Name: Hsuan-Chih Hsu

1. The file measurement_data_uint8.bin contains 1 billion samples of type uint8. This is representative of actual measurement data, as many oscilloscopes have 8 bit of resolution due to the Analog-to-Digital Converter (ADC) being used. **Your task is to compute the mean and variance** of this data by applying different methods. Note: this is about incrementally processing the data and updating intermediate variables, to then compute the final result. Do not use any readily available functions for mean/variance.

[NumPy] Mean: 133.700091185, Var: 1.7878604616853109.
Code is named as **task1.py** in the .zip file. And the final results by using 4 different methods are almost the same as the result from using NumPy Mean and Variance.

a) Naïve approach

```
----------Naive algorithm----------
Mean:  133.700091185
Var:  1.7878604616835938
Time:  346.3566687107086
```

b) Welford's algorithm

```
----------Welford's algorithm----------
Mean:  133.7000911848925
Variance:  1.7878604616782383
Time:  496.45581340789795
```

c) One-pass arbitrary order method

```
----------One pass----------
Mean:  133.7000911848925
Variance:  1.7878604616782392
Time:  452.0204963684082
```

d) Histogram method

```
----------Histogram method----------
Mean:  133.700091185
Variance:  1.7878604616875
Time:  4.641166687011719
```

e) Compare the runtime of the different methods.

Methods:	Time:
Naïve approach	346.356 seconds
Welford's algorithm	496.455 seconds
One-pass arbitrary order method	452.02 seconds
Histogram method	4.641 seconds

2. In the lecture, we learned that not only key extraction may be a viable goal of a side-channel analysis but also leakage detection. In particular, leakage detection may help to choose points for more computationally demanding attacks or as part of a security certification process to confirm the lack of points of interest. The most basic leakage detection is the SNR (cf. Appendix). To develop a natural understanding of what is happening, first compute the numerator and denominator separately.

a) Compute the numerator ('signal') of the traces and plot the result.

b) Compute the denominator ('noise') of the traces and plot the result.

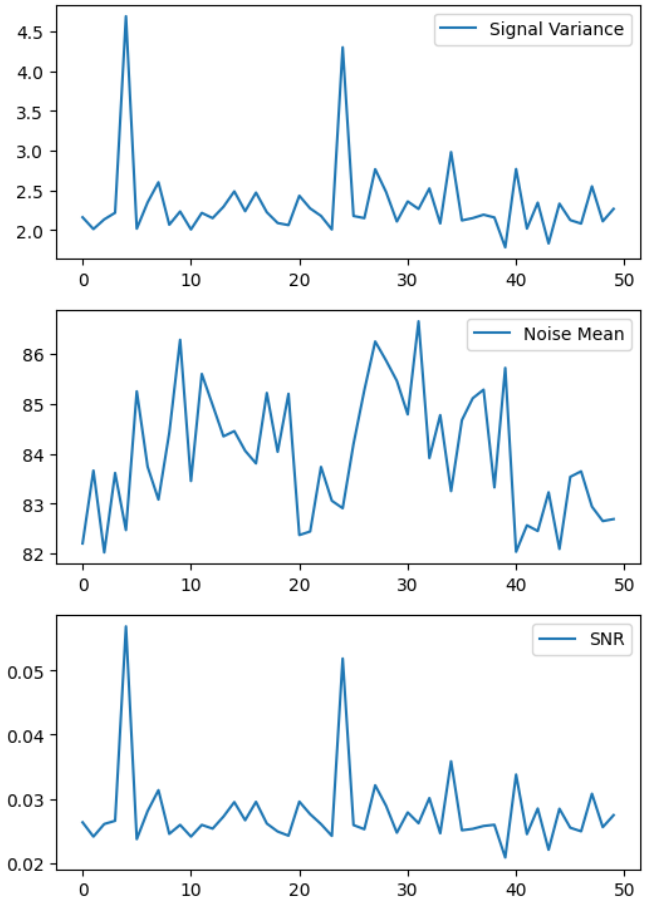
c) Compute the whole SNR and plot the result.

Code is named as **task2.py** in the .zip file. I have plotted all the key byte results, i.e., Signal, Noise, SNR, in the following pages. And the results are all matched the range as shown below.

