Power Analysis Side-channel Attacks

Federico Zanca

Computer Science and Engineering
Politecnico di Milano

Table of Contents

Table of Contents

What Are Side-Channel Attacks?

Definition

Side-channel attacks exploit unintended **physical leakages** from cryptographic devices to extract secret information.

- Instead of attacking the mathematical foundations of a cryptographic algorithm, they target information leaked through physical phenomena.
- This happens if there is a relationship between data/operations and a certain physical phenomena that can be observed
- Common leakage sources include **power consumption**, electromagnetic emissions, timing variations, and even sound or heat.

Why Are They Important?

A strong encryption algorithm is not secure if the underlying implementation is flawed

- Many devices thought to be secure can be compromised by analyzing side-channels.
- Attackers can retrieve secret keys without breaking the cryptographic math.
- Real-world cryptographic implementations often leak enough information to be vulnerable.

First publicized power analysis attacks appeared in the late 1990s. Since then, the field has evolved to include a variety of sophisticated techniques and powerful countermeasures.

Table of Contents

What is a Power Analysis Attack?

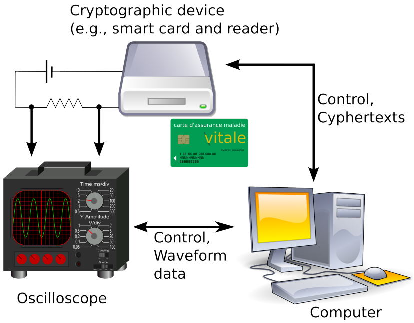
Switching $1 \to 0$ and $0 \to 1$ means providing or removing power to transistors.
A power analysis attack is a technique that exploits the relationship between a device's **power consumption** and the **data it processes** or the **operations** performed.
By carefully measuring power usage, an attacker can learn about the secret data being used

The Attack Setup

The setup for a power analysis attack is composed of:

- A **target device** that performs cryptography.
- A small **resistor** placed in the power line of the device.
- A digital **oscilloscope** to measure the tiny voltage changes across the resistor.

The oscilloscope captures a **power trace**: a graph of power usage over time.



Possible target devices

Electronic devices that perform cryptographic computations can be power analysis targets. This includes:

- Smart Cards and Security Tokens
- Embedded Microcontrollers (MCUs)
- Internet of Things (IoT) Devices
- Mobile Phones and Tablets

The Goal of Power Analysis attacks

Retrieving the Secret Key

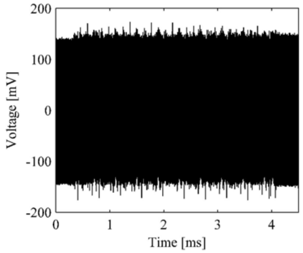
The primary goal of most power analysis attacks is to recover secret **cryptographic keys**.
This is rarely done by attacking the full key at once. Instead, attackers target **intermediate values** that are processed during the cryptographic algorithm. By learning small pieces of the key one at a time, they can eventually reconstruct the entire secret.

The Power Trace: A Measurement Vector

Definition

A power trace is a discrete time-series vector representing the instantaneous power consumption of a cryptographic device during a specific operation.

- Each point in the trace corresponds to a power measurement at a discrete point in time, captured by a high-frequency sampling instrument like a digital oscilloscope.
- The trace represents the electrical activity of the device's components.



The Non-Deterministic Nature of Measurements

The Problem of Measurement Variance

Repeated measurements of the same operation, using the **exact same inputs** (key and plaintext), will yield different power traces.
This variability is due to the presence of **noise**.

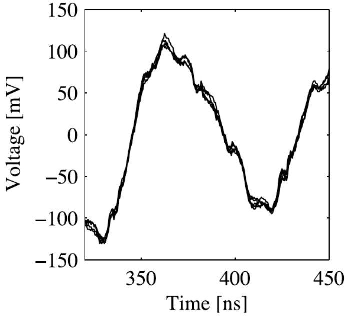


Figure: Power traces are different (but look very similar) even processing the same input because of noise

Noise Source 1: Electronic Noise

The Stochastic Component

Electronic Noise, $P_{electronic\_noise}$, is originated by the physical properties of the electronic components and the measurement setup.
It has many sources:

- Power supply
- Clock generator
- Radiated emissions of neighboring components

The distribution of the electronic noise $P_{el.noise}$ is a normal distribution

$$P_{el.noise} \sim \mathcal{N}(\prime, \sigma)$$

Noise Source 2: Switching Noise

The Uncorrelated Component

Switching Noise is the variations of power traces that are caused by cells that are not relevant for the attack.

- The amount of switching noise also depends on the architecture of the attacked device
- More computations performed in parallel by device → More switching noise
- The two most relevant properties of the measurement setup that introduce switching noise are the **clock frequency** of the device and **bandwidth** of the connection between the target logic cells and the oscilloscope

Composition of Power Traces

Model of each point of a power trace

Each point of a power trace can be modeled as the sum of an operation-dependent component P_{op}, a data-dependent component P_{data}, electronic noise $P_{el.noise}$, and a constant component P_{const}.

$$P_{total} = P_{op} + P_{data} + P_{el.noise} + P_{const}$$

All of these components are a function of time; we do not write them explicitly that way as we usually only analyze single points based on this model.
P_{const} is not relevant for power analysis attacks because it does not provide any exploitable information for an attacker.

The distributions of P_{op}, P_{data} and $P_{el.noise}$ can usually be approximated by a **normal distribution**.

Signal and Noise

In the context of a given attack scenario, the power consumption of a point of a power trace can be modeled as the sum of the exploitable power consumption P_{exp}, the switching noise $P_{sw.noise}$, the electronic noise $P_{el.noise}$ and the constant cmoponent P_{const}.

$$P_{total} = P_{exp} + P_{sw.noise} + P_{el.noise} + P_{const}$$

These components are independent of each other. Therefore, all parts of the power consumption that depend on the information that the attacker is looking for need to be modeled as P_{exp} and not as $P_{sw.noise}$.

