



POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Design, Implementation and Experimental Validation of FreeSCA, a Python Library for Side Channel Analysis

Tesi di Laurea Magistrale in
Ingegneria Informatica
Computer Science and Engineering

Author: **Valentina Borghi**

Student ID: 10712890
Advisor: Prof. Alessandro Barenghi
Co-advisors: Dott. Marco Gianvecchio
Academic Year: 2025-26

Abstract

The security assessment of cryptographic devices cannot be limited to the mathematical analysis of the implemented algorithms. Physical implementations may reveal information through observable effects such as timing variations, electromagnetic emissions, or power consumption. For this reason, side channel analysis plays a central role in the evaluation of device security. In experimental settings, this evaluation involves multiple stages, including dataset handling, preprocessing, leakage detection, feature reduction and attack execution. These tasks are often managed through separate tools and heterogeneous data sources, making the overall workflow less uniform, less reusable and more difficult to reproduce. This thesis addresses this problem through the design and implementation of FreeSCA, a modular Python library for side channel analysis. The framework integrates the main stages of the analysis workflow within a common structure and introduces the FSCA trace format for the uniform management of traces and associated metadata, conversion tools for heterogeneous data sources, and an incremental processing approach suited to realistic experimental conditions. Within this framework, a representative set of methods for practical side channel evaluation is implemented, including different variants of Welch's t-test for statistical leakage detection, Correlation Power Analysis, Template Attack and feature reduction procedures for high dimensional trace data. The framework is validated experimentally on masked and unmasked datasets associated with AES-128 implementations running on an ARM Cortex-M4 microcontroller. The results show that the proposed library supports the main stages of a practical side channel workflow in a consistent manner. The implemented statistical methods highlight meaningful differences between protected and unprotected settings, while the attack techniques support the practical evaluation of leakage exploitability under different analysis conditions. Overall, the thesis contributes a modular and extensible framework for side channel analysis, combining methodological coherence, practical usability and experimental validation.

Keywords: Side channel analysis, Correlation Power Analysis, Template Attack, Statistical tests, AES-128

Abstract in lingua italiana

La valutazione della sicurezza dei dispositivi crittografici non può limitarsi all'analisi matematica degli algoritmi implementati. Le implementazioni fisiche possono infatti rivelare informazioni attraverso effetti osservabili, quali variazioni temporali, emissioni elettromagnetiche o consumo di potenza. Per questo motivo, la side channel analysis svolge un ruolo centrale nella valutazione della sicurezza del dispositivo. In contesti sperimentali, tale valutazione coinvolge più fasi, tra cui gestione dei dataset, preprocessing, leakage detection, riduzione delle feature ed esecuzione degli attacchi. Queste attività sono spesso gestite tramite strumenti separati e sorgenti di dati eterogenee, rendendo il workflow meno uniforme, meno riusabile e più difficile da riprodurre. Questa tesi affronta tale problema attraverso la progettazione e l'implementazione di FreeSCA, una libreria Python modulare per la side channel analysis. Il framework integra le principali fasi del workflow di analisi all'interno di una struttura comune e introduce il formato FSCA per la gestione uniforme di tracce e metadati associati, strumenti di conversione per sorgenti di dati eterogenee e un approccio di elaborazione incrementale adatto a condizioni sperimentali realistiche. All'interno di questo framework è implementato un insieme rappresentativo di metodi per la valutazione pratica side channel, comprendente diverse varianti del Welch t-test per la rilevazione statistica del leakage, Correlation Power Analysis, Template Attack e procedure di riduzione delle feature per tracce ad alta dimensionalità. Il framework è validato sperimentalmente su dataset masked e unmasked associati a implementazioni AES-128 eseguite su un microcontrollore ARM Cortex-M4. I risultati mostrano che la libreria proposta supporta in modo coerente le principali fasi di un workflow pratico di side channel analysis. I metodi statistici implementati evidenziano differenze significative tra scenari protetti e non protetti, mentre le tecniche di attacco supportano la valutazione pratica della sfruttabilità del leakage in diverse condizioni di analisi. Nel complesso, la tesi contribuisce con un framework modulare ed estendibile per la side channel analysis, combinando coerenza metodologica, usabilità pratica e validazione sperimentale.

Parole chiave: Side channel analysis, Correlation Power Analysis, Template Attack, test statistici, AES-128

Contents

Abstract	i
Abstract in lingua italiana	iii
Contents	v
Introduction	1
1 Background and state of the art	5
1.1 Statistical Background	5
1.2 AES-128 and Leakage Modeling	12
1.3 Metrics for Side-Channel Analysis	16
1.3.1 Leakage detection metrics	16
1.3.2 Localization and Feature Selection Metrics	19
1.3.3 Information-Theoretic Metrics	20
1.3.4 Scoring, Ranking, and Performance Metrics	22
1.4 Classification and Overview of Side-Channel Attacks	27
1.4.1 Non-profiled Attacks	28
1.4.2 Profiled Attacks	28
2 Design and Implementation of a Python Library for SCA	31
2.1 Overview, Design Goals and General Workflow	31
2.2 The FSCA File Format and Data Converters	33
2.2.1 The FSCA File Format	34
2.2.2 Data Converters	38
2.3 Feature Selection and Dimensionality Reduction	39
2.3.1 Score-based Feature Selection	40
2.3.2 Principal Component Analysis (PCA)	43
2.3.3 Generation of reduced FSCA datasets	45

2.4	Intermediate Computation and Leakage Models	47
2.5	Statistical Tests	49
2.5.1	First Order t-test	50
2.5.2	Higher Order t-test	52
2.5.3	Bivariate t-test	58
2.6	Implemented Side Channel Attacks	61
2.6.1	Correlation Power Analysis	63
2.6.2	Template Attack	67
3	Experimental Evaluation	73
3.1	Experimental Setup	73
3.2	Statistical Test Results	74
3.2.1	First Order t-test	74
3.2.2	Higher Order t-test	79
3.2.3	Bivariate t-test	81
3.3	Attack Results	83
3.3.1	Correlation Power Analysis	84
3.3.2	Template Attack	87
4	Conclusion	93
	Bibliography	95
A	Appendix A	97
B	Appendix B	101
	List of Figures	103
	List of Tables	107
	List of Symbols	109
	Acknowledgements	111

Introduction

Cryptographic algorithms play a central role in protecting sensitive data in modern digital systems. Many of these algorithms have been extensively studied and are considered secure from a mathematical point of view. However, the security of a real cryptographic device does not depend only on the algorithm itself. Even when the implementation is correct, the execution of a cryptographic computation may still produce observable physical effects that reveal information.

This information may become accessible through quantities that can be observed while the device is operating, such as execution time, electromagnetic emissions or power consumption. Attacks that exploit this form of leakage are known as side channel attacks. In this context, the vulnerability does not arise from the cryptographic algorithm itself, but from the information that may be unintentionally exposed during execution.

Among the different forms of side channel leakage, power consumption is one of the main sources of information. During a cryptographic computation, the device continuously updates its internal state while processing data. These changes require electrical switching, and switching consumes power. Since the internal values being processed depend on both the input data and the secret key, the corresponding power consumption is not completely independent of the computation. As a result, measurements of the power signal may contain information about operations involving secret data. By collecting and analyzing many such measurements, an attacker may detect data dependent leakage and recover information about the secret key.

Problem statement

The practical evaluation of cryptographic implementations through side channel analysis (SCA) still presents several challenges. In real experimental settings, the analysis typically involves multiple steps, including trace acquisition or conversion, dataset organization, leakage assessment, feature reduction and attack execution. These steps are often managed through heterogeneous tools, ad hoc scripts and different file formats, rather than within a unified framework. As a result, the overall workflow becomes fragmented

and less uniform, which complicates the reuse of methods, the comparison of results, and the reproducibility of the analysis.

This motivates the need for a software framework able to support the main stages of side channel analysis within a coherent structure. Such a framework should make it possible to manage trace datasets in a uniform way, apply standard statistical leakage detection methods, execute representative attack techniques and compare the obtained results under different experimental settings.

Within this context, the problem addressed in this thesis is the design and implementation of a modular library for side channel analysis, together with its validation on masked and unmasked trace datasets. The goal is not only to provide individual implementations of established methods, but also to integrate them into a common workflow that supports practical experimentation and the coherent application of different analysis methods.

Main contributions

The main contribution of this thesis is the introduction of FreeSCA, a modular framework for side channel analysis conceived as a coherent environment for practical experimentation. Its value lies not only in the implementation of established analysis methods, but also in the definition of a common structure that supports the different stages of the workflow in a uniform and extensible way, within a Python library designed for direct use in experimental analysis settings.

This contribution takes concrete form in a library that combines a dedicated trace format, introduced to provide a common representation for traces and associated metadata throughout the analysis workflow, with conversion tools for heterogeneous data sources, so that externally acquired or publicly available datasets can be imported into the same environment and processed consistently. At the same time, the framework adopts an incremental approach across the library, aimed at making dataset handling and method execution compatible with realistic experimental conditions and large trace collections. On this basis, the thesis provides a representative set of methods for practical side channel analysis, including statistical leakage detection through different variants of Welch's *t*-test, representative attack techniques such as Correlation Power Analysis and Template Attack and feature reduction procedures for high dimensional trace datasets. The resulting framework is naturally suited for future extension to additional attacks, statistical tests and evaluation metrics, and its practical usefulness is finally assessed through an experimental validation on masked and unmasked datasets.

Structure of the Thesis

This section provides an overview of the structure of the thesis and briefly describes the content of each chapter.

Chapter 1: Background and state of the art

The first chapter introduces the theoretical background required for the rest of the thesis. It presents the main statistical concepts used in the analysis, introduces the target algorithm and the considered leakage setting, and reviews the principal metrics and attack techniques relevant to side channel analysis.

Chapter 2: Design and Implementation of a Python Library for SCA

The second chapter presents the design and implementation of the proposed Python library for side channel analysis. It describes the general architecture of the framework, the adopted trace format, the implemented conversion tools and the main analysis modules, including feature reduction procedures, statistical tests and attack techniques.

Chapter 3: Experimental Evaluation

The third chapter reports the experimental evaluation of the proposed framework. It introduces the considered datasets and discusses the results obtained with the implemented statistical tests and attack methods on both masked and unmasked settings.

Chapter 4: Conclusion

The final chapter summarizes the main results of the thesis, discusses the achieved objectives and outlines possible future developments.

1 | Background and state of the art

This chapter is organized into four sections. The first section presents the statistical background used in the remainder of the work. The second section introduces the target algorithm and defines the concepts used in the considered setting, including traces, intermediate values, and leakage models. The third section presents and classifies the main metrics used in side-channel analysis, which operate on traces and intermediates to quantify relevant properties of the measurements that can be used by attacks. The final section provides an overview of the main attack techniques and their classification.

1.1. Statistical Background

This section introduces the statistical background used throughout the thesis, covering estimation, uncertainty quantification, main reference distributions and hypothesis testing.

Random variables and estimators. When a measurement or an experiment is repeated under nominally identical conditions, the outcome is not perfectly reproducible: it varies due to noise, environmental effects, and acquisition variability. To represent this variability, the quantities of interest are modeled as random variables. A random variable, denoted by $\mathcal{X}$, is a quantity whose value is governed by a probability distribution.

In the continuous case, this distribution can be described by a Probability Density Function (PDF), denoted by $f_{\mathcal{X}}$, which specifies how probability is distributed over the possible values of $\mathcal{X}$. When a parametric distribution is considered, the density is written as $p(x; \theta)$, where θ denotes the distribution parameters.

The Cumulative Distribution Function (CDF) $F_{\mathcal{X}}$ gives the probability that $\mathcal{X}$ takes a value not greater than x ,

$$F_{\mathcal{X}}(x) = \Pr(\mathcal{X} \leq x). \quad (1.1)$$

For continuous random variables, the PDF and CDF are related by

$$F_{\mathcal{X}}(x) = \int_{-\infty}^x f_{\mathcal{X}}(t) dt. \quad (1.2)$$

Rather than knowing this distribution explicitly, the available information consists of a set of samples of finite cardinality, denoted by $\{x_1, \dots, x_n\}$. Consequently, distribution parameters such as the mean $\mu = \mathbb{E}[\mathcal{X}]$ and the variance $\sigma^2 = \text{Var}[\mathcal{X}]$ are replaced by estimators, i.e., functions of the observed data designed to approximate the corresponding parameters. In this thesis, the most common examples are the sample mean $\hat{\mu}$, used as an estimate of μ , and the sample variance $\hat{\sigma}^2$, used as an estimate of σ^2 . Given n observations $x_1, \dots, x_n$, the sample mean is defined as shown in 1.3

$$\hat{\mu} = \frac{1}{n} \sum_{i=1}^n x_i \quad (1.3)$$

and the sample variance is defined as in 1.4

$$\hat{\sigma}^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \hat{\mu})^2 \quad (1.4)$$

where normalization by $n-1$, instead of n , corresponds to Bessel's correction and yields the standard unbiased estimator of the population variance when the mean is unknown and replaced by $\hat{\mu}$. In other words, one degree of freedom is used to estimate the mean, and the variance is computed over the remaining $n-1$ degrees of freedom.

Standard Error (SE). The estimators 1.3 and 1.4 are computed from a set of samples of finite cardinality $\{x_1, \dots, x_n\}$, where n denotes the number of available observations. As a consequence, repeating the acquisition yields different estimates, even when the acquisition procedure is repeated under the same conditions. A quantitative measure of this variability is given by the standard error. For an estimator $\hat{\theta}$ of a parameter θ , the standard error is the standard deviation of $\hat{\theta}$ under repeated sampling from the same distribution. In practical terms, SE indicates how stable an estimate is: a small SE means that repeating the experiment is expected to give similar values of $\hat{\theta}$, while a large SE indicates higher variability. For the sample mean $\hat{\mu}$, assuming independent observations with finite variance σ^2 , the standard error is

$$SE(\hat{\mu}) = \frac{\sigma}{\sqrt{n}}. \quad (1.5)$$

Since σ is unknown, 1.5 is not computable. The estimated standard error is obtained by

using the sample standard deviation $\hat{\sigma}$ in place of σ , resulting in

$$\widehat{\text{SE}}(\hat{\mu}) = \frac{\hat{\sigma}}{\sqrt{n}}. \quad (1.6)$$

As shown by 1.6, the sample size n directly affects the standard error: increasing the number of observations reduces $\widehat{\text{SE}}(\hat{\mu})$, and therefore the uncertainty associated with the estimate $\hat{\mu}$.

