Side-channel Metrics

Federico Zanca

Computer Science and Engineering
Politecnico di Milano

Table of Contents

1. Number of Traces
2. Signal to Noise Ratio
3. Score and Rank
4. Success Rate and Guessing Entropy
5. Correlation as a Metric
6. Information-Theoretic Metrics
   - Motivation for IT Metrics
   - Mutual Information
   - Hypothetical Information
   - Perceived Information
   - Limitations of IT Metrics
7. Key Enumeration and Rank Estimation
8. Statistical Tests

Table of Contents

1. Number of Traces
2. Signal to Noise Ratio
3. Score and Rank
4. Success Rate and Guessing Entropy
5. Correlation as a Metric
6. Information-Theoretic Metrics
   - Motivation for IT Metrics
   - Mutual Information
   - Hypothetical Information
   - Perceived Information
   - Limitations of IT Metrics
7. Key Enumeration and Rank Estimation
8. Statistical Tests

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

1. Number of Traces
2. Signal to Noise Ratio
3. Score and Rank
4. Success Rate and Guessing Entropy
5. Correlation as a Metric
6. Information-Theoretic Metrics
   - Motivation for IT Metrics
   - Mutual Information
   - Hypothetical Information
   - Perceived Information
   - Limitations of IT Metrics
7. Key Enumeration and Rank Estimation
8. Statistical Tests

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: 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.

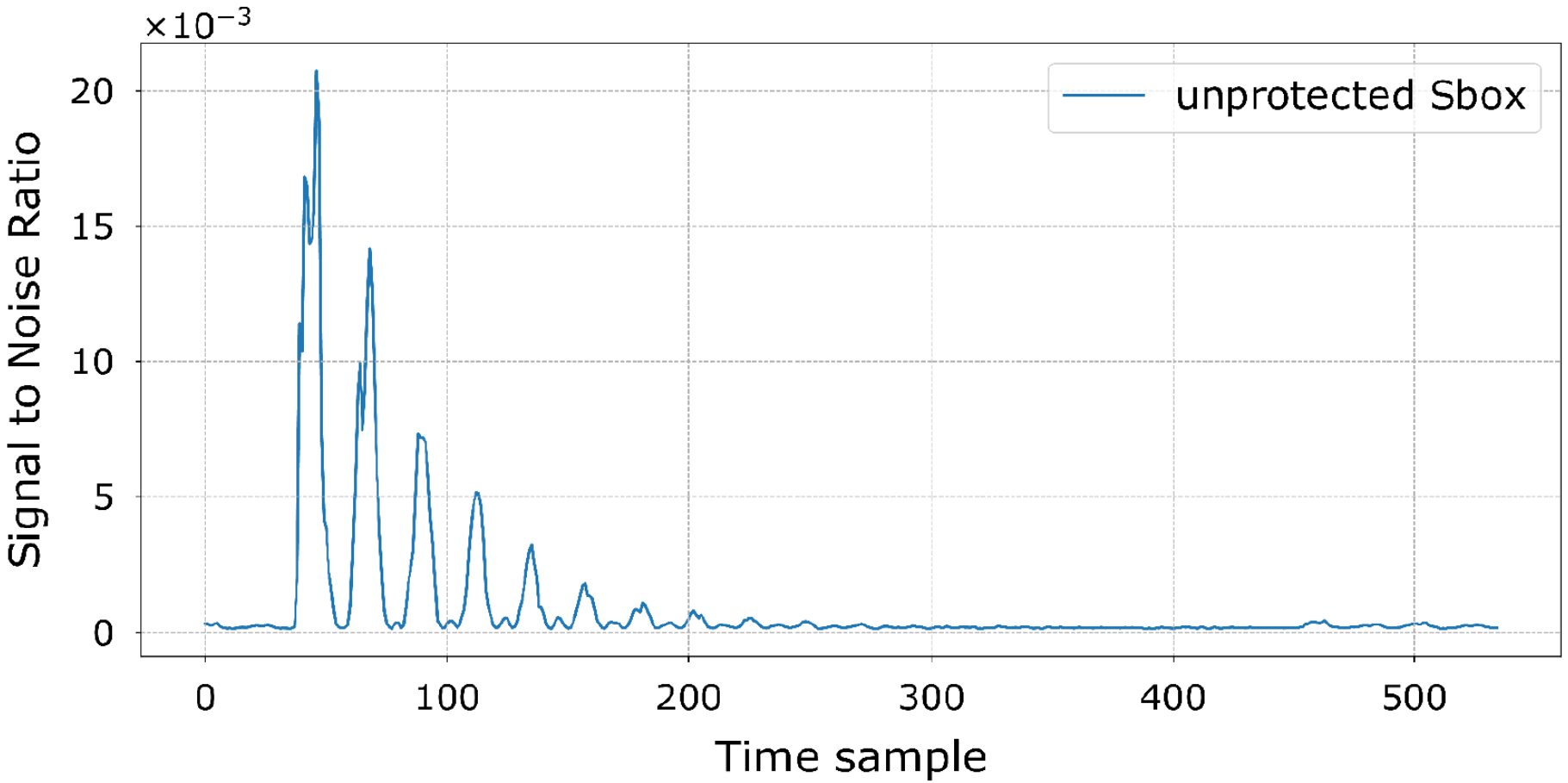


Figure: SNR plot for an unprotected AES S-box implementation. Peaks indicate key-dependent leakage.

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

1. Number of Traces
2. Signal to Noise Ratio
3. Score and Rank
4. Success Rate and Guessing Entropy
5. Correlation as a Metric
6. Information-Theoretic Metrics
   - Motivation for IT Metrics
   - Mutual Information
   - Hypothetical Information
   - Perceived Information
   - Limitations of IT Metrics
7. Key Enumeration and Rank Estimation
8. Statistical Tests

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

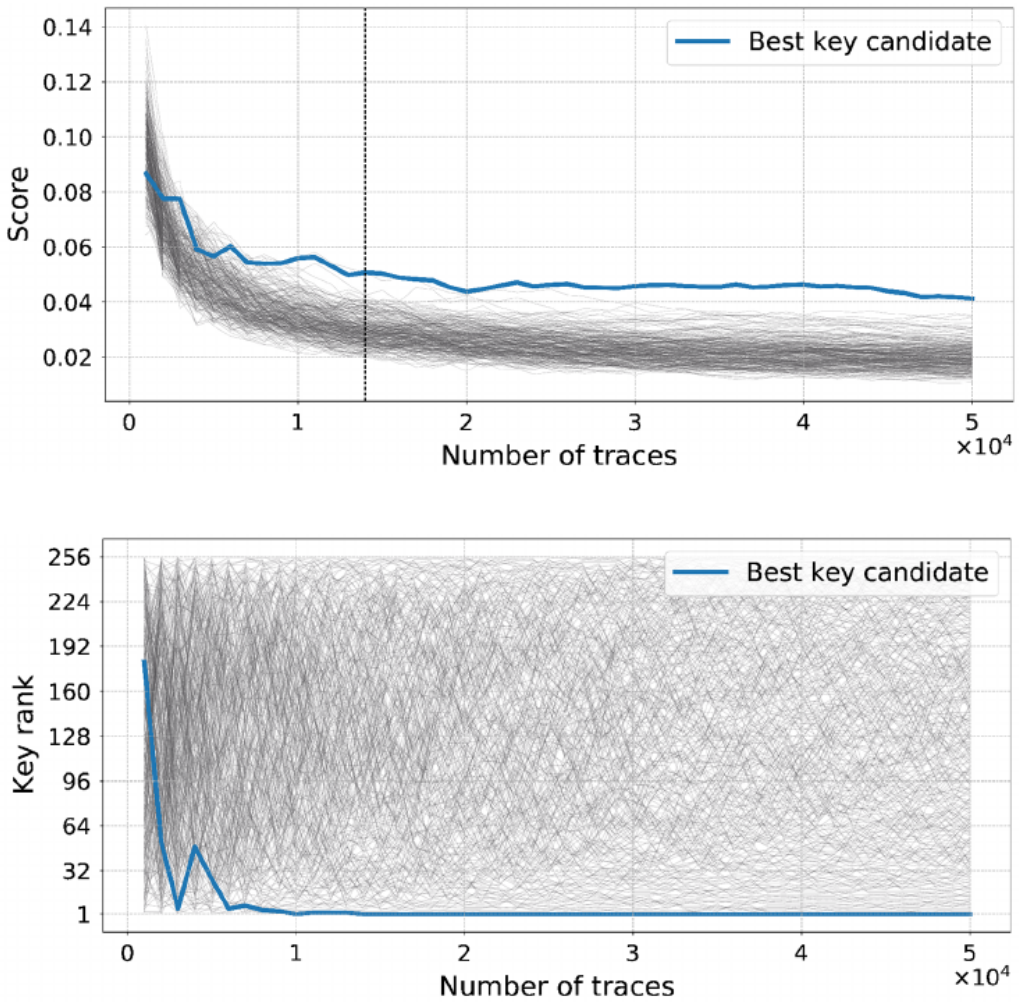


Figure: Top: Score plot where the correct key (K=203) is in blue
Bottom: Rank plot where the same key's rank converges to 1 after approx. 15,000 traces.

