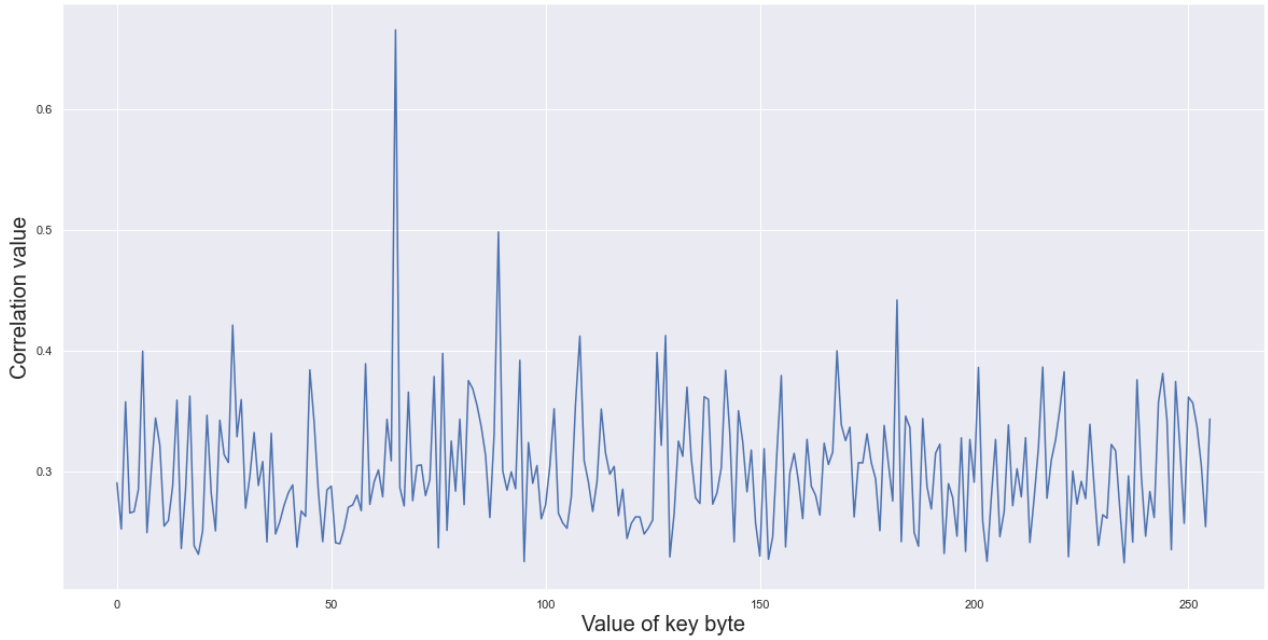
In [52]:
```
# Time to generate the rest of the graphs and find out our recovered key:
m = powermodelgenerator(8)
correlationmatrix = correlationmatrixgenerator(m)
recoveredkeybytes.append(plotgraph(correlationmatrix,8))
```

```
Graph for byte number: 8
Highest correlation value: 0.6326019650687962
Recovered key byte: 0x45
```



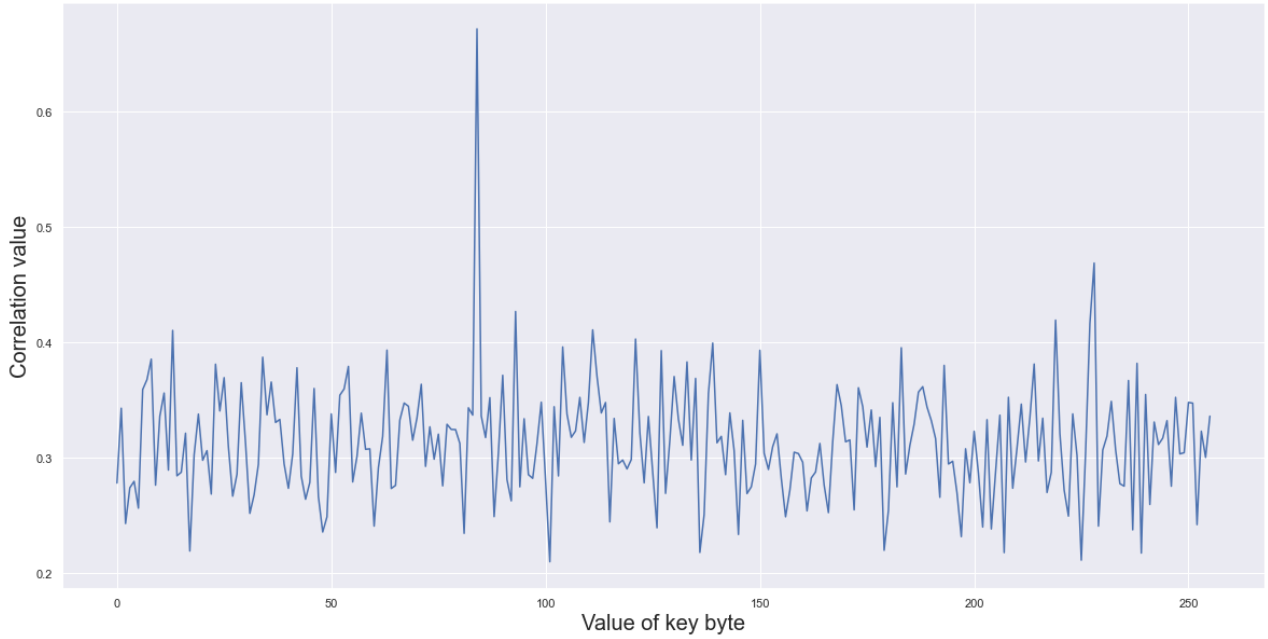
In [53]:
```
# Time to generate the rest of the graphs and find out our recovered key:
m = powermodelgenerator(9)
correlationmatrix = correlationmatrixgenerator(m)
recoveredkeybytes.append(plotgraph(correlationmatrix,9))
```

```
Graph for byte number: 9
Highest correlation value: 0.6655987689790792
Recovered key byte: 0x41
```



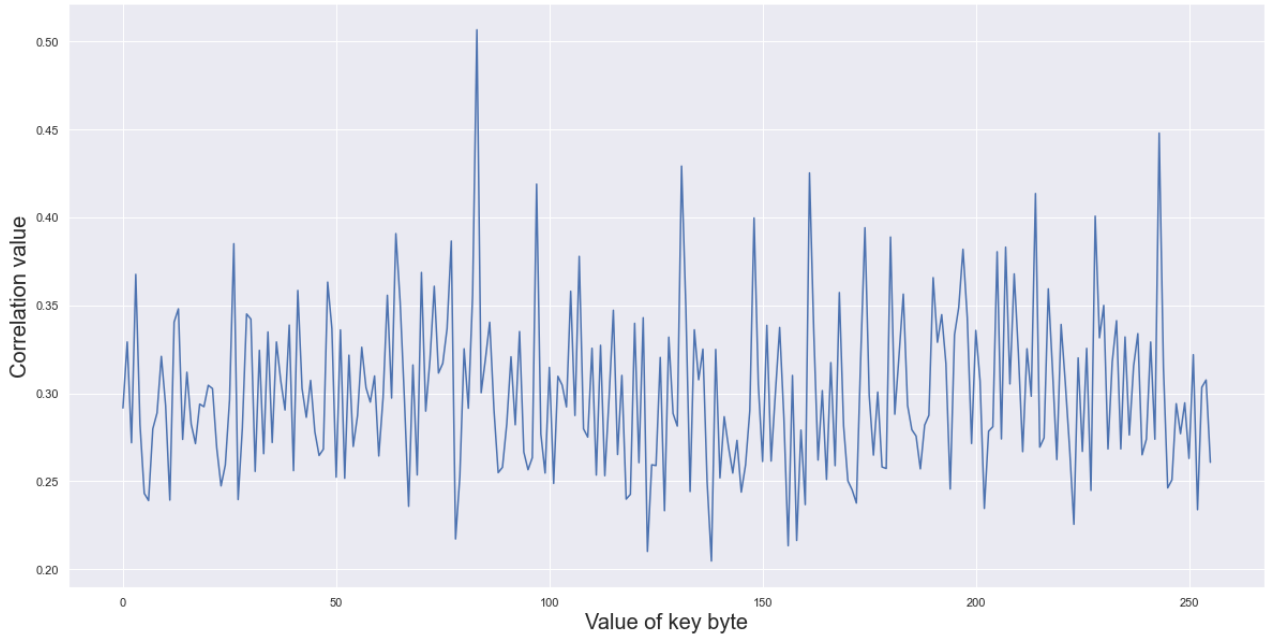
In [54]:
```
# Time to generate the rest of the graphs and find out our recovered key:
m = powermodelgenerator(10)
correlationmatrix = correlationmatrixgenerator(m)
recoveredkeybytes.append(plotgraph(correlationmatrix,10))
```

```
Graph for byte number: 10
Highest correlation value: 0.6716498484069444
Recovered key byte: 0x54
```