Confidence Intervals (CI). A point estimate $\hat{\theta}$ computed from a finite dataset is affected by sampling variability, which can be quantified through its estimated standard error $\widehat{\text{SE}}(\hat{\theta})$. For this reason, it is often useful to report $\hat{\theta}$ together with a confidence interval, i.e., an interval $[l, u]$ that provides a lower bound l and an upper bound u for the unknown parameter θ . The interval is constructed at confidence level $1 - \alpha$, where $\alpha \in (0, 1)$ is the corresponding significance level (e.g., $\alpha = 0.05$ corresponds to a 95% confidence level). It means that, if the same procedure is repeated on independent datasets of size n , then $1 - \alpha$ of the computed intervals include the true value θ . A confidence interval can be expressed as

$$\hat{\theta} \pm q \widehat{\text{SE}}(\hat{\theta}). \quad (1.7)$$

Here, $\hat{\theta}$ is the estimate of the parameter of interest θ , $\widehat{\text{SE}}(\hat{\theta})$ is the estimated standard error of $\hat{\theta}$, and q is a critical value determined by the confidence level $1 - \alpha$ and by the reference distribution assumed for the standardized estimator.

Reference distributions. A probability distribution describes the variability of a random variable by specifying how its values are spread over the domain. The reference distributions introduced in this paragraph provide standard models that will be used throughout the thesis whenever explicit distributional assumptions are required.

- **Normal (Gaussian) distribution:** a scalar random variable $\mathcal{X}$ is said to follow a normal distribution with mean μ and variance σ^2 , denoted $\mathcal{X} \sim \mathcal{N}(\mu, \sigma^2)$, if its probability density function is

$$p(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2} \frac{(x - \mu)^2}{\sigma^2}\right), \quad x \in \mathbb{R} \quad (1.8)$$

where x is a realization of $\mathcal{X}$, $\mu = \mathbb{E}[\mathcal{X}]$ and $\sigma^2 = \text{Var}[\mathcal{X}]$. The normalization factor $1/\sqrt{2\pi\sigma^2}$ ensures that the total area under the PDF is equal to one.

Figure 1.1 illustrates the Equation 1.8 for different values of the parameters μ and σ^2 . As shown in Figure 1.1, the mean μ controls the horizontal position of the density, i.e., changing μ shifts the curve along the x -axis without changing its shape. The

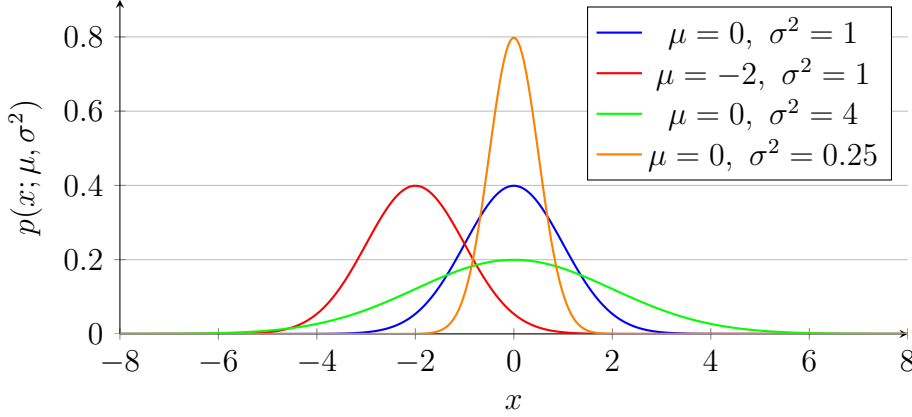


Figure 1.1: Gaussian probability density functions for different values of μ and σ^2 .

variance σ^2 controls the spread, i.e., a larger σ^2 produces a wider distribution with a lower peak, while a smaller σ^2 produces a narrower distribution with a higher peak. In all cases, the area under the curve remains equal to one.

For $\mathcal{X} \sim \mathcal{N}(\mu, \sigma^2)$, probabilities are conveniently expressed by standardizing $\mathcal{X}$ into a standard normal variable. The standardized variable is

$$Z = \frac{\mathcal{X} - \mu}{\sigma}, \quad (1.9)$$

where σ is the standard deviation, i.e., the square root of the variance. This transformation shifts the distribution to zero mean and rescales it to unit variance, so that $Z \sim \mathcal{N}(0, 1)$. Using this standardization, the CDF of $\mathcal{X}$ can be written in terms of the standard normal CDF Φ , defined as

$$\Phi(z) = \Pr(Z \leq z). \quad (1.10)$$

In a two-sided setting, α specifies how much probability mass is left in the extreme parts (the tails) of the standard normal distribution. The total tail probability is α , split equally between the two sides: $\alpha/2$ in the left tail and $\alpha/2$ in the right tail. The value $z_{1-\alpha/2}$ is the $(1 - \alpha/2)$ -quantile of the standard normal distribution such that

$$\Phi(z_{1-\alpha/2}) = \Pr(Z \leq z_{1-\alpha/2}) = 1 - \alpha/2. \quad (1.11)$$

Because the standard normal distribution is symmetric, the left threshold is simply $-z_{1-\alpha/2}$. As a result, the probability of falling outside the interval $[-z_{1-\alpha/2}, z_{1-\alpha/2}]$ is α , while the probability of falling inside the interval is $1 - \alpha$. Figure 1.2 shows the standard normal density and highlights the two-sided critical values $\pm z_{1-\alpha/2}$.

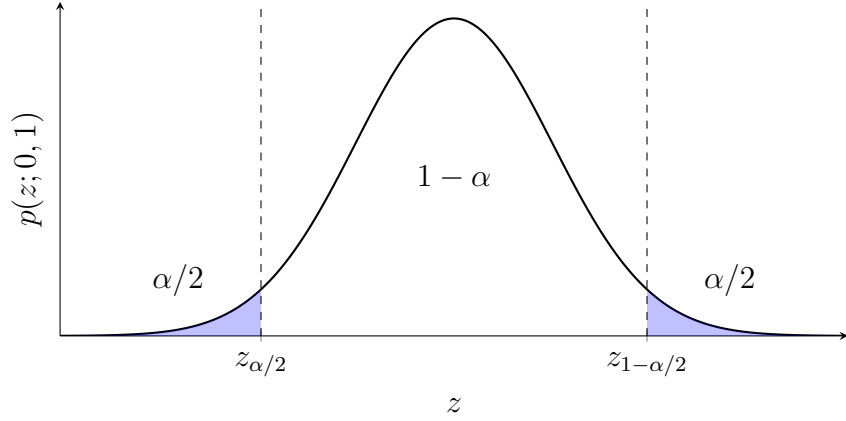


Figure 1.2: Two-sided critical values for the standard normal distribution.

The shaded regions correspond to the two tails, each having probability mass $\alpha/2$, so that the total probability outside the interval $[-z_{1-\alpha/2}, z_{1-\alpha/2}]$ is α .

- **Multivariate normal (Gaussian) distribution.** A random vector $\mathcal{X} \in \mathbb{R}^d$ is said to follow a multivariate normal distribution with mean vector $\boldsymbol{\mu} \in \mathbb{R}^d$ and covariance matrix $\Sigma \in \mathbb{R}^{d \times d}$, denoted $\mathcal{X} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$, if its probability density function is

$$p(\mathbf{x}; \boldsymbol{\mu}, \Sigma) = \frac{1}{\sqrt{(2\pi)^d \det(\Sigma)}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^\top \Sigma^{-1}(\mathbf{x} - \boldsymbol{\mu})\right), \quad \mathbf{x} \in \mathbb{R}^d. \quad (1.12)$$

where d is the dimension of the random vector and $\mathbf{x}$ denotes a realization of $\mathcal{X}$. In Equation 1.12, $\boldsymbol{\mu} = \mathbb{E}[\mathcal{X}]$ and $\Sigma = \text{Cov}[\mathcal{X}]$. The diagonal entries of Σ are the variances of the individual components, while the off-diagonal entries represent pairwise covariances. The normalization factor $1/\sqrt{(2\pi)^d \det(\Sigma)}$ ensures that the total integral of the density over $\mathbb{R}^d$ is equal to one.

- **Student's t distribution:** Let $x_1, \dots, x_n$ be independent observations of a scalar random variable $\mathcal{X}$. Denote by $\hat{\mu}$ the sample mean and by $\hat{\sigma}$ the sample standard deviation. Standardizing the mean using the estimated standard deviation yields the statistic

$$\mathcal{T} = \frac{\hat{\mu} - \mu}{\hat{\sigma}/\sqrt{n}}. \quad (1.13)$$

A scalar random variable $\mathcal{T}$ is said to follow a Student's t distribution with $\nu = n - 1$ degrees of freedom, i.e., $\mathcal{T} \sim t_\nu$, if its PDF is

$$p(t; \nu) = \frac{\Gamma(\frac{\nu+1}{2})}{\sqrt{\nu\pi} \Gamma(\frac{\nu}{2})} \left(1 + \frac{t^2}{\nu}\right)^{-\frac{\nu+1}{2}}, \quad t \in \mathbb{R}. \quad (1.14)$$

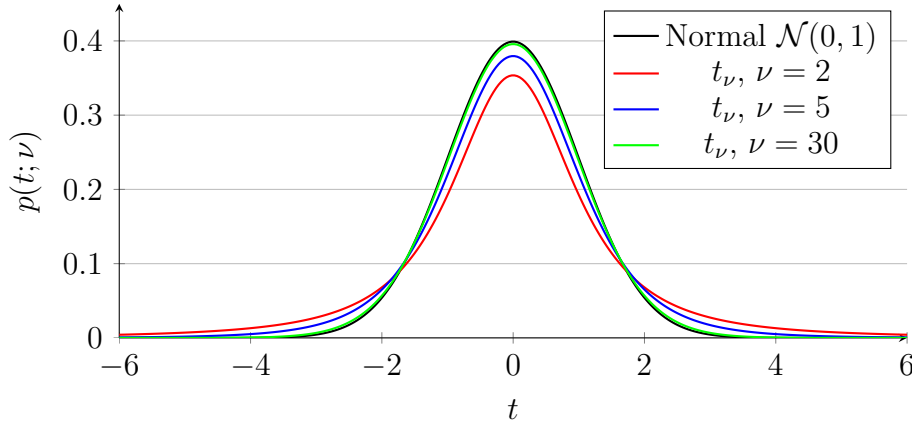


Figure 1.3: Student's t densities for different degrees of freedom ν , compared with the standard normal density.

where $\Gamma(\cdot)$ denotes the Gamma function, which extends the factorial to real arguments, and $\nu > 0$ is the number of degrees of freedom.

Figure 1.3 illustrates the Equation 1.14 for different values of ν and compares it with the standard normal density. As shown in Figure 1.3, the Student's t distribution is symmetric around zero and its shape depends on the degrees of freedom ν . For small ν , the density has heavier tails than the normal distribution. As ν increases, the the Student's t density becomes progressively closer to the standard normal density.

After defining the PDF $p(t; \nu)$, the CDF is introduced in the same way as in 1.1. In a two-sided setting, the critical value $t_{1-\alpha/2, \nu}$ is defined as the $(1 - \alpha/2)$ -quantile of the t_ν distribution, i.e., the value such that

$$\Pr(\mathcal{T} \leq t_{1-\alpha/2, \nu}) = 1 - \alpha/2 \quad \text{for } \mathcal{T} \sim t_\nu. \quad (1.15)$$

This quantile depends not only on α but also on the degrees of freedom ν . As ν increases, the t_ν distribution approaches the standard normal distribution, and $t_{1-\alpha/2, \nu}$ approaches the corresponding normal critical value $z_{1-\alpha/2}$.

Hypothesis testing. A hypothesis test [3] is a decision procedure used to assess whether the available data are compatible with a reference assumption. In the following, the focus is on the two-group mean comparison problem. Let Q_0 and Q_1 denote two groups of observations, with unknown population means μ_0 and μ_1 . The objective is to determine whether the available data support the presence of a mean difference between the two

groups. This is formulated through the hypotheses

$$H_0 : \mu_1 = \mu_0 \quad H_1 : \mu_1 \neq \mu_0, \quad (1.16)$$

where H_0 states that the two groups have the same mean and H_1 states that their means differ. A test produces a decision rule that maps the observed samples to one of two outcomes: "reject H_0 " or "do not reject H_0 ". Two types of decision error are possible. A Type I error (false positive) occurs when H_0 is rejected even though the two population means are equal. The probability of a Type I error is denoted by α and is chosen in advance:

$$\alpha = \Pr(\text{reject } H_0 | H_0 \text{ true}). \quad (1.17)$$

A Type II error (false negative) occurs when H_0 is not rejected even though the two population means are different. Its probability is denoted by β :

$$\beta = \Pr(\text{do not reject } H_0 | H_1 \text{ true}). \quad (1.18)$$

A hypothesis test is completed by specifying a test statistic and a reference distribution under H_0 . A test statistic is a scalar quantity computed from the observed data and designed to capture the evidence against H_0 . In this setting, the statistic is built as:

$$t_{\text{obs}} = \frac{\hat{\mu}_1 - \hat{\mu}_0}{\widehat{SE}}, \quad (1.19)$$

where $\hat{\mu}_0$ and $\hat{\mu}_1$ are the sample means computed on Q_0 and Q_1 , and $\widehat{SE}$ is an estimate of the standard error of the mean difference. The specific form used for $\widehat{SE}$ depends on the assumptions adopted and is specified later for the tests considered in this thesis.

