ECE599 - Assignment Homework 1

Opeyemi Ajibuwa

Question 1

1. Please consider the following AES inputs and compute the output after the first SubBytes operation. (10 pts)

Plaintext = 00 00 00 00 00 00 C1 A5 51 F1 ED C0 FF EE B4 BE

Key = 00 00 01 02 03 04 DE CA F0 C0 FF EE 00 00 00 00

Round Number	Start of Round	After Subbytes	After Shift Rows	After Mixcolums		Round Key value	
Input	00 00 51 FF 00 00 F1 EE 00 C1 ED B4 00 A5 C0 BE				⊗	00 03 F0 00 00 04 C0 00 01 DE FF 00 02 CA EE 00	=
1	00 03 A1 FF 00 04 31 EE 01 1F 12 B4 02 6F 2E BE	63 7B 32 16 63 F2 C7 28 7C C0 C9 8D 77 A8 31 AE			⊗		

The output after the first SubBytes

Question 2

2. Your coworker needs to implement the AES MixColumns operation in software. He found some code on stackexchange.com. You remember the best practice to review (and test) code before using it in production. Therefore, you offer to review the code. (10 pts)

a. **What is the type of problem here?**
"MixColumns_Mult_by2" function introduces timing dependencies based on the conditional checks on the MSB of the input. Depending on the value of the input, the execution time of the function will vary. This can potentially be exploited by an attacker to extract sensitive information through a side-channel attack, such as a timing attack.

b. **Identify the function(s) with undesired behavior.**
The function with the undesired behavior is “MixColumns_Mult_by2”. However, since the second function “MixColumns_Mult_by3” calls the first function, it also inherits the timing dependency.

c. **Suggest a code fragment that solves the problem under idealized assumptions.**
In the code fragment shown below, the “MixColumns_Mult_by2” function uses a ternary operator which does not introduce any timing dependency. The execution time of the function is constant regardless of the input value. Similarly, MixColumns_Mult_by3 function also does not introduce any timing dependency.

```c
unsigned char MixColumns_Mult_by2(unsigned char input) {
    return (input & 0x80) ? ((input << 1) ^ 0x1b) : (input << 1);
}

unsigned char MixColumns_Mult_by3(unsigned char input) {
    return (input ^ MixColumns_Mult_by2(input));
}
```

d. d) Is your solution processor independent? Please provide appropriate reasoning.

The code snippet provided above is not code dependent. It uses bitwise operations and logical operations which are implemented in the same way across different processors. The operations used in the code such as bitwise and, left shift, and xor are standard operations that are supported by most processors. The ternary also works in the same way across different processors. It is possible that the code may have different performance characteristics on different processors, but the output of the code will be same regardless of the processor.