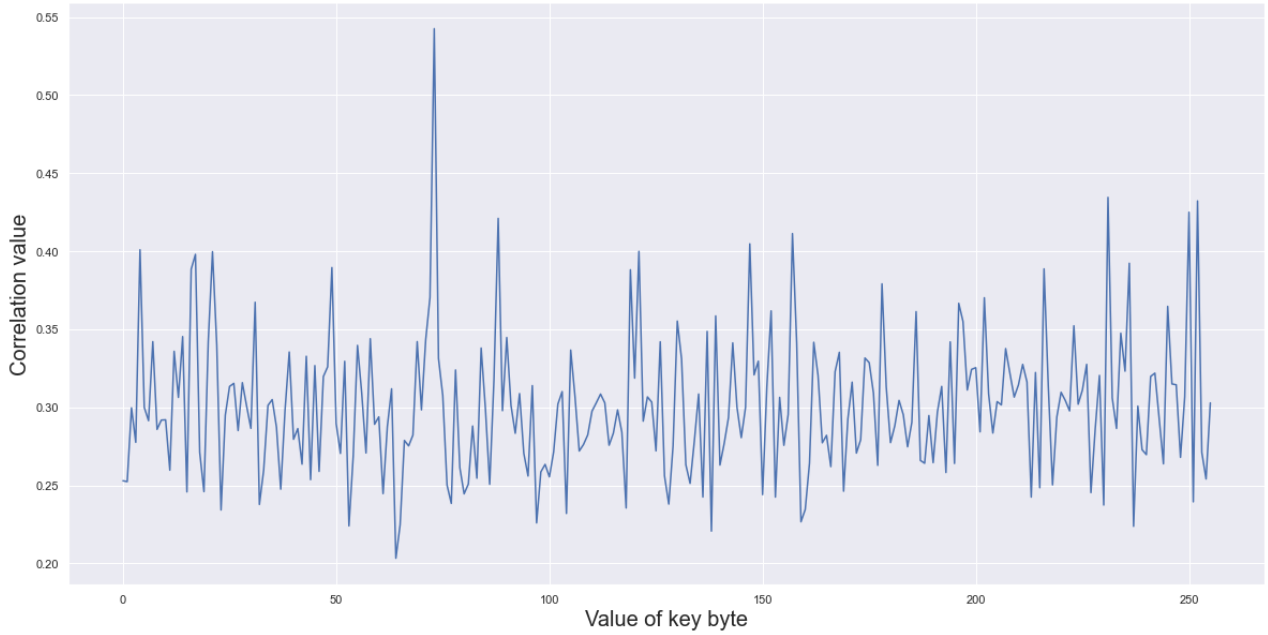
In [55]:
```
# Time to generate the rest of the graphs and find out our recovered key:
m = powermodelgenerator(11)
correlationmatrix = correlationmatrixgenerator(m)
recoveredkeybytes.append(plotgraph(correlationmatrix,11))
```

```
Graph for byte number: 11
Highest correlation value: 0.5067094344412493
Recovered key byte: 0x53
```



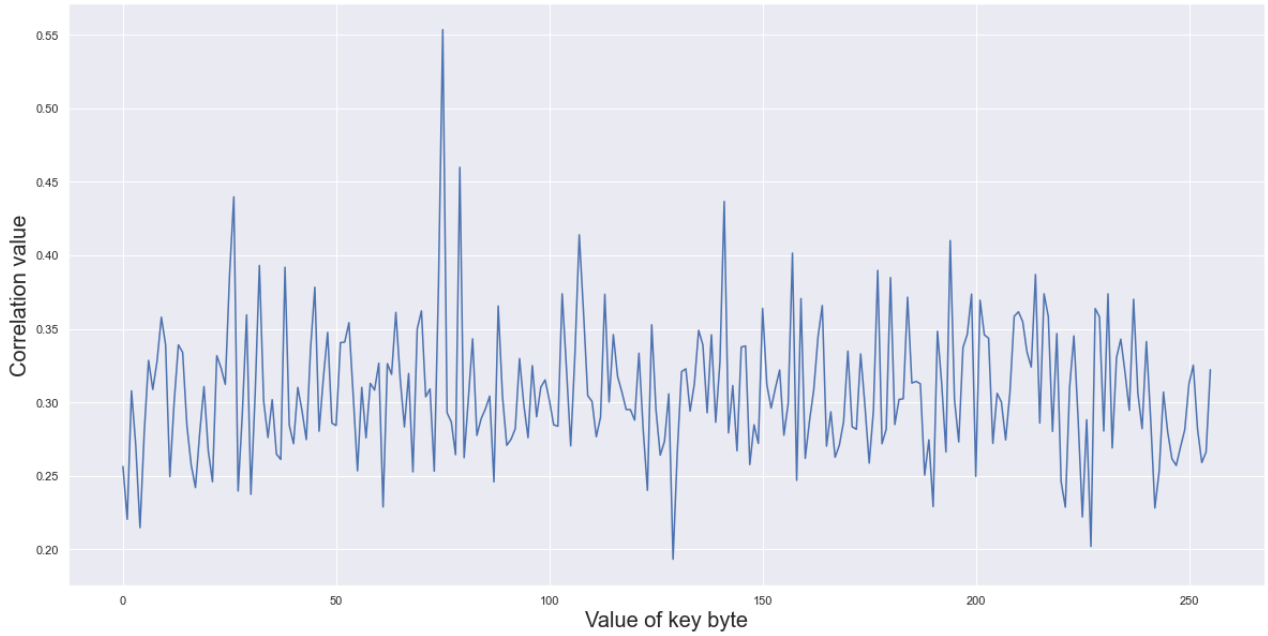
In [56]:
```
# Time to generate the rest of the graphs and find out our recovered key:
m = powermodelgenerator(12)
correlationmatrix = correlationmatrixgenerator(m)
recoveredkeybytes.append(plotgraph(correlationmatrix,12))
```

```
Graph for byte number: 12
Highest correlation value: 0.5425991847253301
Recovered key byte: 0x49
```



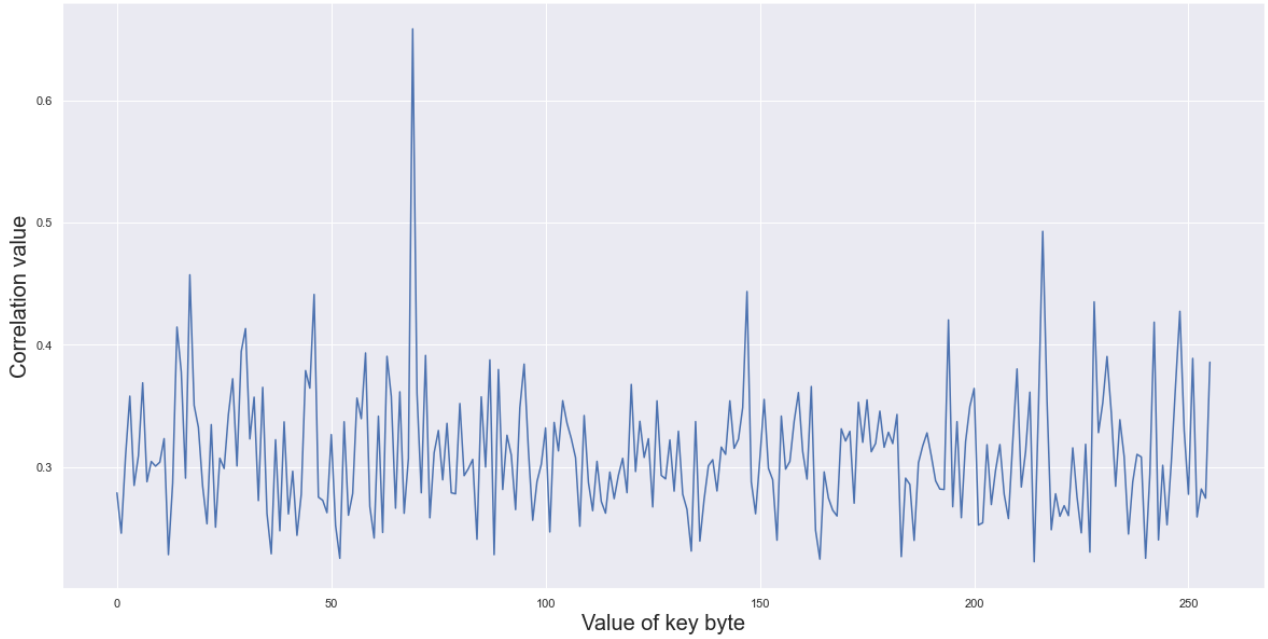
In [57]:
```
# Time to generate the rest of the graphs and find out our recovered key:
m = powermodelgenerator(13)
correlationmatrix = correlationmatrixgenerator(m)
recoveredkeybytes.append(plotgraph(correlationmatrix,13))
```

```
Graph for byte number: 13
Highest correlation value: 0.5533618078257962
Recovered key byte: 0x4b
```