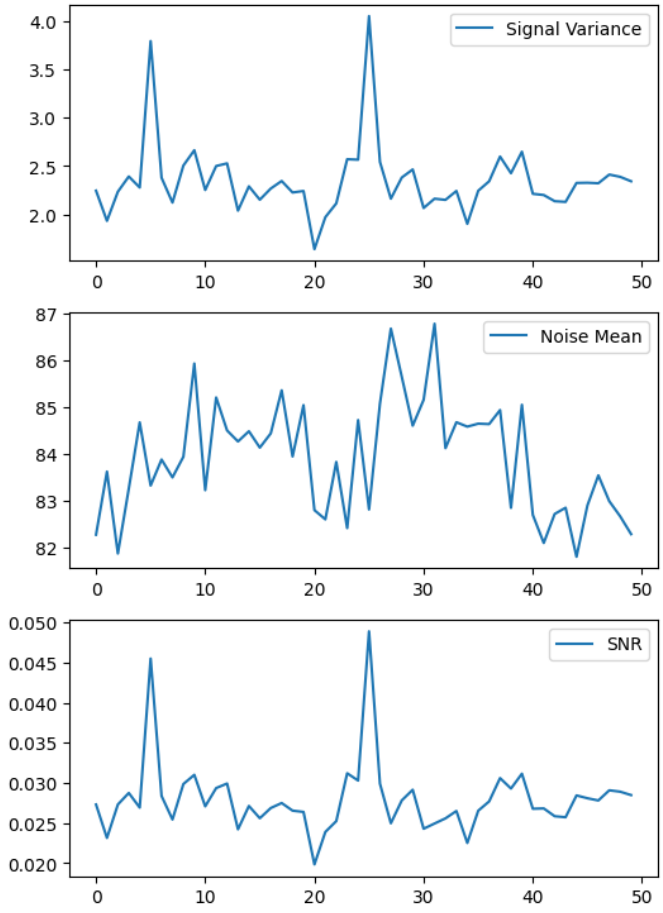
Signal y-range: approx. 1 to 5 with 2 nice peaks.
Noise y-range: approx. 80 to 90
SNR y-range: approx. 0.02 to 0.06 with 2 nice peaks.

Note: I do not list all the result in the report. Instead, I have them printed out in the console of PyCharm.

