# Assignment 1

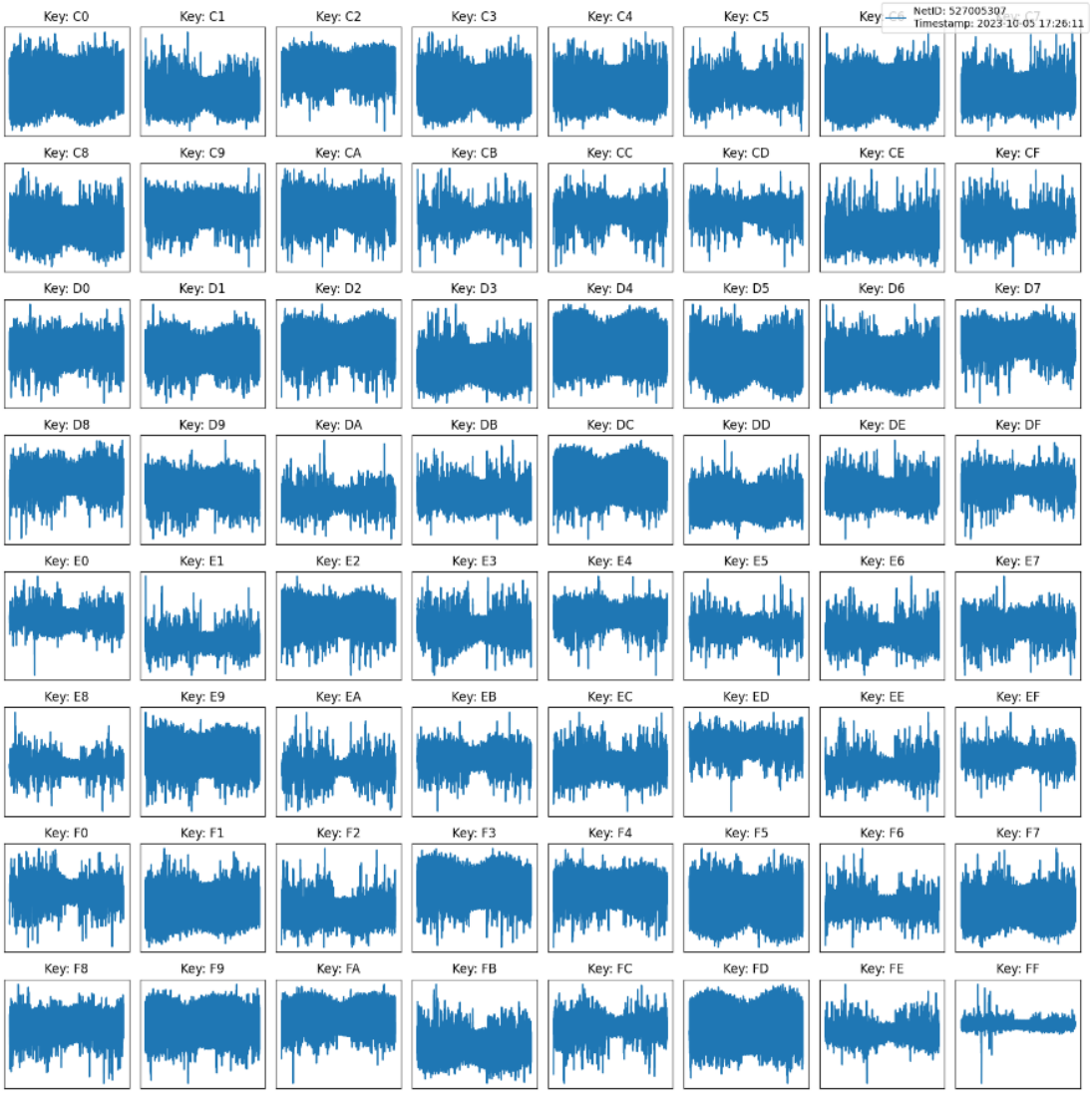
ECEN 426

Name: Daniel Horan

UIN: 527005307

**Overview:**
For this assignment, we were asked to launch a Differential Power Analysis (DPA) attack and extract the key for the Advanced Encryption Standard (AES). We were suggested to use MATLAB for this assignment but due to a lack of knowledge of it, I've proceeded with using Python through Google Colab which unfortunately caused some complications in integrating ML algorithms

