Side-channel Metrics

Federico Zanca

Computer Science and Engineering
Politecnico di Milano

Table of Contents

Table of Contents

Number of Traces (N.O.T.): Definition

What it Measures

- The most commonly used metric to quantify the efficiency of a side-channel attack
- It represents the minimum number of side-channel measurements (traces) an attacker needs to successfully recover the secret key

Its primary purpose is to assess how "data-hungry" an attack is.

A lower N.O.T. indicates a more effective or
faster attack, as fewer measurements are required to compromise the secret.

Number of Traces (N.O.T.): Limitations

Challenges and Limitations

- **Attack Dependent:** The N.O.T. value is inherently tied to the specific side-channel attack method employed (e.g., DPA, CPA, Template Attacks), making direct comparisons between different attacks difficult
- **Vague Key Recovery:** The term "key recovery" can be imprecise; it might refer to recovering the full key, a partial key, or simply reducing the entropy of the key space, leading to ambiguity in evaluation
- **Conflates Device and Attacker:** N.O.T. is not an inherent measure of a device's security. It conflates the device's inherent physical leakage with the attacker's computational capabilities (e.g., ability to collect many traces, search the key space). This coalescence makes it difficult to obtain a clear picture of the device's true vulnerability and to identify countermeasures that are robust against any attacker

Table of Contents

Signal-to-Noise Ratio (SNR): Introduction and Definition

What is SNR?

- Standalone metric, it can be obtained directly from side-channel traces without needing to launch a full attack
- Provides a quick assessment of the quality of information leakage present in the measurements

- SNR quantifies the ratio between the strength of the "signal" L_d (the useful, data-dependent part of the leakage) and the strength of the "noise" L_n (the random, useless part)

$$L = L_d + L_n = g(V) + L_n$$

where $L_n \sim N(0, \sigma^2)$.

- A higher SNR indicates stronger and clearer leakage, making it potentially easier to exploit

Signal-to-Noise Ratio (SNR): Introduction and Definition

Mathematical Definition

Formally, SNR is defined as the ratio of the variance of the deterministic leakage to the variance of the noise:

$$\text{SNR} = \frac{\text{Var}(L_d)}{\text{Var}(L_n)}$$

SNR: Visualizing Leakage & Interpretation

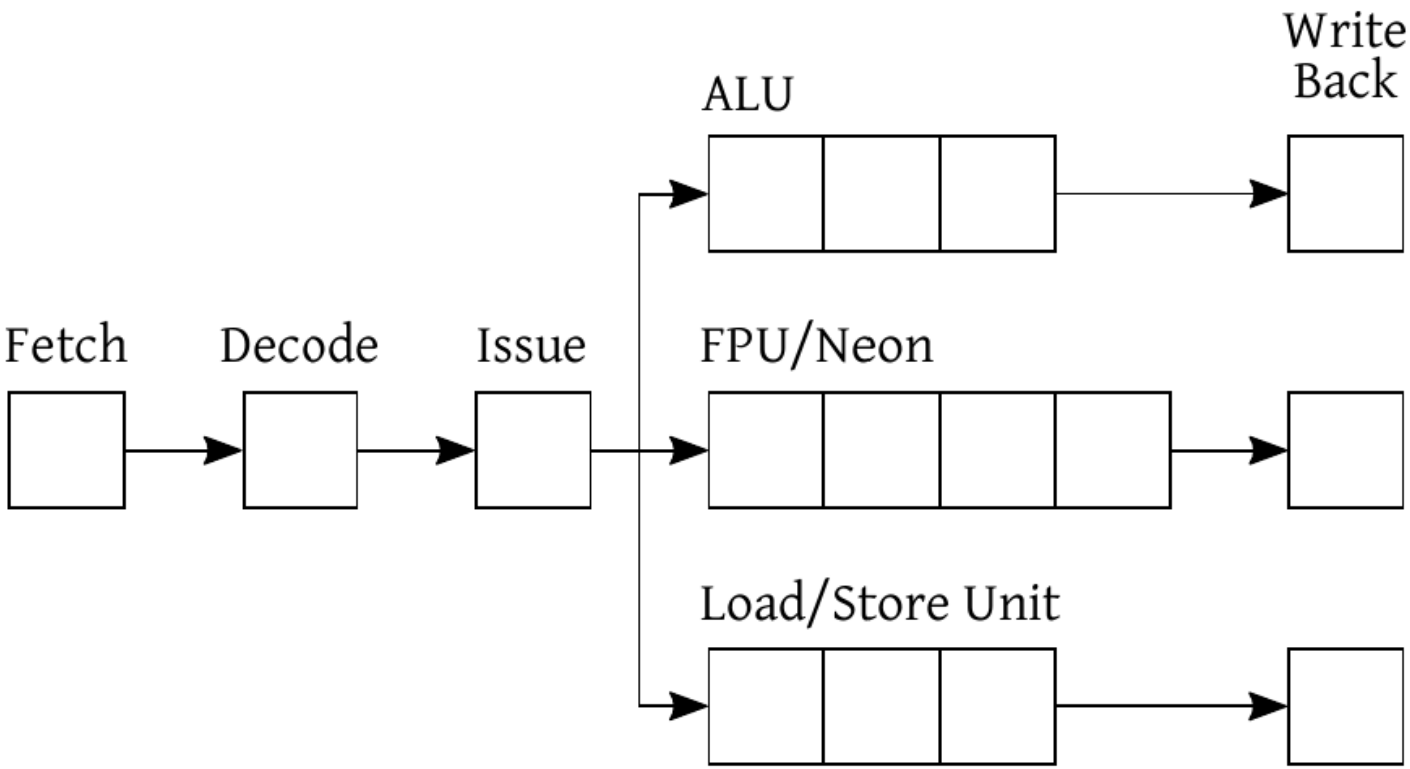


Figure: Example of a side-channel trace (placeholder image)

SNR: Understanding and Utilizing Leakage

Interpreting SNR

- A high SNR indicates that the signal component dominates the noise, suggesting clear leakage points
- Helps in identifying the most promising Points of Interest (POIs) within a trace, where the leakage is strongest
- Can be used to compare the leakage characteristics of different implementations or countermeasures

Computing SNR with Simulated Leakage

In this scenario, the attacker explicitly simulates the leakage L
This is done by precisely defining its deterministic part (L_d) and its noise component (L_n)

Example: 8-bit Value Leakage

- Simulate the leakage of an 8-bit value V (uniformly random)
- Choose a leakage function $g(V) = \alpha V + \beta$, where α, β are known constants
- Thus, $L_d = \alpha V + \beta$, representing a scaled and offset version of identity leakage
- Choose noise $L_n \sim \mathcal{N}(0, \sigma^2)$, specifying σ^2
- The simulation can be noiseless ($\sigma = 0$) to identify leakage, or noisy ($\sigma \neq 0$) to analyze device/countermeasure behavior in the presence of noise

SNR with Simulated Leakage: Computation

In this controlled simulation, we can proceed to compute the SNR directly using the expectations $E(\cdot)$

Expectations of Deterministic Leakage

- Expected value of L_d:

$$E(L_d) = E(g(V)) = \sum_{v=0}^{255} g(v)Pr(v) = \frac{1}{256}\sum_{v=0}^{255}(\alpha v + \beta)$$

- Expected value of L_d^2:

$$E(L_d^2) = E(g(V)^2) = \frac{1}{256}\sum_{v=0}^{255}(\alpha v + \beta)^2$$

-

$$\text{SNR} = \frac{\text{Var}(L_d)}{\text{Var}(L_n)} = \frac{E(L_d^2) - E(L_d)^2}{\sigma^2}$$

SNR with Real Measurements

- In a real-world scenario, we cannot directly observe the deterministic leakage (L_d) or the noise (L_n). We can only measure their sum, the total leakage (L).
- Our goal is to estimate the SNR from these real measurements without prior knowledge of the leakage function.

Let's consider once again an example in which the leakage L of an 8-bit value V is observed, obtaining n traces $l_1, ..., l_n$, each trace l_i containing a single time sample that leaks information about V

Estimation Steps

- 1. Partition traces: Group traces based on the value of the intermediate V (e.g., 256 groups for an 8-bit value). This way we obtain $\mathcal{L}_0, \mathcal{L}_1, ..., \mathcal{L}_{255}$. The set $\mathcal{L}_v$ contains all traces l that are observed when $V = v$
- 2. Estimate Means: Compute the mean trace $\hat{\mu}_v$ for each $\mathcal{L}_v$ and the overall mean $\hat{\mu}$.