Under H_0 , the statistic is compared to a reference distribution, in this thesis a Student's t reference distribution with ν degrees of freedom is used (see section 1.1). For a two-sided test at significance level α , the corresponding critical value is the quantile $t_{1-\alpha/2, \nu}$ (see 1.15). The standard decision rule rejects H_0 when $|t_{\text{obs}}| \geq t_{1-\alpha/2, \nu}$, and does not reject H_0 otherwise.

The outcome can also be summarized by the two-sided p -value, defined as the probability under H_0 of obtaining a statistic at least as extreme as the observed one:

$$p = \Pr(|\mathcal{T}| \geq |t_{\text{obs}}| \mid H_0), \quad \mathcal{T} \sim t_{\nu}. \quad (1.20)$$

The rule "reject H_0 " is equivalent to $p \leq \alpha$.

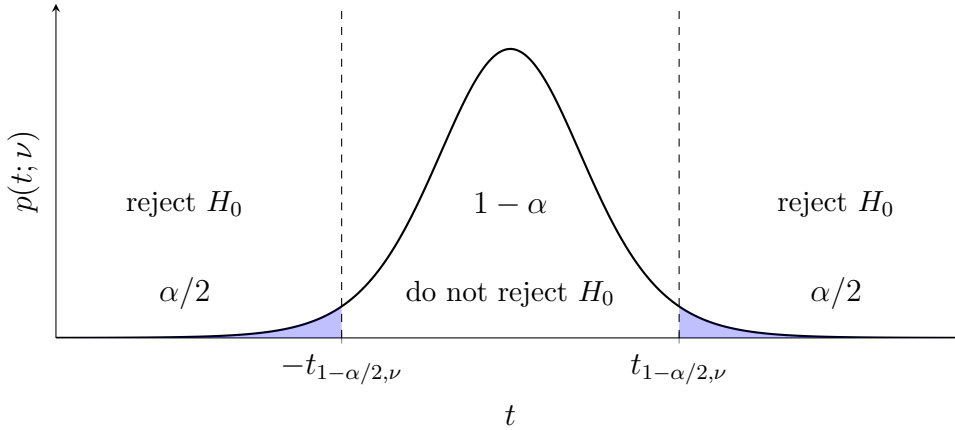


Figure 1.4: Two-sided rejection regions for a Student's t reference distribution.

Figure 1.4 illustrates the corresponding two-sided acceptance and rejection regions for a Student's t reference distribution (example with $\alpha = 0.05$ and $\nu = 10$).

1.2. AES-128 and Leakage Modeling

This section presents the Advanced Encryption Standard (AES) in the AES-128 variant considered in this thesis and the modeling elements used in the remainder of the work. It first summarizes the algorithm and highlights the internal computations that are relevant in side-channel analysis. It then describes how measurements are recorded and represented as traces, and how they are associated with known inputs. Finally, it defines intermediate values derived from AES and introduces the leakage models used to relate those intermediates to the measured signals.

The AES-128 [4] is a symmetric-key block cipher that encrypts data in fixed-size blocks of 128 bits and uses a 128-bit secret key. Encryption updates an internal state through an initial key addition followed by ten rounds of transformations. Each round applies a fixed set of operations to the state and incorporates a round key derived from the key schedule. Figure 1.5 summarizes the high-level structure of AES-128 encryption and the order of the round transformations. The AES round is composed of the following state transformations:

- **SubBytes.** Each byte of the state is substituted independently through the AES substitution box (SBox). This is the only non-linear operation in AES and is therefore central in side-channel analysis. In practice, intermediate values from the first and last round are often preferred because they can be expressed in terms of known data (plaintext or ciphertext) together with a single key byte. For this reason,

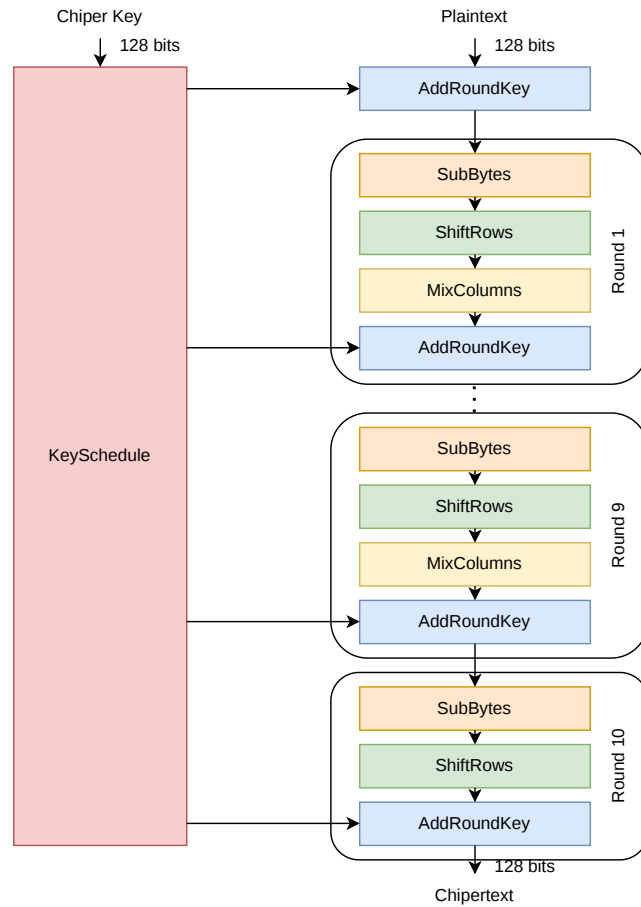


Figure 1.5: High-level structure of AES-128 encryption and round transformations.

the SBox input or output is commonly used to build key-dependent predictions in subsequent attacks.

- **ShiftRows.** The state bytes are permuted by cyclically shifting each row by a fixed offset. This operation does not change the byte values but changes their positions, contributing to diffusion across rounds.
- **MixColumns.** A fixed linear mixing is applied independently to each column of the state by applying matrix multiplication that combines its bytes. This step spreads the influence of each byte within the column and it is applied only in the first nine rounds.
- **AddRoundKey.** The state is combined with the round key through a xor operation. This operation introduces key dependence at each round.

In the considered setting, each execution of AES-128 is accompanied by a physical measurement of the device, such as power consumption. The measurement is recorded as a time-series signal, referred to as a trace. Traces are collected because the power consumption of the device during AES execution is not constant but it depends on the operations being performed and on the data values manipulated by the implementation. As a result, across repeated executions, samples at specific time instants can show a statistical dependence on internal computations. Let n denote the number of collected traces, corresponding to n executions performed under a fixed secret key, and let t denote the number of sampled time instants per trace. The acquired dataset is represented as a matrix $M \in \mathbb{R}^{n \times t}$, where each row $\mathbf{m}_{i,:}$ corresponds to the i -th trace (one measurement), and each column $\mathbf{m}_{:,j}$ contains the samples of all traces at time j -th.

$$M = \begin{bmatrix} m_{1,1} & m_{1,2} & m_{1,3} & \dots & m_{1,t} \\ m_{2,1} & m_{2,2} & m_{2,3} & \dots & m_{2,t} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ m_{n,1} & m_{n,2} & m_{n,3} & \dots & m_{n,t} \end{bmatrix} \quad (1.21)$$

For each given time index j , the column $\mathbf{m}_{:,j}$ is treated as a set of observations of an underlying random variable describing the device leakage at that time sample. In controlled laboratory settings, each trace is associated with known inputs (e.g., plaintext or ciphertext) and with the corresponding measurement metadata. These known values are later used to define internal intermediate variables and to build key-dependent predictions.

After arranging the measurements in the trace matrix M , the next step is to relate the observed signals to values computed during the execution of AES-128. An intermediate value is an internal quantity that depends on known inputs (e.g., plaintext or ciphertext) and on a subset of the secret key. In side-channel analysis, the objective is to identify time samples for which the measured leakage is statistically dependent on such intermediates. For AES-128, a common choice is to focus on first-round SBox intermediates at a given byte index b . For trace i , the corresponding intermediate can be written as

$$v_i^{(b)} = \text{SBox}(\mathbf{P}_i[b] \oplus \mathbf{K}[b]), \quad (1.22)$$

where $\mathbf{P}_i[b]$ denotes the b -th plaintext byte of trace i , and $\mathbf{K}[b]$ denotes the b -th key byte. Most practical attacks follow a Divide and Conquer strategy, where the secret key is decomposed into smaller subkeys (typically one byte) and each byte is attacked independently. For a fixed byte index b , all key hypotheses $k \in \{0, \dots, 255\}$ are evaluated. For each hypothesis k and each trace i , a predicted intermediate value is computed, which

results in an intermediate values matrix $V^{(b)} \in \mathbb{R}^{n \times 256}$, where n denotes the number of traces and 256 are the key hypotheses for byte b . Since intermediate values are not directly observable, leakage models allow to associate each predicted intermediate to a scalar quantity that can be compared with the observed leakage. Formally, a leakage model is a function $s(\cdot)$ that maps an intermediate to a scalar feature. Typical leakage models include:

- Hamming weight (**hw**): the **hw** model assumes that the power consumption is proportional to the number of bits set to 1 in the intermediate value. For a byte value x , it is defined as:

$$\text{hw}(x) = \sum_{r=0}^7 x_r \quad (1.23)$$

where x_r denotes the r -th bit of x .

- Hamming distance (**hd**): the **hd** model assumes that the power consumption is proportional to the number of bit transitions between two values x and y . For two byte values x and y , it is defined as:

$$\text{hd}(x, y) = \text{hw}(x \oplus y) \quad (1.24)$$

Applying a leakage model to the predicted intermediate values results in a prediction matrix $H^{(b)} \in \mathbb{R}^{n \times 256}$, where n denotes the number of traces.

Each row corresponds to one trace, while each column corresponds to one key hypothesis:

$$H^{(b)} = \begin{bmatrix} h_{1,0}^{(b)} & h_{1,1}^{(b)} & h_{1,2}^{(b)} & \dots & h_{1,255}^{(b)} \\ h_{2,0}^{(b)} & h_{2,1}^{(b)} & h_{2,2}^{(b)} & \dots & h_{2,255}^{(b)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ h_{n,0}^{(b)} & h_{n,1}^{(b)} & h_{n,2}^{(b)} & \dots & h_{n,255}^{(b)} \end{bmatrix}. \quad (1.25)$$

For instance, when using the Hamming weight model, the entries of $H^{(b)}$ are defined as

$$h_{i,k}^{(b)} = \text{hw}(\text{SBox}(\text{P}_i[b] \oplus k)), \quad (1.26)$$

where $i \in \{1, \dots, n\}$ indexes the trace, b denotes the targeted byte index, and $k \in \{0, \dots, 255\}$ represents a key hypothesis.

1.3. Metrics for Side-Channel Analysis

This section introduces the main metrics used throughout the thesis. Several metrics can serve multiple purposes and here they are organized according to their primary role in a typical side-channel analysis workflow. The first group of metrics is used for leakage detection, i.e., to verify whether the measurements exhibit statistically significant data dependence. The second group supports localization and feature selection, i.e., identifying time samples or combinations of samples that are most informative. The third group consists of information-theoretic measures used to quantify the amount of information carried by the measurements. Finally, the section introduces scoring and ranking concepts and the metrics used to evaluate attacks, from subkey to full-key level.

1.3.1. Leakage detection metrics

This subsection focuses on leakage detection metrics, which are used to determine whether the measurements exhibit statistically significant data dependence. In addition, these metrics can also indicate where the dependence is stronger across time samples and can therefore support the selection of points of interest.

Welch’s t -test. Welch’s t -test is widely used in side-channel analysis as a leakage detection tool [6, 9]. In this context, the goal is not to recover a key, but to verify whether the measured traces exhibit a statistically significant dependence on data values at specific time samples. The test is applied in a two-sided setting at a chosen significance level α , following the hypothesis testing framework introduced in 1.1. The procedure operates by comparing two groups of observations, denoted Q_0 and Q_1 , and testing whether their mean values differ. Two main strategies are commonly used to form the two groups: non-specific and specific. In the non-specific setting, the most common choice is the fixed-vs-random test. One group contains traces collected with a fixed input (e.g., fixed plaintext), while the other group contains traces collected with varying inputs (e.g., random plaintexts).

$$Q_0 : \text{Fixed}, \quad Q_1 : \text{Random.} \quad (1.27)$$

In the specific setting, traces are partitioned according to a label. In a controlled evaluation setting where the key is known, the grouping can be defined using the true intermediate values. A common choice in AES is the first-round SBox output:

$$v_i^{(b)} = \text{SBox}(\text{P}_i[b] \oplus \text{K}[b]) \quad (1.28)$$

where b denotes a targeted byte index and $K[b]$ is the corresponding key byte. A binary label s_i is obtained from the intermediate by applying a mapping $f(\cdot)$ (e.g., selecting one bit):

$$s_i = f(v_i^{(b)}) \in \{0, 1\}. \quad (1.29)$$

The two groups are defined by this label:

$$Q_0 = \{i : s_i = 0\}, \quad Q_1 = \{i : s_i = 1\}. \quad (1.30)$$

Welch's t -test is computed independently at each time sample j . To keep the notation compact, the time index j is omitted in the expressions below. Let $n_0 = |Q_0|$ and $n_1 = |Q_1|$ denote the group sizes. At the considered time sample, let $\hat{\mu}_0$ and $\hat{\sigma}_0^2$ be the sample mean and sample variance computed from the values $\{m_i\}_{i \in Q_0}$, and let $\hat{\mu}_1$ and $\hat{\sigma}_1^2$ be the corresponding quantities computed from $\{m_i\}_{i \in Q_1}$. Welch's t statistic is defined as

$$t = \frac{\hat{\mu}_1 - \hat{\mu}_0}{\sqrt{\frac{\hat{\sigma}_1^2}{n_1} + \frac{\hat{\sigma}_0^2}{n_0}}}. \quad (1.31)$$

The numerator is the difference between the two sample means. The denominator estimates the standard error of this difference by combining the two variance terms $\hat{\sigma}_1^2/n_1$ and $\hat{\sigma}_0^2/n_0$, which makes the test suitable when the two groups have unequal variances. Under the hypotheses H_0 (see 1.16), the statistic is compared against a Student's t reference distribution with ν degrees of freedom. The degrees of freedom are approximated by the Welch-Satterthwaite equation

$$\nu \approx \frac{\left(\frac{\hat{\sigma}_1^2}{n_1} + \frac{\hat{\sigma}_0^2}{n_0}\right)^2}{\frac{\left(\frac{\hat{\sigma}_1^2}{n_1}\right)^2}{n_1 - 1} + \frac{\left(\frac{\hat{\sigma}_0^2}{n_0}\right)^2}{n_0 - 1}}. \quad (1.32)$$

Applying the test to many time samples produces N_{tests} hypothesis tests. Using the same significance level α for each test would increase the family-wise error rate (FWER), defined as the probability of observing at least one false positive among N_{tests} . The Bonferroni correction controls the FWER by using a per-test significance level $\alpha^* = \alpha/N_{\text{tests}}$, which replaces α in the decision rule.