**Explaining The Code for Task 1:**
The code performs a DPA attack on AES encryption using power consumption data from the provided .mat file. For each possible key guess (from 0 to 255; 256 overall), it XORs a specific plaintext byte with the key, retrieves an output from the AES S-box, and sorts power traces into two groups based on the output's LSB. It calculates the Difference of Means (DoM) for the two trace groups, aiming to find a distinct pattern that might give us the correct key byte. The results are visualized in a grid of plots, each corresponding to a key guess.
**Results of Task 1:**
These are the plots I got as a result:




As we can see, Key: FF (bottom right plot) has the most distinct peak which tells us that this is the correct key.

**Explaining The Code for Task 2:**
The code performs a DPA attack on AES encryption to try to recover the secret key. This is done by analyzing power consumption traces during the encryption process. The code loads power consumption data from the given .mat file. It then systematically guesses each byte of the AES key (from 0 to 255; 256 overall). For each guess, it computes an S-box output based on the plaintext and the guessed key byte, groups traces based on the LSB of the S-box output, and calculates the DoM for these groups. A large DoM indicates a potential correct key guess. The process is repeated for all 16 bytes of the AES key. The final recovered key is then compared with a given actual key to calculate the accuracy of the recovery in terms of correctly guessed bits.

To improve performance, the second code performs a DPA attack on AES encryption using the Adaline Stochastic Gradient Descent (AdalineSGD) algorithm to predict the secret key. This code has a few differences in comparison with the original code. For example, labels are created based on whether the LSB of the S-box output is 1 or -1. The traces are then split into training and testing sets. AdalineSGD is trained on the training data and evaluated on the testing data. The key guess with the highest accuracy is chosen as the predicted byte value. This process is repeated for all 16 bytes of the AES key. Finally, the recovered key is compared to a predefined actual key to determine the accuracy of the attack.

**Results of Task 2:**
These are the results I got after running code one:

```
Recovered Key: 00 2A 22 33 44 55 66 77 04 99 AA BB B5 91 CC FF
Accuracy: 33.59%
```

As we can see, the standard DPA attack on the AES algorithm does not perform well resulting in 33.59%.

To improve the performance of this program, I decided to use the AdelineSGD algorithm. This assignment seemed to me like a classification problem so I hypothesized that this algorithm would perform well for this task. These are the results I got after running code two:

```
<ipython-input-5-ad5bf5c761da>:62: RuntimeWarning: overflow encountered in double_scalars
  cost = 0.5 * error**2
<ipython-input-5-ad5bf5c761da>:60: RuntimeWarning: invalid value encountered in add
  self.w_[1:] += self.eta * xi.dot(error)
<ipython-input-5-ad5bf5c761da>:61: RuntimeWarning: invalid value encountered in double_scalars
  self.w_[0] += self.eta * error
Recovered Key: 8F 67 60 1E 0C 65 C8 DA BD 6B 05 03 8F 61 DA E8
Accuracy: 50.78%
Timestamp: 2023-10-05 03:32:57
NetID: 527005307
```

As we can see, the DPA attack in combination with AdelineSGD performed better than the standard DPA attack.

**Explaining The Code for Task 3:**
The code executes a DPA attack to decipher an AES encryption key. First, it loads data, including power consumption traces, from the given .mat file. The core of the attack is the dpa_attack function, which, given a byte position and a specific number of power traces, tries to determine the most probable value of that byte in the secret key. The attack distinguishes power traces into two groups based on the LSB of the S-box lookup result after

XORing the plaintext with the guessed key byte. It calculates the DoM between the two groups for each key guess and determines the most probable byte value as the one with the highest absolute DoM value. The code does this for all 16 bytes of the AES key. The complete process is repeated for different numbers of power traces (20, 50, 100, 200) to examine the effect of the number of traces on the attack's accuracy. The final recovered key is compared to a known correct key to compute and display the accuracy of the recovery.