$$\hat{\mu}_v = \frac{1}{|\mathcal{L}_v|} \sum_{l \in \mathcal{L}_v} l$$

$$\hat{\mu} = \frac{1}{256} \sum_{v=0}^{255} \hat{\mu}_v$$

where $|\mathcal{L} - v|$ is the number of traces for the group

- 3. Estimate Variances: Compute the variance ($\hat{\sigma}_v^2$) for each group and the average of these variances (σ^2) as the noise variance.

$$\hat{\sigma_v^2} = \frac{1}{|\mathcal{L}_v| - 1} \sum_{l \in \mathcal{L}_v} (l - \hat{\mu_v})^2$$

$$\hat{\sigma}^2 = \frac{1}{256} \sum_{v=0}^{255} \hat{\sigma}_v$$

- 4. Compute SNR: The signal variance is the variance of the group means, while the noise variance is the average of the group variances. The SNR is their ratio.

$$\text{SNR} = \frac{\text{Var}(L_d)}{\text{Var}(L_n)} = \frac{\sum_{v=0}^{255} (\hat{\mu}_v - \hat{\mu})^2}{\hat{\sigma}^2}$$

Interpreting an SNR Plot

Example: Unprotected AES S-box

- An SNR plot over time reveals when and where the most significant leakage occurs in an operation.
- High peaks in the plot correspond to specific time samples where the leakage is strongly correlated with the secret key.
- For instance, an unprotected AES S-box implementation on a Cortex-M0 microcontroller shows distinct peaks where key-dependent operations are performed, with an SNR of approximately 0.02.
- The overall shape of the SNR plot provides an overview of the inherent leakage of the device for a given implementation.