Signal-to-Noise Ratio (SNR)

Signal-to-noise ratio is the ratio between the signal and the noise component of a measurement

$$SNR = \frac{Var(Signal)}{Var(Noise)}$$

Definition

In the context of a given attack scenario, the signal-to-noise ratio of a point of a power trace is given by the following equation

$$\mathrm{SNR} = \frac{\mathrm{Var}(P_{exp})}{\mathrm{Var}(P_{sw.noise} + P_{el.noise})}$$

The SNR quantifies how much information is leaking from a point of a power trace. The higher the SNR, the higher is the leakage

The Role of SNR in Attack Efficiency

SNR as an Indicator of Attack Feasibility

The SNR is a primary determinant of the efficiency and success of a power analysis attack.

- **High SNR**: Indicates that the data-dependent signal is strong relative to the noise. This allows for a successful attack with a smaller number of power traces.
- **Low SNR**: Indicates that the signal is weak and obscured by noise. A much larger number of traces is required to average out the noise and isolate the leakage.

The number of traces (N) required for a successful attack is inversely proportional to the SNR:

$$N \propto \frac{1}{\text{SNR}}$$

An attacker wants to maximize the SNR of their measurement setup, defender aims to design systems with the lowest possible SNR.

Table of Contents

Simple Power Analysis (SPA)

What is SPA?

- Attack technique based on the **direct visual interpretation** of power consumption traces from a cryptographic device.
- SPA attacks exploit key-dependent patterns within a trace
- They use only one trace (or very few traces)
- This was one of the first power analysis techniques to be publicly described.

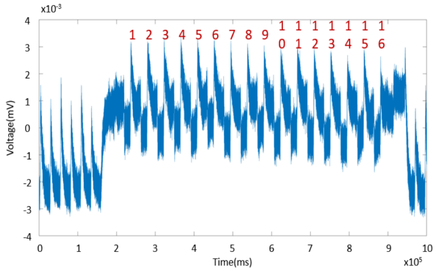


Figure: An SPA trace showing the 16 rounds of a DES encryption.

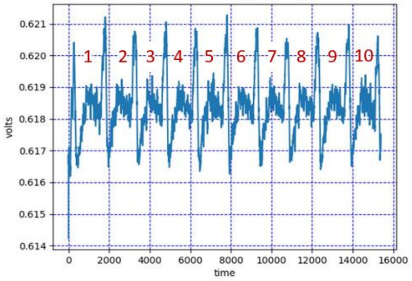


Figure: An SPA trace revealing the 10 rounds of an AES encryption.

Averaging Traces to Reduce Noise

The electrical signals measured are very small and inherently noisy. The random electronic noise component can obscure the underlying patterns. To improve the clarity of the trace for visual inspection, an attacker can capture multiple traces of the same operation and **compute their average**.

Since the electronic noise is typically **normally distributed** with a mean of zero, **averaging causes the noise to cancel out**.
The underlying deterministic signal, which is consistent across traces, is reinforced.

This technique significantly improves the signal-to-noise ratio (SNR) for visual analysis.

Interpreting SPA Traces

By visually inspecting a power trace, an attacker can identify large-scale features of the cryptographic algorithm.

- **Algorithmic Rounds:** Iterative ciphers like DES and AES produce repetitive patterns, where each repetition corresponds to one round of the algorithm.
- **Distinct Operations:** Different operations within a round, such as substitutions (S-boxes) and permutations (MixColumns), can consume different amounts of power and time, making them distinguishable in the trace.

This allows an attacker to map the power trace to the known structure of the cryptographic algorithm.

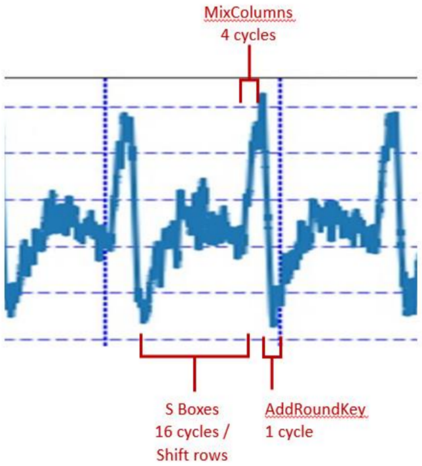


Figure: A detailed view of an AES round.

`path/to/your/AES_diagram.png`

Figure: The standard block diagram of an AES round for comparison.

Classic SPA Attack: RSA Exponentiation

A classic example of SPA is the attack on RSA implementations that use the **square-and-multiply** algorithm for modular exponentiation.

- The algorithm processes the private key one bit at a time.
- If the key bit is a '1', it performs a **square** operation followed by a **multiply** operation.
- If the key bit is a '0', it performs **only a square** operation.

The additional 'multiply' operation for a '1' bit consumes a different amount of power and takes a different amount of time, creating a distinct and easily recognizable pattern in the power trace.

Visualizing the RSA Attack

By observing the sequence of power consumption patterns, an attacker can directly read the bits of the private exponent from the trace.

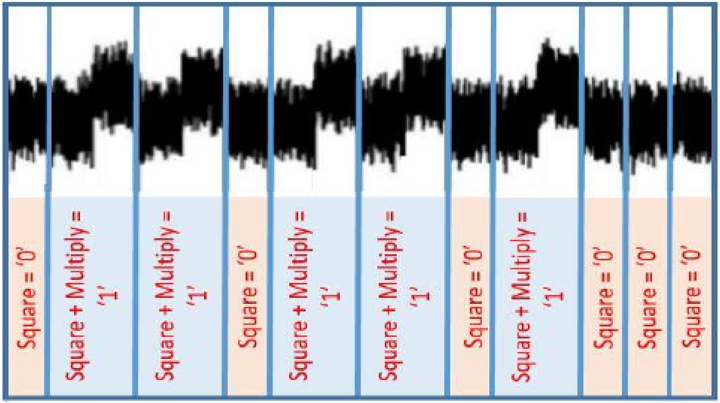


Figure: A power trace of an RSA exponentiation. The distinct patterns for 'square' and 'square-and-multiply' operations reveal the secret key bits.