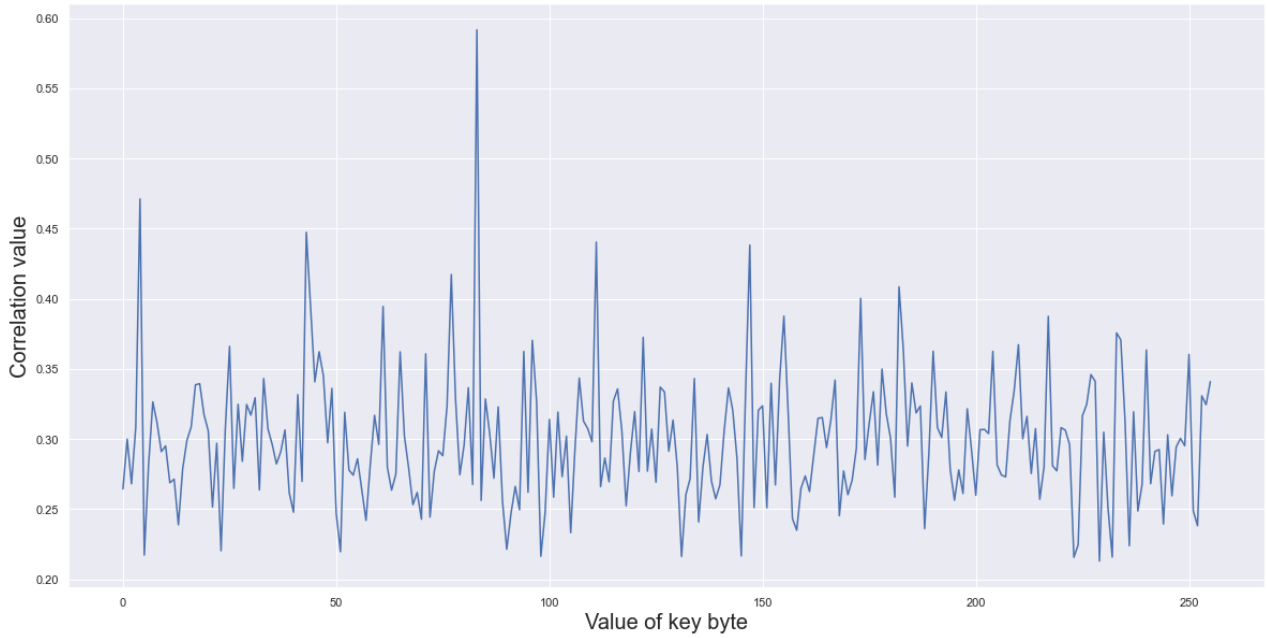
In [58]:
```
# Time to generate the rest of the graphs and find out our recovered key:
m = powermodelgenerator(14)
correlationmatrix = correlationmatrixgenerator(m)
recoveredkeybytes.append(plotgraph(correlationmatrix,14))
```

```
Graph for byte number: 14
Highest correlation value: 0.6582853672652147
Recovered key byte: 0x45
```



In [59]:
```
# Time to generate the rest of the graphs and find out our recovered key:
m = powermodelgenerator(15)
correlationmatrix = correlationmatrixgenerator(m)
recoveredkeybytes.append(plotgraph(correlationmatrix,15))
```

```
Graph for byte number: 15
Highest correlation value: 0.5916939618914787
Recovered key byte: 0x53
```



In [60]:
```
# Let's see what key we recovered:
key = ""
for byte in recoveredkeybytes:
    key += str(byte[2:])
print(key)
```

```
4c4946454953475245415453494b4553
```

In [61]:
```
##############################################################################################
```

In [114…

```
# second power model generator for Plot-2
# arguments: key byte index (plaintext byte index) to build a power model for, and numb
# return a 256 x n power model M for that byte index

# byteindex ranges from 0 to 15
def secondpowermodelgenerator(byteindex, numberoftraces):
    byteindex *= 2
    singleresultrow = []
    resultpowermodel = []

    # key ranges from 0 to 255
    for key in range(0x00,0x100):
        for i in range(0,numberoftraces):
            tmp = "0x" + df.iloc[i,0][byteindex:byteindex+2]
            tmpagain = int(tmp,16) ^ key
            tmpfinal = Sbox[tmpagain]
            singleresultrow.append(hw(tmpfinal))
        resultpowermodel.append(singleresultrow)
        singleresultrow = []

    return resultpowermodel
```

In [115…

```
# second correlation matrix generator for Plot-2
# m is a 256 times n power model: m[0 to 255][0 to n-1]
# time to correlate each m[0] to m[n-1] with each df.iloc[:n,2] to df.iloc[:n,2501]
# power traces captured in lab ranges from df.iloc[:,2] to df.iloc[:,2501]

# arguments: powermodel we built earlier containing 256 sets of n traces, number of tra
def secondcorrelationmatrixgenerator(powermodel, numberoftraces):
    singleresultrow = []
    correlationmatrix = []

    for i in range(0,256):
        for j in range(2,2502):
            # note that powermodel is already of size 256 times n
            singleresultrow.append(abs(stats.pearsonr(powermodel[i],df.iloc[:numberoftr
        correlationmatrix.append(singleresultrow)
        singleresultrow = []

    # returns a correlation matrix of size 256 times 2500 which was built on the number
    return correlationmatrix
```

In [116…

```
# the input to the graph set for a key byte needs to be
# a 256 times 10 matrix: 256 lines for 256 key values, and 10 columns for the number of
# this function is for producing this input to the graph set properly

def secondgraphinputproducer(byteindex):

    # Our graph input should have 256 lines for each key, and each line has 10 data poi
    # initialize all entries to 0
    graphinput = [[0 for x in range(10)] for y in range(256)]

    # I want to increase my number of traces in steps of 10 from 10 to 100.
    tracearray = [10,20,30,40,50,60,70,80,90,100]
    for numberoftraces in tracearray:
        corrmat = secondcorrelationmatrixgenerator(secondpowermodelgenerator(byteindex,

        # time to find the highest correlation value in each row of corrmat, so 256 hig
```

```
        highestvaluearray = []
        for row in corrmat:
            highestvaluearray.append(max(row))

        # now we have the 256 data points for a certain number of traces, put them into
        for i in range(0,256):
            graphinput[i][(numberoftraces//10)-1] = highestvaluearray[i]

    return graphinput
```

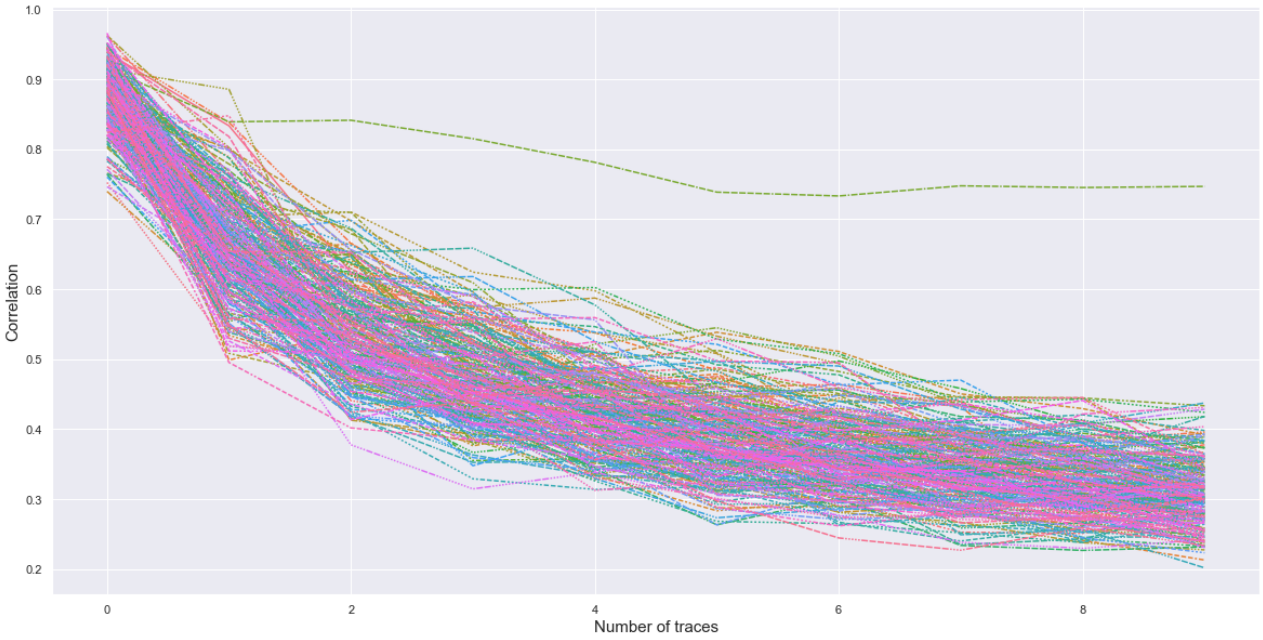
In [117…

```
# this function is for plotting the graphs for Plot-2
# the legend is removed as there are 256 different lines for 256 different key values

def secondplotgraph(graphinput, byteindex):
    print("Plot for key byte " + str(byteindex))

    graphdata = {}
    for i in range(0,256):
        graphdata[hex(i)] = graphinput[i]

    secondplot = seaborn.lineplot(data=graphdata, legend=False)
    secondplot.set_xlabel("Number of traces", fontsize = 15)
    secondplot.set_ylabel("Correlation", fontsize = 15)
```

