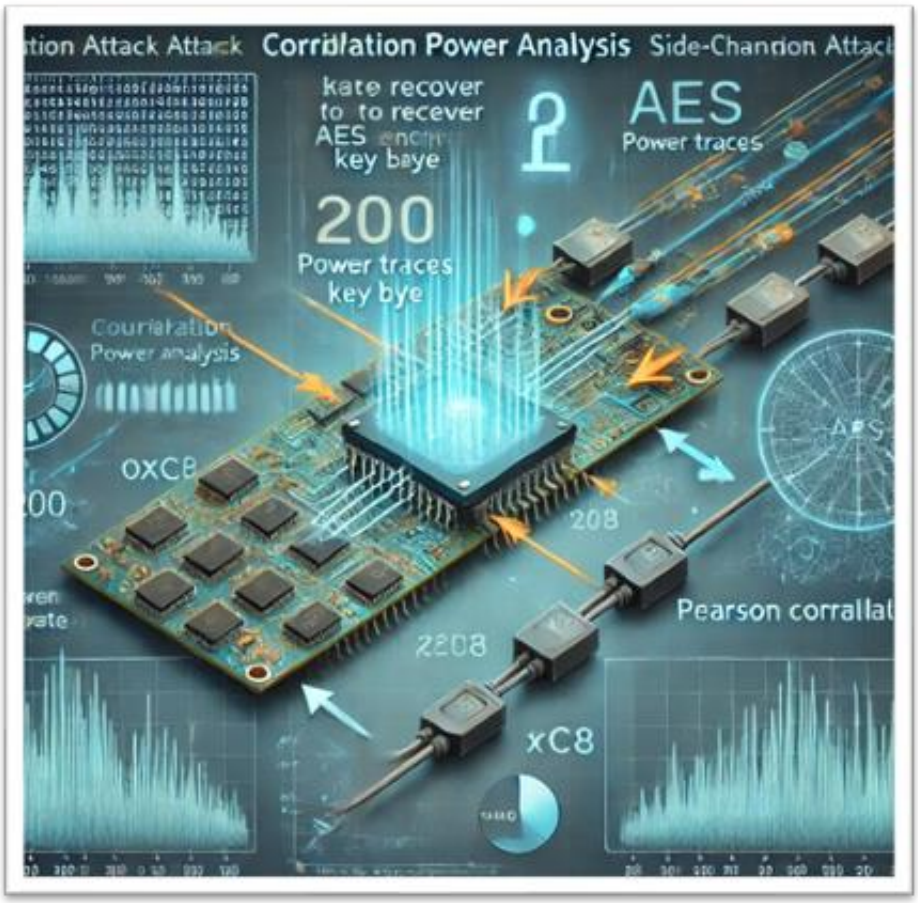


Understanding CPA Attacks: Recovering AES Key Byte K11 from Power Traces

CS984- ASSIGMENT-1

Student ID	Name	Email
233560019	Mohammed Jawed	mjawed23@iitk.ac.in
233560020	Mohit Paritosh	pmohit23@iitk.ac.in
233560021	Mohit Srivastava	mohitsri23@iitk.ac.in
233560023	Nikita Shetty	narayans23@iitk.ac.in
233560025	Ajit Madhusudan Paranjape	paranjape23@iitk.ac.in

Recovering AES Key Byte K11 from Power Traces

Contents

Recovering AES Key Byte K11 from Power Traces

Abstract

This report focuses on the application of Correlation Power Analysis (CPA) to recover a specific key byte (K11) from an AES hardware implementation. The attack leverages side-channel information obtained from power traces collected during the encryption process. The goal is to analyze these traces to extract the secret key used in the AES encryption.

The report begins by outlining the problem, where Daniel, a security engineer, is tasked with performing a side-channel analysis on an AES implementation using power traces stored in a CSV file. The task is to recover a specific key byte (K11) of the 128-bit AES key by examining the power consumption model based on Hamming Weight. This method assumes that power leakages correlate linearly with the number of '1' bits in the binary representation of a target value.

The solution involves several steps:

1. Reading the power traces and associated plaintexts and ciphertexts from the provided CSV file.
2. Modeling power consumption using the Hamming Weight of the intermediate values during encryption.
3. Calculating the Pearson correlation coefficient between the observed power traces and the hypothesized Hamming Weights for different key guesses (0x00 to 0xFF).
4. Determining the correct key guess that shows the highest correlation with the power traces, indicating the correct value for K11.