Table of Contents

Two Classes of Power Analysis Attacks

Over time, power analysis attacks have divided into two main categories:

- **Model-Based Attacks:**
    - Rely on assumed **power leakage models** to relate device power consumption to secret data.
    - Use statistical techniques (difference of means, correlation, etc.) to test if a key guess matches observed traces.
    - **Differential Power Analysis** and **Correlation Power Analysis** fall in this category
- **Profiling Attacks:**
    - Use a profiling device (a surrogate of the target) to empirically build a model of how the key affects leakage.
    - Attack proceeds by matching new traces to the prebuilt "stencil" of leakages.
    - **Template Attacks** fall in this category

In both approaches, the effectiveness depends on how accurately the power model describes the real leakage behavior.

Power Models

What is a Power Model?

A **power model** mathematically predicts how a device's power consumption depends on the intermediate values manipulated by a cryptographic algorithm.

Power models are essential for model-based statistical attacks; they map the algorithm's internal computations (e.g., AES S-box outputs) to hypotheses about power consumption, and allow attackers to use observed traces to distinguish key guesses.

Typical Power Models Used in SCA

Standard models include:

- **Hamming Weight (HW):** Number of "1" bits in a byte or word ($HW(x) = \sum_i x_i$) is related to power consumption
- **Hamming Distance (HD):** Number of bits that flip between two values ($HD(x, y) = HW(x \oplus y)$) is related to power consumption
- **Least Significant Byte (LSB):** Power consumption is related to the value of LSB
- **Zero Value (ZV):** Power spikes when intermediate value $= 0$ (due to gate level effects)
- **Multivariate/Learned Models**: Combines multiple leakage sources

How Power Models Enable Statistical Attacks

- Statistical attacks use the power model to **make hypotheses**: for each key guess, predict the hypothetical leakage at every trace.
- Measurements are grouped/binned by predicted value.
- Statistics (such as mean, correlation, or likelihood score) are computed for each hypothesis.

The correct key is identified as the one whose hypothesis gives the highest statistical match to the observed physical traces.

Table of Contents

Differential Power Analysis

Differential Power Analysis (DPA) is a statistical method designed to identify data-dependent correlations in power measurements.
The technique partitions sets of power traces into two subsets and computes the difference of the averages for those subsets.

Key Insight

If the subsets are **truly correlated** with secret-dependent power, their **average power traces will differ**.
If not, the difference between averages approaches zero as the number of traces increase.

This allows even very small correlations to be statistically isolated despite noise.

Detecting Leakage

By computing the difference between the average traces of the two subsets at each time point:

- Significant non-zero differences indicate a correlation with secret data.
- As the dataset grows, random noise averages out, revealing the leakage signal.
- This process can be automated across entire traces without prior knowledge of where leakage occurs.

DPA: General Overview of the Algorithm

1. **Measure:** Capture a large number power traces with the victim device given a set of inputs.
2. **Define a Selection Function:** A function that predicts a single bit of an intermediate value inside the cryptographic algorithm, based on a known value (like plaintext or ciphertext) and a *guess* for a part of the key.
3. **Partition Traces:** For a specific *key guess*, classify all captured traces into two bins: one where the selection function predicts '0', and one where it predicts '1'.
4. **Compare the Sets:** Calculate the average trace for each of the two sets. Then, compute the pointwise difference between these two average traces.

Key guess correct: a "spike" will appear in the difference trace.
Key guess wrong: partitioning is random, and the difference trace will be close to zero.

Visualizing DPA Results: Case Study

- The **top trace** is the average of all measured traces where the LSB was 1 (over the first two rounds of AES).
- The **second trace** is the average where the LSB was 0.

The two averages are visually very similar—differences due to leakage are much smaller than overall power consumption variations.

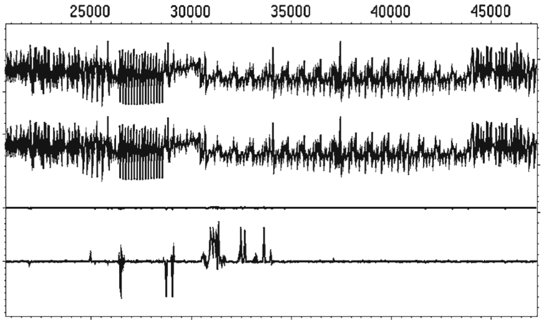


Figure: Outcome of a DPA experiment against an AES smart card. The analysis targets the LSB of the first S-box output in the first round.

Making Leakage Visible

- The **third trace** in the figure shows the difference between the two subset averages. This line appears mostly flat, as the differences are small and masked by the y-axis scale.
- The **bottom trace** is the same difference but scaled up (e.g. $\times 15$), making the leakage spikes clearly visible.

Leakage Interpretation

Spikes in the scaled difference trace correspond to moments when the targeted S-box output bit is manipulated inside the device. Spikes disappear after the first round, when the internal state is no longer correlated with the targeted bit.

High numbers of traces (e.g., 4,000) reduce noise, making the leakage clearer.

Selection Functions in Differential Power Analysis

The effectiveness of a DPA test critically depends on the choice of **selection function**.

Definition

A selection function is **used to assign traces to subsets** and is typically based on an educated guess as to a possible value for one or more intermediate values within a cryptographic calculation

Selection functions can be as simple as the predicted output bit of an S-box, or more complex, combining multiple bits or difference values.

- If the final DPA trace shows **significant spikes**, the **selection function is correlated** with an internal value computed by the device.
- If not, the prediction is either **incorrect** or too weakly correlated to detect.

DPA Attack on AES-128: Overview

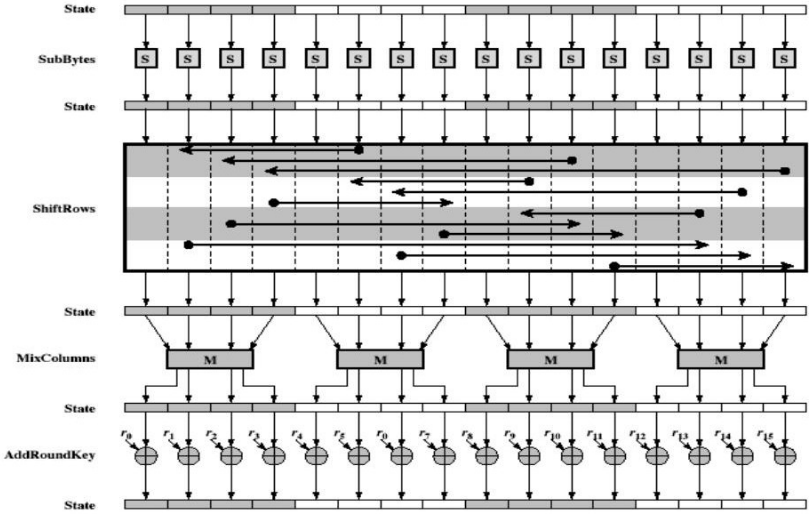
To illustrate the DPA process, we examine a classic attack on AES-128 encryption using real power traces from a smart card device.