The Welch's t test captures differences in the first-order statistics of the leakage (in particular, differences in the mean between two groups). In masked implementations, sensitive

intermediate values are combined with random masks to reduce direct first-order dependence on the unmasked value. As a result, first-order leakage may be significantly reduced even though higher-order leakage is still present. For this reason, leakage detection is often extended by applying the same statistical test to derived quantities that are designed to capture higher-order effects or joint behavior across multiple samples.

Higher-Order Welch Test. Higher-order tests [9] extend leakage detection by applying the same Welch framework to derived features, rather than to the raw trace samples. This is motivated by masked implementations, where sensitive intermediates are combined with random masks so that first-order mean differences may be strongly reduced, even though higher-order leakage can still be present. For each time sample j , a higher-order feature is built from the sample values $m_{i,j}$. For an order $d \geq 2$, a common choice is a centered power, where centering is performed with respect to the corresponding group. Specifically, for each group Q_g , where $g \in \{0, 1\}$, the mean $\hat{\mu}_g(j)$ is computed and the feature values are defined as

$$y_{i,j}^{(d)} = (m_{i,j} - \hat{\mu}_g(j))^d, \quad i \in Q_g. \quad (1.33)$$

Welch's test is then applied as in the first-order case, by treating $\{y_{i,j}^{(d)}\}_{i \in Q_0}$ and $\{y_{i,j}^{(d)}\}_{i \in Q_1}$ as the two samples to compare. The computation is repeated for all time indices j and the decision rule and the correction for multiple tests are the same as in the first-order case. These features capture higher-order effects at a single time sample. When higher-order leakage is distributed across multiple time samples, bivariate constructions are often used to capture the joint behavior of two samples, as discussed next.

Fork Welch Test. Bivariate (fork) tests [1] extend leakage detection by applying Welch's t test to features built from pairs of time samples, rather than from a single sample. This variant is useful when the relevant leakage is distributed across time and becomes more evident when two samples are considered jointly. Let j_1 and j_2 be two time indices. For each trace i belonging to group Q_g , with $g \in \{0, 1\}$, a bivariate feature is constructed as the product of the two centered samples:

$$y_{i,j_1,j_2} = (m_{i,j_1} - \hat{\mu}_g(j_1))(m_{i,j_2} - \hat{\mu}_g(j_2)), \quad i \in Q_g, \quad (1.34)$$

where $\hat{\mu}_g(j_1)$ and $\hat{\mu}_g(j_2)$ denote the group mean at the corresponding time indices. Welch's test is then applied by treating $\{y_{i,j_1,j_2}\}_{i \in Q_0}$ and $\{y_{i,j_1,j_2}\}_{i \in Q_1}$ as the two samples to compare. The test is computed for each pair of time indices (j_1, j_2) within the considered time range and the decision rule and the correction for multiple tests are the same as in the first-order case.

1.3.2. Localization and Feature Selection Metrics

This subsection introduces metrics used to identify points of interest (POIs) and to support feature selection.

Signal-to-Noise Ratio (SNR). The SNR [6] is evaluated independently at each time sample j and the explicit dependence on j is omitted for readability. At a fixed time sample, the leakage is modeled as a random variable $\mathcal{M}$ that can be decomposed into a data-dependent component and a noise component:

$$\mathcal{M} = \mathcal{M}_d + \mathcal{M}_n. \quad (1.35)$$

The SNR is used to quantify how strong the data-dependent component is relative to the noise component and it is defined as

$$\text{SNR} = \frac{\text{Var}[\mathcal{M}_d]}{\text{Var}[\mathcal{M}_n]} \quad (1.36)$$

In practice, $\mathcal{M}_d$ and $\mathcal{M}_n$ are not observed separately. Their variances are therefore estimated from the samples $m_{i,j}$ by partitioning them into classes and comparing the dispersion within each class to the dispersion across class means. In a profiling setting, each trace i is assigned a class label v_i derived from a chosen intermediate. Let $\mathbf{V}$ denote the set of possible class values. Two common choices are:

- the exact intermediate value, leading to $v_i \in \{0, \dots, 255\}$;
- the Hamming weight of the intermediate, leading to $v_i \in \{0, \dots, 8\}$.

For each class $v \in \mathbf{V}$, define the set $Q_v = \{i : v_i = v\}$, which contains the traces belonging to class v . For each class v , the class mean $\hat{\mu}_v$ is computed as

$$\hat{\mu}_v = \frac{1}{|Q_v|} \sum_{i \in Q_v} m_{i,j} \quad (1.37)$$

and the global mean is computed as the average of the class means:

$$\hat{\mu} = \frac{1}{|\mathbf{V}|} \sum_{v \in \mathbf{V}} \hat{\mu}_v. \quad (1.38)$$

The within-class variance for each class v is computed as

$$\hat{\sigma}_v^2 = \frac{1}{|Q_v| - 1} \sum_{i \in Q_v} (m_{i,j} - \hat{\mu}_v)^2 \quad (1.39)$$

and the average within-class variance is defined as

$$\hat{\sigma}^2 = \frac{1}{|\mathbf{V}|} \sum_{v \in \mathbf{V}} \hat{\sigma}_v^2. \quad (1.40)$$

Intuitively, samples within the same class share the same expected data-dependent level, so their spread mainly reflects noise, while samples across different classes have different expected levels, so the dispersion of class means captures the data-dependent component. Accordingly, the noise term is estimated with Equation 1.40. The data-dependent term is estimated by the variance of the class means around the global mean:

$$\widehat{\text{Var}}_d = \frac{1}{|\mathbf{V}|} \sum_{v \in \mathbf{V}} (\hat{\mu}_v - \hat{\mu})^2. \quad (1.41)$$

The resulting SNR estimate is then defined as

$$\widehat{\text{SNR}} = \frac{\widehat{\text{Var}}_d}{\hat{\sigma}^2}. \quad (1.42)$$

Normalized Inter-Class Variance (NICV). NICV is a normalized variant of the SNR. Instead of expressing the ratio between the data-dependent variance and the noise variance, NICV measures the fraction of the total variance that can be attributed to the separation between classes. Using the same quantities already defined for SNR, NICV is computed as

$$\text{NICV} = \frac{\widehat{\text{Var}}_d}{\widehat{\text{Var}}_d + \hat{\sigma}^2}, \quad (1.43)$$

where $\widehat{\text{Var}}_d$ is the variance of the class means around the global mean and $\hat{\sigma}^2$ is the average within-class variance. By construction, $\text{NICV} \in [0, 1]$, and values close to 0 indicate that the measurements are dominated by noise.

1.3.3. Information-Theoretic Metrics

This subsection introduces Mutual Information (MI), Hypothetical Information (HI), and Perceived Information (PI) as information-theoretic measures [6] that express in bits the relation between leakage features and intermediate-derived labels.

Mutual Information (MI). Mutual information is used to quantify, in bits, the dependence between a leakage feature extracted from the traces and a discrete label derived

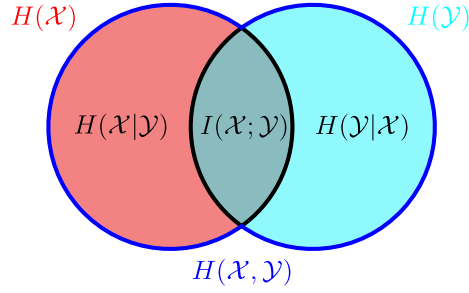


Figure 1.6: Graphical decomposition of Shannon entropies: the overlap of $H(\mathcal{X})$ and $H(\mathcal{Y})$ corresponds to $I(\mathcal{X}; \mathcal{Y})$.

from an AES intermediate. For two random variables $\mathcal{X}$ and $\mathcal{Y}$, MI is defined as:

$$I(\mathcal{X}; \mathcal{Y}) = H(\mathcal{X}) - H(\mathcal{X} | \mathcal{Y}) = H(\mathcal{Y}) - H(\mathcal{Y} | \mathcal{X}), \quad (1.44)$$

where $H(\cdot)$ is Shannon entropy. $I(\mathcal{X}; \mathcal{Y})$ measures the reduction in uncertainty about one variable obtained by observing the other. Figure 1.6 provides a compact view of the entropy terms used in the MI definition. In the considered setting, $\mathcal{X}$ can be defined as a discrete leakage feature derived from the measured trace samples at a fixed time index (e.g., by classifying each sample $m_{i,j}$ according to its raw value, whenever it is expressed as an integer), and $\mathcal{Y}$ as a discrete label derived from an AES intermediate (e.g., the SBox output value or its Hamming weight). For discrete random variables, MI can be computed as

$$I(\mathcal{X}; \mathcal{Y}) = \sum_x \sum_y p_{\mathcal{X}, \mathcal{Y}}(x, y) \log_2 \frac{p_{\mathcal{X}, \mathcal{Y}}(x, y)}{p_{\mathcal{X}}(x) p_{\mathcal{Y}}(y)}. \quad (1.45)$$

where x and y denote possible values of $\mathcal{X}$ and $\mathcal{Y}$, $p_{\mathcal{X}}(x)$ and $p_{\mathcal{Y}}(y)$ are the corresponding marginal probabilities and $p_{\mathcal{X}, \mathcal{Y}}(x, y)$ is the joint probability. In practice, the required probabilities are estimated from data using histograms, i.e., by counting occurrences of the discrete pairs (x_i, y_i) .

Alternatively, MI can be estimated in a model-based way by assuming the shape of the distribution (i.e., to which family it belongs), which is the basis for HI and PI.

Hypothetical Information (HI). HI is defined in a profiled setting (see 1.4), where a model for the leakage feature is first learned for each label value. From the profiling set, a conditional distribution $\hat{p}(x|y)$ is estimated, together with a prior $p(y)$ over the labels, often chosen as uniform, as key values during profiling are usually uniformly selected. HI

is then computed by evaluating the MI Equation 1.45 using the model-implied distribution $\hat{p}_{\mathcal{X},\mathcal{Y}}(x, y) = p(y)\hat{p}(x|y)$, and the corresponding marginals. Conceptually, HI corresponds to the MI value that would be obtained if the true joint distribution $p_{\mathcal{X},\mathcal{Y}}(x, y)$ were equal to the model-implied joint $p(y)\hat{p}(x|y)$.

Perceived Information (PI). PI evaluates the fitted profiling model on an independent test set. The conditional model $\hat{p}(x|y)$ is learned on the profiling set, while the test set is used to estimate $\hat{p}_{\text{test}}(x|y)$. PI is then computed by evaluating the MI expression in 1.45 using $\hat{p}_{\text{test}}(x|y)$ for the averaging, and $\hat{p}(x|y)$ for the probabilities in the logarithmic term. In practice, PI reflects the information effectively obtained on unseen measurements and decreases when the model does not match the data. By construction, HI is an upper bound with respect to PI and $\text{PI} \leq \text{MI} \leq \text{HI}$ is typically expected [6].

1.3.4. Scoring, Ranking, and Performance Metrics

This subsection introduces standard concepts used to describe key-recovery results in side-channel analysis. It starts from the notion of assigning a score to each key hypothesis and ranking hypotheses accordingly, and then summarizes common scoring functions and evaluation measures [6].

Scores and ranks. Consider the recovery of one AES key byte indexed by b . For each key-byte hypothesis $k \in \{0, \dots, 255\}$, a scalar score $s_b(k)$ is computed to quantify how well the hypothesis matches the measurements under the adopted intermediate definition and leakage model. Sorting the 256 hypotheses by decreasing score produces an ordered list, called guess list. The rank of a hypothesis k is its position in this ordering (by convention, rank = 1 corresponds to the highest score). Since the scores are computed from a finite and noisy set of measured traces, the resulting rank is a random variable. In particular, the rank of the correct key byte $K[b]$, denoted by rank_b , summarizes how close the correct hypothesis is to the top of the guess list. A minimal workflow for one byte b is:

1. extract a one dimensional leakage feature from the traces (e.g., the column vector $\mathbf{m}_{:,j}$);
2. compute a prediction for each hypothesis k , forming a prediction matrix $H^{(b)} \in \mathbb{R}^{n \times 256}$;
3. compute a score $s_b(k)$ for each k by applying a scoring function that compares the measured leakage feature with the corresponding predictions;
4. sort the hypotheses to obtain the guess list and the corresponding ranks.

The next paragraph introduces Pearson correlation as a widely used scoring function.

Pearson correlation. It is a standard measure of linear correlation and is widely used as a scoring function in side-channel analysis. For a fixed byte index b , the dependence on b is omitted for readability. Let $\mathbf{m} \in \mathbb{R}^n$ be the selected leakage feature and let $\mathbf{h}_k \in \mathbb{R}^n$ be the prediction vector associated with hypothesis k (the k -th column of the prediction matrix H). The Pearson coefficient is:

$$r_k = \frac{\sum_{i=1}^n (m_i - \bar{m})(h_{i,k} - \bar{h}_k)}{\sqrt{\sum_{i=1}^n (m_i - \bar{m})^2} \sqrt{\sum_{i=1}^n (h_{i,k} - \bar{h}_k)^2}}, \quad r_k \in [-1, 1] \quad (1.46)$$

where $\bar{m}$ and $\bar{h}_k$ denote the sample means of $\mathbf{m}$ and $\mathbf{h}_k$, respectively. The value r_k measures the linear association between measurements and predictions and the score is commonly defined as $|r_k|$ and hypotheses are ranked accordingly.