In [118…

```
# for byte 0
secondplotgraph(secondgraphinputproducer(0), 0)
```

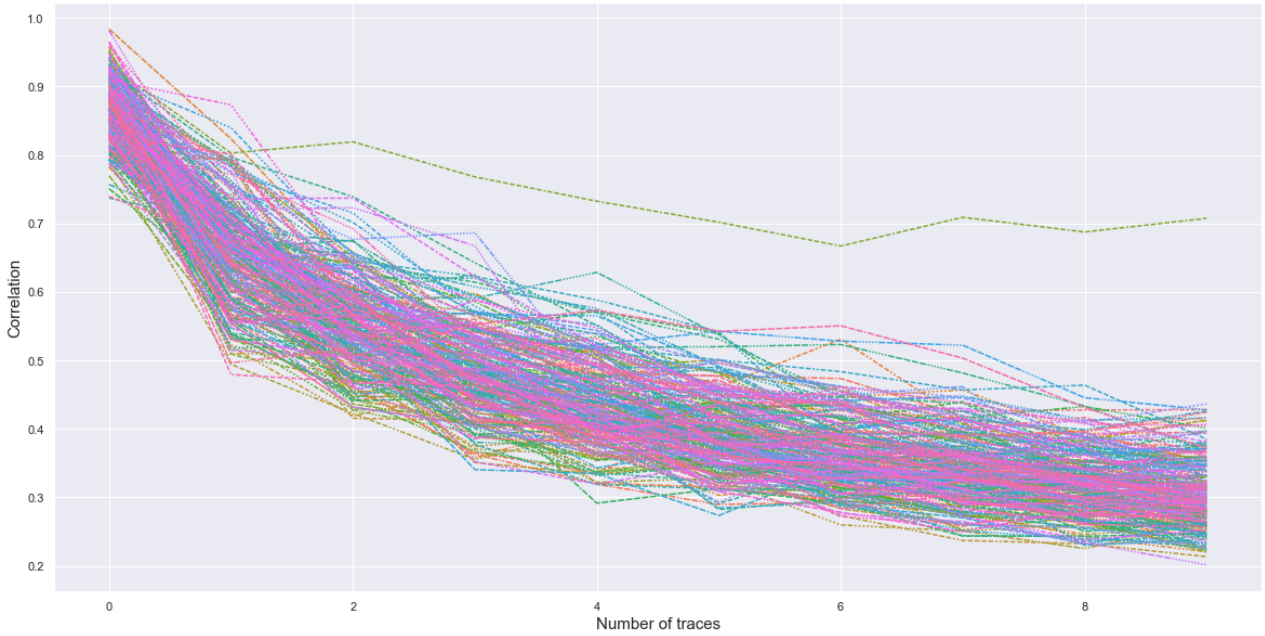
```
Plot for key byte 0
```



In [119…

```
# for byte 1
secondplotgraph(secondgraphinputproducer(1), 1)
```

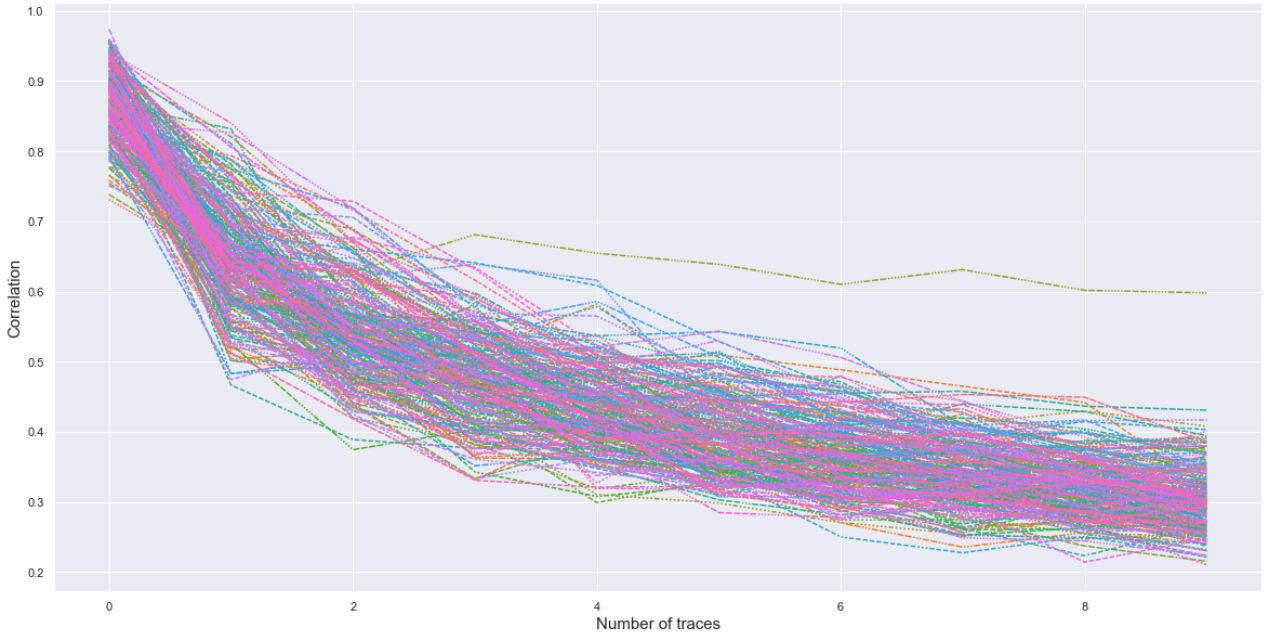
```
Plot for key byte 1
```



In [120…

```
# for byte 2
secondplotgraph(secondgraphinputproducer(2), 2)
```

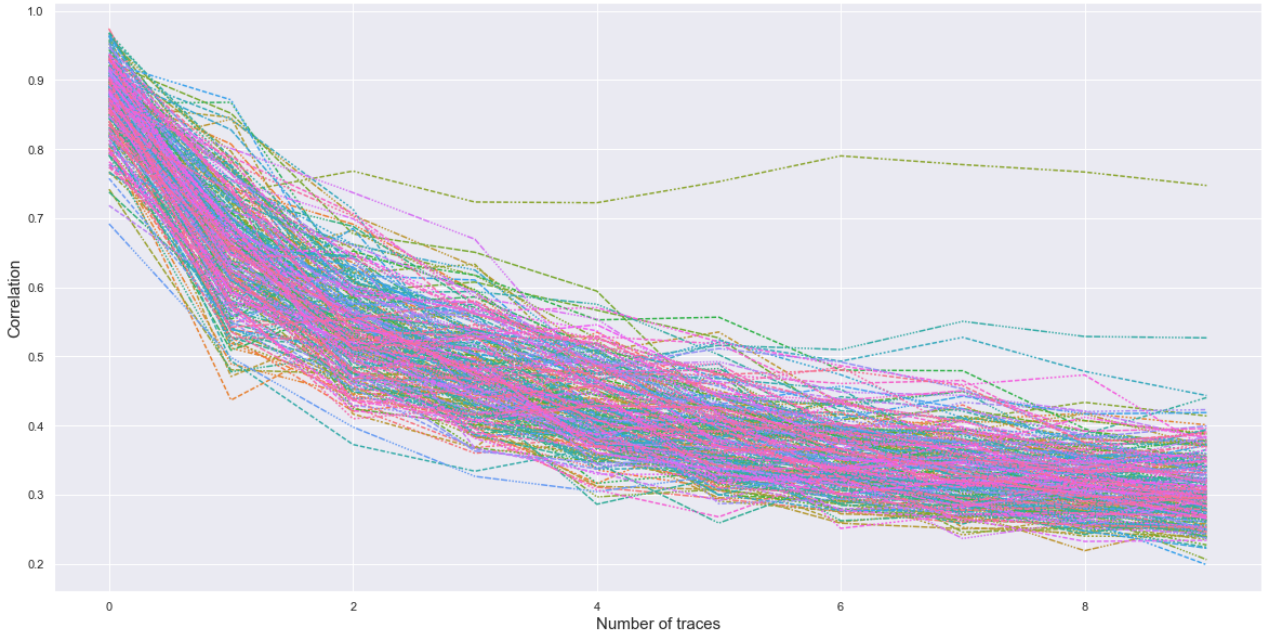
```