Quick memory refresher on AES

- AES is a block cipher (symmetric)
- AES runs in multiple **rounds**; each round performs 4 operations
- AES takes blocks of 16 bytes as input and a key that is 16 (AES-128), 24 (AES-192) or 32 (AES-256) bytes long
- Plaintext and ciphertext are always the same size

We will attack the first round of AES.

AES-128 Rounds


State
SubBytes
S
State
ShiftRows
State
MixColumns
M
State
AddRoundKey
r_0 r_1 r_2 r_3 r_4 r_5 r_6 r_7 r_8 r_9 r_{10} r_{11} r_{12} r_{13} r_{14} r_{15}
State

AES-128 Round 1: Steps and Target

The first round of AES-128 consists of these steps:

1. **Initialization:** State is set to the 16-byte plaintext.
2. **AddRoundKey:** The 16-byte secret key is XORed with the state.
3. **SubBytes:** Each state byte is substituted using the S-box.
4. **ShiftRows:** Each row is cyclically shifted.
5. **MixColumns:** Each column is mixed by a linear operation.

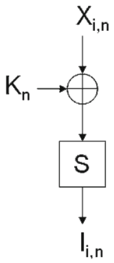


Figure: The DPA attack focuses on the output of *AddRoundKey* and *SubBytes*, as shown in the figure.

Intermediate Value & Attack Notation

For each power trace i:

- $X_{i,n}$: The n^{th} byte of plaintext in trace i (known).
- K_n: The unknown n^{th} byte of the round key (to be recovered).
- $I_{i,n}$: The n^{th} byte of the state after SubBytes in round 1.

These terms are related:

$$I_{i,n} = S[X_{i,n} \oplus K_n]$$

where $S[\cdot]$ denotes the AES S-box.

Key Search Strategy in DPA

Since $X_{i,n}$ is known, K_n is unknown, and $S[\cdot]$ is public:

- For a candidate K_n, compute $I_{i,n}$ for each trace.
- Use a **selection function** (e.g., take the LSB of $I_{i,n}$) to assign each trace to subset 0 or 1.
- Repeat for all 256 possible values of K_n. Each round key requires only 256 guesses.

DPA Advantage

This divide-and-conquer method makes brute-force keyspace attack on AES feasible by turning a 2^{128} search into a succession of practical 2^8 (256) searches.
DPA tests which candidate K_n aligns with actual device leakage, byte by byte, to reveal the full 128-bit AES key.

Key Guess 0x01 Binning Example

InputData	*KeyGuess*	*XOROutput*	*S − BoxOutput*	*Subset*
0xC7	0x01	0xC6	0xC7	0
0x1F	0x01	0x1E	0x1F	0
0x2C	0x01	0x2D	0x2C	0
0x89	0x01	0x88	0x89	0
0x01	0x01	0x00	0x01	**1**
0xD2	0x01	0xD3	0xD2	0

More traces are needed of course

Key Guess 0x3D Binning Example

InputData	*KeyGuess*	*XOROutput*	*S − BoxOutput*	*Subset*
0xC7	0x3D	0xFA	0x2D	1
0x1F	0x3D	0x22	0x93	1
0x2C	0x3D	0x11	0x82	0
0x89	0x3D	0xB4	0x8D	1
0x01	0x3D	0x3C	0xEB	**1**
0xD2	0x3D	0xEF	0xDF	1

More traces are needed of course

Interpreting Differences of Averages in DPA

After partitioning traces using a selection function, the difference of subset averages is examined:

Key Insight

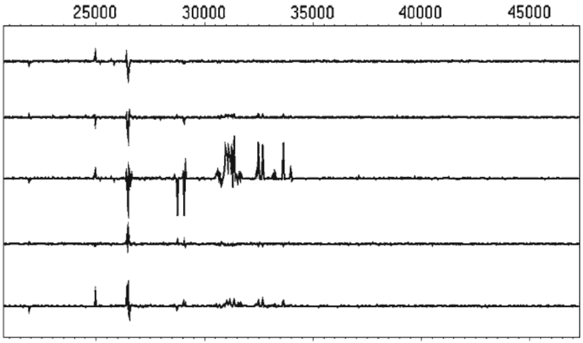
If the predicted S-box output bit correlates with the power measurements, large spikes appear in the differential trace for the correct key guess.

For incorrect guesses, the predicted values are unrelated to the device data, yielding little to no significant spike.

DPA Difference Traces for AES Key Byte Candidates

Differential traces were computed for all possible values of the first round key byte K_0 (0 to 255).

- The figure shows traces for guesses $K_0 = 101$ to 105.
- The correct key byte $K_0 = 103$ is identified by strong spikes in the differential trace.
- Incorrect guesses produce smaller spikes or flat traces.



Completing the Key Recovery

This analysis can be iterated independently for each of the 16 bytes ($n = 0, \dots, 15$) of the AES state.

Important Note

The same set of traces can be reused for each round key since each attack tests different hypotheses.

This divide-and-conquer approach enables full recovery of the 128-bit AES key.

Mathematical Formulation of DPA

Let:

- $T = \{T_i\}_{i=1}^m$ be the set of collected power traces,
- $T_i[j]$ be the power measurement at the j^{th} time sample in trace i,
- $C = \{C_i\}_{i=1}^m$ be the set of known inputs or outputs corresponding to each trace,
- $D(C_i, K_n)$ be a binary selection function dependent on known data C_i and a key guess K_n.