Under an additive decomposition $\mathcal{M} = \mathcal{M}_d + \mathcal{M}_n$ as in 1.35, where $\mathcal{M}_n$ is uncorrelated with $\mathcal{M}_d$ and the prediction captures the data-dependent component of the leakage, r_k^2 corresponds to the fraction of variance explained by the data-dependent term and can be expressed as:

$$r_k^2 = \frac{\text{Var}[\mathcal{M}_d]}{\text{Var}[\mathcal{M}_d] + \text{Var}[\mathcal{M}_n]}. \quad (1.47)$$

In this case, Pearson correlation can be expressed in terms of SNR as:

$$\text{SNR} = \frac{r_k^2}{1 - r_k^2}. \quad (1.48)$$

Success Rate (SR). Success Rate evaluates how often a scoring procedure ranks the correct key-byte hypothesis first when a fixed number of traces is used. Consider a target byte index b and n traces. In a known-key evaluation setting, the correct value $K[b]$ is available. For each run, a score $s_b(k)$ is computed for every hypothesis $k \in \{0, \dots, 255\}$, and hypotheses are sorted by decreasing score 1.3.4. Let rank_b denote the position of $K[b]$ in the resulting ordered list, the top-1 success rate is defined as:

$$\text{SR}_b(n) = \Pr(\text{rank}_b = 1). \quad (1.49)$$

In practice, $\text{SR}_b(n)$ is estimated by repeating the evaluation over p independent experiments. In experiment i , let $\text{rank}_b^{(i)}$ be the rank of $K[b]$. The estimated success rate is

computed as the average of the success indicator:

$$\widehat{\text{SR}}_b(n) = \frac{1}{p} \sum_{i=1}^p \mathbf{1}\{\text{rank}_b^{(i)} = 1\}. \quad (1.50)$$

A top- o success rate is defined in the same way by counting an experiment as successful whenever the correct hypothesis appears within the first o candidates of the guess list:

$$\text{SR}_b^{(o)}(n) = \Pr(\text{rank}_b \leq o), \quad \widehat{\text{SR}}_b^{(o)}(n) = \frac{1}{p} \sum_{i=1}^p \mathbf{1}\{\text{rank}_b^{(i)} \leq o\}. \quad (1.51)$$

As n increases, SR should increase, assuming that the model of the power consumption is correct.

Guessing Entropy (GE). Guessing Entropy is a metric used to quantify the uncertainty remaining after scoring and sorting key hypotheses. In a single experiment, GE is based on the rank of the correct key-byte hypothesis in the guess list. This rank indicates how many candidates would be tested before reaching the correct one if the hypotheses were tried in score order. For a target byte index b and n traces, let rank_b denote the position of the correct key byte $K[b]$ in the ordered list. GE is defined as:

$$\text{GE}_b(n) = \mathbb{E}[\text{rank}_b]. \quad (1.52)$$

It is common to report a logarithmic form, which gives the residual effort in bits:

$$\text{GE}_b^{\log}(n) = \mathbb{E}[\log_2(\text{rank}_b)]. \quad (1.53)$$

In a known-key evaluation setting, both quantities are estimated over p experiments by averaging the observed ranks:

$$\widehat{\text{GE}}_b(n) = \frac{1}{p} \sum_{i=1}^p \text{rank}_b^{(i)}, \quad \widehat{\text{GE}}_b^{\log}(n) = \frac{1}{p} \sum_{i=1}^p \log_2(\text{rank}_b^{(i)}). \quad (1.54)$$

As n increases, GE tends to decrease.

The metrics above are defined per key byte. To assess the effort associated with recovering the full AES key, the per-byte rankings must be combined into a full-key view. This is addressed by key enumeration, which lists full-key candidates in decreasing order of plausibility, and by rank estimation, which approximates the rank of the correct full key when explicit enumeration is computationally infeasible.

Key Enumeration (KE). Key enumeration is a procedure used to assess full-key effort starting from byte-wise scores. It orders full-key candidates from the most to the least plausible and counts how many candidates must be tested before reaching the correct key. To compare full-key candidates consistently, the per-byte information is expressed through scores that can be combined across bytes, typically probabilities or log-probabilities. Pearson correlation scores are not probabilities and therefore do not directly define a combined full-key score. Let $w_b(k)$ denote an additive log-score for hypothesis $k \in \{0, \dots, 255\}$ at byte b . For a full-key candidate $\mathbf{k} = (k_0, \dots, k_{15})$, the combined score is

$$w(\mathbf{k}) = \sum_{b=0}^{15} w_b(k_b). \quad (1.55)$$

Here, larger values of $w_b(k)$ indicate more plausible hypotheses and candidates are then tested in decreasing $w(\mathbf{k})$ order, up to a maximum of n_{en} tested keys. When explicit enumeration is too costly, rank estimation provides an alternative by approximating the position of the correct key in the same ordering without enumerating candidates explicitly.

Rank Estimation (RE). Rank estimation approximates the rank of the correct full key in the ordering induced by the additive score $w(\mathbf{k})$, without explicitly enumerating full-key candidates. The method operates on the same convention used above, i.e., higher scores correspond to more plausible keys. The procedure for RE follows these steps:

1. For each key byte $b \in \{0, \dots, 15\}$ and each hypothesis $k \in \{0, \dots, 255\}$, an additive log-score $w_b(k)$ is available.
2. For each key byte b , a histogram $\mathbf{h}_b$ of the scores is built, using the same fixed number of bins n_{bin} where $\mathbf{h}_b[u]$ counts how many hypotheses k fall into the u -th score interval.
3. The histogram of full-key scores is obtained by iteratively convolving the per-byte histograms:

$$\mathbf{f} = \mathbf{h}_0 * \mathbf{h}_1 * \dots * \mathbf{h}_{15}. \quad (1.56)$$

4. The resulting vector $\mathbf{f}[u]$ counts how many full keys have a total score falling into the u -th global interval.
5. The total score of the correct key $\mathbf{k}^*$ is computed as $w(\mathbf{k}^*) = \sum_{b=0}^{15} w_b(k_b^*)$, and the corresponding global bin index is denoted u^* .
6. Since candidates are ordered by decreasing score, the rank is obtained by counting how many keys fall into intervals with higher score than u^* . The rank is bounded

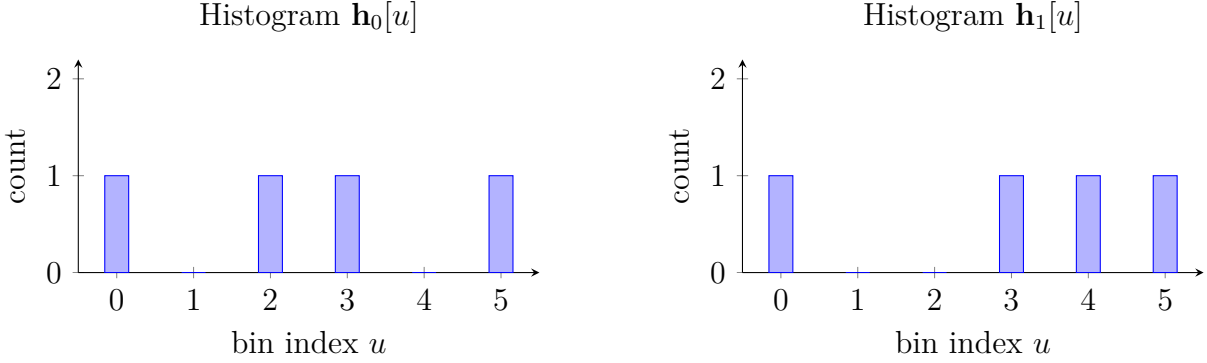


Figure 1.7: Example with $n_{\text{bytes}} = 2$: per-byte histograms $\mathbf{h}_0$ and $\mathbf{h}_1$ obtained by binning the log-scores $w_0(k)$ and $w_1(k)$ into $n_{\text{bin}} = 6$ uniform intervals.

depending on whether the correct key is assumed to be the first or the last element within its score interval:

$$r_{\text{lower}} = 1 + \sum_{u > u^*} \mathbf{f}[u], \quad r_{\text{upper}} = 1 + \sum_{u \geq u^*} \mathbf{f}[u]. \quad (1.57)$$

The example illustrates the procedure in a simplified setting with $n_{\text{bytes}} = 2$ key bytes and with the correct key $\mathbf{k}^* = (01, 00)$. The per-byte probabilities for the first and second byte are, respectively,

$$\begin{aligned} p_0 &= \{(00, 0.8), (01, 0.1), (10, 0.08), (11, 0.02)\}, \\ p_1 &= \{(00, 0.09), (01, 0.5), (10, 0.01), (11, 0.3)\}. \end{aligned}$$

The corresponding log-scores are defined as $w_b(k) = \log_2 p_b(k)$,

$$\begin{aligned} w_0 &= \{(00, -0.322), (01, -3.322), (10, -3.644), (11, -5.644)\}, \\ w_1 &= \{(00, -3.474), (01, -1), (10, -6.644), (11, -1.737)\}. \end{aligned}$$

The values $w_b(k)$ are grouped into $n_{\text{bin}} = 6$ uniform score intervals over $[w_{\min}, w_{\max}]$, where $w_{\min} = \min_{b,k} w_b(k)$ and $w_{\max} = \max_{b,k} w_b(k)$. In this example, $w_{\min} = -6.644$, $w_{\max} = -0.322$ and $\Delta = (w_{\max} - w_{\min})/n_{\text{bin}} \approx 1.054$, producing the per-byte histograms $\mathbf{h}_0$ and $\mathbf{h}_1$, shown in 1.7. Convolution of the two histograms provides the global distribution $\mathbf{f} = \mathbf{h}_0 * \mathbf{h}_1$ shown in 1.8. In this example, $u^* = 6$, $r_{\text{lower}} = 7$ and $r_{\text{upper}} = 9$. Since $n_{\text{bytes}} = 2$, the exact rank can also be computed by enumerating all the candidates, leading to an exact rank $r^* = 9$, which falls within the estimated bounds.

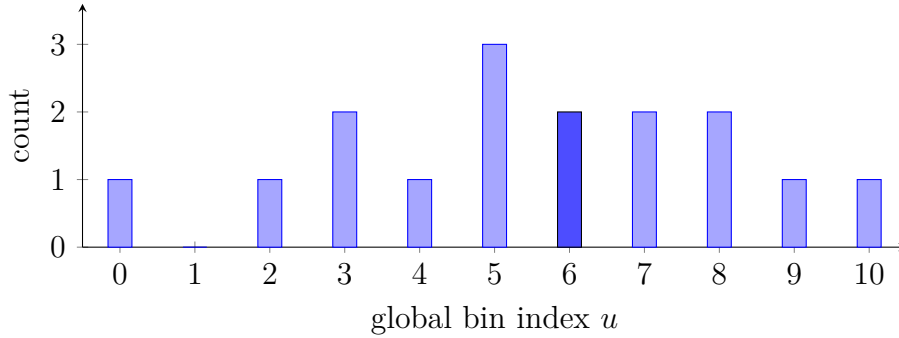


Figure 1.8: Example with $n_{\text{bytes}} = 2$: global histogram $\mathbf{f} = \mathbf{h}_0 * \mathbf{h}_1$ obtained by convolution. The bin $u^* = 6$ corresponds to the correct key.

1.4. Classification and Overview of Side-Channel Attacks

This section presents the main classification for side-channel attacks. Vertical versus horizontal and non-profiled versus profiled distinctions are introduced first, followed by a short overview of representative attack families from the literature.

Side-channel attacks can be classified by how the trace matrix $M \in \mathbb{R}^{n \times t}$ is exploited:

- In **vertical attacks**, the analysis combines many traces acquired from repeated executions of the cryptographic algorithm under the same secret key. The analysis typically focuses on one or more time samples (or a feature derived from a short window) and evaluates how those leakage samples vary across traces as a function of a predicted intermediate. Viewed in terms of the trace matrix M , vertical attacks mainly process selected columns across multiple rows.
- In contrast, **horizontal attacks** focus on the information available within an execution. Rather than relying on many independent traces, they exploit multiple key-dependent operations occurring within the same trace and perform the statistical analysis across different time samples. With respect to M , horizontal attacks mainly process multiple columns within a row (or within a few rows).

Attacks can also be distinguished according to the availability of traces acquired under known keys:

- In **profiled attacks**, a preliminary profiling phase is performed using traces for which the secret key is known, typically collected from the same device or from a closely similar device. This phase is used to learn a statistical leakage model (for example, class-conditional distributions of a selected feature), which is then applied

to traces acquired under an unknown key.

- In **non-profiled attacks**, no profiling phase is assumed: key hypotheses are scored directly on the traces acquired under the unknown key by measuring the statistical dependence between predicted intermediates and the measured leakage.

After introducing the main classification criteria, the remainder of this section summarizes representative attack families, highlighting their core idea and the type of information they require. Since the attacks described in the remainder of this section are typically formulated in a vertical setting, the following overview groups them primarily by profiling availability, distinguishing non-profiled and profiled approaches.

1.4.1. Non-profiled Attacks

This subsection summarizes representative non-profiled attacks, where key hypotheses are evaluated directly on traces acquired under the unknown key, without a separate training phase [3].

Differential Power Analysis (DPA). DPA follows a divide-and-conquer strategy and evaluates each subkey hypothesis by splitting the traces into two groups according to a binary label derived from a predicted intermediate (for example, one selected bit as in 1.30). The hypothesis is scored by measuring how different the leakage is between the two groups and hypotheses are ranked accordingly. Welch’s t -test (section 1.3.1) provides a standard statistic for this setting.

