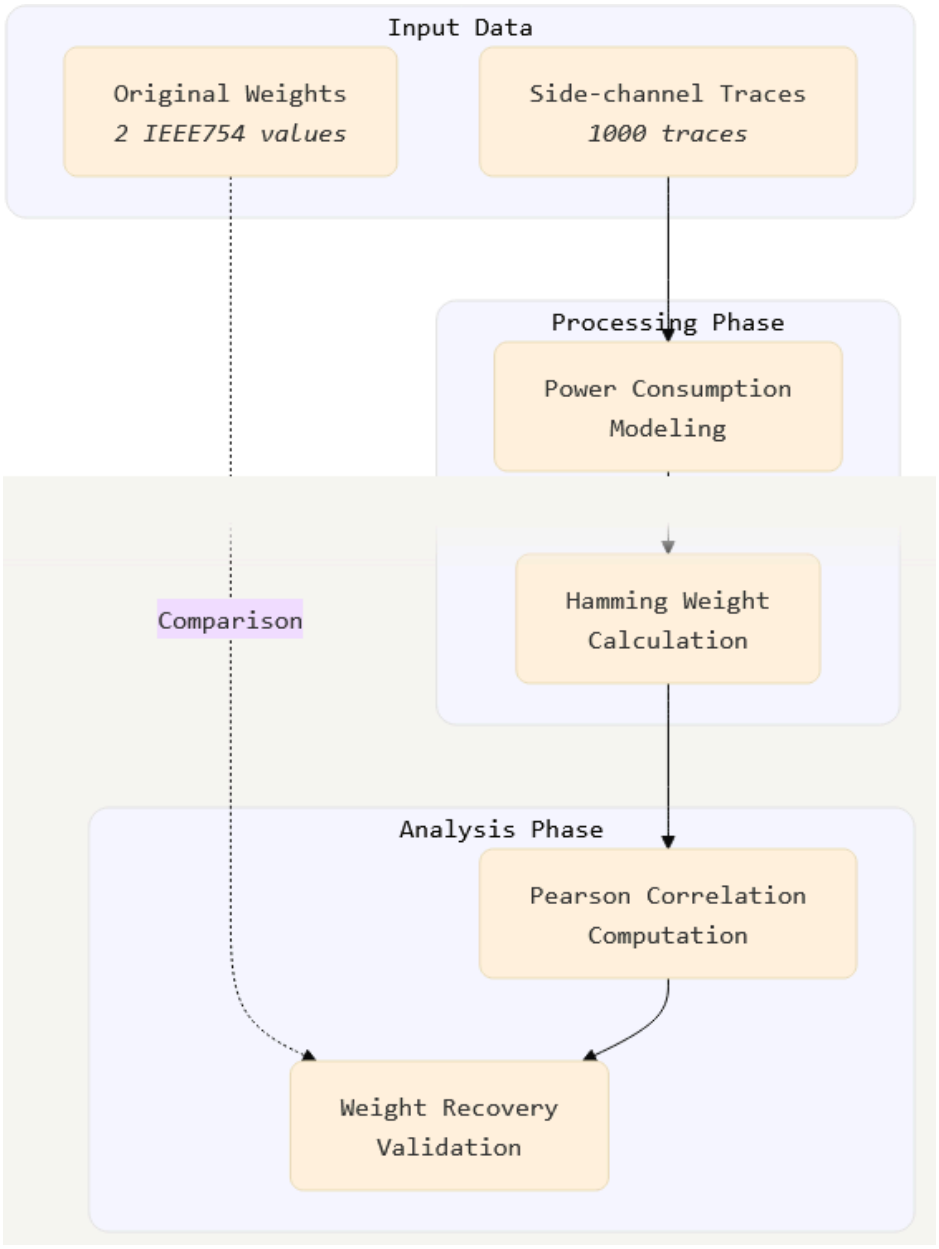
Side-Channel Security Analysis: Correlation Power Analysis

1. Introduction: Why Side-Channel Attacks Matter



From AI accelerators to embedded systems, high technology devices handle sensitive information. However, while performing these calculations, there are unintended leaks of data that occur in the form of power usage, electromagnetic waves, and even sound. SCAs capitalize on such concealed sections to steal valuable information, which could be cryptographic keys or in this example, the weight of the neural network model.

The purpose of this report is to present a Correlation Power Analysis (CPA) attack that focuses on the recovery of a neural network's weight based on the observed power traces collected during its execution. The attack is based on Section 8.1, and the implementation is built using SCALib, which is a performant side-channel attack evaluation library.