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

1. Number of Traces
2. Signal to Noise Ratio
3. Score and Rank
4. Success Rate and Guessing Entropy
5. Correlation as a Metric
6. Information-Theoretic Metrics
   - Motivation for IT Metrics
   - Mutual Information
   - Hypothetical Information
   - Perceived Information
   - Limitations of IT Metrics
7. Key Enumeration and Rank Estimation
8. Statistical Tests

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):

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

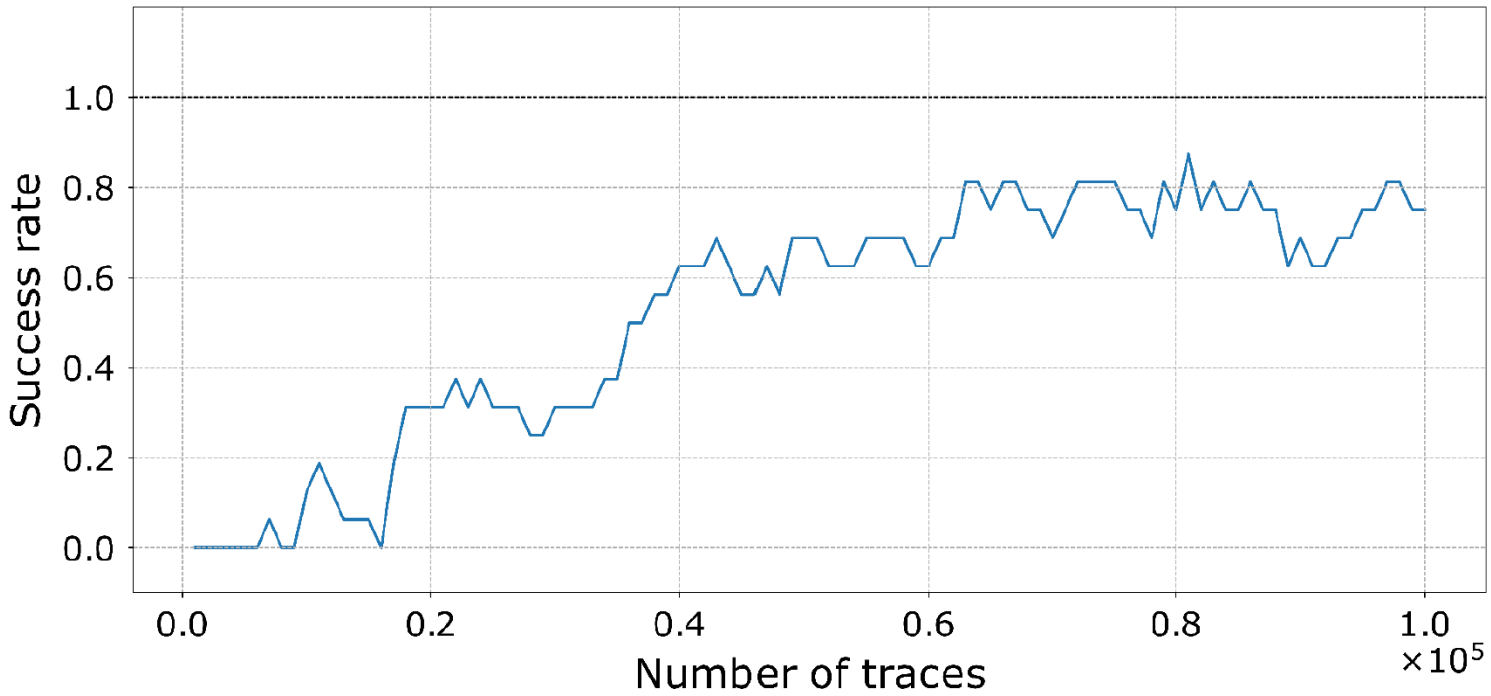
To ensure statistical stability, the final SR metric is estimated by averaging over p experiments:

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

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

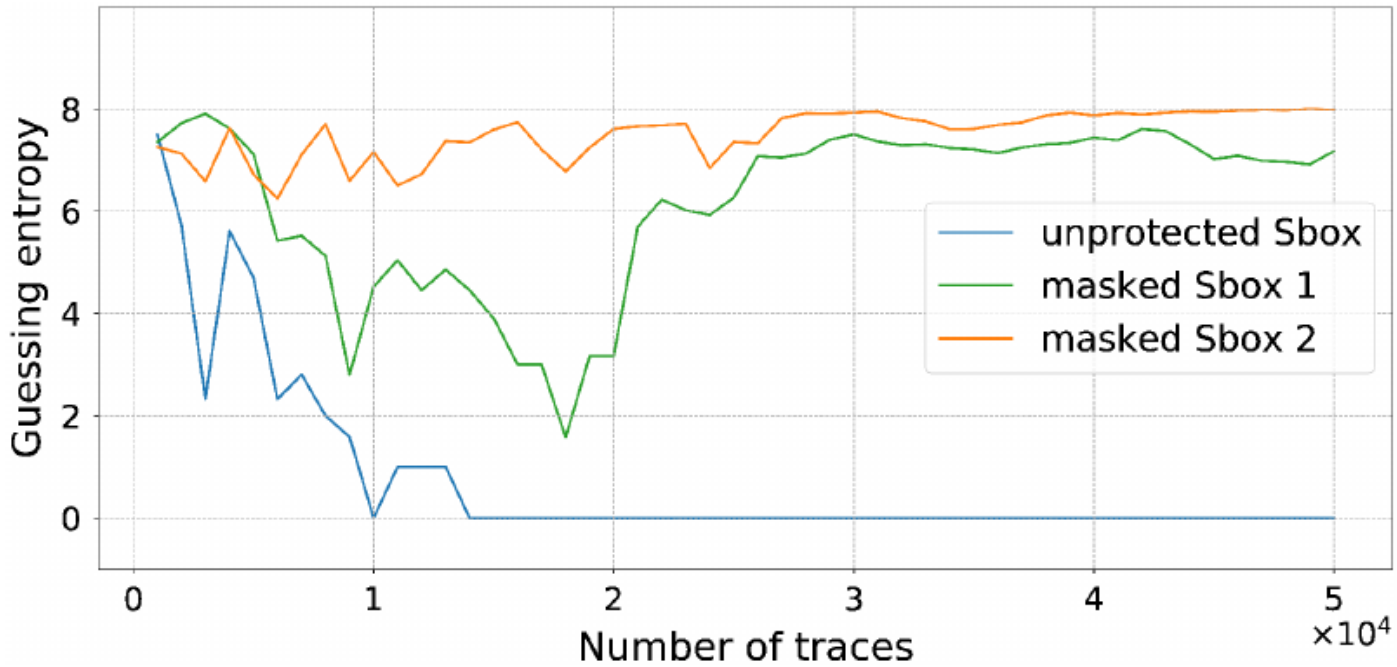


Figure: Guessing Entropy plot for three AES S-box implementations. GE drops to 0 for the unprotected S-box after roughly 15,000 traces; for masked S-boxes, GE remains high even with 50,000 traces.