Correlation Power Analysis (CPA). CPA evaluates each subkey hypothesis by comparing a prediction vector derived from an intermediate (under a chosen leakage model) with a leakage feature extracted from the traces. CPA uses Pearson correlation as score (see 1.3.4), and ranks hypotheses by the corresponding correlation score.

1.4.2. Profiled Attacks

This subsection summarizes representative profiled attacks, which rely on an initial profiling phase under known keys to learn a leakage model that is later applied to traces acquired under the unknown key.

Template Attacks (TA). Template attacks build an explicit leakage model during profiling. During profiling phase, traces collected under a known key are used to build one statistical template per class and a Gaussian model is commonly adopted (see 1.1). In the attack phase, each hypothesis is assigned a score by the template model, and

hypotheses are ranked accordingly.

Deep-Learning SCA (DL-SCA). DL-SCA is a profiled attack in which the leakage model is learned from data using a neural network. Compared to template attacks, DL-SCA does not require a Gaussian assumption and learns both features and scoring directly from labeled traces. In the profiling phase, traces are collected under a known key and each trace is assigned a label derived from an intermediate value. The network is trained as a classifier that takes a leakage feature as input (e.g., a set of POIs or a time window) and outputs a vector of class probabilities. In the attack phase, the trained network is evaluated on traces acquired under the unknown key. Its outputs are converted into a score for each key hypothesis by mapping the hypothesis to the corresponding class label, and the scores are aggregated across traces to obtain a ranking of the key candidates. [7]

Soft-Analytical SCA (SASCA). SASCA is a side-channel attack approach that combines probabilistic information extracted from measurements with explicit constraints derived from the target algorithm. The core idea is to represent the implementation as a factor graph. Variable nodes represent unknown quantities, including key bytes and intermediate values, while known inputs (e.g., plaintext bytes) are represented as fixed variables. Two types of factors are used:

- Leakage factors: to selected intermediate nodes and provide soft evidence in the form of a probability vector over the 256 possible byte values, obtained from profiling.
- Operation factors: encode deterministic constraints of the algorithm (e.g., xor or SBox operation), assigning non-zero weight only to combinations of values that satisfy the corresponding relation.

Figure 1.9 illustrates how measurement evidence on an intermediate is combined with algorithmic constraints. Given this graph, SASCA runs a message-passing procedure, called belief propagation, to combine the soft evidence with the operation constraints. Each factor sends a message to its neighboring variables expressing how compatible each variable value is with the current information coming from the other neighbors. Each variable combines incoming messages and produces an updated probability distribution (belief) over its 256 possible values. Message passing is run until convergence criteria are met or a maximum number of iterations is reached. The key nodes yield marginal distributions that can be used to rank key-byte hypotheses [10].

Exact Soft-Analytical SCA (EXSASCA). EXSASCA is a variant of SASCA that aims at exact results when combining measurement evidence with AES constraints. In SASCA, message passing is run on the full AES graph. Because AES contains cycles,

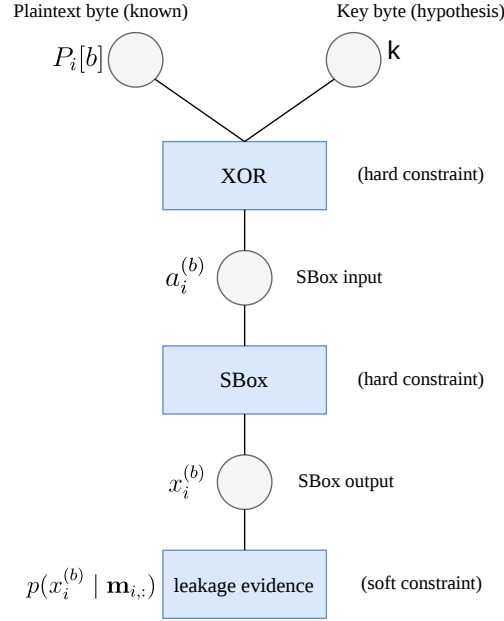


Figure 1.9: Example of factor-graph fragment for AES, with variable nodes (known plaintext, key-byte hypothesis, and intermediates), factor nodes (xor and SBox) and soft evidence attached to an intermediate.

introduced by MixColumns (see 1.2), the procedure is executed on a loopy structure and does not provide guarantees on convergence. If termination is forced after a fixed number of iterations, the correctness guarantees of exact inference are lost. EXSASCA restructures the graph by isolating the constraints related to MixColumns and replacing that subgraph with a single aggregate factor. This factor represents the same MixColumns consistency relation but is stored in a compiled form that supports efficient computation of the sums required by message passing. The compiled factor is built once from the deterministic MixColumns relations and then reused when processing the measurement evidence. After this replacement, message passing runs on a tree structure, where it terminates after a finite number of updates and returns updated probability distributions for the key bytes [11].

2 | Design and Implementation of a Python Library for SCA

This chapter presents the design and implementation of FreeSCA, the Python library developed in this work for side channel analysis. The chapter is organized into six sections. The first section introduces the library, its design goals and its general workflow. The second section presents the FSCA file format and the data converters developed to support dataset import and representation. The third section describes the preprocessing, feature selection and dimensionality reduction techniques implemented in the library. The fourth section discusses the leakage models and the computation of intermediate values used throughout the analysis tools. The fifth section presents the statistical testing methods implemented for leakage assessment. Finally, the last section describes the side channel attacks currently supported by the library.

2.1. Overview, Design Goals and General Workflow

This section introduces FreeSCA and outlines the main objectives that guided its development. In particular, it discusses the motivation for designing a Python library for side channel analysis and presents the general workflow adopted to support data handling, preprocessing, statistical testing and attack execution.

FreeSCA is a Python library developed in this work to support side channel analysis within a unified software framework. The library is designed to cover the main stages of the analysis process, including dataset handling, trace preprocessing, feature selection and dimensionality reduction, leakage assessment through statistical testing and the execution of side channel attacks. Its development was motivated by the need to reduce the fragmentation that often characterizes experimental side channel analysis, where different tasks are frequently carried out through separate scripts and ad hoc implementations. The proposed library addresses this issue by collecting these functionalities within a single package and exposing them through a unified environment so that, once installed, users can run different analysis procedures without relying on disconnected software com-

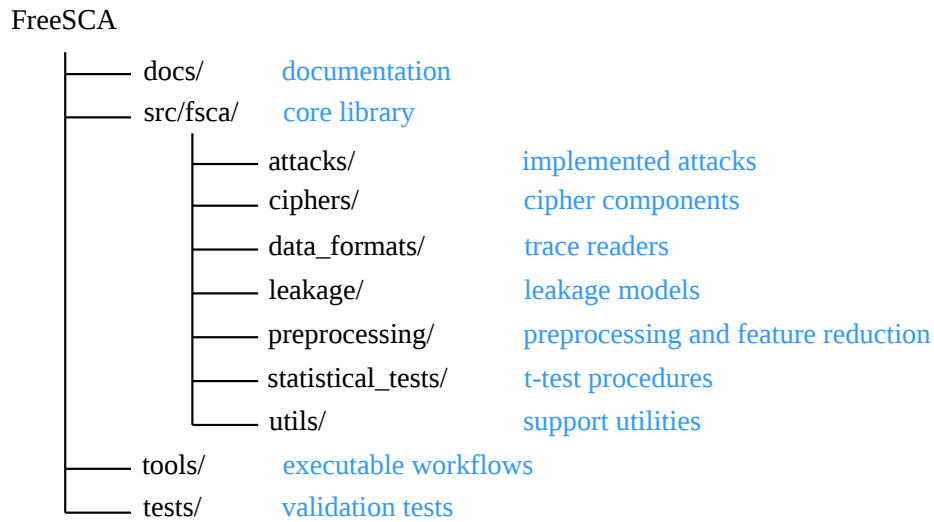


Figure 2.1: General structure of the FreeSCA library, with blue annotations indicating the role of the main components.

ponents. To facilitate future distribution, reuse and reproducibility, the library has been organized and packaged according to the standard model adopted by PyPI, the Python Package Index. PyPI is the main repository used for the distribution of Python packages. Adhering to this model means that the library is structured as a standard Python package, with an explicit description of its modules, dependencies and version information. As a consequence, the software can be installed and integrated into a conventional Python environment through standard Python installation tools. This choice improves portability, simplifies installation and maintenance and makes the distributed version of the library easier to identify and reproduce. Figure 2.1 provides an overview of the organization of the FreeSCA library. The tree highlights the main directories of the project and the structure of the core package, whereas the annotations shown in blue indicate the corresponding functional role of each component.

The package organization is complemented by internal documentation embedded directly in the source code through docstrings. Within the Python ecosystem, docstrings are part of the software interface itself, as they can be accessed through introspection mechanisms, displayed by help utilities, and reused in the generation of technical documentation. In FreeSCA, the main modules, classes and functions are documented in this form, with concise indications of their purpose, inputs, outputs and usage conditions. This contributes both to the practical usability of the library and to its future maintenance.

The design of FreeSCA was guided by a set of objectives related to both methodological coherence and practical usability. In particular, the development of the library followed three main design criteria:

1. **Unified analysis workflow.** A first objective was to provide a single environment in which the main stages of side channel analysis could be integrated within a coherent workflow, from data representation and trace handling to leakage assessment and attack execution.
2. **Modularity and extensibility.** A second objective was to preserve a modular software organization, so that the different analysis components could be developed, extended and reused independently, while still remaining part of a consistent library structure.
3. **Flexible data handling.** Particular attention was devoted to the management of trace data during execution. Rather than treating the dataset as a single monolithic input, the library was designed so that individual procedures can operate incrementally and, when needed, on selected time intervals in the time series which represent measurements. This makes it possible to support different analysis scenarios while retaining flexibility in the processing of large tracesets.

From an operational perspective, Figure 2.2 summarizes the workflow supported by FreeSCA, which is not organized as a rigid sequence of mandatory steps. A common data representation based on the FSCA format facilitates the interaction between the different analysis modules. Datasets can be used directly when already available in this form, or converted into it when necessary. Once represented consistently, they can be processed by different modules according to the analysis objective. As shown in Figure 2.2, the library supports preprocessing procedures such as feature selection and dimensionality reduction, statistical leakage assessment, and the execution of attack procedures including CPA and Template Attacks. Some preprocessing modules can also produce derived FSCA files, which may be stored on disk and reused as input for subsequent stages of the analysis.

2.2. The FSCA File Format and Data Converters

This section is divided into two parts. The first part describes the FSCA file format, which acts as the common data representation used throughout FreeSCA and discusses the main design requirements that motivated its introduction together with its internal organization. The second part presents the data converters currently implemented in the library,

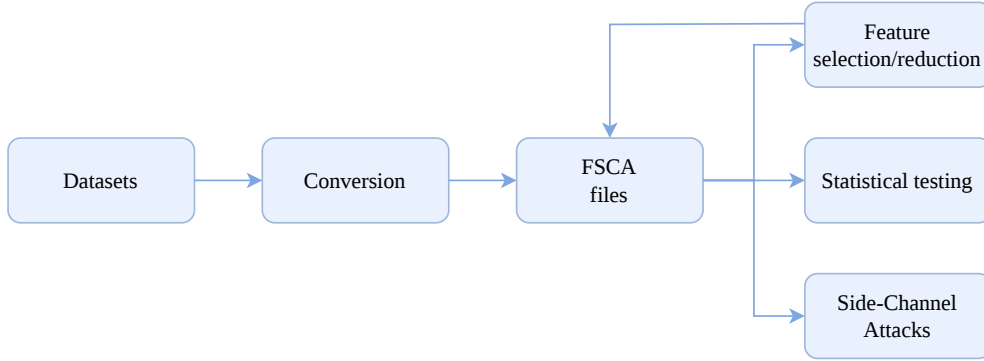


Figure 2.2: Simplified overview of the FreeSCA workflow.

which allow supported input datasets to be transformed into the FSCA representation.

2.2.1. The FSCA File Format

The introduction of a dedicated file format was motivated by the need for a common data representation across the different modules of the library. In the context of side channel analysis, datasets are often stored in heterogeneous formats and may expose different combinations of traces and associated metadata, such as plaintexts, ciphertexts, keys or masks. This heterogeneity complicates the integration of analysis tools, especially when feature selection, statistical testing and attack modules are expected to operate within the same workflow. For this reason, we designed a dedicated file format, named FSCA, to provide a common internal representation for traces and relevant per-trace metadata, while preserving enough flexibility to accommodate different analysis settings and to enable the generation of derived datasets that can be stored and reused in subsequent analysis stages.

From a structural perspective, a FSCA file consists of a fixed-size 1024-byte header followed by a sequence of trace records. Figure 2.3 illustrates the organization of a FSCA file.

Header structure. The header stores the information required to identify the file format and to determine how the subsequent binary data must be interpreted. Figure 2.4 shows the internal organization of the FSCA header in terms of the metadata fields currently defined by the format.

This organization keeps the structural information of the dataset explicitly available within the file itself. The use of a fixed size header also leaves reserved space for possible future extensions of the format, without requiring a redesign of the overall file structure.

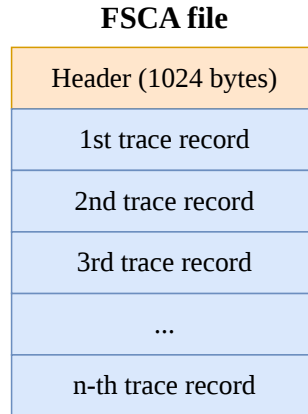


Figure 2.3: General structure of a FSCA file.

The header includes fields devoted to format validation and compatibility. In particular, the magic value (an 8-byte ASCII sequence) allows the reader to verify that the file conforms to the FSCA format. The version field (an unsigned 16-bit integer) makes it possible to distinguish between different revisions of the format and to reject unsupported ones. The header size (an unsigned 16-bit integer) explicitly specifies the extent of the header, which facilitates consistency checks, while the endianness marker (an unsigned 8-bit integer) identifies the byte order used to interpret multi-byte fields (e.g., little endian).