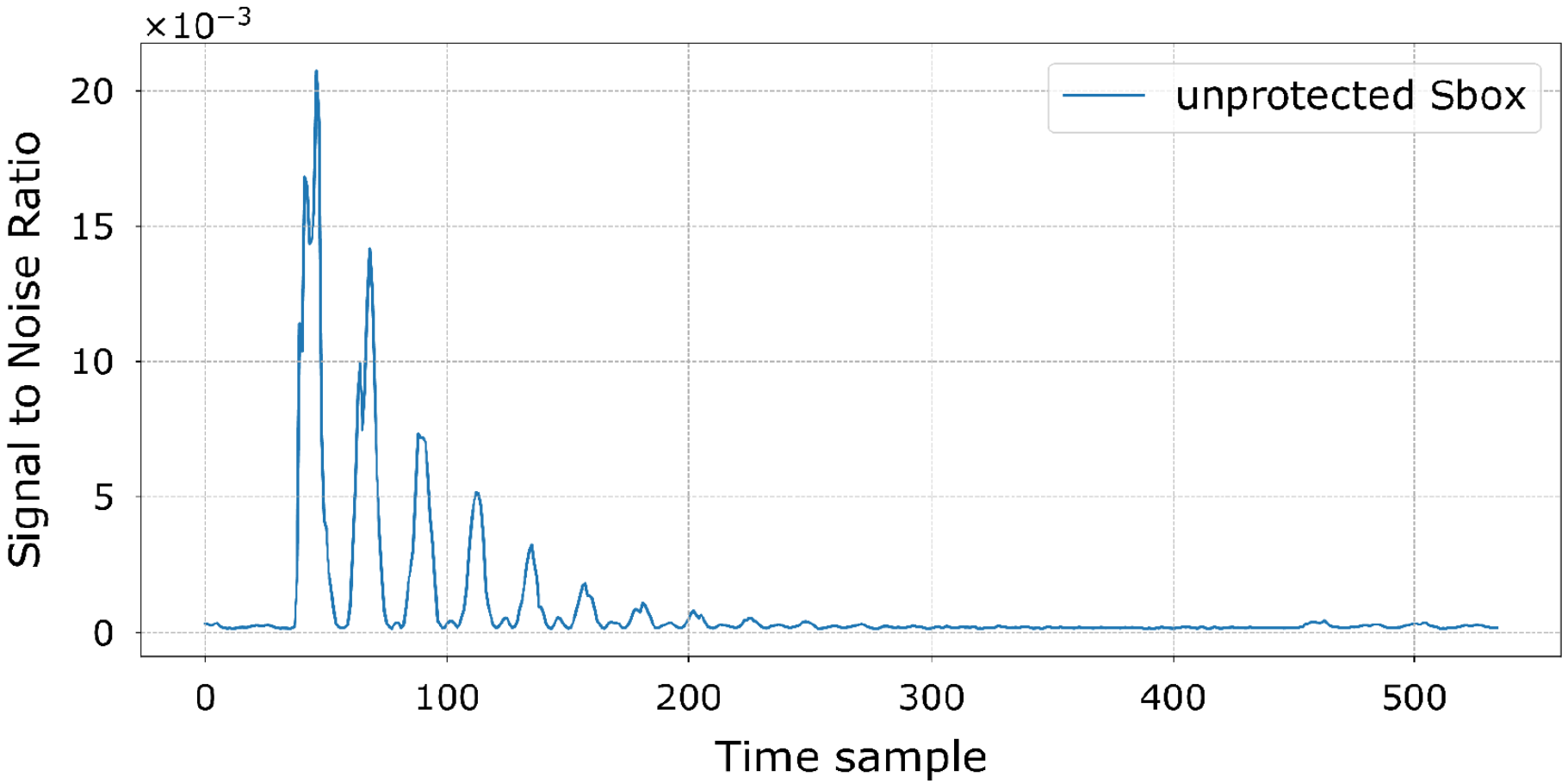


Figure: SNR plot for an unprotected AES S-box implementation. Peaks indicate key-dependent leakage.

SNR: Limitations

- **Not a direct attack feasibility metric:** It maintains a level of generality by not being tied to a specific attack method, but this also means it can't directly convey the actual security level of a device.
- **Simplified leakage model:** The metric implicitly relies on a simplified model of leakage (e.g., assuming a Gaussian noise distribution). In real-world scenarios, the actual leakage may diverge from these assumptions.
- **Parameter estimation issues:** The statistical parameters (mean and variance) must be adequately estimated from the available traces. A poor estimation due to a limited N.O.T. can lead to misleading SNR results.
- **Univariate nature:** The standard definition of SNR focuses on a single time sample at a time (univariate), ignoring the joint multivariate leakage that an adversary can exploit across multiple samples.

Table of Contents

Score and Rank: Why?

The majority of side-channel attacks employ a **divide-and-conquer** strategy.
Instead of targeting the full cipher key at once, the attack partitions it into smaller, more manageable parts. For example, a 16-byte (128-bit) AES key is typically attacked as 16 independent key bytes.

The Goal

This approach allows the attacker to recover each small key part individually, dramatically reducing the complexity of the attack. The metrics of **score and rank** are used to evaluate the success of recovering each of these parts.

The Scoring and Ranking Process

For a Single Key Part (e.g., one AES key byte)

For each key portion:

1. **Hypothesize:** The attacker considers all possible values for the key part. For an 8-bit key byte, this means 256 key candidates (0 to 255).
2. **Score:** A score is computed for each key candidate. This score, generated by a distinguisher like correlation, measures how well the hypothesis for that candidate matches the observed side-channel leakage.
3. **Guess:** The scores are sorted to produce an ordered list of guesses, from best to worst. The key candidate with the highest score becomes $\texttt{guess}_1$, the attacker's top choice. For instance, if candidate k=42 had the best score, then $\texttt{guess}_1$ would be 42.
4. **Rank:** A rank is assigned to each key candidate based on its position in the sorted list. The best-scoring candidate receives rank 1.

Algorithm for a Standard Attack with Score and Rank

Input: Attack score function $f(\cdot)$, Key candidates K
Data: Simulated or real side-channel traces
Output: $score_k, guess_k, rank_k$, for all $k \in K$ and all key partitions

1: $partitions \leftarrow$ divide-and-conquer(full key)
2: **for all** $partition \in partitions$ **do**
3: **for all** $k \in K$ **do**
4: $score_k \leftarrow f(\text{traces}, k)$
5: **end for**
6: $[score_i, score_j, \ldots, score_m] \leftarrow \text{sort}([score_0, score_1, \ldots, score_{|K|-1}])$
7: $[guess_1, \ldots, guess_{|K|}] \leftarrow [i, j, \ldots, m]$
8: **for all** $guess_i$, with $i \in \{1, 2, \ldots, |K|\}$ **do**
9: $rank_{guess_i} \leftarrow i$
10: **end for**
11: **end for**

Interpreting the Results: Convergence

- By plotting score/rank against the number of traces, we can visualize the attack's progress.
- **What to look for:** A key candidate that **stands out** from the rest.
- A successful attack shows **convergent behavior**: one candidate consistently achieves the highest score (and thus, rank 1) as more traces are added.
- **Example:** Considering a CPA attack on AES-128, key candidate $K = 203$ reaches rank 1 after about 15,000 traces and maintains that top position, demonstrating a successful recovery of that key byte (as shown in the following pictures).

