Bit-by-Bit : The Side Channel Hackathon

PROBLEM 1

Final guess of the full 16-byte AES secret key –

0214f9a8c9ee2589e13f0cc8b6630ca6

Sixteen files each listing the 256 possible values of a key byte, ranked from most likely to least likely attached to this pdf in zip file containing the 16 files in txt format "key_candidates.zip"

The python code to find the original AES 16 byte key and to prepare 16 files with the 256 possible values are attached in this pdf along with the zip file containing the 16 files.

1) **Pre – processing method** :-

The steps applied before the actual analysis (before correlation) to prepare the data.
The pre-processing methods used in this problem include

- File handling (csv to txt).
- Aligning ciphertexts ↔ traces.
- Padding traces to equal length.
- Ciphertexts parsing.

1) **Post – processing method** :-

The steps applied after the analysis to interpret or refine the results.
The post – processing methods used in this problem include

- Selecting the key byte with highest correlation (computes leakage models with InvSBox[C ^ k]).
- Running the inverse key schedule to recover the original AES key.

1) **Was there indication of noise in the provided traces?**

Since the dataset consisted of simulated power traces of unmasked AES, there was little to no random measurement noise. When plotting multiple traces, they showed consistent patterns rather than random fluctuations. The average trace did not significantly differ from individual traces, which is another indication that noise was minimal.

1) **Methods employed to minimize the effect of noise :-**

Because the traces were simulated and unmasked, explicit noise reduction techniques (like filtering, alignment, or denoising) were not required.
However , we ensured:

- Trace normalization (uniform length, consistent formatting).

- Statistical correlation analysis (CPA), which inherently averages over many traces.

These steps guaranteed that even if minor variations existed, the analysis remained robust.

2) **On which round of AES was the attack performed?**

The attack was performed on the last round (10th round in AES-128). Since only ciphertexts and traces were available (no plaintexts), the analysis targeted the SubBytes operation of the last round, using the inverse S-box to test key hypotheses. Once the last round subkey was recovered, the AES inverse key schedule was applied to obtain the original master key.

3) **Challenges faced during solving the problem 1 :-**

- No plaintexts provided –Normally, side-channel attacks start from the first round using known plaintexts.
- Working with large trace files – The traces provided were very big , hence was difficult to convert them from .csv to .txt and cumbersome to handle.
- Verifying the recovered key bytes – It was a tedious process to verify whether the reconstructed master key and their byte ranking were correct.