Then, the differential trace $\Delta D[j]$ at time offset j for key guess K_n is computed as:

$$\Delta D[j] = \frac{\sum_{i=1}^m D(C_i, K_n) \cdot T_i[j]}{\sum_{i=1}^m D(C_i, K_n)} - \frac{\sum_{i=1}^m (1 - D(C_i, K_n)) \cdot T_i[j]}{\sum_{i=1}^m (1 - D(C_i, K_n))}$$

This difference highlights points in time where the power consumption correlates with the selection function's prediction, helping identify the correct key guess.

Extending DPA Across Modes and Devices

DPA can be adapted for different cipher modes or hardware.
For example, the figure in the next slide shows a DPA attack on an AES-CBC implementation in an FPGA:

- The ciphertext is available instead of plaintext; so the selection function targets last round key bytes.
- A single oscilloscope trace capturing all AES operations is split into many operations for analysis.

Extending DPA Across Modes and Devices

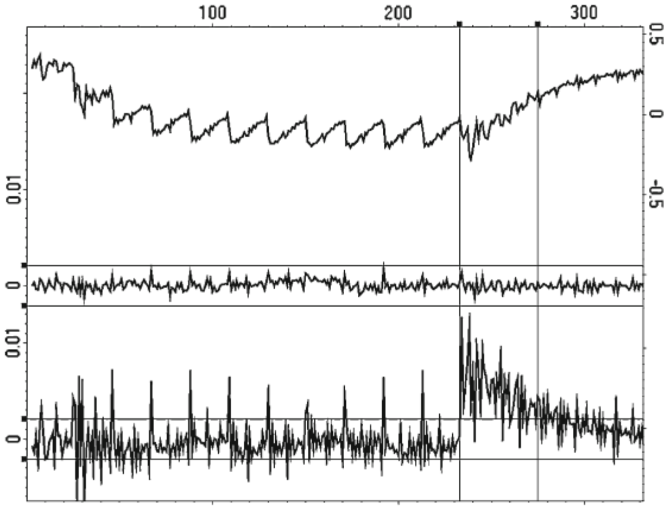


Figure: DPA results against AES-CBC on FPGA; top: average trace, middle: incorrect key guess, bottom: correct key guess.

Table of Contents

From DPA to CPA: Motivation

CPA was introduced by Brier, Clavier, and Olivier (2004) as a refinement of Differential Power Analysis.

Key Difference from DPA

- DPA: compares subset averages at a single point in time.
- CPA: computes the **statistical correlation** between predicted leakage (based on a model) and actual power measurements.

Advantage: Instead of just testing "is there a measurable difference?", CPA quantifies *how well the model matches reality*, across the whole trace.

Leakage Model 1: The Hamming Weight (HW)

The simplest model for data-dependent power leakage assumes consumption is proportional to the number of bits set to '1' in a register or on a bus.

Definition: Hamming Weight

For an m-bit data word $D = \sum_{j=0}^{m-1} d_j 2^j$, its Hamming Weight is the count of its '1' bits:

$$HW(D) = \sum_{j=0}^{m-1} d_j$$

- **Key Insight:** In this model, power consumption depends on the **number of active bits**, not the integer value they represent.
- For example, the 8-bit values 0x01, 0x02, 0x40, and 0x80 all have $HW = 1$ and are predicted to leak similar amounts of power.
- For uniformly random m-bit data, the average Hamming Weight is $m/2$.

Leakage Model 2: The Hamming Distance (HD)

The HW model has a limitation: it ignores the hardware's previous state. A more physically accurate model states that power is consumed when bits **flip state** (e.g., on a data bus).

Definition: Hamming Distance

The Hamming Distance between a new data word D and the previous state R is the number of bits that have changed:

$$HD(D, R) = HW(D \oplus R)$$

- This **transition model** is a better fit for real CMOS logic, where charging and discharging capacitors during bit flips is a major source of dynamic power consumption.
- The $D \oplus R$ operation isolates exactly which bits have toggled.
- The Hamming Weight model is a special case of the HD model where the reference state R is assumed to be all zeros ($R = 0$).

The Linear Power Model for CPA

The Hamming Distance model leads to the fundamental assumption of CPA: the data-dependent portion of the power consumption is linearly related to the number of bit transitions.

The Linear Power Model

This relationship is expressed as:

$$W = a \cdot HD(D, R) + b$$

- W: The total power measured at a specific point in time.
- $HD(D, R)$: The **hypothesized leakage** based on the HD model.
- a: A scaling factor connecting the HD() to the actual power.
- b: A constant offset representing static power and measurement noise.

The entire goal of CPA is to find a key guess that makes our hypothesized leakage $HD(D, R)$ strongly correlate with the measured power W.

Limitations of Classic DPA

Differential Power Analysis (DPA) distinguishes key guesses by comparing **average traces across two partitions**.

- It only considers the **difference of means** at each time sample.
- It is sensitive to noise: many traces required to average out randomness.
- May produce misleading results (**"ghost peaks"**) when leakage does not align perfectly with the partition.
- Wastes information: each sample is treated independently instead of considering correlation across the whole trace.

Consequence

DPA is useful, but not statistically optimal; it tests for a binary difference instead of measuring **how strongly traces align with the hypothesis**.

Why CPA is a More Powerful Refinement

Correlation Power Analysis (CPA) addresses DPA's shortcomings:

- Uses a **parametric model** of data-dependent leakage (Hamming Weight or Hamming Distance).
- Instead of simply asking *"are two averages different?"*, CPA asks:

  How well does my predicted power model correlate with the measured power trace?
- Employs the **Pearson correlation coefficient** across all traces and all times, giving a continuous score for each hypothesis.
- Strong signals stand out as **statistically significant correlations**, reducing the chance of ghost peaks.

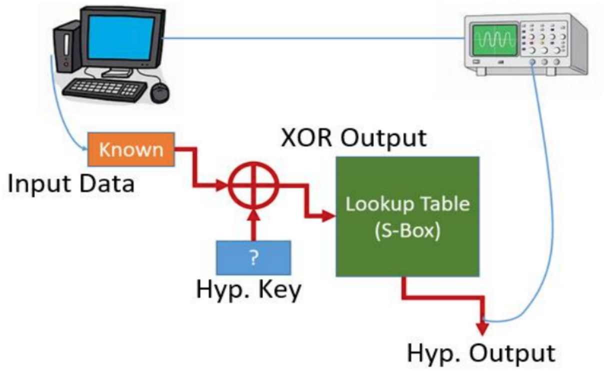
Intuition

CPA turns DPA from a binary detector into a **statistical matching process**, making better use of each collected trace.