Plot for key byte 2
```



In [121…

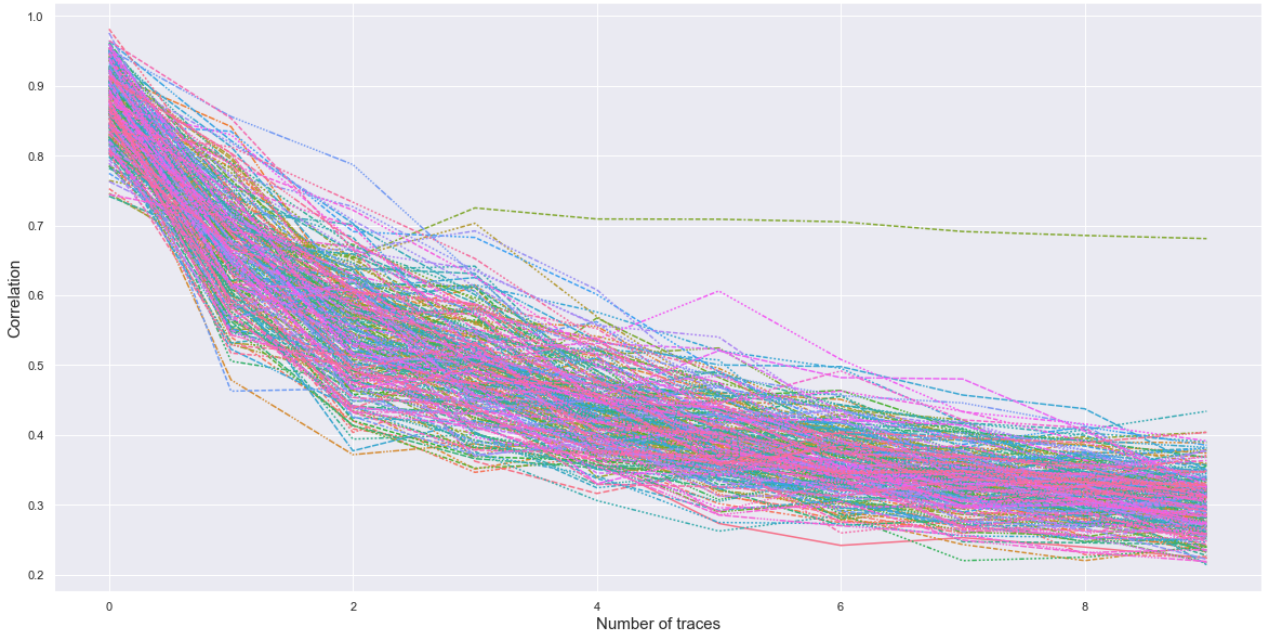
```
# for byte 3
secondplotgraph(secondgraphinputproducer(3), 3)
```

```
Plot for key byte 3
```



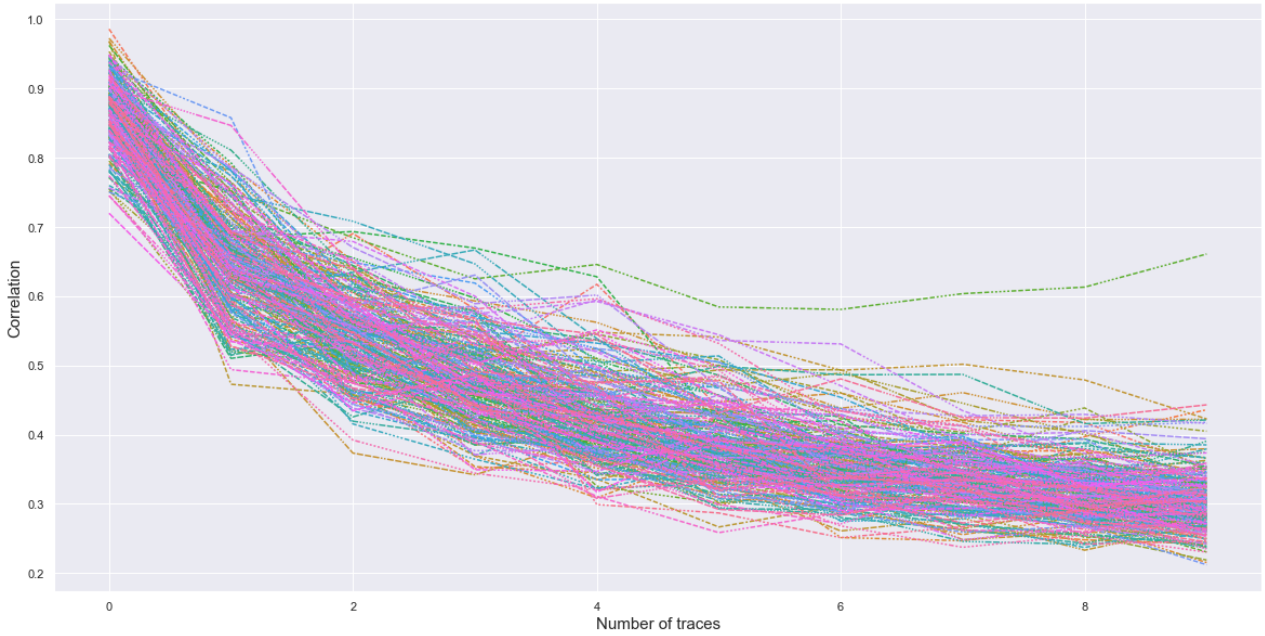
In [122…
```
# for byte 4
secondplotgraph(secondgraphinputproducer(4), 4)
```

Plot for key byte 4



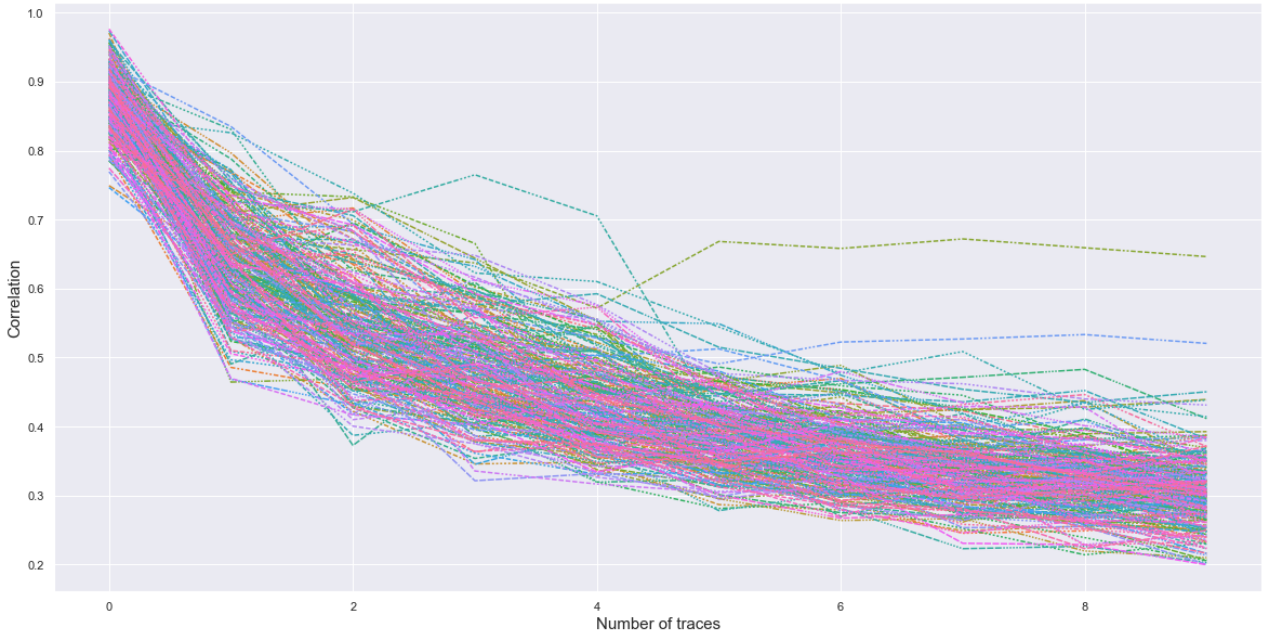
In [123…
```
# for byte 5
secondplotgraph(secondgraphinputproducer(5), 5)
```