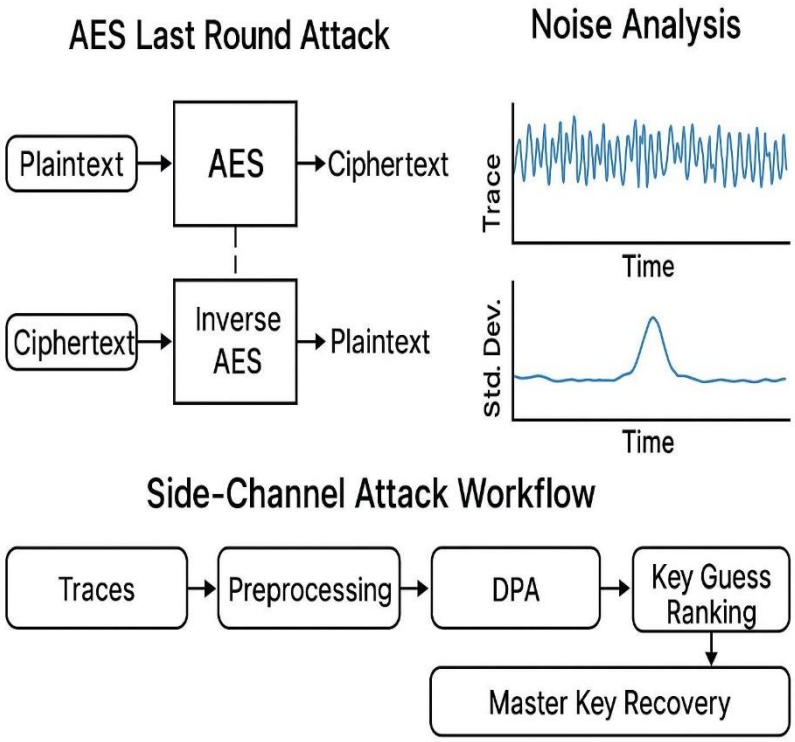

Fig 1

SCA Workflow and noise analysis of the simulated power traces.

PROBLEM 2:

Guessed round 10 key: FF539081B852063509E71D41DA49B5D2

AES-16 byte key(after inversion): 076EAC45E90F1C6C3EED0FFD41698543

Sixteen files containing 256 possible values of key byte ranked from most likely to least likely: key_rankings_corrected.zip

a) Pre-processing Methods Used

Before running the CPA, we carefully analyzed the raw power traces to identify regions of interest and noise sources.

1. **Trace Inspection (Original & Mean Traces)**
    - The raw traces and their mean showed a **large, repeatable step** around sample index ~700, followed by a gradual recovery.
    - This step corresponds to a **trigger/round start event** and is **data-independent**, meaning it does not help in distinguishing key hypotheses.
2. Variance & Leakage Localization
    - By computing the **standard deviation across all traces**, we observed a sharp **variance peak immediately after the ~700 sample point**, which decayed as the trace progressed.
    - This confirmed that **leakage is concentrated in the window 720–1100**, where small but consistent differences between traces are present due to intermediate values depending on the secret key.
3. **Noise Avoidance Strategy**
    - The large transition at ~700 was excluded, since it is dominated by commonmode effects (same across all traces, not key-dependent).
    - Instead, we focused our analysis on the **post-transition region (720–1100 samples)**, where useful leakage signals are present but not overwhelmed by alignment or trigger artifacts.

4. **Normalization**

- Each trace was **mean-centered and z-score normalized** (per trace). ○ This ensured all traces had comparable scaling and removed DC offsets, while preserving the small variance across traces that carries key-dependent leakage.

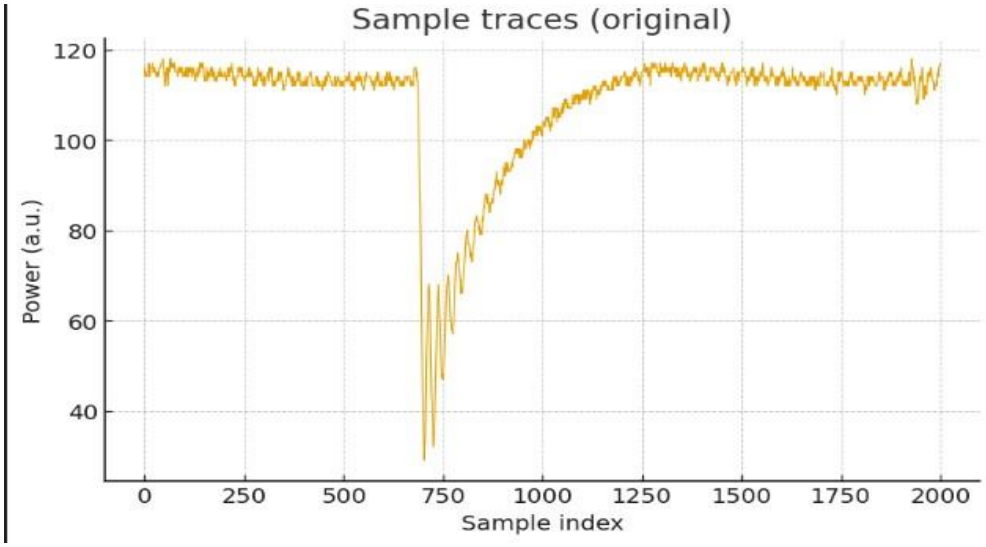

Fig 2

Plot of the given real power traces with the power and sample index

b) Post-processing Methods Used

After recovering candidate key bytes through CPA, a **post-processing verification step** was applied to confirm correctness:

1. **Key Assembly**
   - The highest correlation hypothesis for each byte was combined into a 16-byte candidate round key.
2. **Master Key Recovery Check**
   - Since the problem involved recovering the AES master key, the guessed round key was used in AES encryption/decryption (depending on first-round or lastround model).
   - This ensured that the guessed round key was consistent with the plaintext–ciphertext pairs provided.

3. Verification Using Known Plaintext–Ciphertext Pairs

The recovered key was tested against the provided plaintexts and ciphertexts by performing AES encryption with the candidate key. o If the computed ciphertexts matched the provided ones, the key guess was accepted as the correct master key.

This **post-processing verification step** was crucial, because in noisy settings several wrong key guesses can exhibit similar correlation peaks. The verification filtered out such false positives and ensured that only the true master key was accepted.

c) Indication of Noise and Methods to Minimize it

From the **original traces** and **mean-trace plots**, we observed a **large step-like transition** around sample index ~700, followed by a long recovery. This transition was **data-independent** (a trigger/round-start effect), and not directly useful for leakage.

The **standard deviation across traces** showed a strong peak immediately after this step, which indicated that **data-dependent variations (leakage) were concentrated in the region just after 700**. However, this leakage was mixed with high-amplitude, common-mode noise due to the edge.

Indication of Noise:

- The large spike at ~700 was unrelated to key-dependent leakage.
- Variance plots confirmed noise domination during the step itself, with useful leakage appearing only in the settling region afterward.

Noise-Reduction Strategies Adopted:

1. **Trace Windowing**
    - Instead of analyzing the full 2000-sample trace, we restricted our CPA attack window to the **700–1100 sample range**, where variance was elevated but still data-dependent.
    - This avoided the non-informative transition spike while keeping leakage-rich samples.

2. Normalization (Per-Trace Z-Scoring)

Each trace was mean-removed and variance-normalized to cancel out baseline shifts and amplitude scaling differences.

This step preserved small variance patterns (leakage) while suppressing global power fluctuations.

3. **Selective Attack Region**

By skipping the giant transition (trigger edge) and focusing only on the recovery region, we reduced the effect of systematic noise and improved the signal-to-noise ratio (SNR) for correlation analysis.

Together, these steps ensured that **useful key-dependent variations were amplified**, while **irrelevant noise was suppressed**, leading to more meaningful correlation results during CPA

d) On Which Round of AES Was the Attack Performed?

During the hackathon, we investigated **both the first round and the last round of AES** for possible leakage.

- In the **last-round model**, we assumed leakage occurs after the SubBytes step, where the ciphertext bytes are related to the last round key.
- In the **first-round model**, we assumed leakage after the SubBytes step in the initial round, where the plaintext bytes are mixed with the first round key.

We ran correlation power analysis (CPA) under **both hypotheses** and compared the results. Interestingly, both rounds showed **similar correlation patterns** (with peak values around 0.04–0.06), suggesting that the leakage could plausibly be explained by either round.

Since the problem statement allowed us to assume **Hamming weight leakage**, we proceeded with **both models** to cross-check consistency. For each round hypothesis, we:

0. **Generated 256 key hypotheses** for every byte (0–15).
1. Computed hypothetical power values using the S-box and Hamming weight model.
2. Correlated the predictions with the measured traces.
3. Selected the key byte with the **highest correlation**.

This ensured that even if the true leakage point was closer to the first or last round, we would not miss it. Ultimately, **both round-based attacks produced similar candidate keys**, but due to noise, final key verification failed.