Example: Score and Rank Plot Convergence

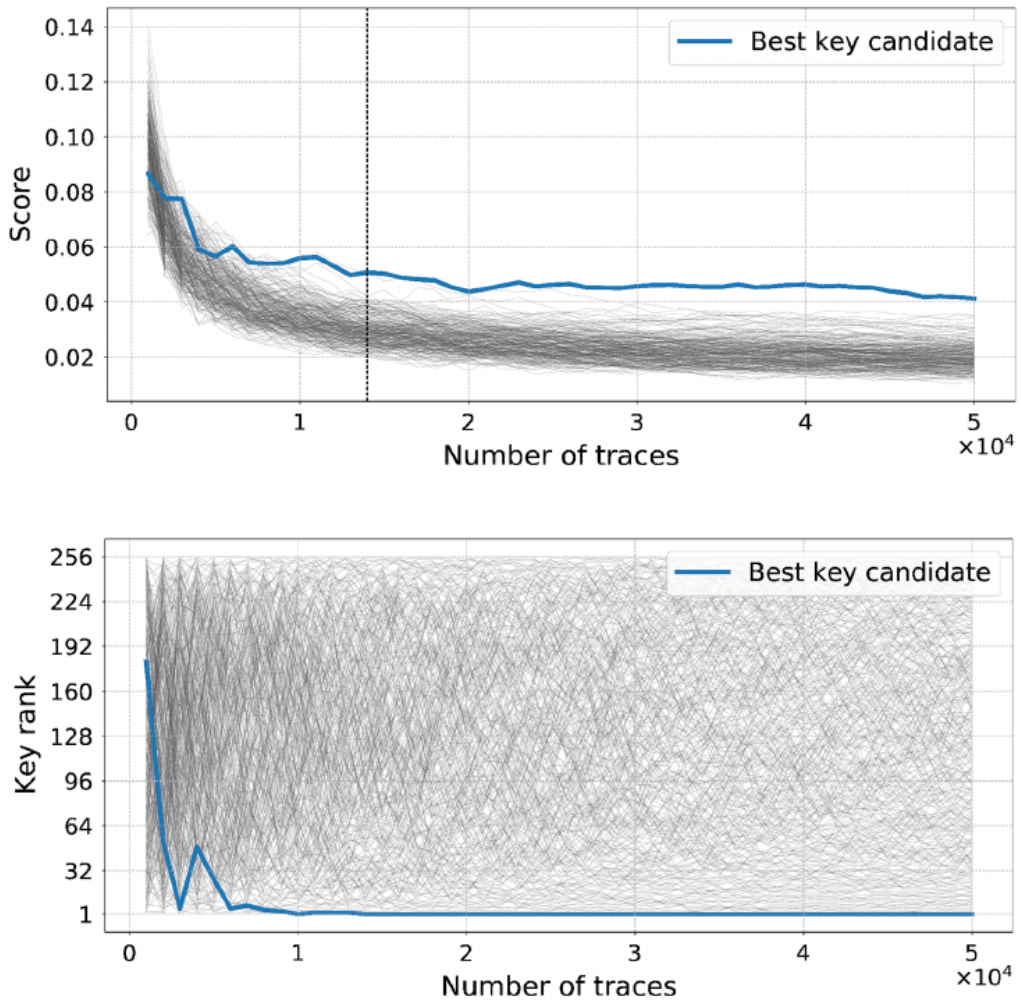


Figure: Top: Score plot where the correct key (K=203) is in blue
Bottom: Rank plot where the same key's rank converges to 1 after approx. 15,000 traces.

Limitation: Statistical Stability

The pitfall of a single experiment

A score/rank plot is typically generated from one traceset and one fixed key. This can be deceptive.
A single result might be an **outlier**, as some keys can be accidentally easier or harder to attack than others.

To ensure the results are meaningful, we must follow a more rigorous process:

- **Repeat** the attack multiple times.
- **Vary** the conditions for each run (e.g., use different keys and different sets of traces).
- **Aggregate** the outcomes by calculating the average score/rank and the standard deviation.