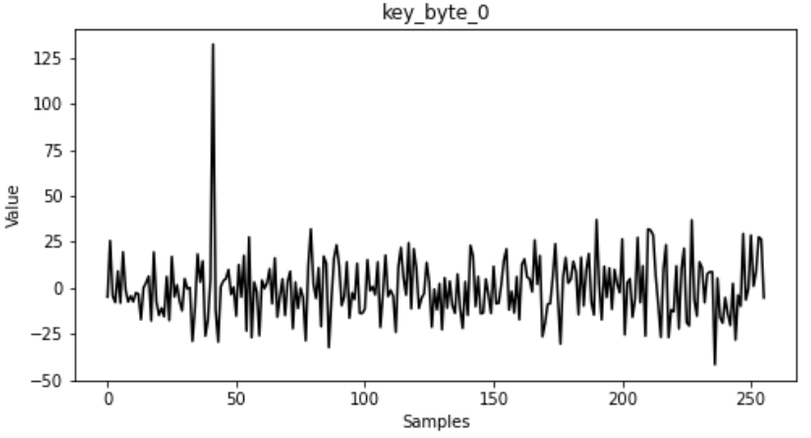
Question 3

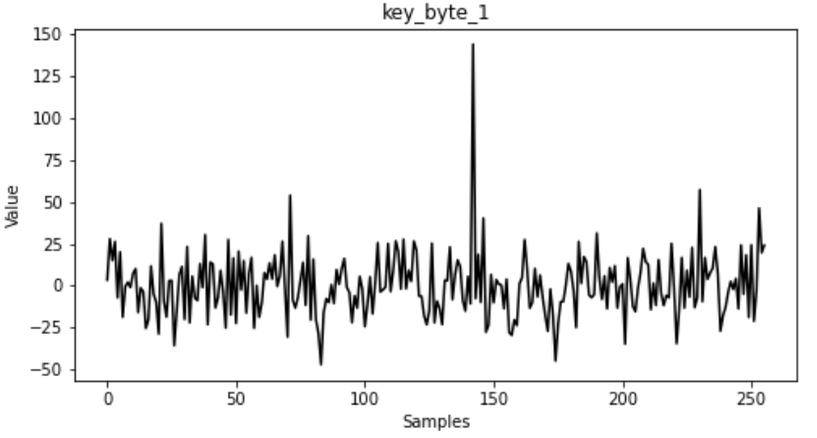
a. Recover the key. Please assume that the Most Significant Bit (MSB) is a good hypothesis. Provide figures to support that you found the correct key.

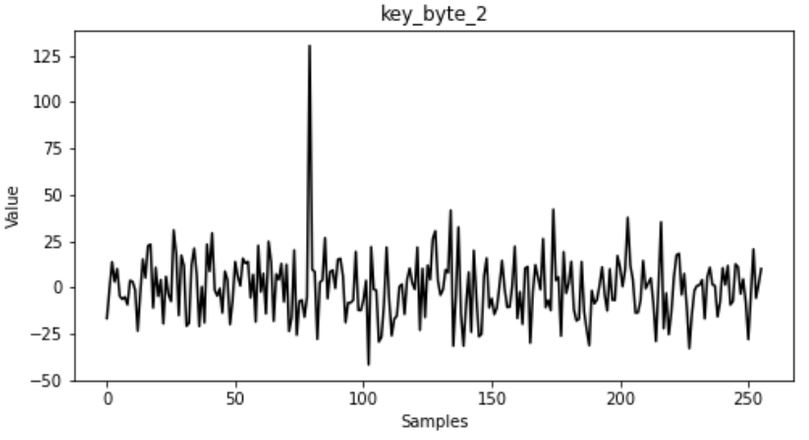
The 16 keys recovered are listed below as follows:

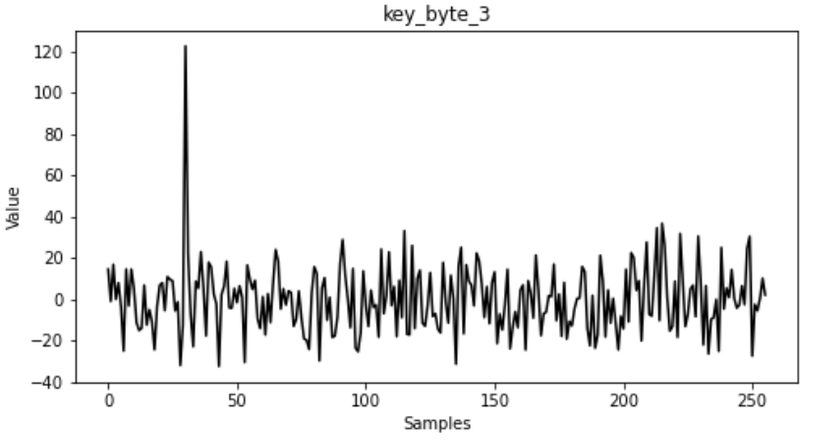
Keys	41, 142, 79, 30, 183, 104, 193, 19, 15, 246, 189, 223, 236, 119, 47, 176