Plot for key byte 5



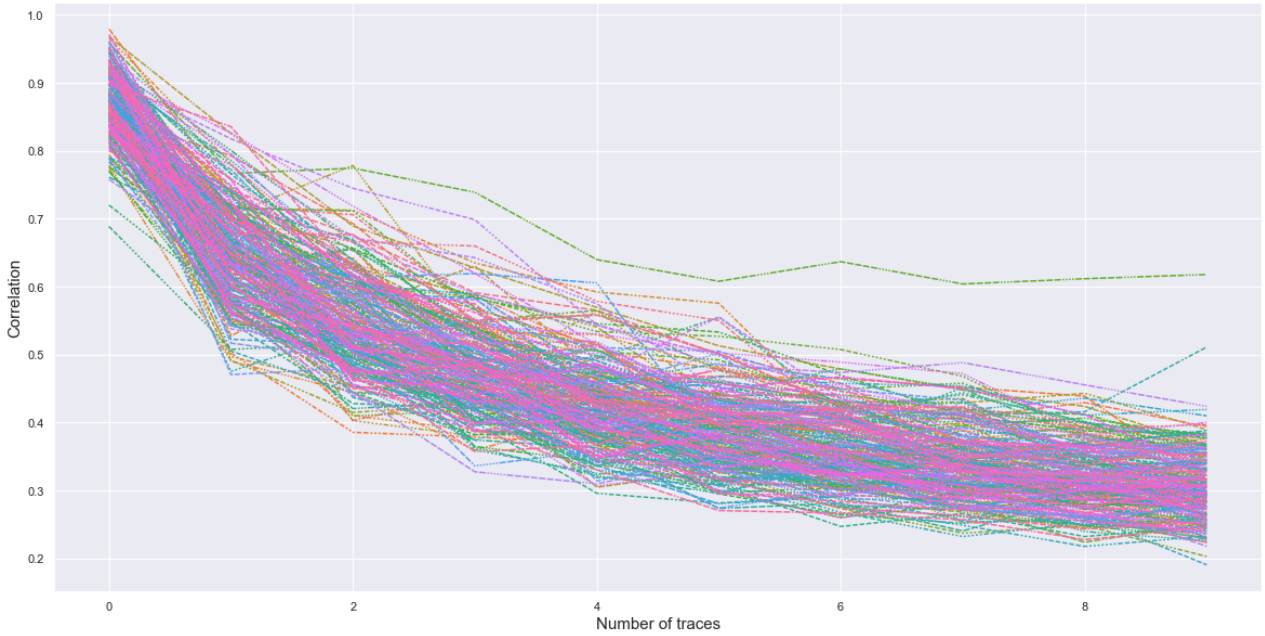
In [124…
```
# for byte 6
secondplotgraph(secondgraphinputproducer(6), 6)
```

Plot for key byte 6



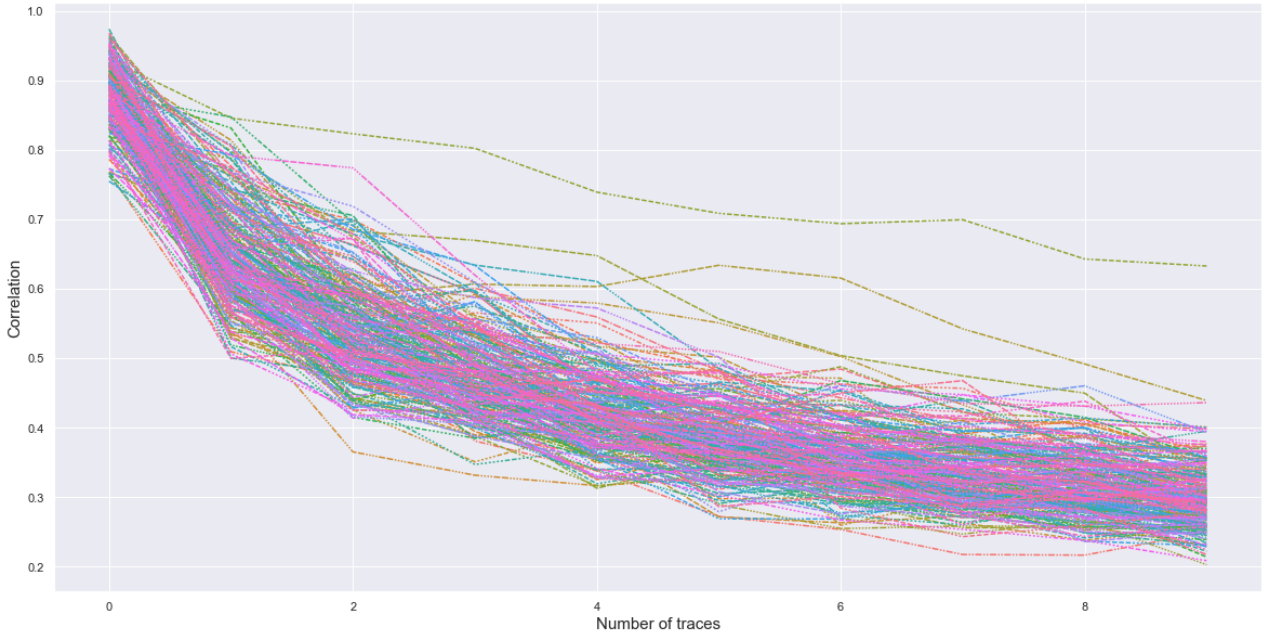
In [125…
```
# for byte 7
secondplotgraph(secondgraphinputproducer(7), 7)
```

Plot for key byte 7



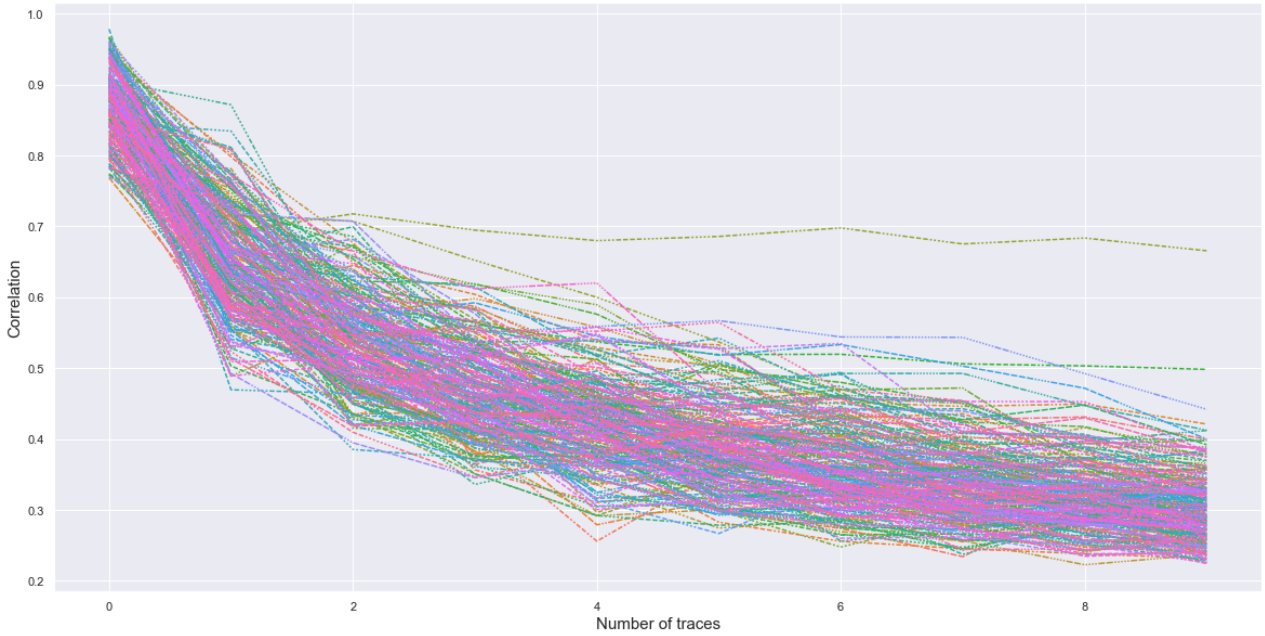
In [126…
```
# for byte 8
secondplotgraph(secondgraphinputproducer(8), 8)
```

```
Plot for key byte 8
```