This gives us a **statistically stable** metric that truly reflects the device's security.

Limitation: The Full Key Recovery Problem

In a real-world attack, the key is unknown. If the attack is suboptimal or uses too few traces, the correct key parts might not achieve Rank 1. Even if the correct byte is at a low rank (like 2 or 3), the **device is still vulnerable**, but the full key remains hidden.

Key Enumeration

To find the full key, the attacker must perform **key enumeration**. This is a time-consuming, trial-and-error process:

- Construct full key candidates from combinations of the top-ranking key parts.
- Test each full key candidate (e.g., by decrypting a known ciphertext) until the correct one is found.

This is often a computationally expensive task.

The Evaluator's Solution: Known-Key Analysis

A More Efficient Approach for Evaluation

To assess security without the high cost of key enumeration, evaluators use a different strategy: a **known-key analysis**.

- In this scenario, the evaluator knows the secret key from the start.
- This allows them to check the rank of the *correct key* directly, even if the attack fails to place it at Rank 1.
- This answers the question "How close are we to breaking it?" and provides a more precise risk assessment than a simple pass/fail.

Paving the way for new metrics

This concept of known-key analysis is the foundation for the next metrics we will discuss: **Success Rate (SR)** and **Guessing Entropy (GE)**.

Table of Contents

Success Rate (SR) and Guessing Entropy (GE): Introduction

Motivation

- In known-key analysis, **evaluators** have access to the secret key. This allows for more precise security assessments.
- SR and GE metrics move beyond a simple "key recovered/not recovered" statement to quantify the attack's probabilistic progress.

What is Success Rate (SR)?

- **SR** measures the probability that an attack successfully places the correct key candidate at **rank 1**.
- It is derived from the attack's guess vector: if the top guess ($\texttt{guess}_1$) matches the correct key, the experiment is a success.
- To ensure statistical stability, SR is typically computed by averaging results over many repeated attack experiments.

Success Rate (SR): Definition

Recall that a standard side-channel attack produces a guess vector $[guess_1, guess_2, \ldots, guess_{|K|}]$ where $guess_1$ is the best candidate.
Let k_c be the correct key.

Formal Definition

For a side-channel experiment i, the success rate SR_i is 1 if the best guess equals the correct key ($guess_1 = k_c$), otherwise it's 0. Alternatively, using the rank of the correct key (rank k_c):

$$SR_i = \begin{cases} 1, & \text{if } rank_{k_c} = 1 \\ 0, & \text{otherwise} \end{cases}$$

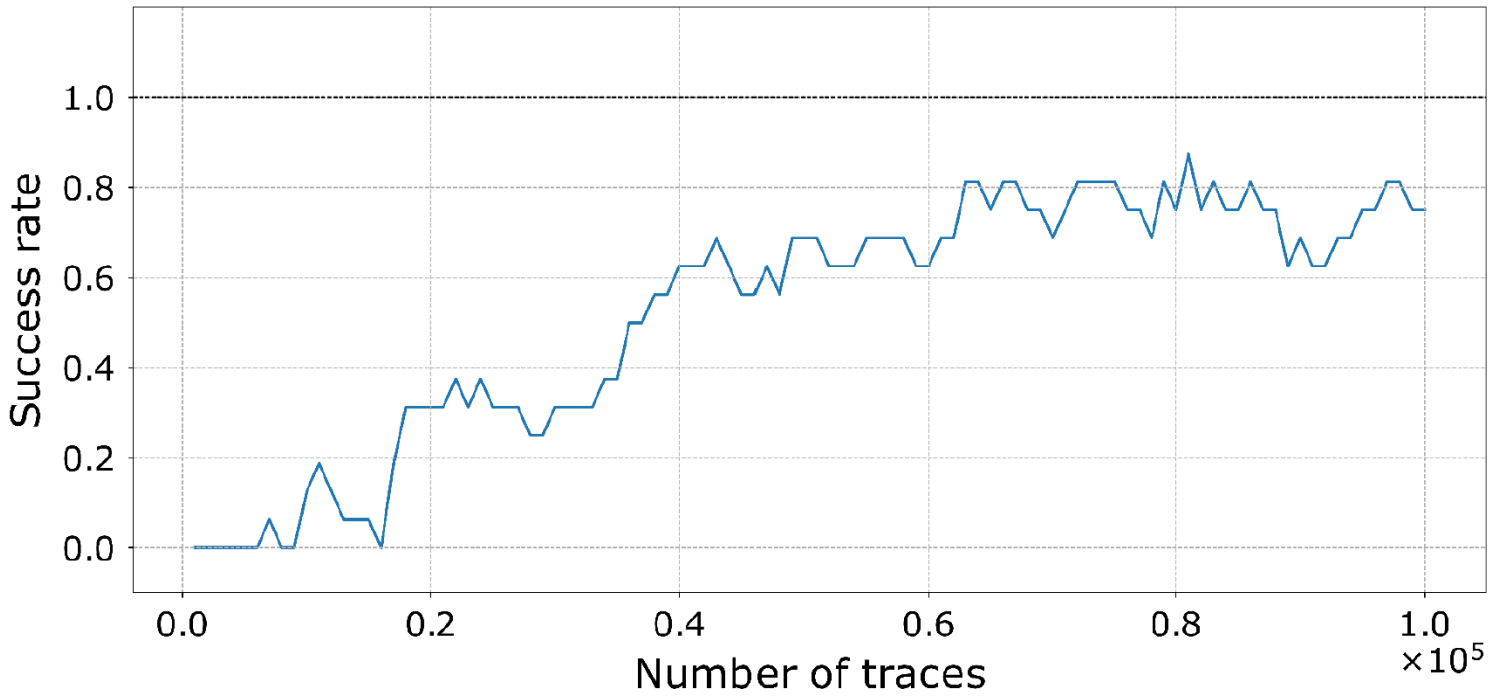
To ensure statistical stability, the final SR metric is estimated by averaging over p experiments:

$$SR = \frac{1}{p} \sum_{i=1}^{p} SR_i$$

Success Rate (SR): Interpretation

Increasing the number of attack traces typically improves the success rate gradually. An SR value closer to 1 implies a high probability of correct key recovery.

Full key recovery implies $SR = 1$. However, we often want the SR metric to reflect cases where the correct key is ranked highly, yet > 1.



SR of Order o

The SR metric can be extended to reflect cases where the correct key is ranked highly, but not necessarily 1st.
SR_i^o is 1 if the correct key k_c is found within the top o key guesses (i.e., $k_c \in [guess_1, \ldots, guess_o]$ or $rank_{k_c} \leq o$), otherwise 0.

$$SR_o^i = \begin{cases} 1, & \text{if } rank_{k_c} \leq o \\ 0, & \text{otherwise} \end{cases}$$

The overall SR_o is:

$$SR_o = \frac{1}{p} \sum_{i=1}^{p} SR_o^i$$

If $o = 1$, this reverts to the original SR definition.

SR of Order o: Link to Attacker Effort

If an evaluator finds $SR^o = 1$ for a key part, an attacker with an unknown key would need to verify at most o candidates to find that key part after performing the attack.
Example: If $SR^5 = 1$ for an AES byte, the attacker needs to check at most 5 candidates for that byte.

Full Key Enumeration Challenge

- This key enumeration process must be repeated for all divide-and-conquer partitions of the full key.
- For example, if $SR^5 = 1$ for all 16 AES-128 key bytes, an attacker might need to verify up to 5^{16} full keys.
- This is a computationally challenging, heuristic task, and determining the appropriate value for o is critical for assessing attacker capability.

Guessing Entropy (GE): Definition

To avoid the heuristic and potentially complex process of SR of order o for full key enumeration, Guessing Entropy (GE) offers a more flexible metric.

GE Formal Definition

Given a rank vector $[rank_0, rank_1, \ldots, rank_{|K|-1}]$ and the correct key k_c, the rank of k_c ($rank_{k_c}$) can be found and plotted against the number of traces.

For a given experiment i, GE_i is defined as:

$$GE_i = \log_2(rank_{k_c})$$

It is typically expressed in bits.