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.

Table of Contents

1. Number of Traces
2. Signal to Noise Ratio
3. Score and Rank
4. Success Rate and Guessing Entropy
5. Correlation as a Metric
6. Information-Theoretic Metrics
   - Motivation for IT Metrics
   - Mutual Information
   - Hypothetical Information
   - Perceived Information
   - Limitations of IT Metrics
7. Key Enumeration and Rank Estimation
8. Statistical Tests

Correlation as a Metric: Motivation

Why Correlation?

Most side-channel metrics (SR, GE, score, rank) require success in the attack, but provide little information on how many traces are theoretically needed for success, especially if the attack fails.

- Correlation gives a way to estimate and project the effort needed to recover the key.
- It enables "security projections": estimating how many traces are needed for the correct key candidate to distinguish itself from others.
- This is particularly useful when the attack hasn't succeeded yet, allowing for predictive evaluation.

Pearson Correlation and Trace Projection

Pearson's Correlation in Side-Channel Analysis

In CPA (Correlation Power Analysis), Pearson's correlation coefficient ρ is computed between:

- The observed leakage traces $l_1, l_2, \ldots, l_n$
- The modeled leakages $g(f(x_i, k))$ for a given key candidate k

For the correct key k_c denote this as ρ_{k_c}

$$\rho_{k_c} = \frac{\sum_{i=1}^{n}(l_i - \frac{1}{n}\sum_{j=1}^{n} l_j)(v_i - \frac{1}{n}\sum_{i=j}^{n} v_j)}{\sqrt{\sum_{i=1}^{n}(l_i - \frac{1}{n}\sum_{j=1}^{n} l_j)^2}\sqrt{\sum_{i=1}^{n}(v_i - \frac{1}{n}\sum_{j=1}^{n} v_j)^2}}$$

where $v_i = g(f(x_i, k_c))$

When Does CPA Succeed?

Success Criterion

A CPA attack succeeds if $|\rho_{k_c}|$ for the correct key is noticeably higher than the absolute correlations for all incorrect key candidates.

- A common practical and theoretical assumption: for all incorrect keys, the expected correlation is close to zero.
- Thus, success is reduced to distinguishing ρ_{k_c} from zero (not from other possible nonzero values).

Why is this Useful?

If you know (or can estimate) the "true" value of ρ_{k_c} post-attack, you can predict how many traces n are required for reliable key recovery.

Mangard's Shortcut: Projecting the Number of Traces

Formula for the Projected Number of Traces

Given an estimate of ρ_{k_c}, the formula for predicting the number of traces needed for successful CPA key recovery is:

$$n' = 3 + 8 \left| \frac{z_\alpha}{\ln\left(\frac{1+\rho_{k_c}}{1-\rho_{k_c}}\right)} \right|^2$$

where:

- z_α is the z-score for desired type I error α (e.g., $z_{0.01}$ for 1% error);
- ρ_{k_c} is the estimated correlation for the correct key hypothesis.

As ρ_{k_c} decreases, the required number of traces grows quadratically. Only the correct key's correlation needs to be estimated: no need to compute all candidate correlations.

Correlation and Success Rate (SR) Projection

Instead of just estimating the number of traces needed for a break, we can predict the **Success Rate (SR)** of a CPA attack for a given number of traces.
This method, developed by Rivain, uses the statistical properties of correlation scores ρ for all key candidates.

The Custom Correlation Coefficient

The analysis uses a slightly different correlation coefficient, $\ddot{\rho}_k$, for each key candidate $k \in K$:

$$\ddot{\rho}_k = \frac{1}{n} \sum_{i=1}^{n} g(f(x_i, k)) \cdot l_i$$

The key candidate k that maximizes this coefficient is chosen as the best guess by the CPA attack.

Statistical Model of Correlation Scores

A Multivariate Normal Distribution

The core insight is that the vector of all correlation scores, $\ddot{\rho} = [\ddot{\rho}_0, \ddot{\rho}_1, \ldots, \ddot{\rho}_{|K|-1}]$, follows a **multivariate normal distribution**. This distribution is defined by a mean vector $\mu_{\ddot{\rho}}$ and a covariance matrix $\Sigma_{\ddot{\rho}}$.

The mean μ_k for each key candidate k is:

$$\mu_k = \frac{1}{|X|} \sum_{x \in X} g(f(x, k) \cdot \hat{\mu}_v$$

where $v = f(x, k_c)$

The covariance Σ_{ij} between candidates i and j is:

$$\Sigma_{ij} = \frac{1}{n'|X|} \sum_{x \in X} g(f(x, i)) \cdot g(f(x, j)) \cdot \hat{\sigma}_v^2 \quad \text{where } v = f(x, k_c)$$

Here, $\hat{\mu}_v$ and $\hat{\sigma}_v^2$ are the mean and variance of the leakage for a given intermediate value v, estimated from experimental traces.

Projecting SR: The Methodology

The core of the method is to model the attack's outcome statistically.

- First, the leakage statistics ($\hat{\mu}_v$ and $\hat{\sigma}_v^2$) are estimated using an experimental set of traces.
- A **projected number of traces**, n', is chosen. This value, which represents the strength of a hypothetical adversary, is used to define the covariance matrix $\Sigma_{\ddot{\rho}}$.
- Many random samples are then drawn from the resulting multivariate normal distribution, $\mathcal{N}(\mu_{\ddot{\rho}}, \Sigma_{\ddot{\rho}})$.
- Each sample vector represents a simulated attack outcome. It is sorted to find the rank of the correct key, $rank_{k_c}$.
- The **projected Success Rate (SR)** is the fraction of simulations where $rank_{k_c} = 1$. This can also be extended to find the projected SR_o.

Limitations of Correlation-Based Projections

- The projection formulas are based on **simplified statistical models**. The behavior of real hardware can deviate from these assumptions, due to electrical and electronic effects.
- **Estimation errors:** Imperfect statistical estimation of the correlation coefficient reduces the reliability of the results.
- **The purpose is Projection:** These are bounds designed to estimate effort when an attack has not yet succeeded, not to guarantee a break.

Table of Contents

1. Number of Traces
2. Signal to Noise Ratio
3. Score and Rank
4. Success Rate and Guessing Entropy
5. Correlation as a Metric
6. Information-Theoretic Metrics
   - Motivation for IT Metrics
   - Mutual Information
   - Hypothetical Information
   - Perceived Information
   - Limitations of IT Metrics
7. Key Enumeration and Rank Estimation
8. Statistical Tests

Motivation for IT Metrics (1): The Rise of Multivariate Attacks

Side-channel analysis evolved beyond simple CPA to include powerful **multivariate attacks** like template attacks and linear regression analysis. These advanced attacks can often recover a secret key with substantially fewer traces than their univariate counterparts.

The Problem

Metrics like correlation and SNR are not easily extended to a multivariate setting. This created the need for new metrics that could fairly assess the risk posed by these stronger attacks.

Motivation for IT Metrics (2): Separating Device from Attacker

A key desire behind IT metrics was to distinguish between two separate concepts:

1. **The implementation:** a device and its countermeasures, which inherently leak information.
2. **The attacker:** the specific methods and capabilities used to exploit that information.

Instead of asking "Can my specific attack succeed?", IT metrics aim to answer:

"How secure is this implementation against side-channel attacks in general?"

This shifts focus away from attack-specific metrics (like SR and GE), allowing for more generic conclusions about the security of a device.

Information Theory Basics (1): Entropy

Entropy H(X)

Entropy measures the average uncertainty of a random variable. Before an attack, we have an initial uncertainty about the secret key, K.
For a discrete variable, it is defined as:

$$H(X) = -\sum_{x \in \mathcal{X}} \Pr(X = x) \cdot \log_2(\Pr(X = x))$$

The result is measured in **bits**.

Example: An AES Key Byte

If the secret key byte K is uniformly random, there are 256 possibilities.
The initial entropy is:

$$H(K) = \log_2(256) = 8 \text{ bits}$$

This represents our total lack of knowledge before observing any leakage.