Upon applying the CPA attack with these steps using python script, the report successfully identifies the 11th byte of the AES key (K11) as **200 (0xC8)**. The Python code provided in the report demonstrates the complete process of key recovery, including data reading, power modeling, correlation computation, and result visualization, leading to the discovery of the key byte.

This finding illustrates the effectiveness of CPA in breaking AES encryption by exploiting side-channel leakage, emphasizing the need for robust countermeasures against such attacks in cryptographic hardware implementations.

Recovering AES Key Byte K11 from Power Traces

1. Problem Statement

Daniel is a security engineer, and he has got a project for side-channel analysis of an AES hardware implementation. He has already collected power traces for that AES implementation with a 128-bit key 'K' and has stored the traces in the "traces_AES.csv" file. The first column in the csv file indicates the plaintext, the second column indicates the ciphertext and the rest of the columns indicate the sample points. Now Daniel has to analyze the power traces and will have to find the AES key using Correlation Power Analysis (CPA). Help Daniel to perform the CPA attack.

Group 6 has to recover key Byte 11.

2. Recover Key Byte

2.1 ANALYSIS

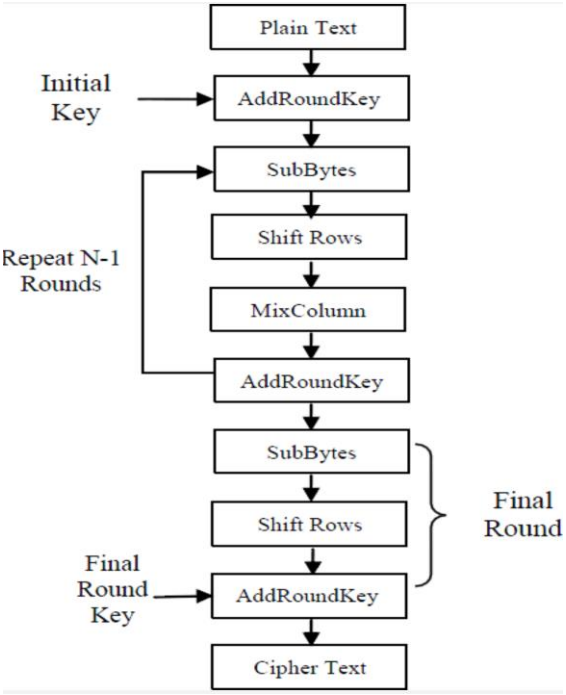
The problem statement says we need to use CPA for the Side Channel Attack carried out by monitoring the power consumption of the AES Hardware implementation.

For the Analysis we are provided with power output measurements with many samples each taken during the encryption process of different plaintexts and also corresponding ciphertexts.

Since we are dealing with 128 bit = 16 bytes AES secret key (10 rounds), for each byte of secret key we have 256 possible guesses (0x00-0xFF).

Ideally, we go over all the bytes of the key, indexed from 0 to 15 and for each byte we guess all the possible values, indexed 0 to 255. However, here we need to recover only key byte 11, so we guess all 256 possible values for key byte 11 only.

Below is an overview of steps performed in AES 128 bit.



Add Round Key	XORs each byte of the buffer with corresponding byte of the key and places it back to the buffer
Inv Shift Rounds	Scrambles the byte locations around
Inv Sub Bytes	It is a lookup table comprised of specific constant values of 256 bytes
Inv Mix Columns	Jumbles the columns

Hamming Weight Model

The chosen power consumption model in our case is Hamming Weight.

Hamming Weight is a leakage model and it is assumed that power leakages vary linearly with the number of 1's in the binary representation of the target value.

Correlation Coefficient

We have two datasets:

- Traces associated to the power consumption of AES operations of a set of plaintexts and the correct key
- The Hamming Weights of the given plaintexts xored with a key guess