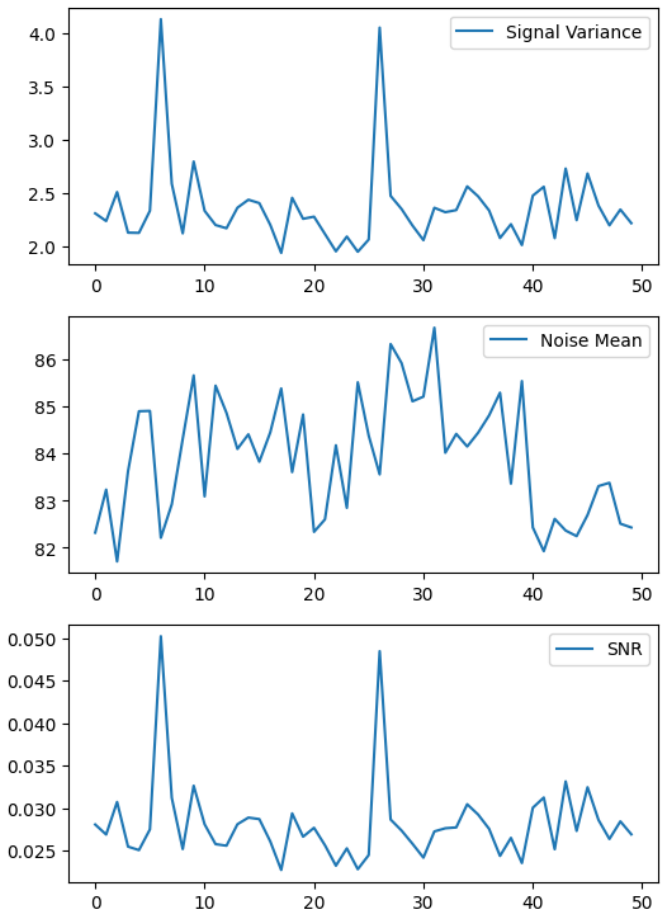
Byte 1



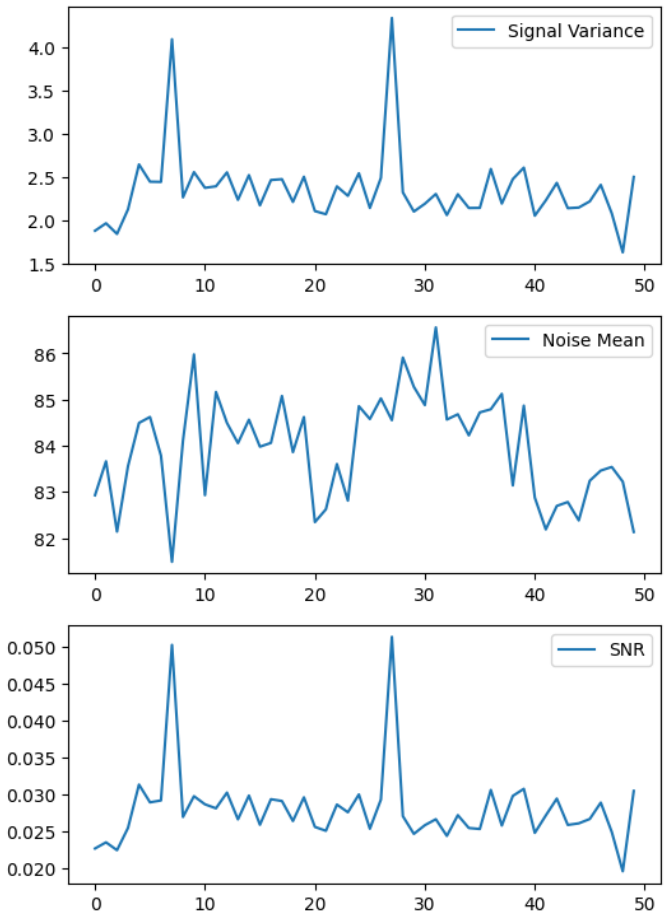
Byte 2



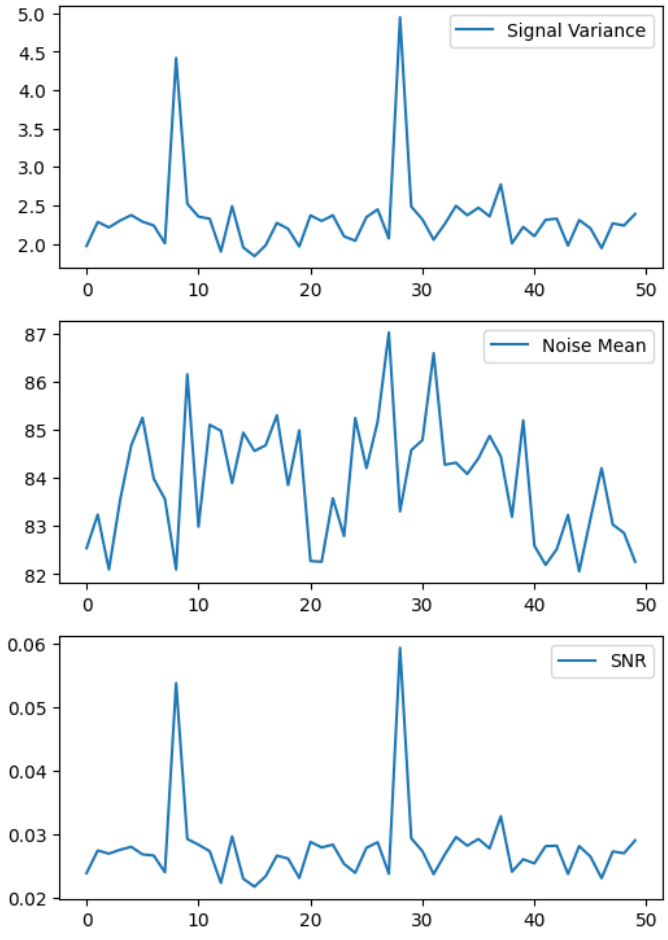
Byte 3



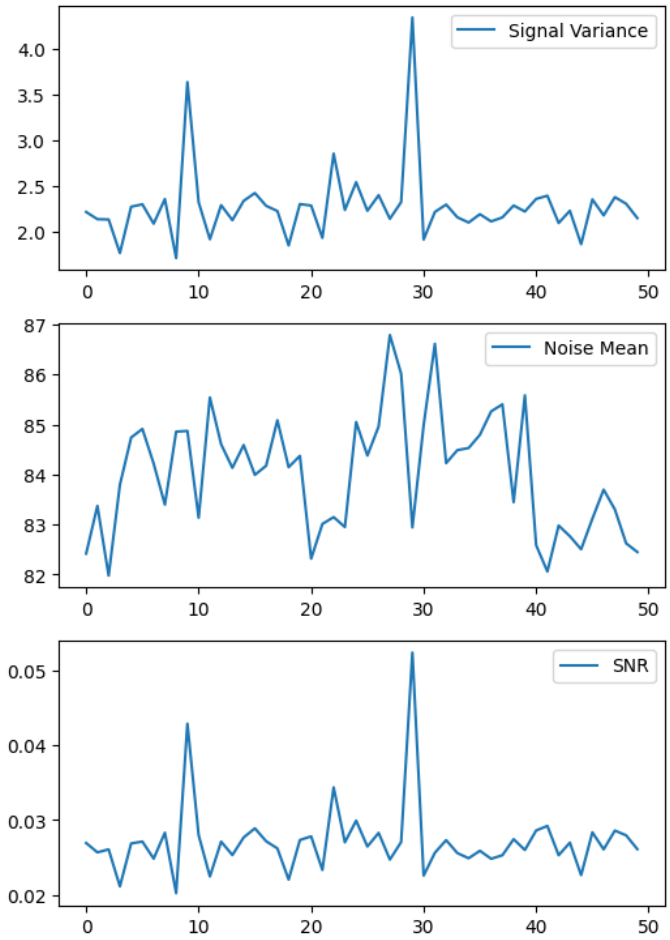