Information Theory Basics (2): Conditional Entropy

Conditional Entropy $H(X|Y)$: Remaining Uncertainty

Conditional Entropy measures the uncertainty of a variable X **given that you already know** the value of another variable Y.

What is the remaining uncertainty of the key **K**, *given that we have observed the leakage* **L**?

This is written as $\mathbf{H}(\mathbf{K}|\mathbf{L})$. From a security perspective, we want this value to be as high as possible after an attack.

$$H(X|Y) = -\sum_{x \in X, y \in Y} \Pr(X = x, Y = y) \cdot \log(\Pr(X = x|Y = y))$$

Information Theory Basics (3): Continuous Variables

Since side-channel leakage is often an analog signal, we extend the concept of entropy to continuous random variables.
For a variable X with a probability density function (PDF) $\Pr(x)$, it is defined as:

$$H(X) = -\int_X \Pr(X = x) \log_2(\Pr(X = x))\, dx$$

The joint and conditional entropies for continuous variables 'X' and 'Y' are defined similarly:

Joint Entropy:

$$H(X, Y) = -\iint_{X,Y} \Pr(X = x, Y = y) \log_2 \Pr(X = x, Y = y)\, dxdy$$

Conditional Entropy:

$$H(X|Y) = -\iint_{X,Y} \Pr(X = x, Y = y) \log_2 \Pr(X = x|Y = y)\, dxdy$$

Information Theory Basics (4): Visual analogy

placeholder.png

Mutual Information

Definition

Mutual Information quantifies the amount of information that one variable (Y) provides about another (X). It is the **reduction in uncertainty** of X after observing Y.
It is fundamentally defined by the relationship between the entropies:

$$I(X;Y) = H(X) - H(X|Y)$$

Theory

When applied to our problem, the formula becomes:

$$I(K;L) = H(K) - H(K|L)$$

$$\text{Information Leaked} = \text{Initial Uncertainty} - \text{Remaining Uncertainty}$$

This metric, $I(K;L)$, represents the total amount of information (in bits) that the leakage L reveals about the secret key K.

Mutual Information: Expanded Formulas

For discrete random variables X and Y:

$$I(X;Y) = \sum_{x \in X} \sum_{y \in Y} \Pr(X = x, Y = y) \cdot \log_2 \left(\frac{\Pr(X = x, Y = y)}{\Pr(X = x) \Pr(Y = y)} \right)$$

For continuous random variables X and Y (using probability density functions):

$$I(X;Y) = \int_X \int_Y \Pr(X = x, Y = y) \cdot \log_2 \left(\frac{\Pr(X = x, Y = y)}{\Pr(X = x) \Pr(Y = y)} \right) dx\, dy$$

Mixed Case (Discrete X, Continuous Y)

$$I(X;Y) = \sum_{x \in X} \int_Y \Pr(X = x, Y = y) \cdot \log_2 \left(\frac{\Pr(X = x, Y = y)}{\Pr(X = x) \Pr(Y = y)} \right) dy$$

Mutual Information: Visual Analogy

placeholder.png

Applying MI to Side-Channels

We want a metric that can capture **multivariate leakages**. To do this, we move from thinking about a single leakage value L to a leakage **vector L**.

The goal is to use Mutual Information to quantify how many bits of information we can learn about the secret key K by observing the multivariate side-channel leakage vector **L**.

The definition remains the same, but the variables now have a specific meaning:

$$I(K; \mathbf{L}) = H(K) - H(K|\mathbf{L})$$

Information Leaked = Initial Uncertainty − Uncertainty after the Attack

The MI Formula for Side-Channels

Applying the definitions of entropy and Bayes' rule, the MI between the key K and the leakage $\mathbf{L}$ can be expressed as:

$$I(K;\mathbf{L}) = -\sum_{k\in K} \Pr(k)\log_2 \Pr(k) + \sum_{k\in K} \Pr(k) \int_{\mathbf{l}\in\mathbb{R}^m} \Pr(\mathbf{l}|k)\log_2 \Pr(k|\mathbf{l})\, d\mathbf{l}$$

What we need to compute

To solve this, an evaluator needs to know several quantities:

- The key probability, $\Pr(k)$.
- The conditional leakage probability, $\Pr(\mathbf{l}|k)$.
- The posterior key probability, $\Pr(k|\mathbf{l}) \rightarrow$ we can get rid of this thanks to Bayes

Simplifying the MI Formula

In the very common case where the secret key K is **uniformly random**:

- The probability of any specific key is $\Pr(k) = 1/|K|$.
- The initial entropy is simply $H(K) = \log_2 |K|$.

Simplified Formula

Under this assumption, the MI formula simplifies significantly:

$$I(K; L) = \log_2 |K| + \frac{1}{|K|} \sum_{k \in K} \int_{\mathbf{l} \in \mathbb{R}^m} \Pr(\mathbf{l}|k) \log_2 \frac{\Pr(\mathbf{l}|k)}{\sum_{k^* \in K} \Pr(\mathbf{l}|k^*)} d\mathbf{l}$$

After all the simplification, the formula reveals that the **only remaining quantity** the evaluator needs to compute is:

$$\Pr(\mathbf{l}|k)$$

This term is the likelihood of observing a side-channel leakage vector $\mathbf{l}$, given a certain key k.

The computation of this quantity is directly linked to **leakage profiling**, **leakage modeling** and parameter estimation.

Modeling the Leakage to Compute $\Pr(\mathbf{l}|k)$

To find a value for $\Pr(\mathbf{l}|k)$, we create a statistical model of the leakage. A common approach, known as a **univariate template**, assumes the leakage for each key k follows a **Normal (Gaussian) Distribution**.

1. This model is built using the workflow of a profiled attack, so the evaluator can use an open device to measure traces $[L_0^{prof}, L_1^{prof}, \ldots, L_{255}^{prof}]$
2. For normal distributions, leakage modeling is equivalent to estimating the mean (μ_k) and standard deviation (σ_k) from the set of profiling traces.
3. This gives us a list of leakage models $[\mathcal{N}(\mu_0, \sigma_0), \ldots, \mathcal{N}(\mu_{255}, \sigma_{255})]$.

The Univariate Template

With the univariate template assumption, the probability of observing leakage **l** for a given key k is given by the Gaussian probability density function:

$$\Pr(l|k) = \frac{1}{\sigma_k \sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{l-\mu_k}{\sigma_k}\right)^2}$$

By plugging this concrete model for $\Pr(l|k)$ into the simplified MI formula we saw earlier, we can now calculate a numerical value for the information leakage.

$$I(K;L) = \log_2 |K| + \frac{1}{|K|} \sum_{k \in K} \int_{-\infty}^{+\infty} \Pr(l|k) \log_2 \frac{\Pr(l|k)}{\sum_{k^* \in K} \Pr(l|k^*)} dl$$

Using MI to make projections on security

Mutual Information allows deriving security bounds, projecting the number of traces an attacker would need for key recovery.

Unprotected Implementation:

$$\text{Number of traces} \geq \frac{H(K)}{I(L;K)}$$

d-th Order Masked Implementation:

$$\text{Number of traces} \geq \frac{H(Y_i)}{I(L_i;Y_i)\cdot d}$$

These formulas are approximations that may overestimate an attacker's capabilities. However, because MI is inherently multivariate, they can provide a more fair security picture than correlation-based shortcuts.

Visualizing Security with MI

The value of MI directly reflects the security of a device, independent of any specific attacker's scores or ranks.

- **High MI** $\Longrightarrow$ High information leakage $\Longrightarrow$ Low security.
- **Low MI** $\Longrightarrow$ Low information leakage $\Longrightarrow$ High security.