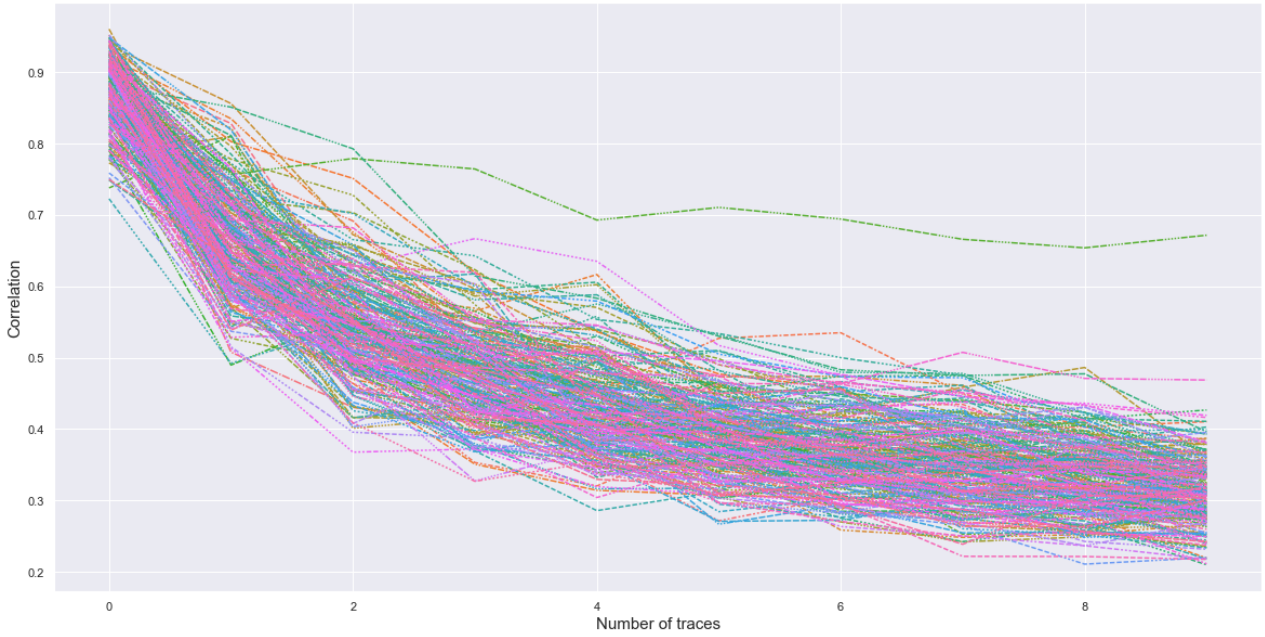
In [127…
```
# for byte 9
secondplotgraph(secondgraphinputproducer(9), 9)
```

```
Plot for key byte 9
```



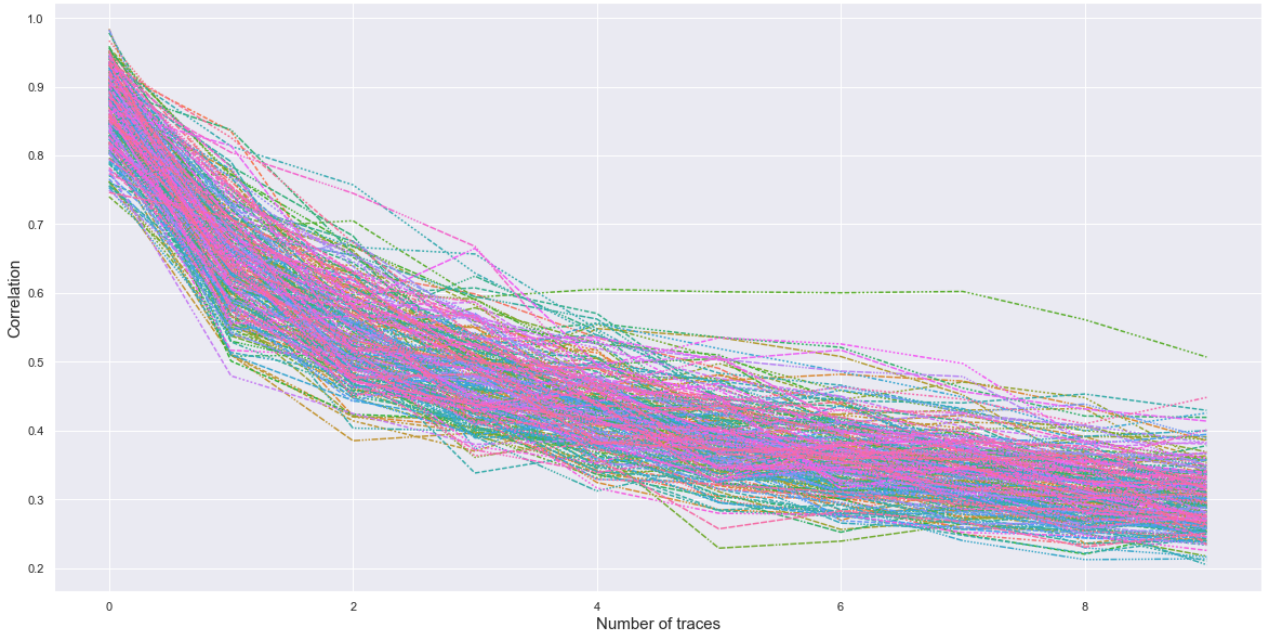
In [128…
```
# for byte 10
secondplotgraph(secondgraphinputproducer(10), 10)
```

Plot for key byte 10



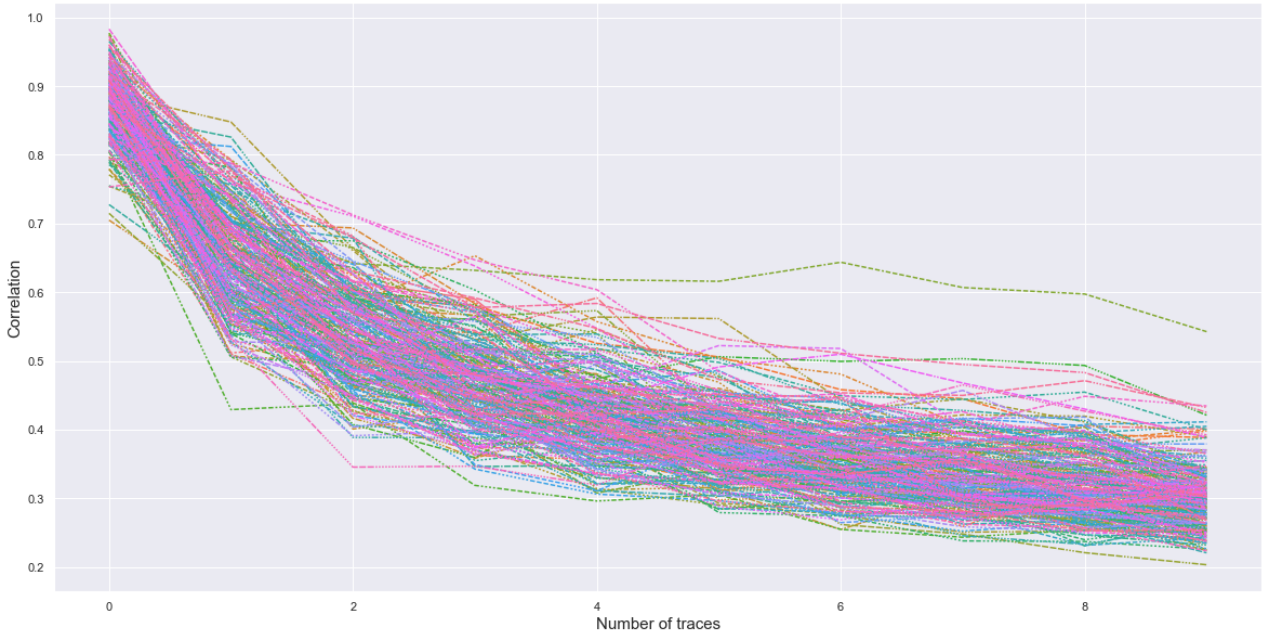
In [129…
```
# for byte 11
secondplotgraph(secondgraphinputproducer(11), 11)
```