The overall GE, averaged over p experiments (where k_c is ideally changed between experiments to ensure statistical stability), is:

$$GE = \frac{1}{p} \sum_{i=1}^{p} GE_i$$

Guessing Entropy (GE): Interpretation and Example

The $rank_{k_c}$ and GE metrics represent the average remaining workload for an attacker after a side-channel attack. GE = 0 means the side-channel attack has eliminated all uncertainty about the secret key

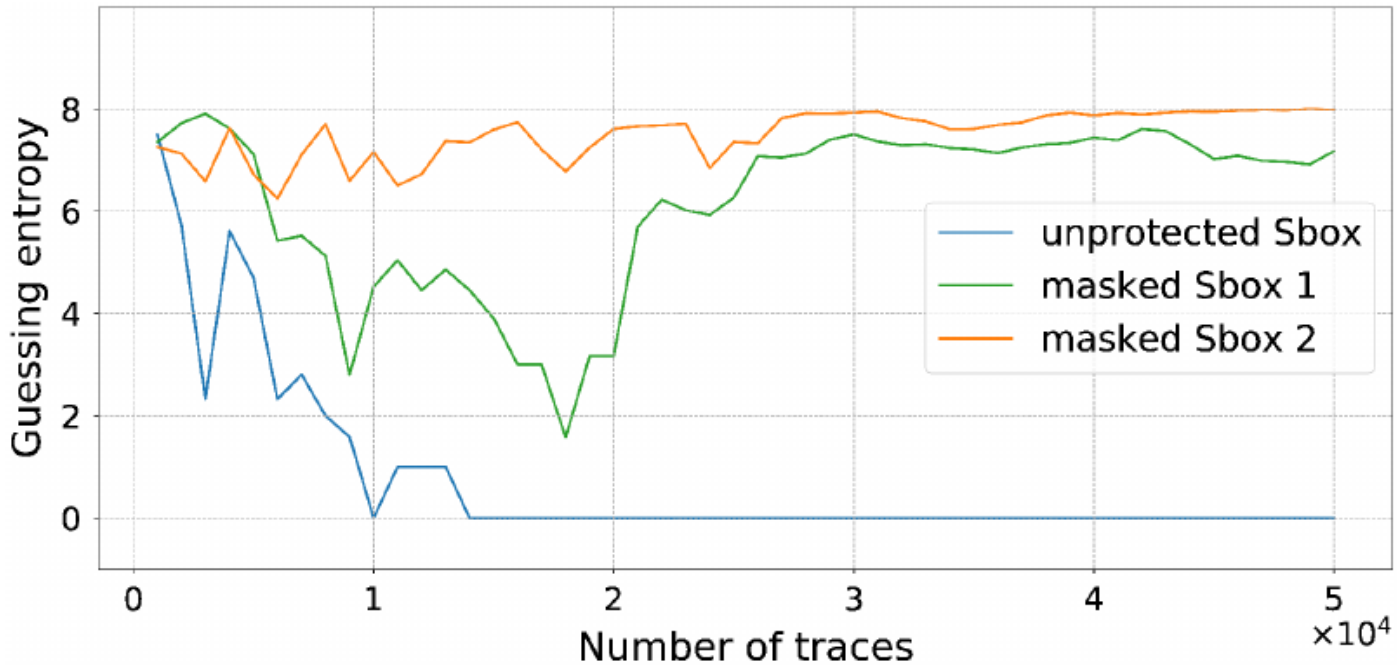


Figure: Guessing Entropy plot for three AES S-box implementations. GE drops to 0 for the unprotected S-box after roughly 15,000 traces; for masked S-boxes, GE remains high even with 50,000 traces.

Limitations of SR and GE: Full Key Security

Byte-wise Metrics Do Not Translate Directly

Both Success Rate (SR) and Guessing Entropy (GE) quantify attack effectiveness for individual key bytes (e.g., in AES). However, knowing the GE or SR for all bytes does **not** allow direct inference of the GE or SR for the full key.

- The process of full key recovery requires **key rank estimation**, which is the known-key analog of key enumeration.
- Full key security assessment is more complex than simply combining values for each byte; detailed estimation methods are required.

Limitations of SR and GE: No Projection of Effort

Trace-Dependency Limits Security Forecast

SR and GE provide attack progress for a specific number of measured traces, but do not project future effort required for successful key recovery.

- If SR or GE is close to random guessing (e.g., 1/256 for AES byte), one can only conclude the attack currently fails.
- The metrics do **not** indicate how many more traces would be needed to break the cipher, nor are they designed to forecast eventual success.
- GE is an expression of the remaining effort, but it is always derived from a specific number of measured traces.

Towards Better Projections

This limitation led to the development of new metrics, which tie attack scores and success probabilities more rigorously to the number of traces.