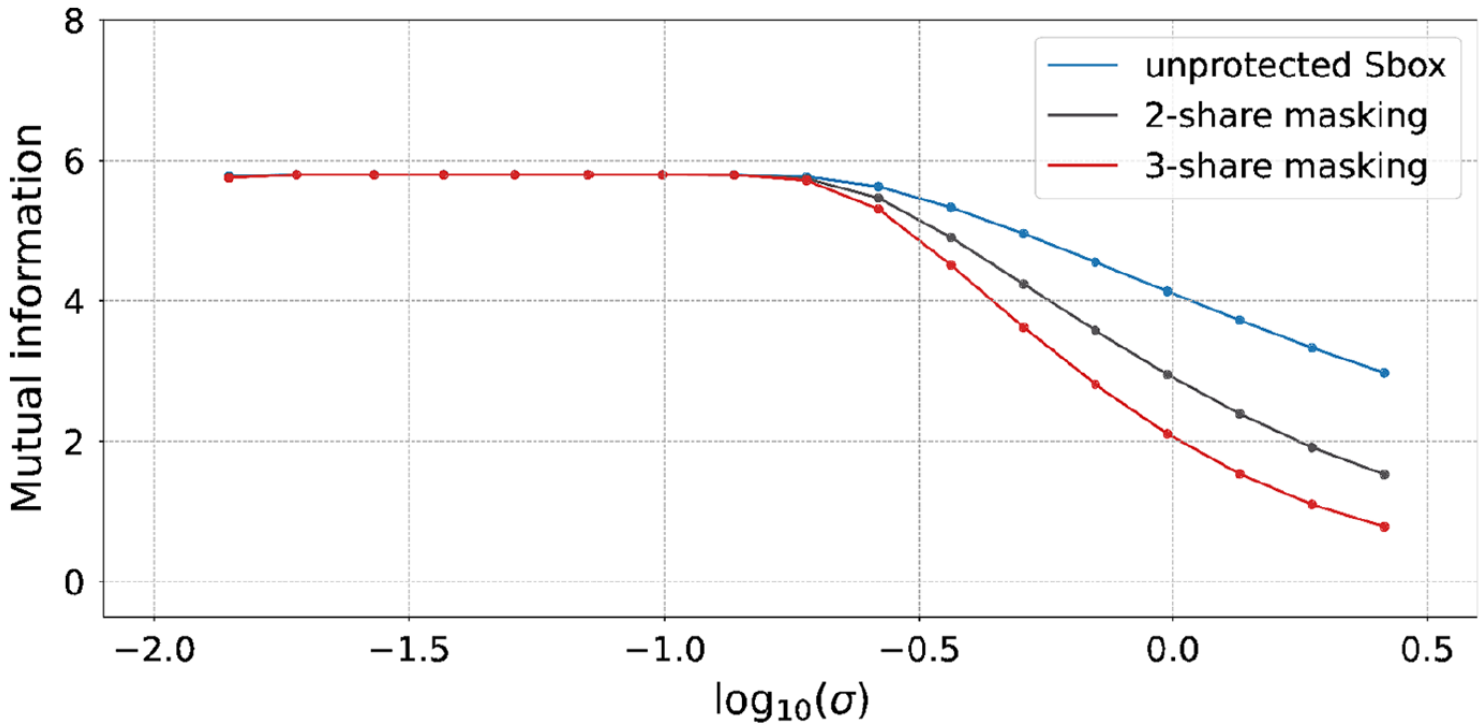


Figure: MI decreases as noise increases. Masking further amplifies this effect, showing its effectiveness as a countermeasure.

The Leakage Model: Two Scenarios

Scenario 1: Theoretical Analysis (True Parameters)

Sometimes, an evaluator wants to test a countermeasure against a **theorized leakage model** without using experimental traces.

- **Example:** Assessing the security of a masking scheme against a perfect, non-linear leakage model.
- In this case, the model parameters (e.g., μ, σ) are not estimated, but are **chosen** by the evaluator. These are considered the *true parameters* for the theoretical model.

Scenario 2: Practical Evaluation (Estimated Parameters)

This is the case we discussed before. An evaluator uses a set of experimental profiling traces, L_k^{prof}, to **estimate** the model parameters $(\hat{\mu}, \hat{\sigma})$.

Introducing Hypothetical Information (HI)

Distinction in Terminology

To separate these two scenarios, the following terms are used:

- **Mutual Information (MI):** The metric calculated using **true parameters** from a theoretical leakage model. This achieves the goal of separating the implementation from the adversary.
- **Hypothetical Information (HI):** The metric calculated using **estimated parameters** $(\hat{\mu}, \hat{\sigma})$ from a real profiling dataset. This is what's computed in a practical evaluation.

The Pitfalls of Hypothetical Information (HI)

Unfortunately, an attacker's poor modeling is not fully reflected in the HI metric, which leads to two major types of errors.

1. **Estimation Errors**
   - Occur when the model is correct, but trained on **too few traces**.
   - The resulting model is weak and may fail in a real attack.
   - HI will still be positive and suggest that key recovery is possible, even when it is not.
2. **Assumption Errors**
   - Occur when the **chosen model is wrong** and does not reflect the device's actual leakage behavior.
   - This can happen due to non-linear effects, measurement drift, or device differences.
   - Even with infinite training data, the model is flawed. Yet, HI will likely be positive, suggesting an attack is possible when it's not.

The Fundamental Flaw of HI

The problem with HI is that it never **tests its own model against reality**.

- HI is calculated using only the profiling and modeling steps.
- It never considers a **test dataset** to see if the model's predictions actually match the device's real leakage.
- HI is "the amount of information that would be leaked from an implementation of which the leakages would be *exactly predicted*" by the model.

HI does not test its estimated model against the actual device leakage. This gap is filled by a new metric that explicitly compares the model to reality: **Perceived Information (PI)**.

Cross Entropy

PI is built on the idea of **cross-entropy** $H_{p,q}(K|\mathbf{L})$. It measures the performance of a model distribution q when it's used to describe a true distribution p.

- The **model distribution** q is our estimated model, $\hat{Pr}_{model}(\mathbf{l}|k)$, built from the profiling traces.
- The **true distribution** p is the actual leakage of the device, which we can only sample using a **test data set**.

Introducing Perceived Information (PI)

PI Definition

Perceived Information is defined as:

$$PI(\mathbf{L}; K) = H(K) - H_{true,model}(K|\mathbf{L})$$

It subtracts the *cross-entropy* between the true leakage and the model from the initial key entropy.

$$PI(\mathbf{L}; K) = (-\sum_{k \in K} \Pr(k) \log_2 \Pr(k)) - (-\sum_{k \in K, l \in L} \Pr_{true}(k|\mathbf{l}) \log_2 \hat{\Pr}_{model}(k|\mathbf{l}))$$

Computing PI

Assuming uniformly random keys and applying Bayes' rule, we get the general formula for Perceived Information:

$$PI(L;K) = H(K) + \sum_{k} \Pr(k) \sum_{l \in L_k^{test}} \Pr_{true}(l|k) \log_2 \left(\frac{\hat{\Pr}_{model}(l|k)}{\sum_{k^*} \hat{\Pr}_{model}(l|k^*)} \right)$$

- Computing $\hat{\Pr}_{model}(l|k)$ is identical to the computation of $\Pr(\hat{l}|k)$ in the HI metric
- To compute $\Pr_{true}(k|l)$ we distinguish between **simulated leakage** and **real traces**

Case 1: PI with Simulated Leakage

Theoretical Scenario

In a theoretical analysis, the evaluator can **define** the true leakage distribution. For example, simulated traces can be generated with known characteristics (known noise level, ...).

- Here, $\Pr_{true}(l|k)$ is known because the evaluator created it.
- The PI metric then quantifies how well the estimated model, $\hat{\Pr}_{model}$, can capture this known, true distribution.

This scenario answers the question: *"Is my training process good enough to learn this specific kind of leakage?"*
A negative PI value in this case clearly indicates an **estimation error** (e.g., not enough training traces).