To determine if there is a relationship and the strength of this relationship between these two datasets we'll use the Pearson's correlation coefficient. The Pearson's correlation can be expressed with the following formula:

$$\rho(X,Y) = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}.$$

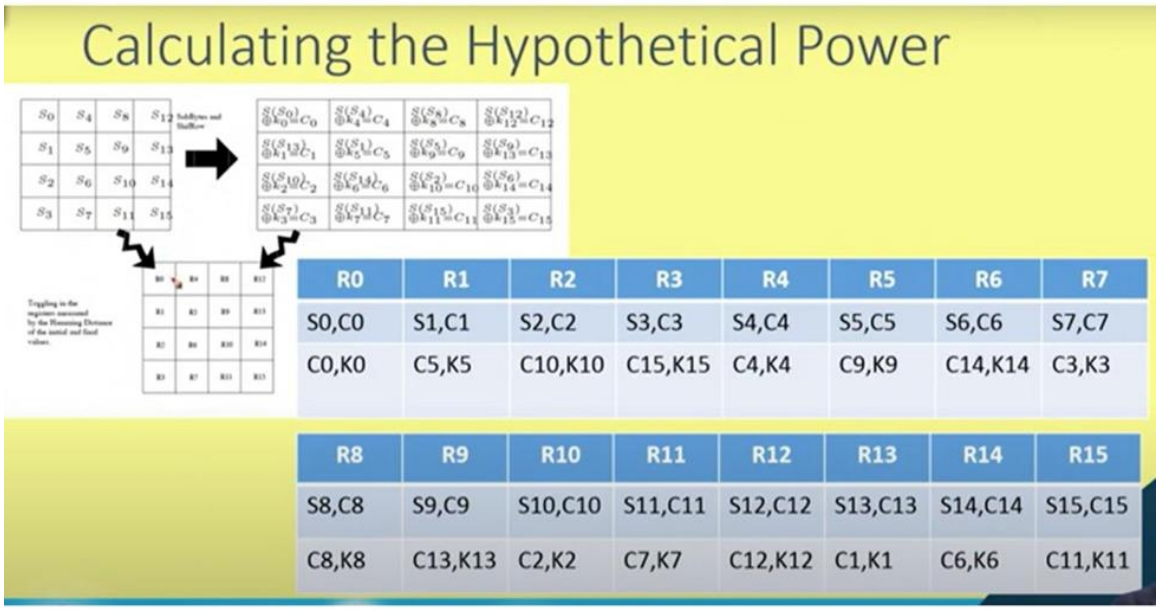
Where y stands for traces and x for hypothesis, respectively x bar and y bar are the values of the hypothesis mean and trace's mean.

The Pearson's coefficient will always return a value between -1 and 1 and stronger the association of the two variables is, closer the Pearson correlation coefficient will be to either 1 or -1, respectively closer the output of the formula is to 0, stronger will be the variation between the two variables.

2.2 SOLUTION

Explanation for Finding the 11th Key Byte (K11):

Our task was to determine the 11th key byte (K11) as part of the side-channel analysis assignment. To achieve this, we mainly referred to the "Power Analysis (Part VII)" YouTube lecture series by Prof. Debdeep Mukhopadhyay, which provided a clear direction for tackling this assignment.



Recovering AES Key Byte K11 from Power Traces

For finding the 11th key byte, we use the Hamming Weight model between two values, C11 and S11. The value of C11 is directly available from the provided power trace data. To determine S11, we analyze the register position and utilize the relationship where:

$$S_{11} = Sbox^{-1}(C_7 \oplus K_7)$$

where SboxInvers denotes the inverse of the AES S-box, C7 is another ciphertext byte, and K7 is the corresponding key byte.

With this relationship, we can express the equation for the Hamming Distance as:

$$HD(C_{11}, S_{11}) = HD(C_{11}, Sbox^{-1}(C_7 \oplus K_{guess}))$$

where K_{guess} is a guessed value of the key byte ranging from 0 to 255.

where Kguess is a guessed value of the key byte ranging from 0 to 255.

Using this approach, we compute the Hamming Distance for each possible key guess and compare it against the power traces to find the guess with the highest correlation. This allows us to determine the correct value for K11.

Below is the Python code (**Assignment1_G6.py**) that implements this approach to find the correct key guess for the 11th byte:

Steps for CPA attack

1. Read the csv file provided which has stored plaintext, cipher text and power traces

```
# Load the data
df = pd.read_csv('traces_AES.csv')
data = df.to_numpy()

# Extract traces and ciphertexts
try:
    traces = df.iloc[:,2:].values
except ValueError as e:
    raise ValueError("Error converting traces to float. Check the format of the input data.") from e

try:
    ciphertexts = df.iloc[:,1].values
except IndexError as e:
    raise IndexError("Error extracting ciphertexts. Check if the expected column exists.") from e
```

2. Choose a model of power consumption, in our case Hamming Weight

```
# Function to compute the Hamming weight of a given value.
def HW(value):
    return bin(value).count('1')
```