The header also contains fields describing the trace dataset. These include the number of traces contained in the file (an unsigned 64-bit integer), the number of samples per trace (an unsigned 64-bit integer), and the sample type used to encode each measurement (a single ASCII byte). This information is necessary to reconstruct the shape of the dataset and to correctly decode the trace values. In particular, the sample type determines the numeric representation of the measurements and therefore how the trace values must be decoded.

The header also stores metadata describing the acquisition setting and the target under analysis. In particular, the sampling frequency (an unsigned 64-bit integer) specifies how the traces were acquired, while the target frequency (an unsigned 64-bit integer) provides information about the operating frequency of the device under analysis. A timestamp (an unsigned 64-bit integer) can be used to associate the file with a specific acquisition time. Further fields identify the considered implementation and the target itself, namely the implementation type (a fixed-length 16-byte ASCII field) and the target identifier (a fixed-

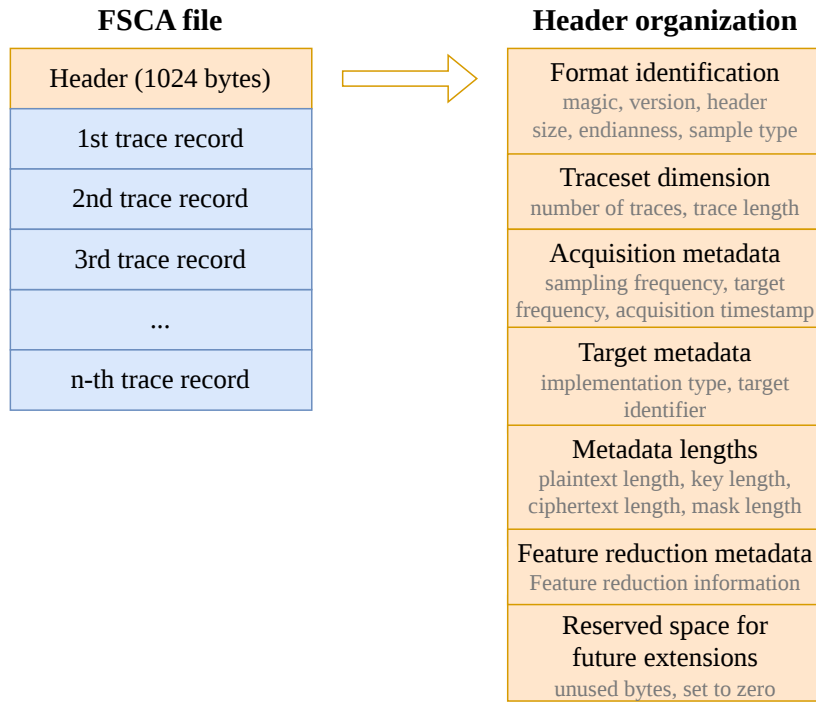


Figure 2.4: Organization of the FSCA header.

length 64-byte ASCII field), so that the traces can be linked to a specific experimental context.

Finally, the header stores the lengths of the optional per-trace metadata fields, namely plaintext, key, ciphertext and mask. These values, each represented as an unsigned 64-bit integer, determine which metadata are present in each record and how they must be parsed.

The header also includes a feature reduction field (a fixed-length 16-byte ASCII field) describing whether feature selection or dimensionality reduction has already been applied to the traces and, when applicable, which method was used. This makes it possible to distinguish raw datasets from derived ones and to keep track of the transformations that produced a given file.

Trace record structure. After the header, the file contains a sequence of trace records, one for each acquired measurement. Each trace record stores the samples of a single trace and, when available, the metadata associated with that trace. In the current version of the format, these metadata may include plaintext, key, ciphertext and mask values.

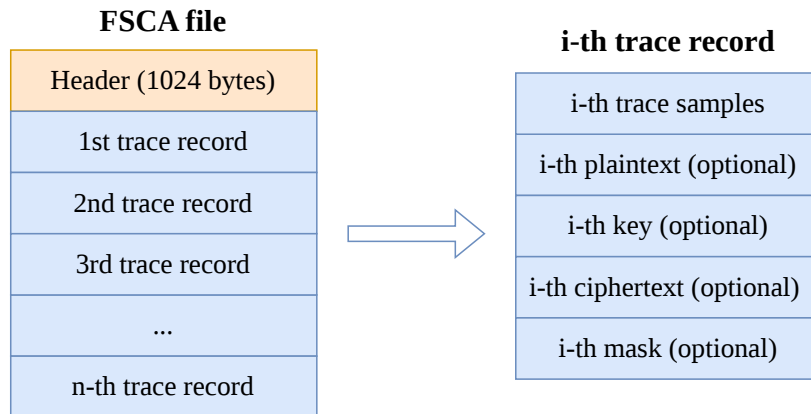


Figure 2.5: General structure of a trace record in the FSCA format.

The internal structure of a trace record is illustrated in Figure 2.5. This representation highlights how the measured samples and the optional metadata fields are organized within each record of the FSCA file.

The presence of these fields depends on the characteristics of the represented dataset and on the requirements of the intended analysis. In practical side channel settings, not all datasets provide the same auxiliary information. Plaintexts and ciphertexts identify the known inputs or outputs associated with each measurement, keys may be available in controlled experimental settings and mask values become relevant when protected implementations are considered. Consequently, the format does not impose a fixed set of per-trace metadata. The metadata stored after each trace are determined by the corresponding length fields specified in the header. A zero length indicates that the field is absent, whereas a non-zero length determines its size within the trace record. In this way, each file can store only the information that is actually available and relevant for the considered acquisition or analysis setting.

Operational characteristics. Beyond its internal organization, the FSCA format was designed so that the stored traces can be accessed and reused efficiently by the procedures implemented in FreeSCA. Since the header specifies the structure and size of each trace record, the file supports both sequential processing of the full traceset and direct access to individual traces, without requiring a scan of the entire dataset. This is useful in analysis settings where traces must be inspected selectively or processed repeatedly by different tools.

The format also supports selective reading of the available information. Depending on the considered procedure, traces can be loaded together with only the metadata fields that

are actually required, rather than with the full set of associated metadata. In addition, the sample vectors can be read with optional trimming, so that the analysis is restricted to a selected temporal portion of the measurement. These possibilities are particularly relevant for large tracesets and for procedures that operate incrementally or batch-wise.

Finally, the same format is used not only for acquisition data, but also for datasets generated during the analysis workflow. As a consequence, procedures such as feature selection or dimensionality reduction can produce derived FSCA files that remain directly compatible with the other components of the library and can therefore be reused in subsequent analysis stages.

2.2.2. Data Converters

The converters implemented in FreeSCA generate FSCA files starting from datasets originally stored according to external formats. For each supported source, the conversion tool reads the traces together with the metadata provided by the original dataset and reorganizes this information according to the FSCA structure.

During this process, the information available in the source dataset is mapped to the corresponding components of the target file. In particular, the header is populated with the parameters required to describe the dataset, while the optional per-trace fields are included only when the corresponding information is present in the source data. In this way, the generated file reflects the actual content of the original dataset without forcing the presence of metadata that are not available.

The conversion therefore produces a new FSCA file, which can then be used directly by the other tools implemented in FreeSCA. At the current stage, converters have been implemented for a limited set of supported sources. The currently available converters are summarized in Table 2.1. More precisely, the currently supported sources include publicly available trace collections for side channel analysis as well as datasets generated within common acquisition environments:

- **ASCAD.** It provides publicly available traces and cryptographic metadata for protected AES implementations [8].
- **DPA Contest.** The DPA Contest traces correspond to an unprotected AES setting and are distributed together with the associated cryptographic information [2].
- **ChipWhisperer.** It denotes an acquisition environment rather than a fixed public dataset. It allows the user to collect traces under configurable experimental conditions and with associated cryptographic metadata [5].

Table 2.1: Summary of the dataset converters currently supported by FreeSCA.

Source	Setting	Available per-trace metadata
ASCAD	Masked	Plaintexts, keys, ciphertexts, masks
DPA Contest	Unmasked	Plaintexts, keys, ciphertexts
ChipWhisperer	Unmasked	Plaintexts, keys, ciphertexts

The resulting datasets can be converted into the FSCA format through the converters implemented in FreeSCA.

2.3. Feature Selection and Dimensionality Reduction

This section describes the methods implemented in FreeSCA for feature selection and trace reduction, which operate on the FSCA datasets introduced in 2.2.1. The implemented functionality includes score based methods, such as the Signal to Noise Ratio (SNR), the Normalized Inter Class Variance (NICV) and the Sum of Squared t-differences (SOST), together with reduction procedures based on Principal Component Analysis (PCA). Although these approaches differ in the way the reduced representation is obtained, they all aim at decreasing the amount of data to be processed while preserving the information that is most relevant for the subsequent analysis.

Several implementation choices are shared across the methods described in this section. In particular:

1. The corresponding tools are designed to process traces in batches rather than loading the entire dataset into memory at once;
2. The required statistics are estimated incrementally, which makes the implemented methods applicable to large datasets while keeping memory usage independent of the total number of traces.

To support this incremental computation, FreeSCA relies on a common update scheme based on **Welford’s algorithm** whenever means and variances must be estimated separately at each sample position of the trace. The corresponding one-pass estimators adopted in the implementation are reported in Appendix A. Since the same update is applied independently at each time sample j , the dependence on j is omitted in the following formulas for readability. Let $m_1, \dots, m_n$ denote the values observed at a fixed sample over the first n traces and let $\hat{\mu}_n$ be the corresponding sample mean after process-

ing n observations. The mean is updated as

$$\hat{\mu}_n = \hat{\mu}_{n-1} + \frac{m_n - \hat{\mu}_{n-1}}{n}, \quad (2.1)$$

while the auxiliary quantity $M_{2,n}$, representing the cumulative squared deviation term after n observations, is updated as

$$M_{2,n} = M_{2,n-1} + (m_n - \hat{\mu}_{n-1})(m_n - \hat{\mu}_n). \quad (2.2)$$

The corresponding variance estimate is then obtained as

$$\hat{\sigma}_n^2 = \frac{M_{2,n}}{n-1}. \quad (2.3)$$

This update scheme is reused throughout FreeSCA whenever means and variances must be estimated incrementally at each sample position of the trace.

2.3.1. Score-based Feature Selection

This subsection describes the score-based feature selection methods implemented in FreeSCA. In these methods, each time sample of the trace is assigned a scalar score computed from a chosen partition of the dataset into classes and from the statistical differences observed among those classes. The resulting score vector can then be used to rank the original trace coordinates and to identify points of interest for subsequent feature selection steps.

SNR and NICV. For SNR and NICV, at each time sample, the dataset is modeled as a set of observations drawn from class-dependent random variables, each describing the leakage distribution of samples associated with a given class. Starting from this interpretation, SNR and NICV quantify how strongly the observed variability is related to the separation between classes, as discussed in Section 1.3.2. The implemented computation therefore assumes that the input traces are organized into labelled subsets, each associated with a distinct class. Depending on the considered analysis, these classes may correspond, for example, to the Hamming weight of an intermediate value, to the two subsets induced by a selected bit or to the 256 possible values of one byte. In the current implementation, the traces belonging to each class are processed separately and the corresponding class-dependent statistics are estimated through the Welford incremental procedure.

More precisely, let $\mathbf{V}$ denote the set of class labels induced by the chosen partition of the dataset. Since the same computation is repeated independently at each time sample

j , the dependence on j is omitted in the following formulas for readability. For each class $v \in \mathbf{V}$, the implementation estimates the corresponding class mean $\hat{\mu}_v$ by processing incrementally the traces associated with that label through Welford's algorithm. Once the class means have been obtained for all $v \in \mathbf{V}$, their average

$$\hat{\mu} = \frac{1}{|\mathbf{V}|} \sum_{v \in \mathbf{V}} \hat{\mu}_v \quad (2.4)$$

is computed and used to estimate the class dependent contribution

$$\widehat{\text{Var}}_d = \frac{1}{|\mathbf{V}| - 1} \sum_{v \in \mathbf{V}} (\hat{\mu}_v - \hat{\mu})^2. \quad (2.5)$$

In the implemented workflow, the class means $\hat{\mu}_v$ are estimated incrementally because they are obtained from the trace data themselves, which are processed batch by batch in order to avoid loading the full dataset into memory. By contrast, the quantity $\hat{\mu}$ is evaluated only after all class means have been determined and is therefore computed directly from their collection. In parallel, the total variance $\widehat{\text{Var}}_{\text{tot}}$ is estimated over the complete set of traces by applying the same Welford based update scheme to all traces taken together.