Visualizing Security with PI

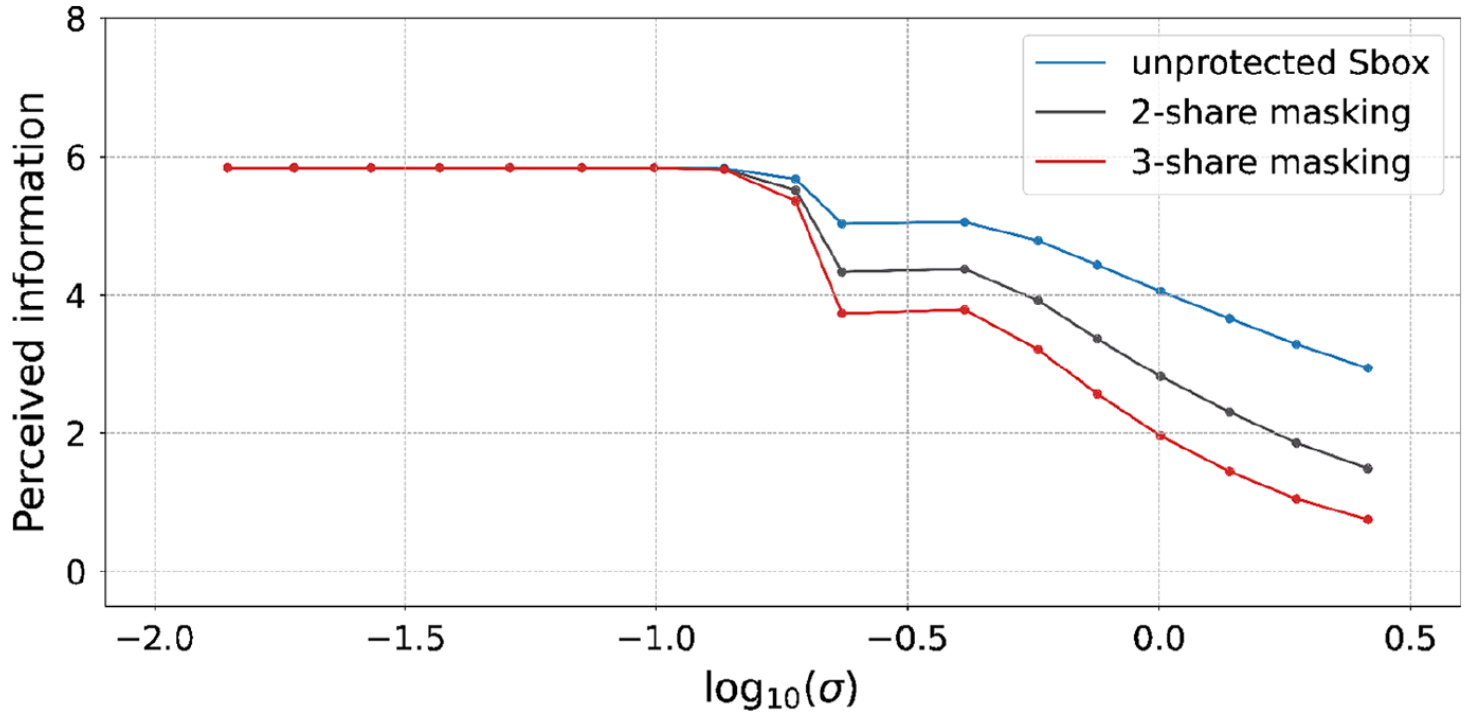


Figure: PI is positive and decreases as artificial white noise increases. The model training adequately captures the simulated leakage

Case 2: PI with Real Traces

Practical Scenario

In a practical evaluation of a real device, the true distribution is unknown. We can only **sample** it by collecting a **test dataset**, L_k^{test}.

- For each possible key k, we capture a new set of n_k test traces.
- Sampling from the test set implies that the probability of observing any specific trace l from that set is $\Pr_{true}(l|k) = 1/n_k$.

The Practical PI Formula

This leads to the final, practical formula for PI:

$$PI(L;K) = H(K) + \sum_{k \in K} \frac{\Pr(k)}{n_k} \sum_{l \in L_k^{test}} \log_2 \left(\frac{\hat{\Pr}_{model}(l|k)}{\sum_{k^* \in K} \hat{\Pr}_{model}(l|k^*)} \right)$$

Comparison: Real and Simulated traces

What Practical PI tells us

This metric directly answers the most important question for an evaluator:

"Does my trained model capture the **actual device leakage** *well enough to be useful?"*

Detecting Errors

- Unlike MI and HI, **PI can be negative**.
- A negative PI value proves that key recovery is not possible with the current model. It serves as a clear indicator of:
  - **Estimation errors** (not enough training data)
  - OR **Assumption errors** (the model is fundamentally wrong)

Summary: The Hierarchy of IT Metrics

The three information-theoretic metrics are bound by the following important inequality:

$$PI \leq MI \leq HI$$

The security bounds we saw earlier (for estimating the number of traces) can be computed with any of these metrics, but a calculation using PI is only meaningful **if the PI value is positive**.

Limitations of Information-Theoretic Metrics

1. **Requires Generative Models:** The need for a probabilistic generative model (one that can describe the full leakage distribution) can exclude powerful discriminative models, like many neural network classifiers.

2. **Computational Hardness with High Dimensions:** IT metrics are naturally multivariate, but computing the required multi-dimensional integrals becomes extremely difficult as the number of leakage samples m or masking shares d increases.

3. **Difficulty with Horizontal Attacks:** Combining the leakage from multiple different intermediate values in the cipher (e.g., Sbox input and output) also dramatically increases the dimensionality and computational complexity.

To resolve these issues evaluators often turn to techniques like Monte-Carlo integration, dimensionality reduction, or other approximations.

Table of Contents

1. Number of Traces
2. Signal to Noise Ratio
3. Score and Rank
4. Success Rate and Guessing Entropy
5. Correlation as a Metric
6. Information-Theoretic Metrics
   - Motivation for IT Metrics
   - Mutual Information
   - Hypothetical Information
   - Perceived Information
   - Limitations of IT Metrics
7. Key Enumeration and Rank Estimation
8. Statistical Tests

The Full Key Recovery Problem

The divide-and-conquer strategy gives us scores and ranks for each key **part** (e.g., each byte of an AES key).

- If the correct byte is Rank 1 for all 16 parts, recovering the full key is trivial.
- But what if some correct bytes have a lower rank? The attacker must test many combinations.

The Central Question

How do we translate the security metrics for 16 independent 8-bit key parts into a single, meaningful security metric for the full 128-bit key?

This is solved by two related, but distinct, processes: **Key Enumeration** and **Rank Estimation**.

Key Enumeration: The Attacker's Approach

Key Enumeration is a practical process performed by an attacker in an **unknown-key** scenario.

- **Input:** The full set of scores for *all* key candidates for *all* key parts (e.g., 16 lists of 256 scores for AES-128).
- **Process:** Systematically construct and test full key candidates, starting with the most probable combinations derived from the scores.
- **Limit:** The process stops after a predefined number of attempts, n_{en}, which reflects the attacker's computational budget.

Binary Outcome

The result is a simple success or failure:

- **Success:** The correct full key is found within n_{en} trials.
- **Failure:** The key is not found, and the attacker gives up.

Key Enumeration

Key Enumeration is a practical process performed by an attacker in an **unknown-key** scenario.

- **Input:** The full set of scores for *all* key candidates for *all* key parts (e.g., 16 lists of 256 scores for AES-128).
- **Process:** Systematically construct and test full key candidates, starting with the most probable combinations derived from the scores.
- **Limit:** The process stops after a predefined number of attempts, n_{en}, which reflects the attacker's computational budget.

Key Enumeration

Measuring the Outcome

The outcome is a precise security level measured in bits of effort:

- **If the key is found in $n_e \leq n_{en}$ trials**, the security of the device against this attack is exactly:

$$\text{security} = \log_2(n_e) \text{ bits}$$

- **If the key is not found within the limit**, the security is only known to be greater than the attacker's effort:

$$\text{security} > \log_2(n_{en}) \text{ bits}$$

Rank Estimation

When key enumeration fails after n_{en} attempts, the evaluator is left with a massive uncertainty gap. The true security could be anywhere from n_{en} to 2^{128} (using AES-128 as our target).

Rank Estimation

To overcome this, the process of **Rank Estimation** was proposed. It is typically a **known-key analysis** that can:

- Quickly estimate the remaining brute-force effort for full key recovery.
- Achieve this without performing the costly process of listing all possible keys.
- Provide a precise security level even when the attack is not fully successful.