Byte 4



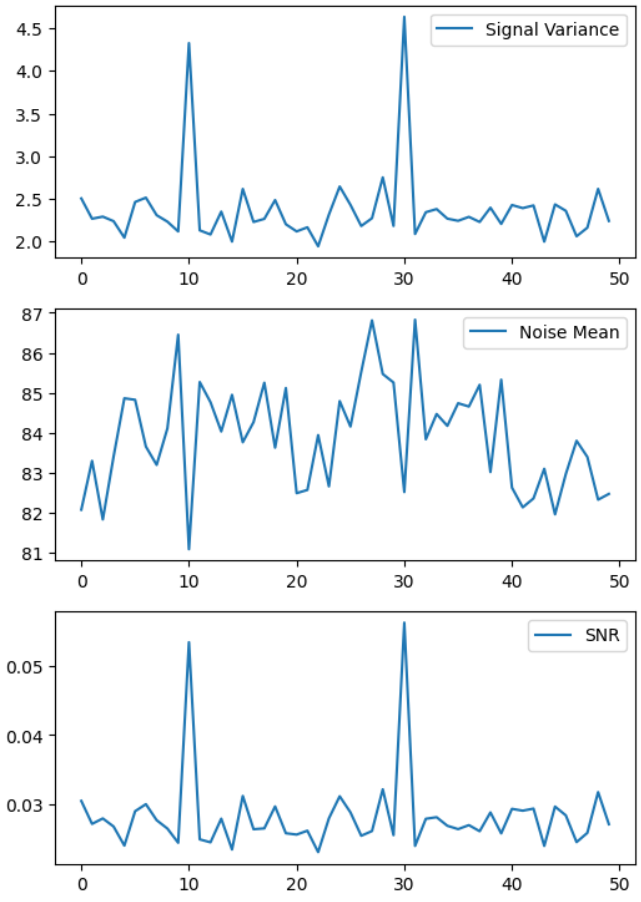
Byte 5



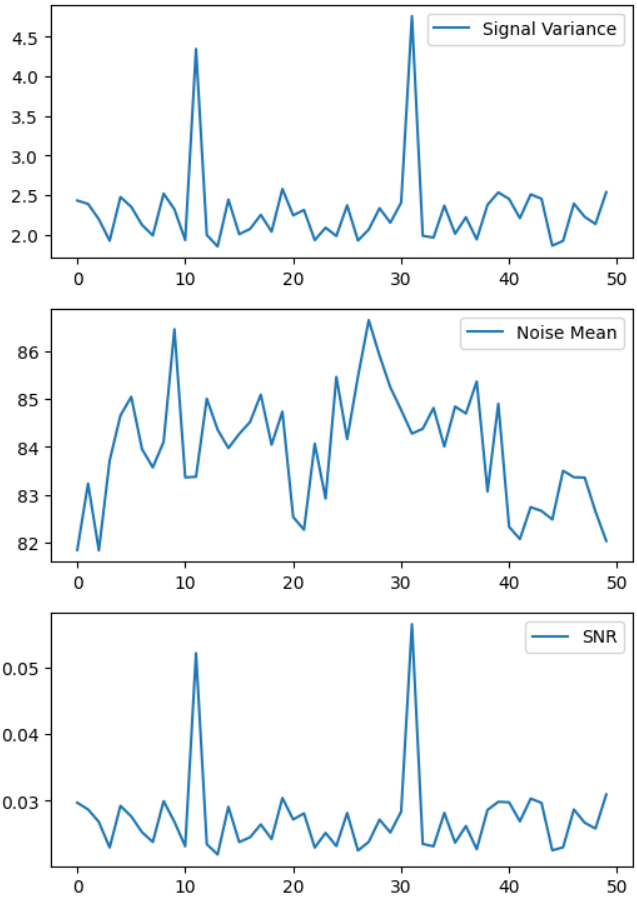
Byte 6



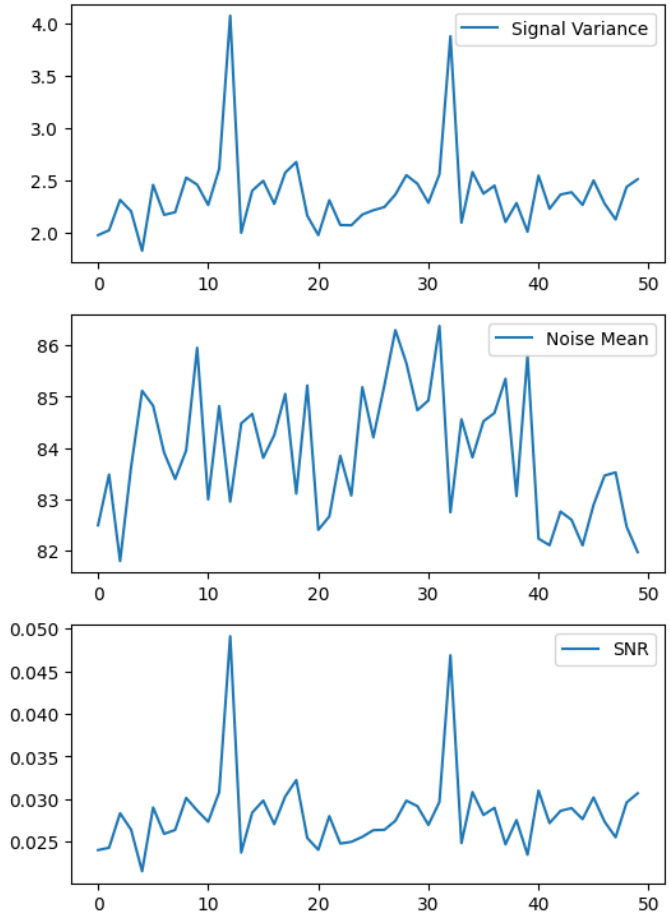
Byte 7



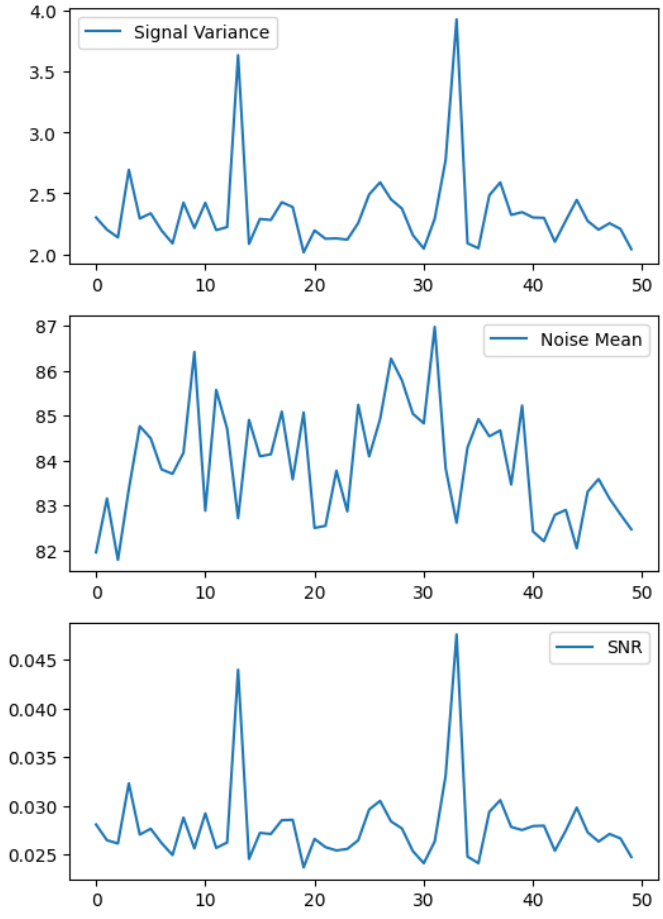