3. For each of the key's bytes, apply the power consumption model. For our assignment, compute the Hamming Weight of the plaintext's each byte and the possible key's bytes hypothesis.

```
# Perform CPA computations
for k in range(numKeyBytes):
    for j in range(numTraces):
        try:
            # Extracting the relevant byte and applying the key byte guess
            c11 = int(ciphertexts[j][22:24], 16) ^ k  #for C11
            x1 = int(ciphertexts[j][14:16], 16) # need to target C7
            x2 = InvSbox[c11]
            xor1 = x1 ^ x2
            hwValue = HW(xor1)
            hypotheticalModel[j][k] = hwValue
        except ValueError:
            hypotheticalModel[j][k] = np.nan
```

4. For each key byte, apply the Pearson's correlation formula on the obtained hypothesis and the recorded actual power consumption.

```
# Calculate correlation matrix
for i in range(numKeyBytes):
    for j in range(numPoints):
        c1 = hypotheticalModel[:, i]
        c2 = traces[:, j]
        try:
            corr, _ = pearsonr(c1, c2)
            correlationMatrix[i, j] = abs(corr)
        except ValueError:
            correlationMatrix[i, j] = np.nan
```

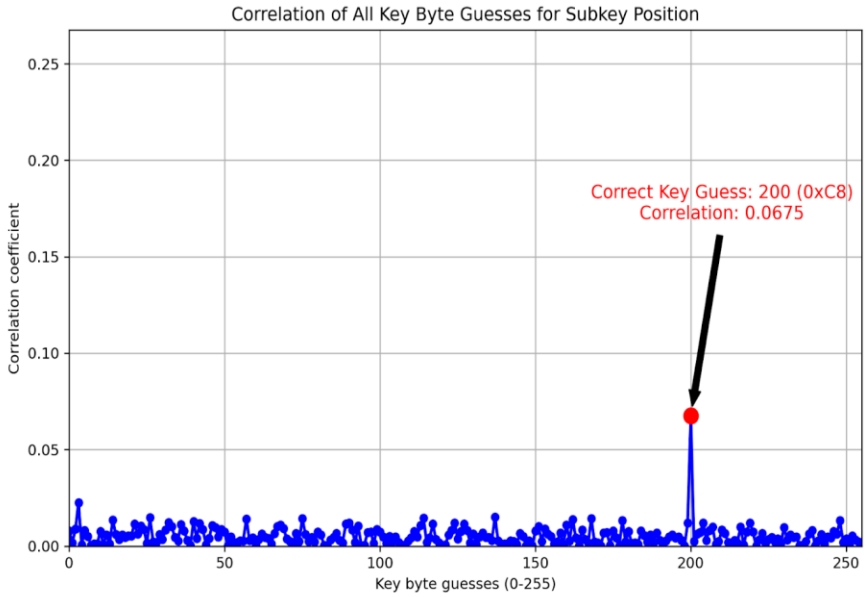
5. Decide which subkey guess correlates best to the measured traces according to Pearson's output interpretation.

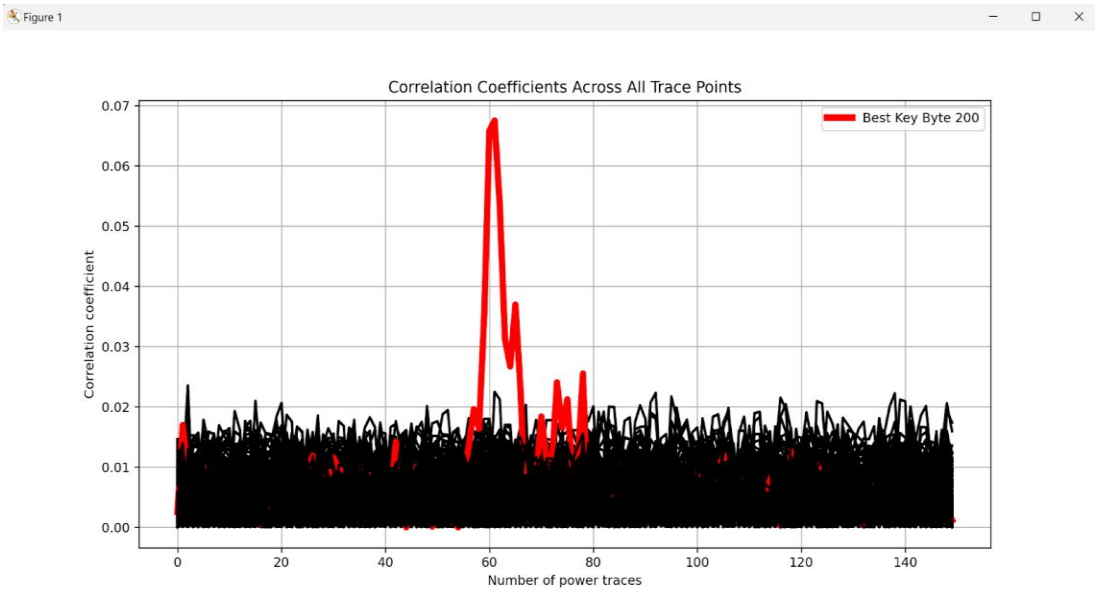
```
bestKeyByte, bestTracePoint = unravel_index(correlationMatrix.argmax(), correlationMatrix.shape)

print(f"Best key byte guess position: {bestKeyByte}")
print(f"Best trace point position: {bestTracePoint}")
print(f"The correct key byte is {bestKeyByte}, hex value = {hex(bestKeyByte)}")
```