Typical Hardware Setup for CPA

CPA requires:

- A device performing cryptographic operations, accessible for power measurement.
- A computer that sends known but random data to the device.
- A power measurement setup recording traces during device operation.



CPA Attack Strategy

The goal of a CPA attack is to find the secret key by identifying which key guess produces a **predicted power consumption** that best matches the **measured power consumption**.

The Core Question

For a given key guess, does the predicted Hamming Weight of an intermediate value (like the S-box output) statistically correlate with the power trace?

- We perform this process for one key byte at a time.
- For AES-128, we target the output of the first round's `SubBytes` operation, just as we did with DPA.
- Intermediate value: $I_{i,n} = S[P_{i,n} \oplus K_n]$.

CPA Attack on AES

The attack iterates through all possible values for a single key byte (e.g., K_0). For each guess:

1. **Hypothesize Intermediate Values:** For each trace $i = 1, \ldots, N$, calculate the hypothetical S-box output based on the known plaintext P_i and the key guess K_{guess}.

$$V_{i,guess} = S[P_i \oplus K_{guess}]$$

2. **Predict Leakage:** Apply the power model to each hypothetical value. For our example, we use the Hamming Weight model.

$$H_{i,guess} = HW(V_{i,guess})$$

   This gives us a vector of N predicted leakage values for this key guess.

3. **Correlate:** Measure the statistical correlation between our vector of predicted leakages (H_{guess}) and the vector of actual power measurements at **each time sample** j in the traces.

The Tool for the Job: Pearson Correlation Coefficient

To quantify how well our predicted leakage matches the real measurements, we use the **Pearson correlation coefficient** (ρ).

What it Measures

The Pearson coefficient measures the **strength and direction of a linear relationship** between two sets of data.

Its value ranges from -1 to +1:

- $\rho = +1$: Perfect positive linear correlation.
- $\rho = -1$: Perfect negative linear correlation (anti-correlation).
- $\rho = 0$: No linear correlation.

In CPA

We compute ρ between our hypothesized Hamming Weights and the measured power trace. A high **absolute value** of ρ indicates our key guess is likely correct.

Pearson Correlation: The Definition

The Pearson correlation coefficient (ρ) is formally defined as the **covariance** of two variables, normalized by the product of their **standard deviations**.

Definitional Formula

$$\rho_{H,T_j} = \frac{\text{cov}(H, T_j)}{\sigma_H \cdot \sigma_{T_j}}$$

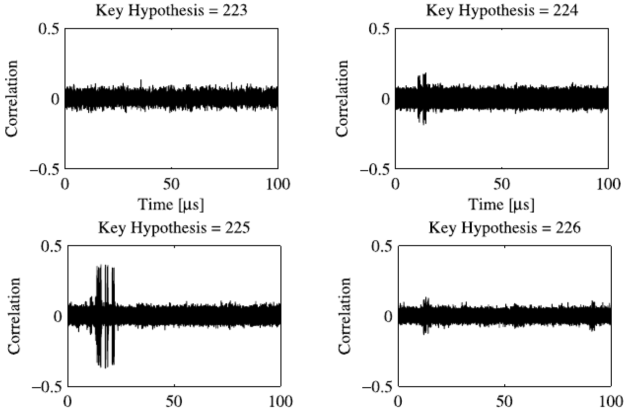
- $\text{cov}(H, T_j)$: The covariance, which measures how our **hypothesized leakage vector (H)** and the **measured power vector (T_j)** vary together.
- σ_H: The standard deviation of our hypothesized leakage values
- σ_{T_j}: The standard deviation of the power measurements at time sample j

Normalizing by the standard deviations removes the units and scale, giving a pure measure of correlation between -1 and +1.

CPA Results: Identifying the Correct Key

After computing the correlation for all 256 key byte guesses at every time sample, we get 256 "correlation traces."

- The plot below shows the correlation traces for a few key candidates.
- The trace corresponding to the **correct key guess** will show a peak
- Traces for **incorrect key guesses** will fluctuate randomly around zero



Interpreting the Results: What the Peak Means

The peak in the correlation trace for the correct key is the "moment of truth."

The Meaning of the Peak

- The **X-axis position** of the peak indicates the *point in time* when the target intermediate value (e.g., the S-box output) is being manipulated by the device.
- The **Y-axis value** (the height of the peak) represents the *strength* of the linear correlation (ρ). A higher peak means a better model fit and less measurement noise.

For the correct key, our hypothesized leakage model, $H = HW(S[P \oplus K_{correct}])$, successfully predicts the device's actual power consumption W.

Interpreting the Results: Incorrect Guesses

When an incorrect key guess (K_{wrong}) is used, the predicted leakage values become meaningless.

- The hypothesized leakage vector, $H_{wrong} = HW(S[P \oplus K_{wrong}])$, is now a set of values that are statistically **independent** of the true intermediate value being processed inside the device.
- Therefore, H_{wrong} is also independent of the true data-dependent power consumption, W.

The Result

When two data vectors are statistically independent, their Pearson correlation coefficient will naturally approach zero as the number of samples (N) increases. This is why the incorrect key traces appear as random noise.

Table of Contents

Countermeasure Philosophy: Hiding

Power analysis attacks work because a device's power consumption **depends on the data** it is processing.

The Goal of Hiding

Hiding aims to break the link between the power consumption and the data values. The cryptographic algorithm itself remains unchanged, but its execution becomes difficult to analyze.

In short, Hiding aims to make the power traces useless to an attacker.

Hiding: Two Fundamental Approaches

To make power consumption independent of the data, there are two ideal (but theoretical) goals:

1. **Random Power**
   - Make the device consume a **random** amount of power in each clock cycle.
   - The real signal is buried in unpredictable noise.