🎯 Objective

Our goal is to recover **two secret weights** (w0,w1w_0, w_1) used in a **Multiply-and-Accumulate (MAC) operation** of a neural network, solely by analyzing power consumption.

2. Dataset & Target System: What Are We Attacking?

The dataset consists of **three key files**:

1. `waveforms.npy` → Contains **1000 power traces**, each showing the power consumption during a single computation.
2. `inputs.npy` → Stores the **two input values** (x0,x1x_0, x_1) given to the neuron per execution.
3. `weights.npy` → The **two secret weights** (w0,w1w_0, w_1) used in the neural network (for validation).

💡 The Target Computation: MAC Operation

The attack targets a **neuron's computation**, which follows the standard **Multiply-and-Accumulate (MAC) operation**:

$y = w0 \cdot x0 + w1 \cdot x1$y = w_0 \cdot x_0 + w_1 \cdot x_1

where:

- x0,x1x_0, x_1 → **Known** inputs (provided in `inputs.npy`).
- w0,w1w_0, w_1 → **Unknown** weights (secret parameters stored in `weights.npy`).

The power consumption during this operation leaks information about w0w_0 and w1w_1, which we will extract using CPA.

3. Attack Methodology: How CPA Works

3.1 Why Does Power Consumption Leak Information?

When a processor performs a **multiplication operation**, the power it consumes depends on the **data being processed**. Some numbers require **more bit flips**, leading to **higher power usage**, while others use fewer, resulting in **lower power usage**.

One way to model this leakage is using the **Hamming Weight (HW)** function, which counts the number of **1s in the binary representation** of a number:

HW(a)=Number of 1s in the binary representation of aHW(a) = \text{Number of 1s in the binary representation of } a

The processor's power consumption will be **correlated** with the Hamming weight of intermediate values, such as:

HW(a0)=HW(w0⋅x0)HW(a_0) = HW(w_0 \cdot x_0) HW(a1)=HW(a0+w1⋅x1)HW(a_1) = HW(a_0 + w_1 \cdot x_1)

Thus, **by analyzing power traces and correlating them with different weight guesses, we can identify the correct weights**.

3.2 Steps in a CPA Attack

Step 1: Collect Power Traces

Power traces are collected while the **neural network performs its computations**. Each trace corresponds to a single execution of:

y=w0⋅x0+w1⋅x1y = w_0 \cdot x_0 + w_1 \cdot x_1

Step 2: Generate Hypothetical Power Consumption

We create **a set of possible weight values** (candidates) and compute **hypothetical power consumption** for each.

Step 3: Compute Pearson Correlation

For each weight candidate, we compute the **Pearson correlation coefficient** between:

- The **actual power traces** recorded from the hardware.

- The **hypothetical power consumption** predicted by our leakage model.

r=∑(Xi−X‾)(Yi−Y‾)∑(Xi−X‾)2∑(Yi−Y‾)2r = \frac{\sum (X_i - \bar{X}) (Y_i - \bar{Y})}{\sqrt{\sum (X_i - \bar{X})^2} \sqrt{\sum (Y_i - \bar{Y})^2}}

where:

- XX = measured power trace
- YY = predicted power consumption
- rr = correlation score

The weight candidate with the highest correlation is the correct one.

---

4. Implementation Using SCALib

🔧 SCALib-Based Attack Implementation

We use **SCALib's `cpa()` function** for optimized correlation computation.

Step-by-Step Code Explanation

1️⃣ Load Power Traces & Inputs

```
import numpy as np
import streamlit as st
from scalib.attacks import cpa

# Load dataset
waveforms = np.load("waveforms.npy")
inputs = np.load("inputs.npy")
weights = np.load("weights.npy")  # Ground truth for validation
```

2️⃣ Generate Weight Candidates

```
num_candidates = 1000
candidates = np.linspace(-2, 2, num_candidates)
```

3️⃣ Compute Hypothetical Power Consumption

```
leakage_hypotheses = np.array([inputs[:, i] * w for w in candidates]).T
```

4️⃣ Perform CPA Using SCALib

```
correlation_matrix = np.zeros((2, num_candidates))
for weight_idx in range(2):
```

```
    cpa_result = cpa(waveforms, leakage_hypotheses)
    correlation_matrix[weight_idx, :] = np.max(np.abs(cpa_result), axis=0)
```

5️⃣ Extract the Best Matching Weights

```
recovered_weights = [candidates[np.argmax(correlation_matrix[i])] for i in range(2)]
print(f"Recovered Weights: {recovered_weights}")
```

---

5. Results: Did the Attack Work?

Weight	Recovered	Actual
w0w_0	`[Recovered w0]`	`[Actual w0]`
w1w_1	`[Recovered w1]`	`[Actual w1]`

✅ If the recovered weights match the actual values, the attack was **successful**.