Plot for key byte 11



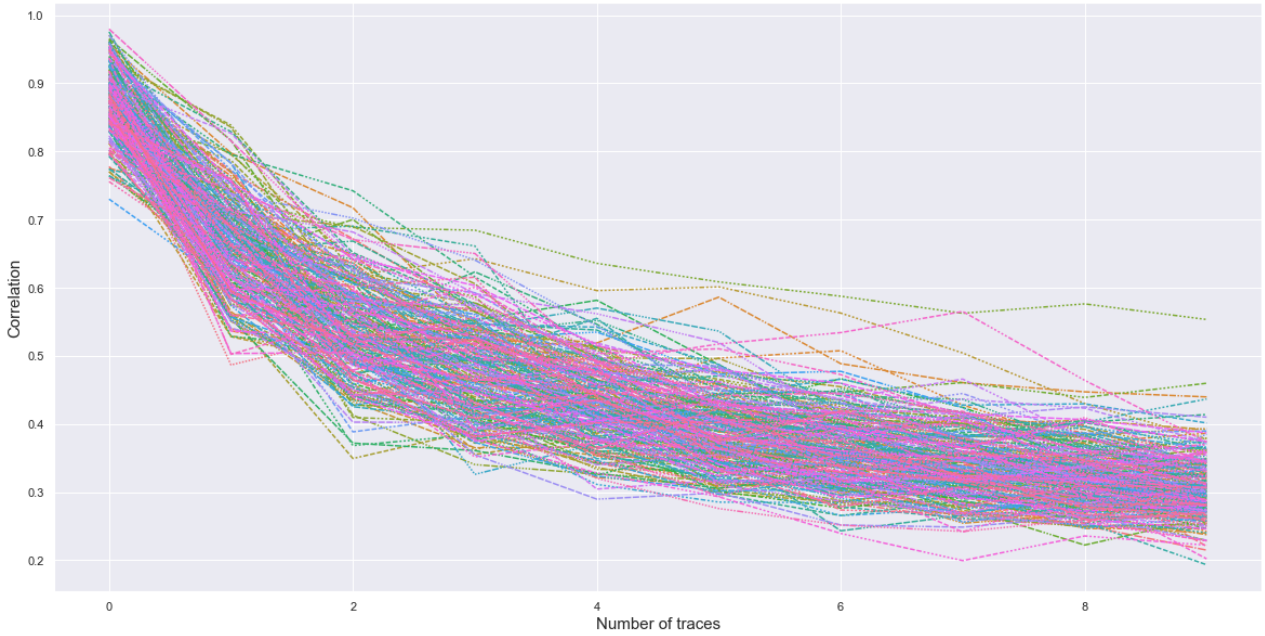
In [130…
```
# for byte 12
secondplotgraph(secondgraphinputproducer(12), 12)
```

Plot for key byte 12



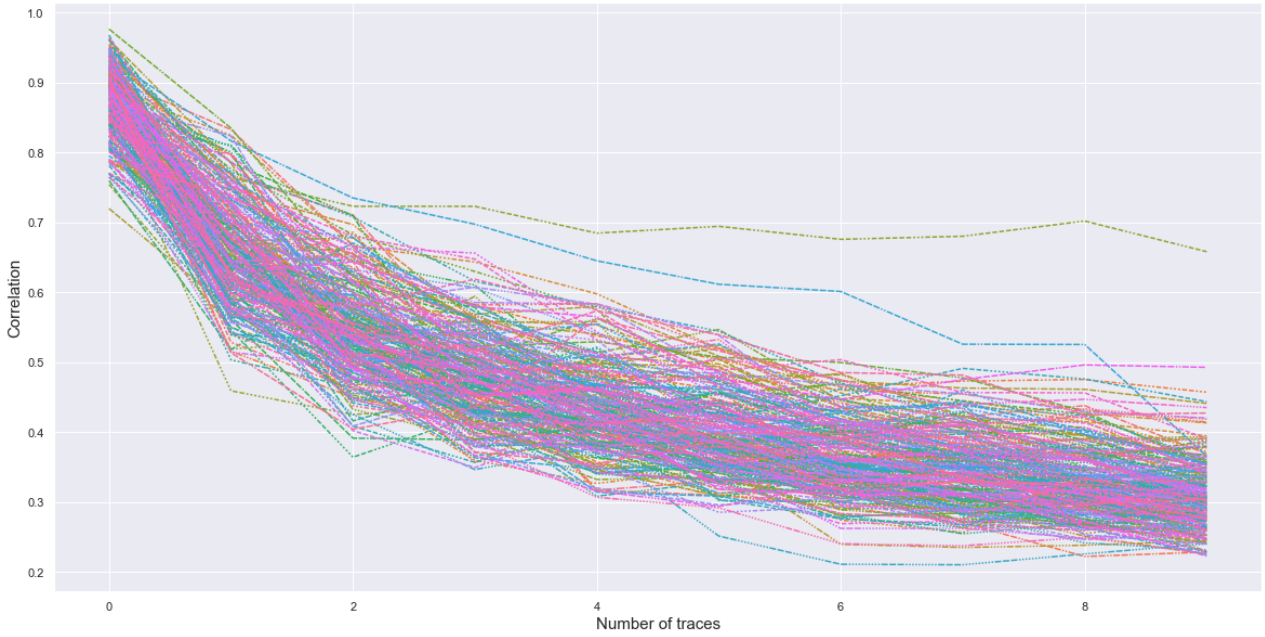
In [131…
```
# for byte 13
secondplotgraph(secondgraphinputproducer(13), 13)
```

Plot for key byte 13



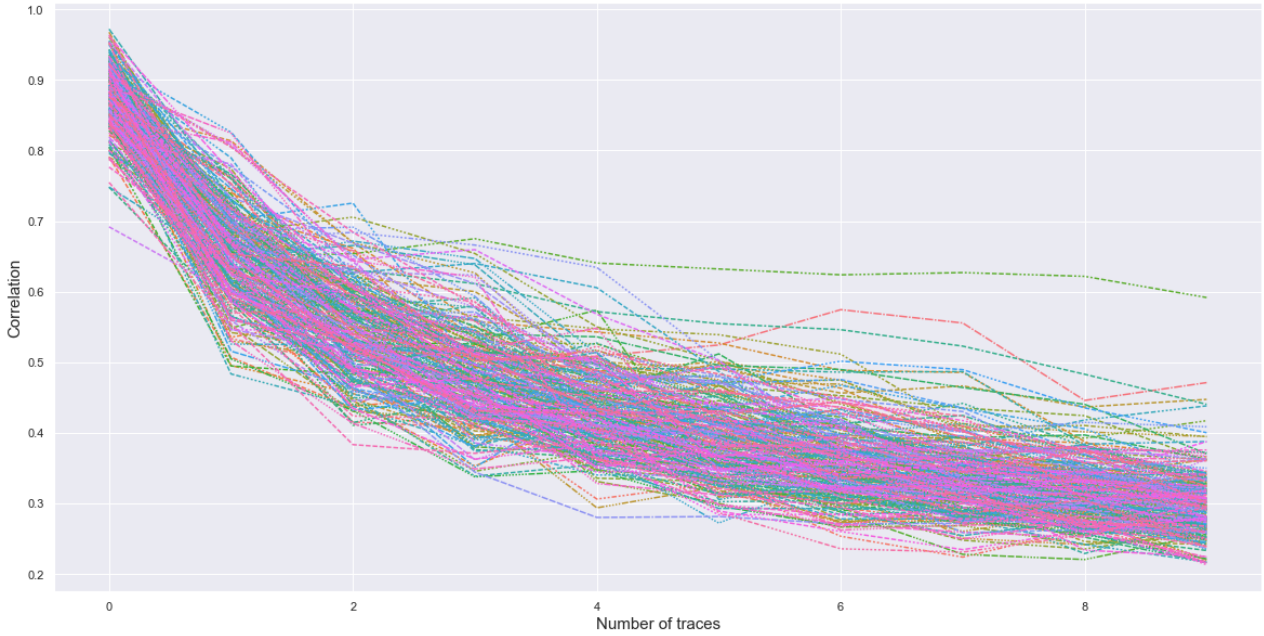
In [132…
```
# for byte 14
secondplotgraph(secondgraphinputproducer(14), 14)
```

Plot for key byte 14



In [133…
```
# for byte 15
secondplotgraph(secondgraphinputproducer(15), 15)
```

Plot for key byte 15


1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
Correlation
0
2
4
6
8
Number of traces