To improve performance, the second code performs a DPA attack on AES encryption using the AdalineSGD machine learning algorithm to recover the secret key. This code has a few differences in comparison with the original code.). For example, it creates a binary label based on the LSB of the S-box output. With the labeled data, the code then splits the traces into training and testing sets. AdalineSGD is trained on the training set and evaluated on the test set. The key value that yields the highest prediction accuracy is chosen as the most likely byte value. This process is repeated for all 16 bytes of the AES key. The key recovery process is performed multiple times, each time considering different numbers of traces (20, 50, 100, 200), to analyze the impact of the number of traces on the accuracy of the key recovery.

**Results of Task 3:**

These are the results I got after running code one:

```
Number of Traces: 20, Accuracy: 49.22%
Recovered Key: 3D 5B 54 FB 34 B4 14 24 04 99 0A 28 0F 91 1B FF
------------------------------------------------
Number of Traces: 50, Accuracy: 51.56%
Recovered Key: 3D 5B 54 FB 34 B4 14 24 04 99 0A 28 0F 91 1B FF
------------------------------------------------
Number of Traces: 100, Accuracy: 53.91%
Recovered Key: 3D 5B 54 FB 34 B4 14 24 04 99 0A 28 0F 91 1B FF
------------------------------------------------
Number of Traces: 200, Accuracy: 33.59%
Recovered Key: 3D 5B 54 FB 34 B4 14 24 04 99 0A 28 0F 91 1B FF
------------------------------------------------
Timestamp: 2023-10-05 19:16:46
NetID: 527005307
```

As we can see, this code performs better with a smaller amount of traces

To improve the performance of this program, I decided to use the AdelineSGD algorithm. This assignment seemed to me like a classification problem so I hypothesized that this algorithm would perform well for this task. These are the results I got after running code two:

```
Number of Traces: 20, Accuracy: 48.44%
Recovered Key: 11 2C 11 0D 09 0F 08 11 07 03 16 0B 06 08 25 0B
------------------------------------------------
Number of Traces: 50, Accuracy: 45.31%
Recovered Key: 0C 10 EA 16 0A 59 03 40 E0 89 46 50 59 31 D3 8C
------------------------------------------------
Number of Traces: 100, Accuracy: 51.56%
Recovered Key: 0D C4 63 FE F3 02 1A 13 55 0A 0A 95 70 D7 BE 2D
------------------------------------------------
Number of Traces: 200, Accuracy: 50.78%
Recovered Key: 8F 67 60 1E 0C 65 C8 DA BD 6B 05 03 8F 61 DA E8
------------------------------------------------
Timestamp: 2023-10-05 20:31:20
NetID: 527005307
```

As we can see, the results have not improved by a lot but stayed in the same range of accuracy in comparison to the standard DPA attack

To further improve my code, I tried using other ML algorithms. My next choice was NNs but since I was running it on the free version of Google Colab my RAM usage was overflown and the code had to be terminated. My next choice was SVMs; results for this code can be seen below:

```
Number of Traces: 20, Recovered Key: 11 31 01 04 06 03 00 13 02 03 13 0B 06 03 1C 08
Accuracy: 54.69%
Number of Traces: 50, Recovered Key: D5 68 EA 16 36 59 14 16 89 89 30 0E 82 50 97 9E
Accuracy: 50.78%
Number of Traces: 100, Recovered Key: 0D C4 42 FE 43 AF 1B 13 70 0A 0F 95 70 24 0C 16
Accuracy: 46.88%
```

As we can see, it had some promising results at the beginning but the accuracy was dropping as the number of traces was increasing. Another issue with this algorithm was the runtime. To output these results, it took over 30 minutes so I decided to terminate the code. This also led to not showing my NetID and the timestamp in the output

I also tried using the Logistic Regression algorithm but after 35 minutes of runtime, it did not give any outputs so I decided to terminate it. Due to the slow and inaccurate performance of LR, SVMs, and NNs, I decided not to include those codes in my final submission

**Conclusion:**

As a conclusion, the results turned out to be not as good as anticipated. I am sure I could have improved the AdelineSGD algorithm if I had tested different learning rates and the number of iterations but due to time restraints I had to keep these results for my final submission. Overall, it was a great experience performing a DPA attack on the AES algorithm. I have learned a lot more about how it works and was able to practice more using ML algorithms