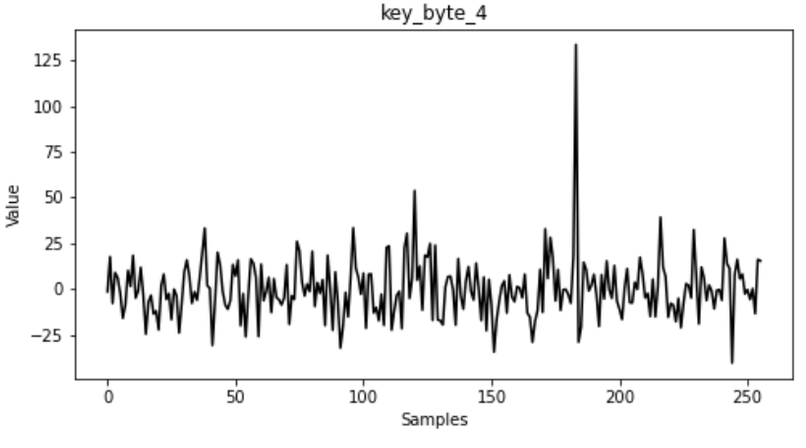
The following figures correspond to each of the recovered keys:

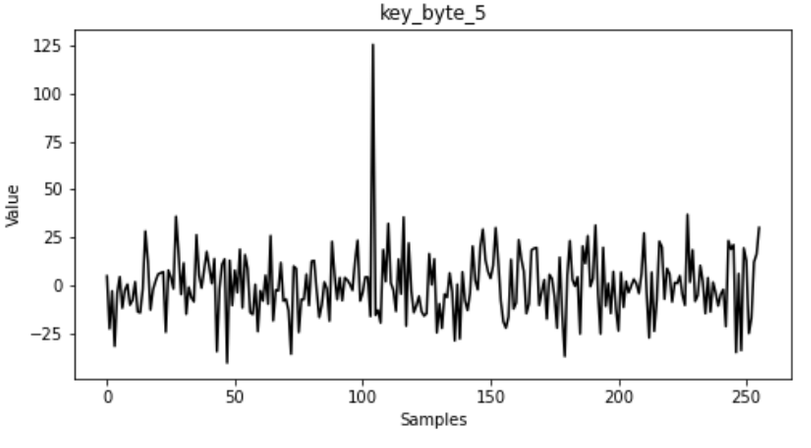


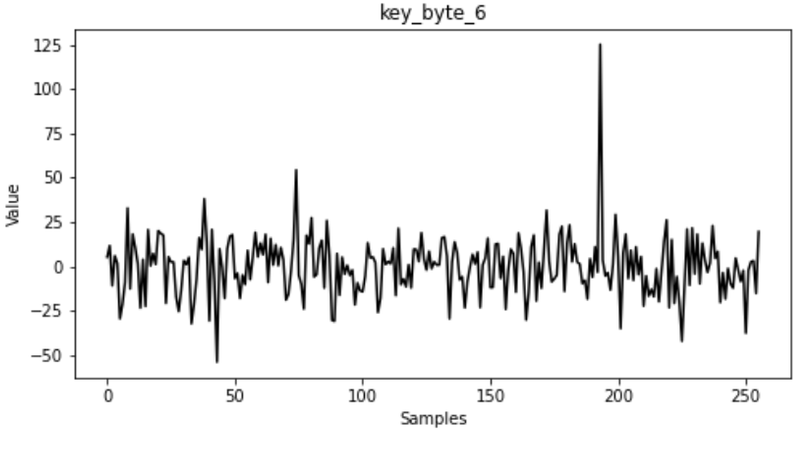


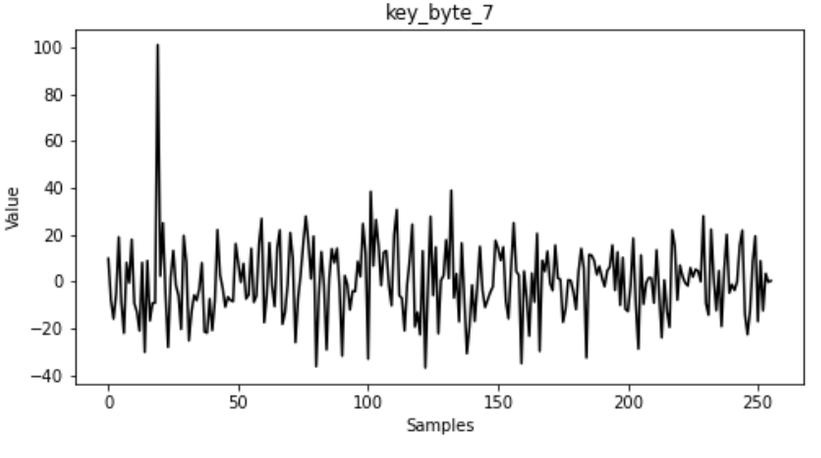

key_byte_2
Value
Samples

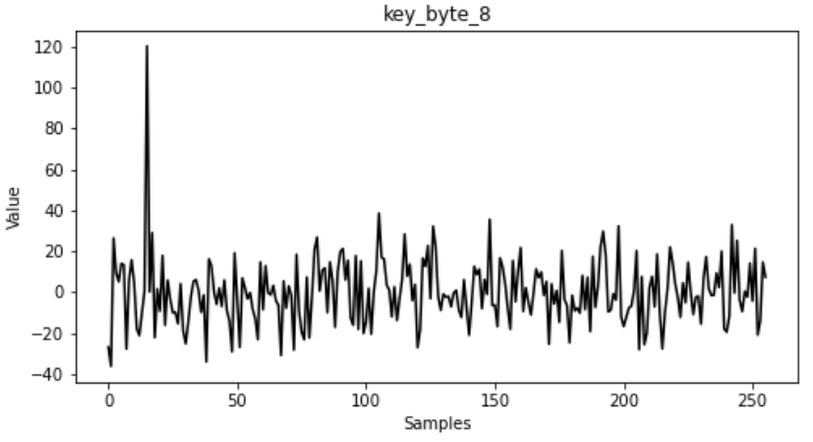

key_byte_3
Value
Samples

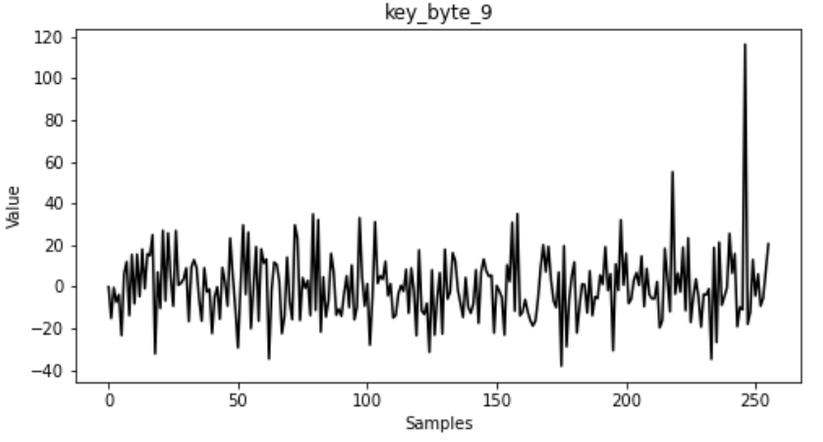

key_byte_4
Value
Samples

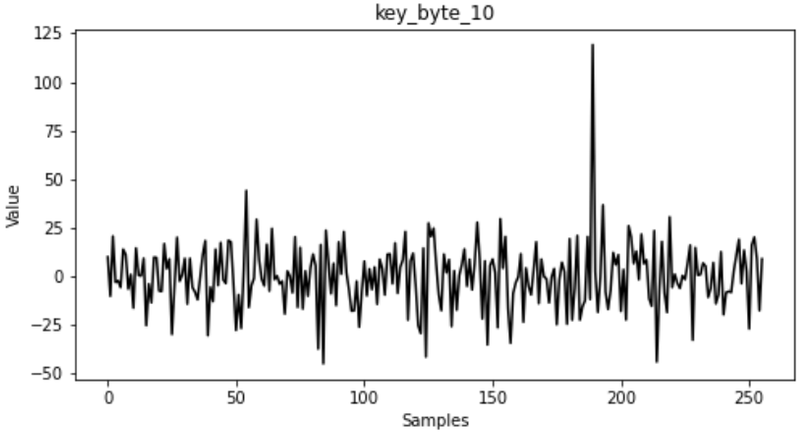

key_byte_5
Value
Samples

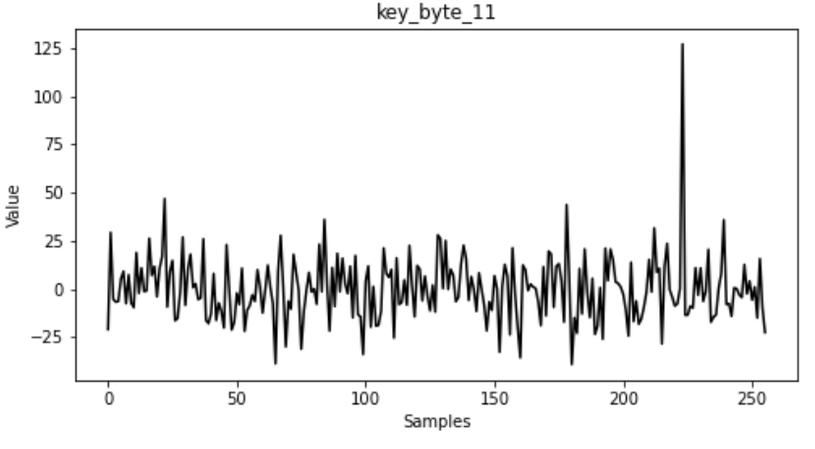

key_byte_6
Value
Samples