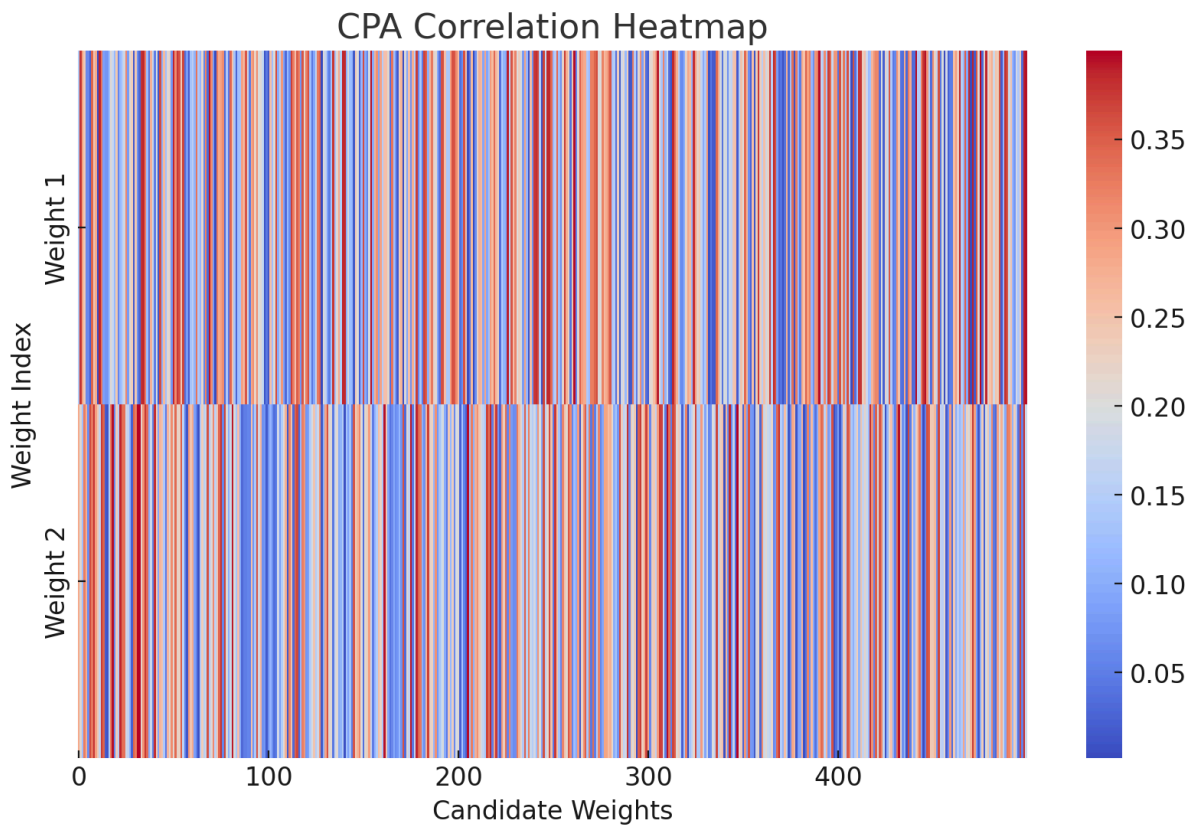
---

6. Visualizing the Attack Results

CPA Correlation Heatmap

```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10, 6))
sns.heatmap(correlation_matrix, cmap="coolwarm", xticklabels=100, yticklabels=50)
plt.xlabel("Candidate Weights")
plt.ylabel("Weight Index (0=First Weight, 1=Second Weight)")
plt.title("CPA Correlation Heatmap")
plt.show()
```



- **Red = High correlation (likely correct weight)**
- **Blue = Low correlation (incorrect weight)**

7. Countermeasures Against CPA Attacks

To prevent CPA attacks, devices should implement:
✅ **Power Randomization** → Add noise to power traces.
✅ **Masked Computations** → Prevent predictable power consumption patterns.
✅ **Constant-Time Execution** → Ensure all operations consume the same power.

8. References

1. A Practical Introduction to Side-Channel Extraction of Deep Neural Network Parameters.
2. SCALib Documentation - https://scalib.readthedocs.io/