2. **Constant Power**
   - Make the device consume an **equal** amount of power in each clock cycle, regardless of the operation or data.
   - There are no data-dependent variations to exploit.

While perfect randomness or equality is impossible in practice, these two ideals guide all hiding countermeasure design.

Hiding in Practice: Time vs. Amplitude

Practical hiding techniques can be divided into two groups based on which dimension of the power trace they manipulate.

1. Manipulating the Time Dimension

- These techniques randomize **when** operations occur.
- The goal is to misalign the power traces from different executions, making averaging and statistical analysis difficult.
- *Example: Random clock jitters or random instruction insertion.*

2. Manipulating the Amplitude Dimension

- These techniques directly alter the power consumption characteristics of the operations themselves.
- The goal is to make the power draw for different operations either more random or more uniform.
- *Example: Special logic gates that draw constant power.*

Hiding: Time Dimension

As we know, DPA and other statistical attacks require **correctly aligned** power traces to average out noise and isolate the signal.

The Countermeasure Strategy

If we can randomize the execution of the algorithm, the operations will occur at different moments in time for each run.

- This misalignment acts as a powerful defense.
- An attacker will require significantly more traces to overcome the induced jitter.
- The more random the execution, the more difficult the attack becomes.

Hiding: Time Dimension

The basic idea is to randomly insert "do-nothing" or dummy operations before, during, and after the real cryptographic operations.

- In each execution, randomly generated numbers decide how many dummy operations are inserted and where.
- The position of each *real* operation now varies randomly from trace to trace.

Critical Detail

The **total number** of inserted dummy operations should be constant for all executions. This prevents an attacker from learning anything by simply measuring the total execution time.

Trade-Off

More dummy operations provide more security but lead to a **lower throughput**. A suitable compromise must be found.

Time Hiding: Shuffling

An alternative to inserting dummies is to randomly change the sequence of operations that can be performed in an arbitrary order.

Example: AES S-Box Look-ups

- In each round of AES, the 16 S-box look-ups are independent of each other.
- Their execution order can be **shuffled**.
- In each execution, a randomly generated sequence determines the order of the 16 look-ups, misaligning the leakage for each byte.

Advantage

- Does not affect throughput nearly as much as inserting dummy operations.

Disadvantage

- Can only be applied to a limited number of independent operations within an algorithm.

Time Hiding: Software vs. Hardware

The effectiveness and options for time-based hiding depend heavily on the implementation level.

Software Implementations

- Dummy operation insertion and shuffling are both easily implemented in software.
- However, their protective power is often limited.
- A key requirement is a source of good on-device random numbers.

Hardware Implementations

- Hardware offers significantly more powerful options.
- In addition to dummy operations and shuffling, hardware allows for direct manipulation of the clock itself.

Hardware Time Hiding: Clock Manipulation

In hardware, it is possible to directly randomize the clock signal to destroy trace alignment. Common techniques include:

- **Random Insertion of Dummy Cycles:** A finer-grained version of dummy operations where individual clock cycles are randomly used for "dummy" computations on random data.
- **Skipping Clock Pulses:** A filter is inserted into the clock path that randomly skips clock pulses, causing jitter.
- **Randomly Changing Clock Frequency:** An internal oscillator's frequency is controlled by random numbers, constantly changing the speed of operations.
- **Multiple Clock Domains:** The device randomly switches between several different on-chip clock signals.

For all these techniques, the attacker must not be able to detect the countermeasure (e.g., by observing the clock signal itself).

Hiding in the Amplitude Dimension

The second major hiding strategy is to directly change the **power consumption characteristics** of the operations themselves.

The Goal: Lower the Signal-to-Noise Ratio (SNR)

Recall that the leakage depends on the SNR. The goal is to make this value as close to zero as possible. This can be achieved in two ways:

- Reduce the signal ($Var(P_{op})$) to zero.
- Increase the noise ($Var(P_{noise})$) to infinity.

While the ideal goal of SNR=0 is unreachable, we can get close by either **increasing the noise** or **reducing the signal**.

Amplitude Hiding: Increasing the Noise

The goal of this approach is to build a device where random switching activity dominates the power consumption, drowning out the real signal.

Hardware Techniques

- **Wide Datapaths:** It is harder to attack a single bit in a 128-bit architecture than in a 32-bit one, as more unrelated bits are switching in parallel.
- **Noise Engines:** These are dedicated circuits, often based on random number generators connected to large capacitors, whose only job is to create random switching noise in parallel with the real computation.

Software Techniques

- **Parallel Activity:** Programmers can increase noise by using other components of the device (e.g., coprocessors, communication interfaces) to perform random activities in parallel with the crypto algorithm.

Amplitude Hiding: Reducing the Signal

The goal of this approach is to make the power consumption as constant as possible, regardless of the operation or data.

Hardware Techniques

- **Dedicated Logic Styles:** The most common strategy. Cells are built using special logic (e.g., Dual-Rail Pre-charge Logic) designed so that their power consumption is constant, regardless of the input data.
- **Power Filtering:** A filter (e.g., using switched capacitors or constant current sources) is inserted between the power supply and the crypto circuit to smooth out data-dependent variations.

Important Limitation

Hardware-level countermeasures like filtering may not protect against attacks on electromagnetic (EM) emanations, as the EM field is generated before the power is filtered.

Amplitude Hiding in Software

While software cannot change the power characteristics of the underlying hardware, programmers can make careful choices to reduce leakage.

- **Choice of Instructions:** Choose instructions that are known to leak less information about their operands.
- **Constant Program Flow:** Avoid conditional jumps or branches that depend on the key or secret data. The sequence of executed instructions should always be the same.
- **Key-Independent Memory Access:** Avoid memory addresses that depend on the key. If necessary, use addresses that have similar leakage characteristics (e.g., the same Hamming weight).

These software techniques are primarily effective against simpler SPA attacks but are generally not sufficient to defeat DPA.