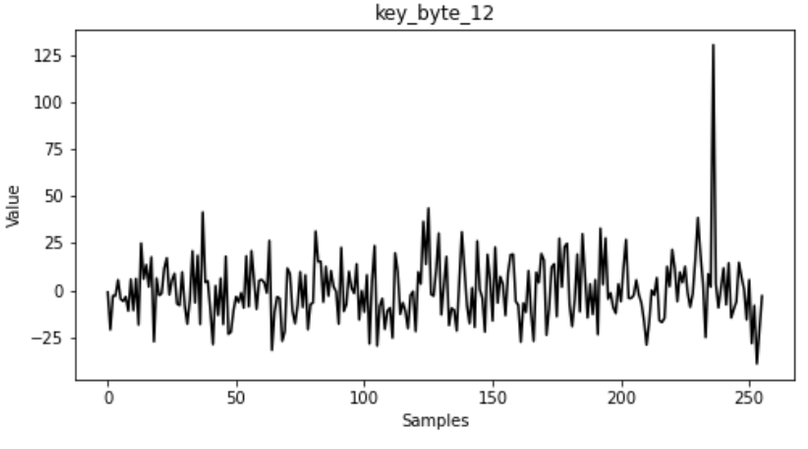

key_byte_7
Value
Samples

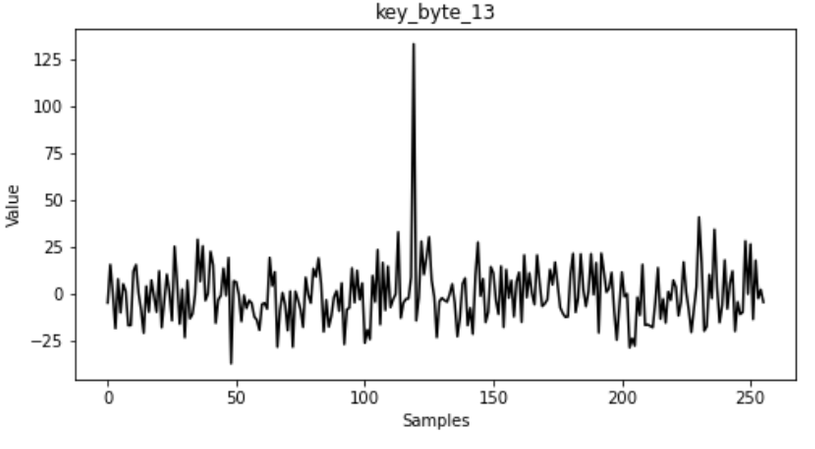

key_byte_8
Value
Samples

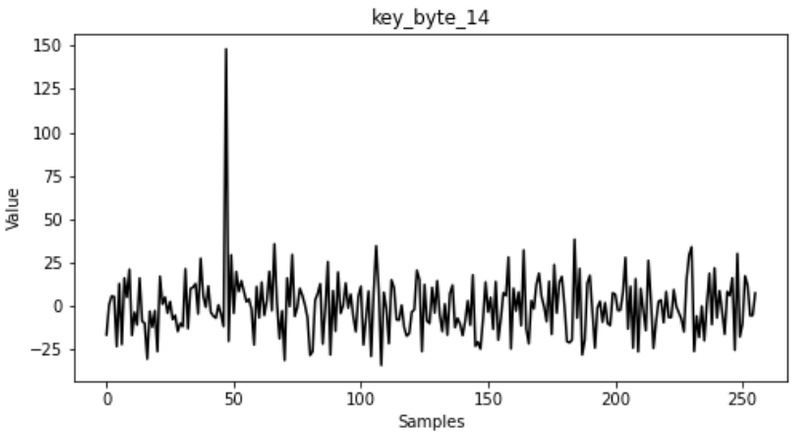

key_byte_9
Value
Samples

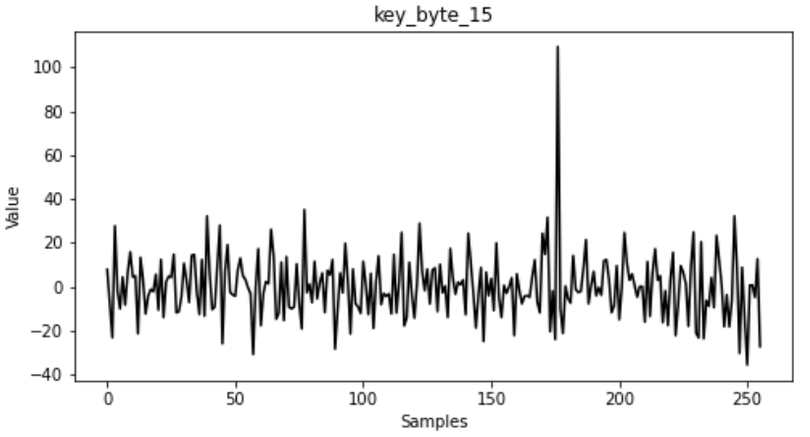

key_byte_10
Value
Samples


key_byte_11
Value
Samples


key_byte_12
Value
Samples


key_byte_13
Value
Samples





b. Check how many samples you need for each key byte. This can be done, e.g., in 1 000 step increments and does not need to be an exact number.

	Key byte	Min no of samples
0	41	100000
1	142	70000
2	79	40000
3	30	50000
4	183	80000
5	104	85000
6	193	107000
7	19	110000
8	15	127000
9	246	103000
10	189	60000
11	223	29000
12	236	48000
13	119	103000
14	47	27000
15	176	30000

c. Optimize your code such that the overall attack time (without incremental steps) is well below 3 minutes. Please report your execution time in addition to your processor and memory specification. Include a note which operating system you used. If you needed to optimize your code to improve runtime, briefly include a note how you optimized

Execution time	2mins, 14secs
Processor	AMD Ryzen 9 5900HX, 3.30Ghz, 8 Core(s), 16 Logical Processors with Nvidia GeForce RTX 3060
Memory	32GB
OS	Windows 11

To optimize my code, I mostly relied on vectorizing the data using NumPy. I tried improving the performance by multithreading the code execution with python multiprocessing library, but that surprisingly didn't yield any improvement. Therefore, I stuck with optimizing the code only with NumPy vectorizations since the execution time is well below the 3 mins threshold.