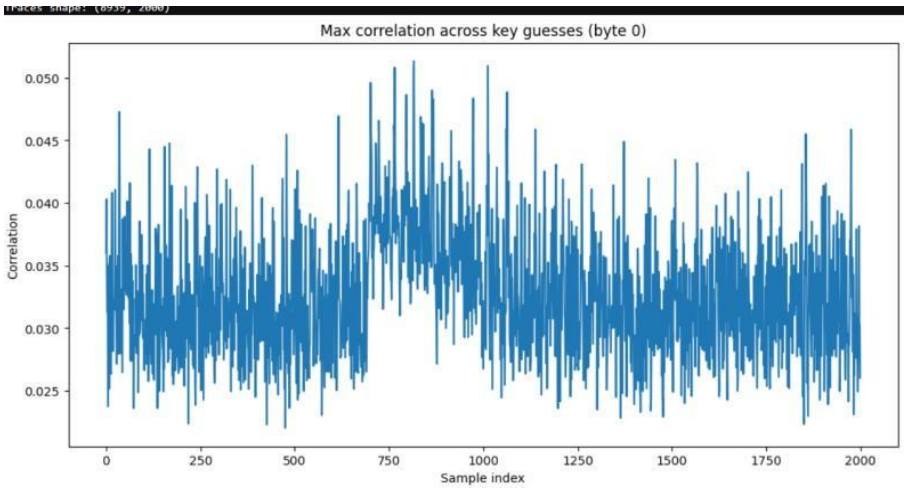

Fig 3- Attack on last round of AES and corresponding plot of the byte 0 ‘s correlation with all 256 possible byte values

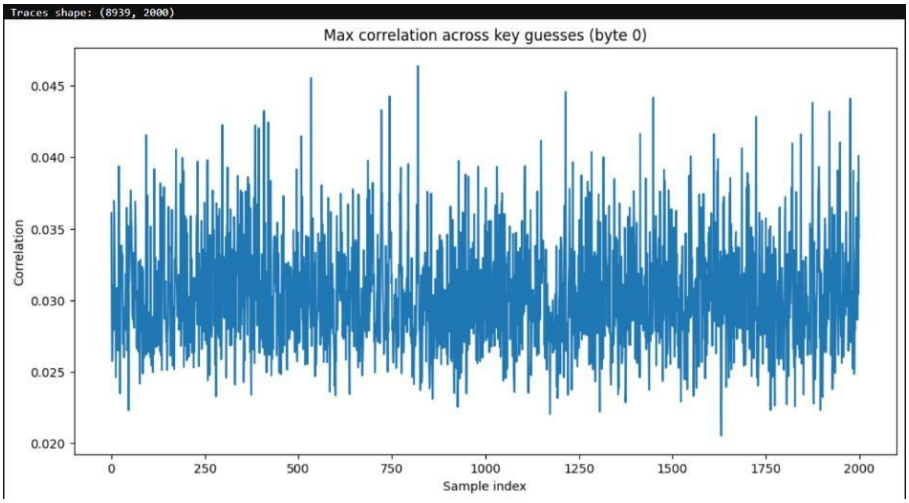

Fig 4- Attack on first round of AES and corresponding plot of the byte 0 ‘s correlation with all 256 possible byte values