Rank Estimation: The Probabilistic Framework

To highlight the process, we use the **histogram-convolution-based algorithm** of Glowacz as our reference.

From Scores to Log-Probabilities

The process begins with the list of probabilities for each key candidate, which are converted to scores in the log domain:

$$\text{score}_k^i = \log \Pr(k_i)$$

where i is the key part $(1, \ldots, p)$ and k is the candidate $(0, \ldots, m-1)$. Assuming independence between key parts, the score for a full key is the sum of the scores of its parts:

$$\Pr(\text{fullkey}) = \prod_{i=1}^{p} \Pr(k^i) \quad \Longleftrightarrow \quad \log \Pr(\text{fullkey}) = \sum_{i=1}^{p} \log \Pr(k_i)$$

Algorithm: Histogram-Based Rank Estimation

Input: score_k^i, for $i = 1, \ldots, p$ and $k = 0, \ldots, m-1$
Output: estimated rank, security (in bits)
for $i = 1$ to p **do**
    $H^i \leftarrow \text{hist}([\text{score}_0^i, \ldots, \text{score}_{m-1}^i])$
end for
$H \leftarrow H_1$
for $i = 2$ to p **do**
    $H \leftarrow \text{conv}(H, H^i)$
end for
$lp_c \leftarrow \sum_{i=1}^{p} \log \Pr(k_c^i)$
$\text{estimated\_rank} \approx \sum_{j=\text{bin}(lp_c)}^{\text{max\_bin}} H(j)$
$\text{security} \leftarrow \log_2(\text{estimated\_rank})$

The Histogram-Based Method: Step-by-Step

The algorithm of Glowacz follows a three-step process to estimate the full key rank:

1. **Create Histograms:** For each of the p key parts, compute a histogram of its m scores (log-probabilities) using a fixed number of bins, n_b. This produces p individual score distributions, $H_1, \ldots, H_p$.
2. **Convolve Histograms:** Iteratively convolve the p histograms to produce a single final histogram, H. This final histogram represents the score distribution for the full key. This process introduces a small, unavoidable *quantization error*.
3. **Calculate the Rank:** Compute the total score of the correct key, $lp_c = \sum_{i=1}^{p} \text{score}_{k_c^i}$. Find which bin this score falls into, $\text{bin}(lp_c)$, and sum the counts of all bins corresponding to higher scores. This sum is the estimated rank.

Handling Quantization Error: Lower and Upper Bounds

The use of histograms introduces quantization errors. To account for this, the estimated rank is bounded to provide a reliable security range.

Rank Bounding Formulas

Given the final convoluted histogram H, the number of key parts p, and the correct key's score bin $\text{bin}(lp_c)$, the bounds are:

Lower Bound:

$$\text{rank}_{\text{lower}} = \sum_{j=\text{bin}(lp_c)+p}^{\text{max\_bin}} H(j)$$

Upper Bound:

$$\text{rank}_{\text{upper}} = \sum_{j=\text{bin}(lp_c)-p}^{\text{max\_bin}} H(j)$$

These bounds provide a tight and accurate estimation of the true key rank, as shown in experimental results on both simulated and real-world devices.

A Tempting (But Wrong) Shortcut

It is tempting to estimate the full key rank by simply multiplying the ranks of the correct key parts (or, equivalently, summing their log-ranks).

This is Incorrect and Misleading

This approach should be avoided for several reasons:

- It does not correspond to any valid key enumeration strategy an attacker would actually use.
- It falsely assumes the attacker has prior knowledge of the individual key part ranks, which they do not.
- As a result, it provides a **pessimistic lower bound** on the true rank, making the attack seem more effective than it is and underestimating the device's security.

A Tempting (But Wrong) Shortcut

Consider a 4-bit key attacked as two 2-bit parts, K_1 and K_2, with the following probabilities:

- $\Pr(K_1) = [00 : 0.8, \mathbf{01{:}0.1}, 10 : 0.08, 11 : 0.02]$
- $\Pr(K_2) = [\mathbf{00{:}0.09}, 01 : 0.5, 10 : 0.01, 11 : 0.3]$

The correct full key is $[01, 00]$.

Shortcut Calculation:

- The rank of $k_c^1 = 01$ is 2.
- The rank of $k_c^2 = 00$ is 3.
- Rank product $= 2 \times 3 = 6$.

Optimal Enumeration:

- An optimal enumeration strategy would list all 16 full keys by their true combined probability.
- The correct key $[01, 00]$ is found at position 9.
- The true rank is 9.

Limitations of Enumeration and Rank Estimation

The optimality and soundness of these techniques depend on a critical condition: the attack scores must represent probabilities or log-probabilities.

- The underlying math assumes that scores for independent key parts can be combined via multiplication (for probabilities) or addition (for log-probabilities) to find the full key's probability.
- **Non-Bayesian scores**, such as raw correlation coefficients, have no such mathematically correct combination function.
- Applying simple heuristics like adding or multiplying these scores leads to inaccurate full key scores and a **suboptimal** enumeration or rank estimation.

Practical Mitigation

Since Bayesian scores are not always available in practice, this limitation can be addressed with other heuristics, such as a **Bayesian extension**, which attempts to convert non-probabilistic scores into a usable format.

Table of Contents

1. Number of Traces
2. Signal to Noise Ratio
3. Score and Rank
4. Success Rate and Guessing Entropy
5. Correlation as a Metric
6. Information-Theoretic Metrics
   - Motivation for IT Metrics
   - Mutual Information
   - Hypothetical Information
   - Perceived Information
   - Limitations of IT Metrics
7. Key Enumeration and Rank Estimation
8. Statistical Tests

Leakage Detection

Historically, side-channel research focused on creating new model-based and profiling attacks (e.g., DPA, CPA, Template Attacks).

New Focus: Leakage Detection

Rather than developing new key-recovery attacks, the field is increasingly asking:

"Does a cryptographic device leak secret-dependent information at all?"

The aim is to provide a reliable, attack-agnostic answer, supporting the validation of countermeasures.

What is TVLA?

Test Vector Leakage Assessment (TVLA) is a standardized methodology for detecting side-channel leakage in a device without attempting full key recovery.

- TVLA analyzes whether a device leaks sensitive information when running known cryptographic algorithms.
- Instead of targeting unknown key bytes, TVLA tests whether any observable variable is correlated with any aspect of the secret.
- TVLA is especially useful for validating the **effectiveness of countermeasures**.

The t-test in TVLA: The Goal and Hypothesis

The goal of the TVLA t-test is to formally certify a device's security status. The test uses two sets of traces captured with a fixed key:

- $\mathbf{L_r}$: A set of n_r traces from **random** plaintexts.
- $\mathbf{L_f}$: A set of n_f traces from a **fixed** plaintext.

The Statistical Hypothesis

The abstract question "is the device secure?" is rephrased into a precise and testable hypothesis comparing the means ($\hat{\mu}$) of the two trace sets:

- **Null Hypothesis (H_0):** The means are equal. The device does not leak information detectable by this test.

$$H_0 : \hat{\mu}_r = \hat{\mu}_f$$

- **Alternative Hypothesis (H_1):** The means are different. The device exhibits leakage.

$$H_1 : \hat{\mu}_r \neq \hat{\mu}_f$$

Welch's t-test: The Calculation

To test the hypothesis, the evaluator uses the two-tailed Welch's t-test, which computes two key values for each time sample in the traces.

- **The t-statistic**, which measures the difference between the means relative to their variance:

$$t = \frac{\hat{\mu}_r - \hat{\mu}_f}{\sqrt{\frac{\hat{\sigma}_r^2}{n_r} + \frac{\hat{\sigma}_f^2}{n_f}}}$$

- **The degrees of freedom (df)**, which defines the shape of the t-distribution:

$$df = \frac{\left(\frac{\hat{\sigma}_r^2}{n_r} + \frac{\hat{\sigma}_f^2}{n_f}\right)^2}{\frac{\hat{\sigma}_r^4}{n_r^2(n_r-1)} + \frac{\hat{\sigma}_f^4}{n_f^2(n_f-1)}}$$

where the sample means and variances are $\hat{\mu}_i = \frac{1}{n_i}\sum_{l \in L_i} l$ and $\hat{\sigma}_i^2 = \frac{1}{n_i-1}\sum_{l \in L_i}(l - \hat{\mu}_i)^2$.