Hiding Countermeasures: Summary

	Time dimension	Amplitude dimension
Goal	Randomize when operations occur to misalign traces	Lower SNR by equalizing or randomizing power amplitude
Methods	Dummy ops; Shuffling; Clock skip; Random freq; Multi-clock	Reduce signal (balanced logic, filtering); Increase noise (noise engines, parallel activity)
Notes	Throughput overhead; limited shufflable ops; often combined	Hardware most effective; software limited; filtering may not help against EM

Table of Contents

Countermeasure Philosophy: Masking

Masking takes a different approach from Hiding. Instead of changing the device's physical properties, it fundamentally **changes the data being processed**.

The Goal of Masking

Masking aims to randomize every sensitive intermediate value within the algorithm, making the device's power consumption statistically independent of the true, secret data.

This is a powerful countermeasure that can provide security even on a "leaky" device.

The Core Idea: Splitting Secrets into Shares

Masking is a form of **secret sharing**. Every sensitive value, v, is split into two or more **shares** that are processed independently.
For a basic first-order scheme, we split v into two shares:

- A random value, m, called the **mask**.
- The masked value, v_m, which is a combination of the secret and the mask.

Example: Boolean Masking with XOR

The most common method is to use the XOR operation:

$$v_m = v \oplus m$$

The device works with v_m and m, but never with v directly. To recover the original value, one simply computes $v = v_m \oplus m$.

How Masking Defeats First-Order DPA

The security of masking comes from statistical independence.

- The mask, m, is a fresh, uniformly random value for each execution.
- This means the masked value, $v_m = v \oplus m$, is **also uniformly random**, regardless of the secret value v.
- An attacker measuring the power consumption of an operation on v_m will see a value that is statistically independent of v.

Security Guarantee

As long as the attacker can only probe the leakage of a single share at a time (the "**first-order**" assumption), the leakage they observe contains no information about the secret value. This provides provable security against first-order DPA.

A Fully Masked Implementation

To be effective, masking must be applied **throughout the entire algorithm**.

1. **Masking the Inputs:** The plaintext and key are masked at the very beginning of the computation.
2. **Processing Masked Values:** All subsequent operations in the algorithm must be redesigned to correctly operate on the shares and produce new, correctly masked intermediate results.
3. **Unmasking the Output:** The final result of the encryption is also masked. The mask must be removed at the very end to produce the correct ciphertext.

A single unmasked operation at any point can create a fatal vulnerability.

The Challenge of Non-Linear Functions

The main difficulty in masking is correctly handling non-linear operations.

Linear Operations are Easy

For a linear function f (with respect to XOR), masking is simple because the function distributes over the operation:

$$f(v \oplus m) = f(v) \oplus f(m)$$

The output mask $f(m)$ is easy to compute.

The Challenge of Non-Linear Functions

Non-Linear Operations are Hard

For a non-linear function like an AES S-box, this is not true:

$$S(v \oplus m) \neq S(v) \oplus S(m)$$

- The relationship between the input mask and the output mask is complex.
- Special, and often computationally expensive, "masked S-box" implementations are required to compute the result while keeping all intermediate values properly masked.

Types of Masking and Blinding

The operation used for sharing defines the type of masking.

Boolean Masking

- Uses XOR ($\oplus$).
- $v_m = v \oplus m$
- Ideal for bitwise or XOR-based algorithms like AES.

Arithmetic Masking

- Uses modular addition or multiplication.
- $v_m = v + m \pmod{n}$
- Used for algorithms based on arithmetic, like RSA.

Blinding

The term **blinding** is typically used to refer to arithmetic masking when applied to asymmetric algorithms like RSA.

- **Message Blinding:** Applying a multiplicative mask to the message.
- **Exponent Blinding:** Applying an additive mask to the private exponent.

Masking at the Architecture Level: Software

The Basic Software Workflow

1. XOR the initial plaintext and/or key with a random mask.
2. Ensure all intermediate values remain correctly masked throughout the computation.
3. Keep track of how the mask itself is transformed by the algorithm's operations.
4. Remove the final mask from the output to get the correct ciphertext.

This process is straightforward for linear operations, but non-linear operations (like table look-ups) require special attention.

The Challenge: Masking Table Look-ups

Many modern ciphers, like AES, use table look-ups (e.g., S-boxes) for their non-linear operations. How do you mask a table look-up?

The Solution: On-the-Fly Masked Tables

Instead of using a fixed table $T(v)$, we must generate a temporary, masked table, T_m, for each execution.

- The goal is to create a table where looking up a masked input gives a masked output:
$$T_m(v \oplus m_{in}) = T(v) \oplus m_{out}$$
- To build this table, the device must iterate through all possible inputs, calculate the output, and store the correctly masked result in the new table.

This process must be repeated for every new mask used. This increases both the **computational effort** (to build the table) and the **memory requirement** (to store it), adding significant performance overhead.

Random Precharging

Random precharging is a simple technique that **implicitly masks** the power consumption without directly masking the data values themselves.

How It Works

This technique is effective against attacks that exploit the **Hamming Distance** leakage model (e.g., of a data bus).

1. Before loading a sensitive value, v, onto the bus (or into a register), first load a random value, m.
2. The device's power consumption will now be related to the Hamming Distance between the random value and the real value: $HD(v, m)$.
3. This is equivalent to leaking the Hamming Weight of the XORed value: $HW(v \oplus m)$.

Since the attacker doesn't know the random value m, they cannot predict the Hamming Distance.

Pitfall: The Independence Assumption

Masking's security proofs often rely on an assumption that can fail in the real world.

Theoretical Assumption

The power consumption of a masked value is independent of the secret.

Flaw

This assumes leakage depends on only **one** value at a time. In reality, leakage often depends on the **relationship between multiple values**, such as the Hamming Distance between consecutive states of a data bus.

This "combined leakage" can completely break a masking scheme.

Example: How a Shared Mask Creates a Flaw

Imagine two values, v_m and w_m, that share the **same mask** (m) are processed back-to-back on a device that leaks Hamming Distance.

$$v_m = v \oplus m \quad \text{and} \quad w_m = w \oplus m$$

The device's leakage is proportional to:

$$HD(v_m, w_m) = HW(v_m \oplus w_m)$$

The math shows the mask cancels out completely:

$$HW((v \oplus m) \oplus (w \oplus m)) = HW(v \oplus w) = HD(v, w)$$

The device leaks information about the **unmasked** secret values, defeating the countermeasure.