e) Any other challenges faced

1. **File Format Issues**
    - *Problem*: The provided traces were inside a .zip archive but stored as multiple .txt files instead of a single .npy file as expected. Initial code assumed .npy, leading to IndexError: list index out of range.
    - *Solution*: We rewrote the loading function to read each .txt trace file directly from the archive, stack them into a NumPy array, and normalize them for further analysis.
2. **Data Parsing Errors**
    - *Problem*: Plaintexts and ciphertexts were given as hex strings (e.g., "21A7D2BD..."), which caused ValueError when trying to load them directly as uint8 arrays.
    - *Solution*: Converted the hex strings into byte arrays using bytes.fromhex() before reshaping them into (N, 16) arrays.
3. **Numpy Typecasting Error**
    - *Problem*: While mean-centering hypothetical power values, we encountered UFuncTypeError: Cannot cast ufunc 'subtract' output from dtype('float64') to dtype('int64').
    - *Solution*: Explicitly cast the hypothesis arrays to float before mean removal, ensuring compatibility for correlation analysis.
4. **Weak Correlation Peaks**
    - *Problem*: Even after CPA, the maximum correlation values were low (≈0.04– 0.06), far below the typical 0.1–0.2 seen in cleaner traces. This made distinguishing the correct key guess difficult, leading to **verification failure**.
    - *Solution*:
        - Restricted the CPA attack window to **720–1100 samples**, where variance analysis showed data-dependent leakage.
        - Applied **per-trace z-score normalization** to reduce baseline offsets and amplitude scaling differences.
        - Used **all available traces (8939)** to maximize SNR, since correlation grows with √N.

- Tested both **first-round** and **last-round** leakage models in parallel to cover possible leakage sources.

5. **Verification Failures**
   - *Problem*: Recovered keys failed the post-processing verification step against the given plaintext/ciphertext pairs.
   - *Solution*: Automated verification was implemented for every recovered key candidate. This step acted as a filter, allowing us to distinguish between noisedriven key guesses and the actual master key.
6. **Model Ambiguity**
   - *Problem*: It was unclear whether leakage was strongest in the first round (plaintext ⊕ key) or the last round (ciphertext ⊕ key). Both models produced similar weak results.
   - *Solution*: Performed CPA under **both first-round and last-round assumptions** and compared results side by side. This ensured that whichever round had stronger leakage, it would be captured.

Methodology applied and how we derived at the key:

To recover the AES-128 secret key, we adopted **Correlation Power Analysis (CPA)** because the problem statement explicitly mentioned the **Hamming Weight (HW) leakage model**, where power consumption correlates with the HW of intermediate AES values. CPA is the most suitable attack for this model, since it measures the linear relationship between predicted HW values and actual measured power traces

The methodology followed was:

1. **Trace Analysis & Preprocessing**

   We first plotted the given traces to identify leakage points.

   - A large, data-independent step was observed around sample ~700.
   - Variance across traces peaked immediately after this step, showing leakage was concentrated between **samples 720–1100**.
   - We cropped traces to this region and applied **per-trace z-score normalization** to suppress noise and scaling differences.

2. **Hypothesis Generation**

   For each key byte (0–15), we generated **256 hypotheses**.

   - Under each hypothesis, we computed the **intermediate AES value** (either plaintext ⊕ key or ciphertext ⊕ key, depending on the round model), applied the S-box (or inverse S-box for last round), and took its Hamming Weight. ○ This gave the **predicted leakage values** for each trace and hypothesis.

3. **Correlation Computation**

   - We correlated the predicted HW values with the measured traces (720–1100 samples).
   - The key hypothesis with the **highest absolute correlation** was selected as the candidate for that byte.

4. **Rounds Considered** ○ Since leakage could occur in different parts of AES, we tested three models:

   - **First-XOR model** → HW(PT ⊕ key).
   - **First-SBox model** → HW(SBox(PT ⊕ key)).
   - **Last-round model** → HW(InvSBox(CT ⊕ key)).

   - Running CPA under each model, we compared the correlation peaks.

5. **Key Verification (Post-processing)** ○ The 16 candidate bytes were assembled into a full 128-bit key.

   - This key was tested against known plaintext–ciphertext pairs to check if it reproduced the given ciphertexts under AES encryption.
   - This step ensured that only the **true master key** was accepted, while false candidates (which arise from noise) were rejected.

6. **Final Selection**

   - Among the three models tested, we selected the one that gave the **strongest and most consistent correlation peaks across all key bytes**.
   - This ensured the most reliable recovery of the AES-128 master key.

Thank you

K.Neha

S.H.Atchaya