Interpreting the Result: The Decision and Errors

The calculated t-statistic is an instance of a random variable T that follows a Student's t-distribution. We use it to make a decision.

The Decision Rule

The null hypothesis H_0 is **rejected** if the absolute value of the t-statistic exceeds a predefined threshold, th:

$$|t| > th$$

A rejection means leakage has been detected.

Interpreting the Result: Errors

Controlling for Errors

- A **Type I Error** (false positive) occurs when we reject H_0 even though the device is secure.
- TVLA seeks to control this error probability at a significance level α.
- This is the probability of detecting flaws when none exist:

$$\alpha = \Pr(\text{reject } H_0 | H_0 \text{ is true}) = 2\Pr(t > th)$$

- TVLA suggests a strict threshold of $\alpha = 10^{-5}$ to avoid this costly error.

Setting Thresholds and Understanding Errors

For the recommended $\alpha = 10^{-5}$ and with a high number of traces (degrees of freedom > 1000), the threshold becomes a well-known value:

$$th \approx 4.5$$

False Negatives

- A **Type II Error** (β) is the failure to reject H_0 when it is false (i.e., declaring a device secure when it actually has flaws).
- The standard TVLA process focuses on controlling the Type I error (α) and conducts independent tests on every sample point of the traces.

Using the t-test as a Metric

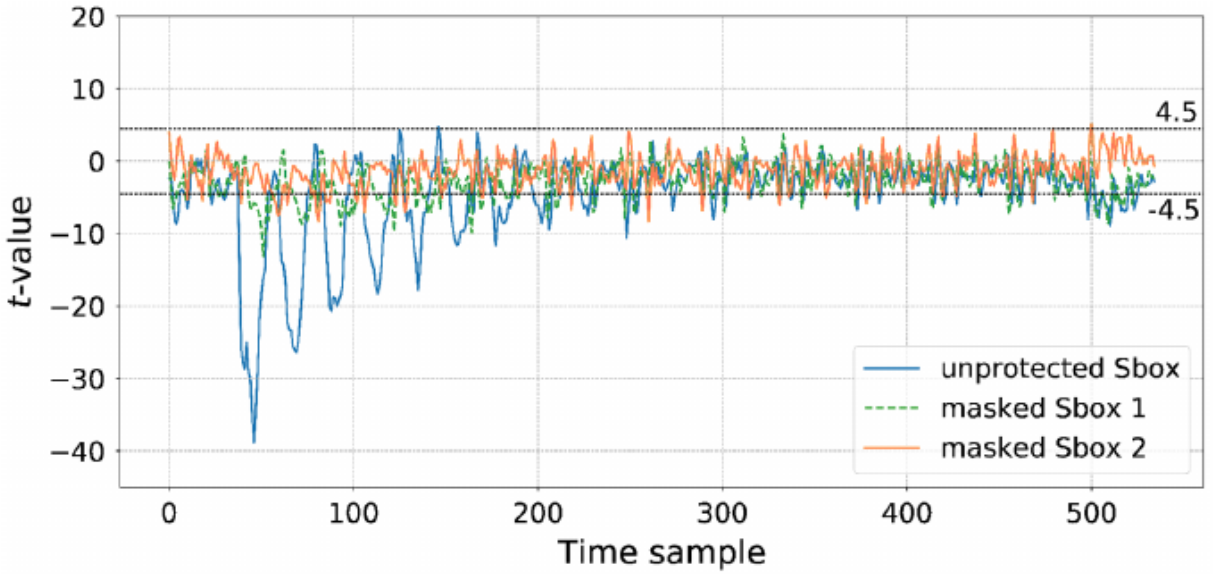
While formally a statistical test, the t-value is often used as a practical metric.

- Plotting the t-statistic over time provides a clear visual of leakage.
- As expected, unprotected implementations show high t-values, but flaws can also be detected in masked implementations, revealing practical imperfections.

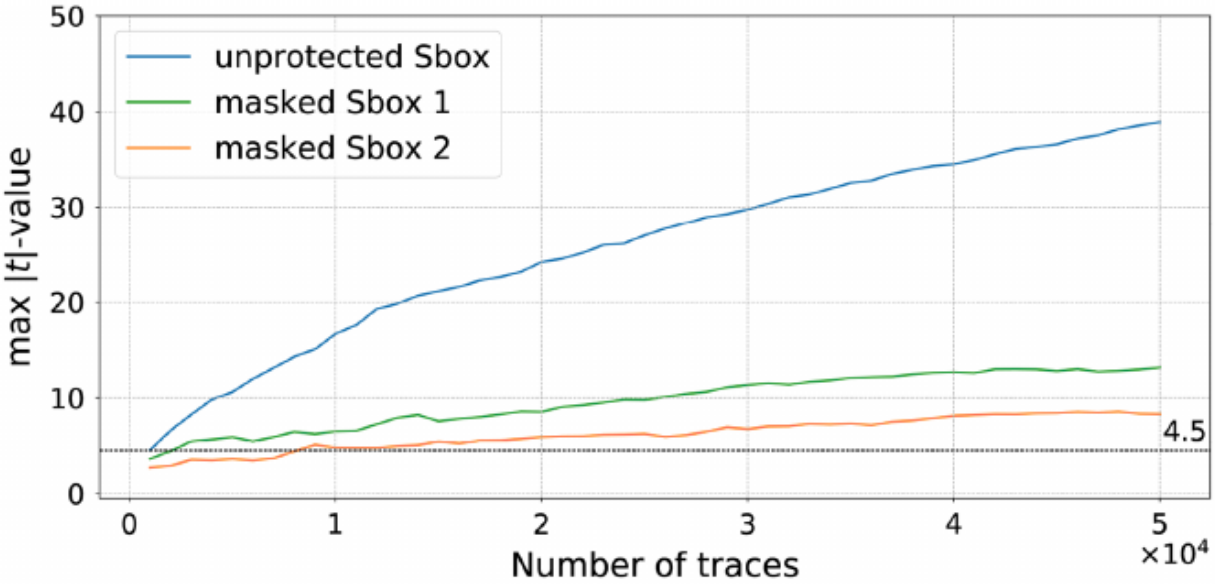
A Heuristic for Further Analysis

High absolute t-values (or low p-values) indicate leaky time samples. This allows the evaluator to use the t-test as a guide for:

- Selecting specific points-of-interest for attacks.
- Performing more detailed leakage profiling on the most vulnerable parts of the operation.



(a) t-value



(b) Maximum absolute t-value

Trade-off for the experimenter

A key insight is that the t-value's significance depends more on the **parameters of the experiment** than on the results alone.

The relationship between the number of traces (n), error rates (α, β), signal variance ($\hat{\sigma}^2$), and the detectable effect size (δ) is given by:

$$n = \frac{(\hat{\sigma}_r^2 + \hat{\sigma}_f^2)(z_{\alpha/2} + z_\beta)^2}{\delta^2}$$

Where $z_{\alpha/2}$ and z_β are quantiles of the standard normal distribution.

This means any use of the t-value as a metric implicitly depends on all these parameters, which the evaluator must set in a meaningful way beforehand.

Variations and Best Practices for TVLA

The standard "random vs. fixed" test is powerful but has limitations and variations.

- **Limitation:** A negative result (no leak detected) from a single test is not conclusive, as it depends on the chosen fixed plaintext. **It is recommended to repeat the test with several different fixed inputs.**
- **Semi-fixed vs. random test:** A variation better suited for devices where the exact start/end of crypto operations is unclear.
- **Higher-order leakage:** The test can be adapted to assess masked implementations by first pre-processing traces to target higher-order statistical moments.
- **Specific tests:** If a non-specific test detects leakage, a specific test (requiring the key) can be used to confirm the finding with higher confidence by targeting a particular intermediate value.