6. Plot graphs for visualization of above key recovery.

```
# Create a new plot & Highlight the correct key byte guess
plt.figure(figsize=(10, 6))
plt.plot(range(numKeyBytes), correlationMatrix[:, bestTracePoint], 'bo-', linewidth=2, markersize=5)
plt.plot(bestKeyByte, correlationMatrix[bestKeyByte, bestTracePoint], 'ro', markersize=10)

# Plot the correlation values for each subkey guess (0-255)
plt.ylabel('Correlation coefficient')
plt.xlabel('Key byte guesses (0-255)')
plt.title('Correlation of All Key Byte Guesses for Subkey Position')

# Annotate the correct key guess with its value and correlation coefficient
plt.annotate(
    f'Correct Key Guess: {bestKeyByte} (0x{bestKeyByte:02X})\nCorrelation: {correlationMatrix[bestKeyByte, bestTracePoint]:.4f}',
    xy=(bestKeyByte, correlationMatrix[bestKeyByte, bestTracePoint]),
    xytext=(bestKeyByte + 10, correlationMatrix[bestKeyByte, bestTracePoint] + 0.1),
    arrowprops=dict(facecolor='black', shrink=0.05, headwidth=8, headlength=10),
    fontsize=12,
    color='red',
    ha='center',
    va='bottom'
)

plt.xlim(0, numKeyBytes - 1)
plt.ylim(0, np.max(correlationMatrix) + 0.2)

# Display the plot
plt.grid(True)
plt.show()

# Plot correlation coefficients across all trace points for each key byte
plt.figure(figsize=(12, 6))
for i in range(numKeyBytes):
    if i == bestKeyByte:
        plt.plot(range(numPoints), correlationMatrix[i, :], 'r', linewidth=5, label=f'Best Key Byte {i}')
    else:
        plt.plot(range(numPoints), correlationMatrix[i, :], 'k', linewidth=2)
plt.ylabel('Correlation coefficient')
plt.xlabel('Number of power traces')
plt.title('Correlation Coefficients Across All Trace Points')
plt.legend()
plt.grid(True)
plt.show()
```

2.3 GRAPHICAL OUTPUT






```
Best key byte guess position: 200
Best trace point position: 61
The correct key byte is 200, hex value = 0xc8
>>>
```

5.References

1. https://gethypoxic.com/blogs/technical/a-practical-guide-for-cracking-aes-128-encrypted-firmware-updates?srsltid=AfmBOoqaBlkFxDYWdvO37xLeikD_NQzkatqjGMEOxguLj732kkkOUgdd
2. https://pkiluke.medium.com/recovering-aes-key-from-side-channel-analysis-bd832bf8e2b7
3. https://yan1x0s.medium.com/side-channel-attacks-part-2-dpa-cpa-applied-on-aes-attack-66baa356f03f
4. Lecture 31: Power Analysis (part – VII) You-Tube lecture series by Prof Debdeep Mukhopadhya
5. A Comparative Study of Power Consumption Models for CPA Attack ,Hassen Mestiri, Noura Benhadjyoussef, Mohsen Machhout and Rached Tourki, https://www.mecs-press.org/ijcnis/ijcnis-v5-n3/IJCNIS-V5-N3-3.pdf