Byte 8



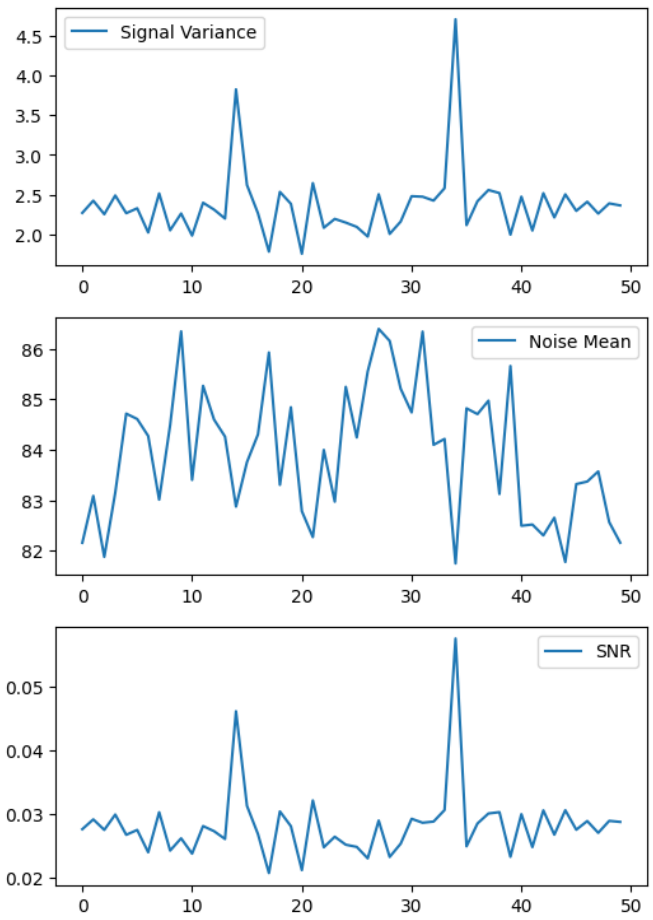
Byte 9



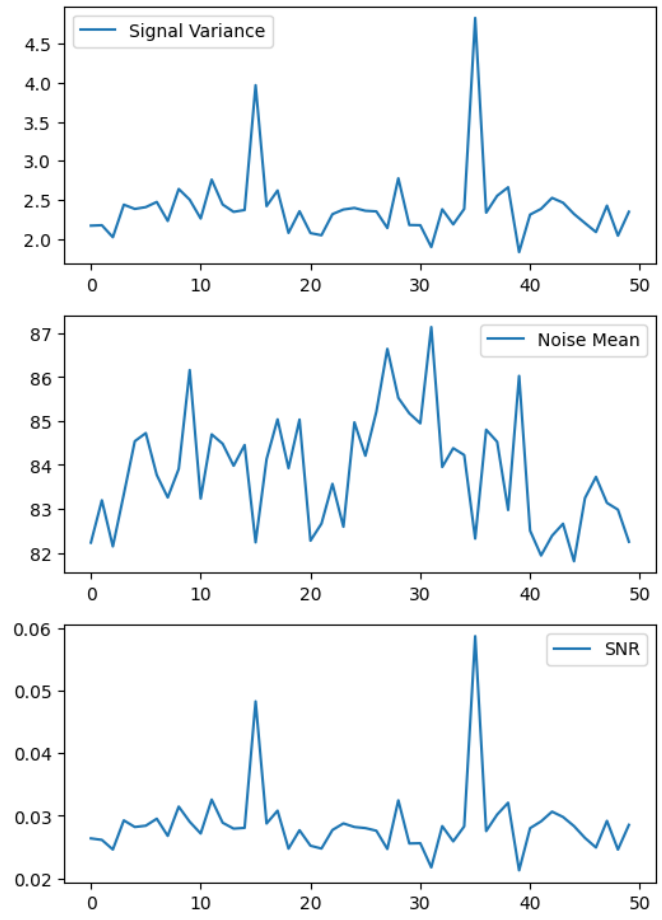
Byte 10



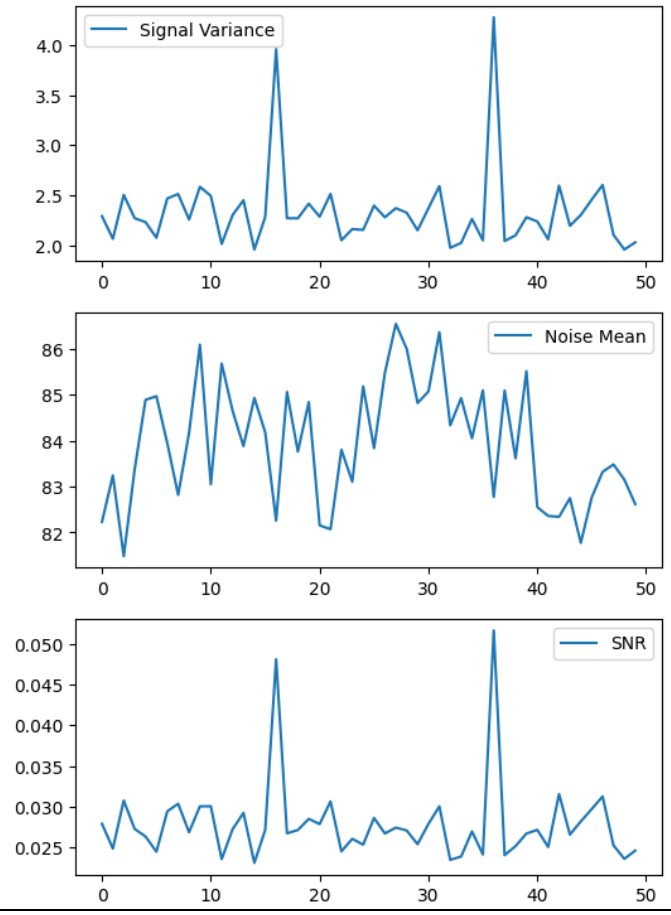
Byte 11



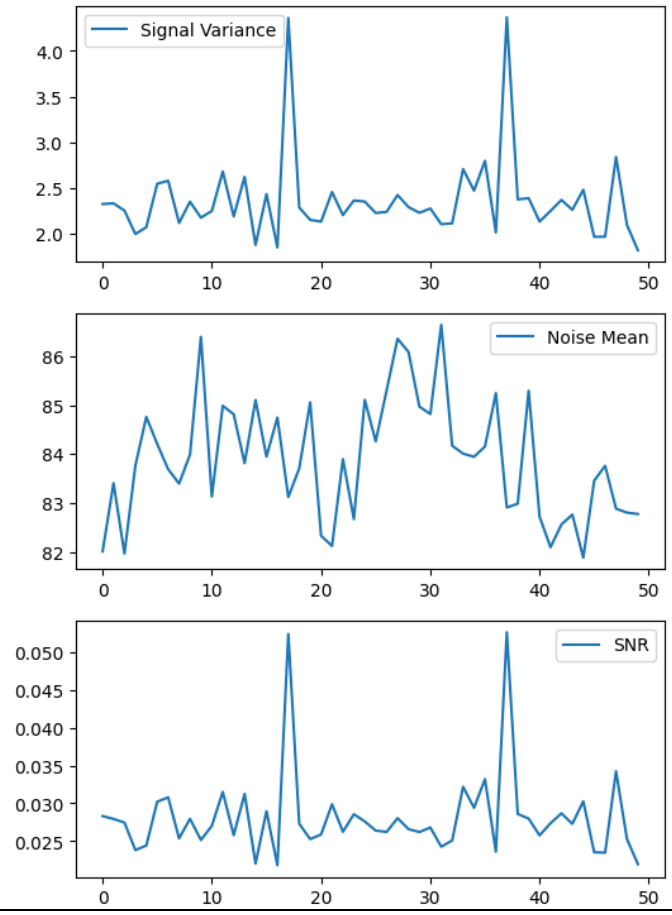
Byte 12



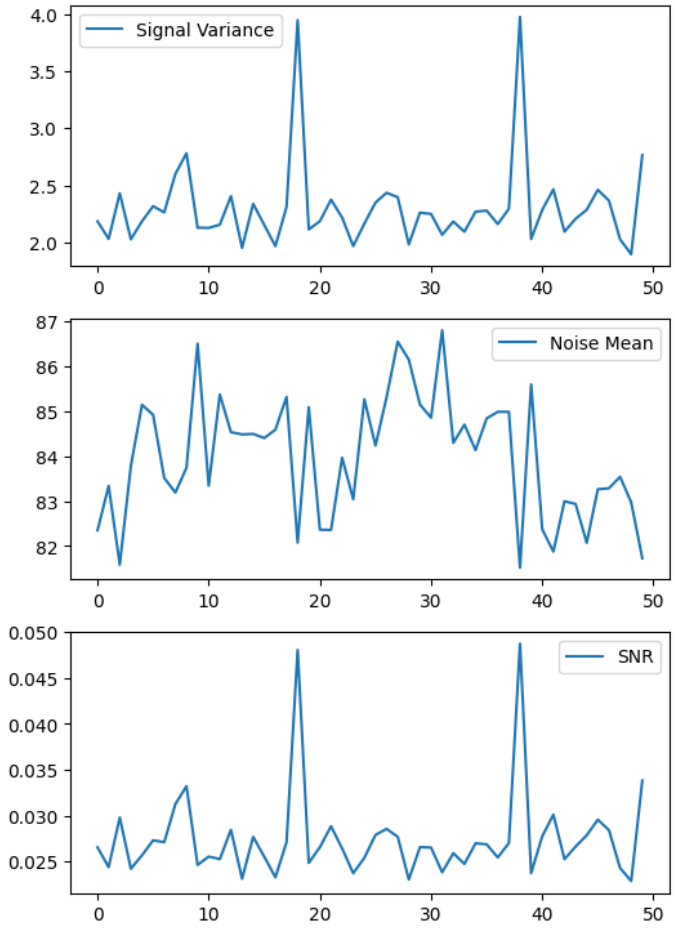
Byte 13



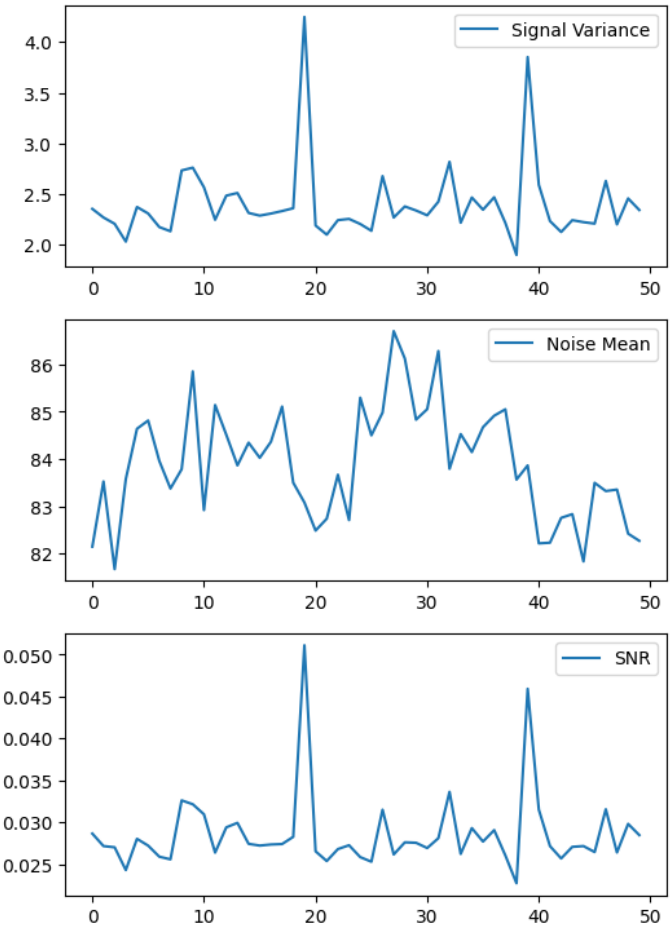
Byte 14



Byte 15



Byte 16



3. Perform a Correlation Power Analysis (CPA) on the traces to extract they key. Do not use any external libraries that provide readily available functions for side-channel analysis. **Please report the execution time of your attack.**

Code is named as **task3.py** in the .zip file. I use the NumPy and Multiprocessing to speed up my attack.
Key: [237, 58, 24, 116, 22, 40, 132, 182, 197, 231, 224, 64, 215, 78, 87, 187]
Time: 103.18786811828613