Rather than estimating the intra-class contribution to the total variance, interpreted as the noise variance $\hat{\sigma}^2$, separately, the implementation derives it from the variance decomposition:

$$\widehat{\text{Var}}_{\text{tot}} = \widehat{\text{Var}}_d + \hat{\sigma}^2, \quad (2.6)$$

which yields

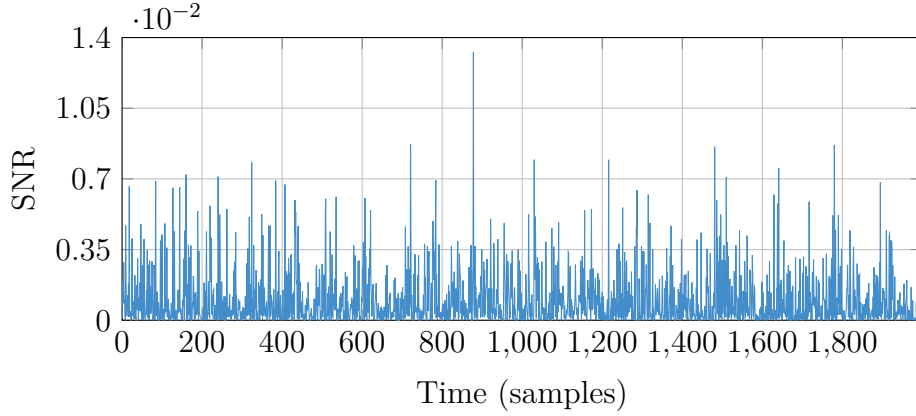
$$\hat{\sigma}^2 = \widehat{\text{Var}}_{\text{tot}} - \widehat{\text{Var}}_d. \quad (2.7)$$

Here, $\hat{\sigma}^2$ is interpreted as the estimated noise variance. The implemented formulas therefore become

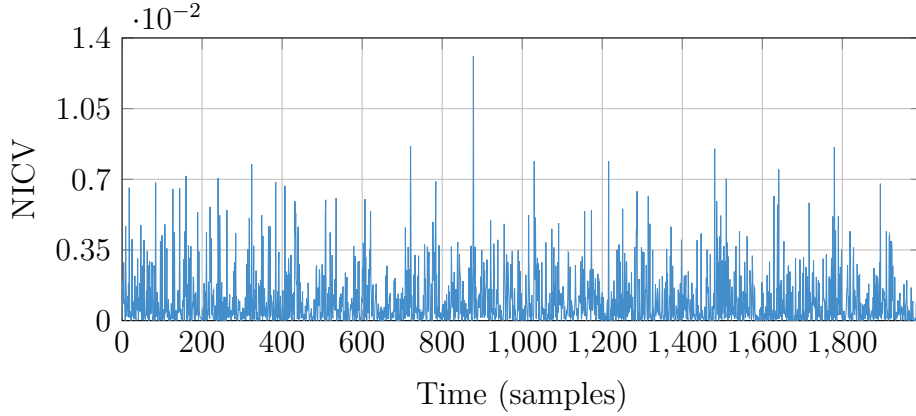
$$\widehat{\text{NICV}} = \frac{\widehat{\text{Var}}_d}{\widehat{\text{Var}}_{\text{tot}}}, \quad \widehat{\text{SNR}} = \frac{\widehat{\text{Var}}_d}{\hat{\sigma}^2}. \quad (2.8)$$

The resulting NICV and SNR values are exported as one dimensional score vectors in CSV form, defined over the original trace coordinates. As an example, Figure 2.6 shows the two metric profiles obtained from a dataset of 1000 traces over a segment of length 2000 samples, and the corresponding local maxima can guide the selection of informative samples.

SOST. FreeSCA also implements SOST as an additional score-based criterion for feature



(a) SNR profile



(b) NICV profile

Figure 2.6: Example SNR and NICV profiles obtained from a two class trace partition over 2000 samples.

selection. As for SNR and NICV, the objective is to associate each time sample of the original traces with a scalar score, using a given class partition of the dataset. In this case, the score measures how strongly the classes are separated at a fixed point of the trace by combining all pairwise class comparisons. Let $\hat{\mu}_v$ and $\hat{\sigma}_v^2$ denote, respectively, the estimated mean and variance for class v , and let $|Q_v|$ be the number of traces belonging to that class. For two distinct classes v and u at a fixed time sample, define the pairwise contribution

$$\Delta_{v,u} = \frac{(\hat{\mu}_v - \hat{\mu}_u)^2}{\sqrt{\frac{\hat{\sigma}_v^2}{|Q_v|} + \frac{\hat{\sigma}_u^2}{|Q_u|}}}. \quad (2.9)$$

This term increases when the difference between the two class means is large relative to the corresponding class variances and cardinalities. For a fixed time sample, the implemented SOST score is obtained by evaluating this contribution for all pairs of distinct classes

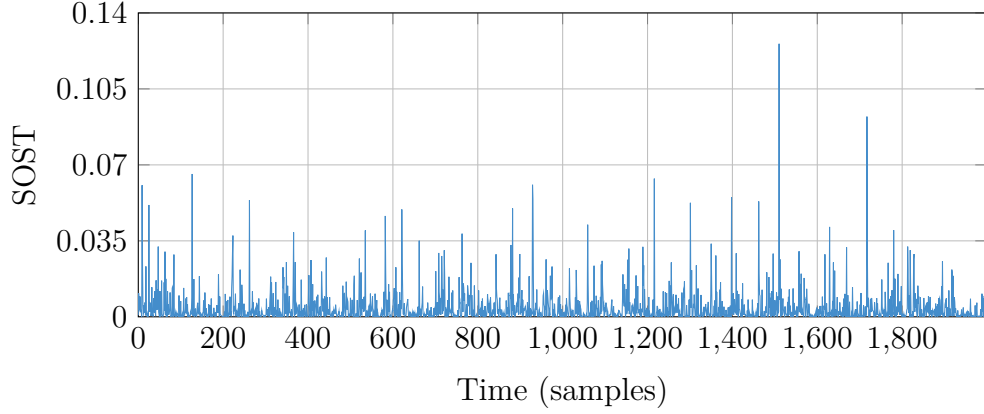


Figure 2.7: Example SOST profile obtained from a two class trace partition over 2000 samples.

and by summing the resulting values. The SOST score at the considered time sample is therefore defined as

$$\widehat{\text{SOST}} = \sum_{v \in \mathbf{V}} \sum_{u \in \mathbf{V} \setminus \{v\}} \Delta_{v,u}. \quad (2.10)$$

In the current implementation, the class means, class variances and class cardinalities required by this computation are estimated from the labelled trace subsets using the Welford incremental procedure. Once these quantities have been accumulated, the pairwise contributions are evaluated at each time sample and combined to produce the final score vector. The resulting SOST values are exported as a one dimensional CSV file defined over the original trace coordinates. As an example, Figure 2.7 shows the SOST profile obtained from a dataset of 1000 traces over a segment of length 2000 samples. The corresponding local maxima indicate sample positions that may be considered as candidate features.

2.3.2. Principal Component Analysis (PCA)

Unlike score based methods, which assign a scalar score to the original time samples, PCA does not evaluate these samples separately. Its purpose is instead to derive a linear transformation to be applied to each trace, obtaining information from the covariance among the coordinates of the trace dataset. In FreeSCA, PCA is therefore used to estimate the transformation associated with the principal components, which is then applied in a separate step to generate reduced datasets.

The implemented procedure starts from the complete set of input traces and first estimates the point-wise mean and variance over the dataset using the Welford update scheme. During the same pass, the implementation also records the point-wise minimum

and maximum values. These quantities make it possible to optionally preprocess the traces before covariance estimation through two dedicated flags enabling standardization or normalization.

Let $\mathbf{m}_i \in \mathbb{R}^t$ denote the i -th trace and let $\hat{\boldsymbol{\mu}}_n \in \mathbb{R}^t$ be the estimated mean trace after processing n traces. The covariance matrix is accumulated incrementally through a matrix valued update, whose corresponding one-pass estimator is reported in Appendix A. More precisely, after reading the n -th trace, the mean trace is updated as

$$\hat{\boldsymbol{\mu}}_n = \hat{\boldsymbol{\mu}}_{n-1} + \frac{\mathbf{m}_n - \hat{\boldsymbol{\mu}}_{n-1}}{n}, \quad (2.11)$$

while the auxiliary matrix $Q_n \in \mathbb{R}^{t \times t}$, used to accumulate the terms required for covariance estimation, is updated as

$$Q_n = Q_{n-1} + (\mathbf{m}_n - \hat{\boldsymbol{\mu}}_{n-1})(\mathbf{m}_n - \hat{\boldsymbol{\mu}}_{n-1})^T. \quad (2.12)$$

The covariance estimate is then obtained as

$$\hat{\Sigma}_n = \frac{Q_n}{n-1}, \quad (2.13)$$

where $\hat{\Sigma}_n \in \mathbb{R}^{t \times t}$ is the estimated covariance matrix after n traces. Once the covariance matrix has been estimated, the PCA transformation is obtained from its singular value decomposition,

$$\hat{\Sigma}_n = U S V^T. \quad (2.14)$$

Since $\hat{\Sigma}_n$ is real and symmetric, its singular value decomposition is equivalent to an eigenvalue decomposition. This is the step from which the PCA transformation is obtained.

In particular:

- The columns of U define an orthogonal basis of the original trace space and therefore represent the principal directions identified from the covariance structure of the dataset. These directions are the principal components. Each of them is a vector in $\mathbb{R}^t$, so its entries specify the coefficients of a linear combination of the original time samples.
- The entries of S are the singular values and, in the present case, coincide with the eigenvalues of the covariance matrix. Each value therefore quantifies the variance associated with the corresponding principal direction given by a column of U . The values are ordered from the largest to the smallest, so that the first principal

components are those associated with the largest contribution to the total dataset variance.

- The matrix V , for a real symmetric covariance matrix, corresponds to the same orthogonal basis in transposed form.

In the implementation, the matrices U , S , and V are exported together with the estimated mean trace. These quantities define the PCA transformation and provide the information required for its subsequent application.

2.3.3. Generation of reduced FSCA datasets

The purpose of this subsection is to show how the quantities computed by SNR, SOST and PCA can be applied to the traces in order to generate reduced datasets. The computation of these methods does not by itself reduce the dimensionality of the data. Instead, it produces scores or transformations that can subsequently be used to construct a reduced representation. Working with reduced datasets is useful when the original traces are long, when the computational cost of the following analysis steps must be limited or when only a restricted part of the available information is useful for the subsequent analysis.

The outputs produced by the considered methods are the following:

- for **SNR** and **SOST**, a score vector defined over the original trace coordinates;
- for **PCA**, the estimated mean trace together with the matrices U , S and V obtained from the decomposition of the covariance matrix.

FreeSCA provides dedicated application tools that use these quantities to generate reduced datasets in FSCA form. The required inputs and the corresponding operations depend on the considered reduction procedure:

- **Score based feature selection.** The application tool takes as input traces stored in FSCA format together with a score vector obtained from SNR or SOST and defined over the same trace coordinates. The user specifies how many time samples must be retained. The implementation first verifies that the score vector is compatible with the trace length of the input dataset and then identifies the indices associated with the largest score values. The same coordinate selection is subsequently applied to all traces. The output therefore consists of new FSCA datasets in which each trace contains only the selected samples, so that the trace length is reduced to the number of retained coordinates.
- **PCA based transformation.** The application tool takes as input traces stored in

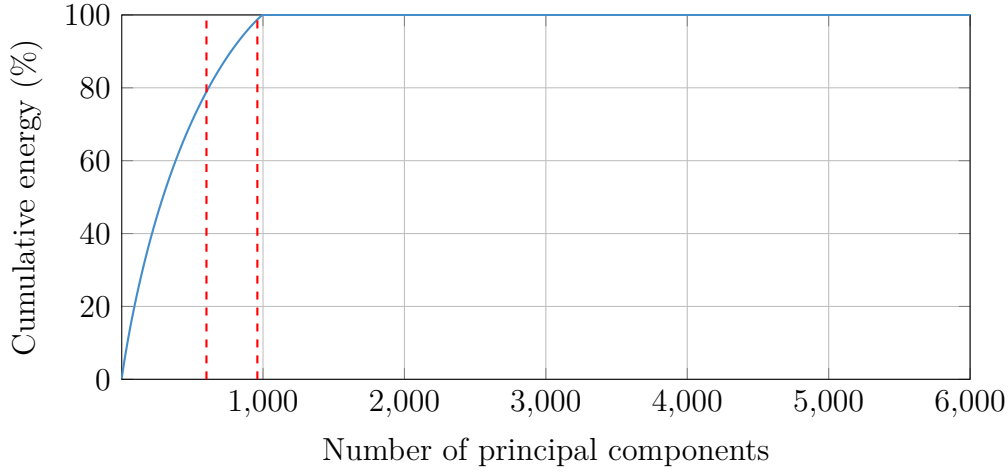


Figure 2.8: Example cumulative energy profile obtained from the PCA decomposition. The red dashed lines indicate the number of principal components required to retain 80% and 99% of the cumulative energy, respectively.

FSCA format together with the mean trace and the matrices U , S and V produced by the PCA computation step. The user specifies how many principal components must be retained. To support this choice, the tool computes from S the cumulative proportion of total variance explained by the first k principal components. This curve is displayed to the user and is used to propose a default number of retained components, defined as the smallest k for which at least 99% of the total variance is explained. The proposed value may then be accepted or modified by the user. As an example, Figure 2.8 shows the cumulative energy profile derived from the eigenvalues returned by the PCA decomposition. The plot reports, for each value of k , the percentage of total variance explained by the first k principal components and therefore provides a practical criterion for choosing how many components to retain.

The implementation also verifies that the trace length is consistent with the dimensions of the stored mean trace and transformation matrices. Once the number of components has been fixed, the matrix U_k formed by the first k columns of U is extracted. Each trace is then centered by subtracting the mean trace and multiplied by U_k . The output therefore consists of new FSCA datasets in which each trace is represented only by the retained PCA coordinates, so that the trace length is reduced to the number of selected principal components.

In all these cases, the reduced traces are written again as new FSCA datasets. During this step, the available per-trace metadata, such as plaintexts, keys, ciphertexts and masks,

are preserved whenever present in the input data. In addition, the applied reduction method is recorded in the corresponding FSCA header field, so that the reduced datasets remain self descriptive and can be reused directly by subsequent statistical tests or attack tools.

2.4. Intermediate Computation and Leakage Models

This section describes the components implemented in FreeSCA to derive, from the available trace metadata, the auxiliary quantities required by several analysis procedures. In contrast with feature selection and reduction methods, which operate directly on the measured traces, statistical tests and attack algorithms often require additional quantities that are not stored explicitly in the dataset, such as class labels, predicted leakage values or key-dependent hypotheses. These quantities must therefore be reconstructed from the available metadata through suitable cryptographic computations and leakage models. The purpose of this section is to describe how FreeSCA implements this intermediate computation layer and how the resulting quantities are made available to the corresponding analysis procedures.

AES support components. Since our testcase implementation is an AES implementation, the reconstruction of these auxiliary quantities relies on reusable implementations of the AES operations and constants. The functioning of AES-128 was discussed in Section 1.2. The present paragraph focuses instead on how the corresponding building blocks are organized and reused within the library.

FreeSCA therefore provides reusable AES components, including AddRoundKey, SubBytes, ShiftRows, MixColumns, round transformations and the constants used in key-schedule computations. These elements are defined by the AES specification and are therefore available independently of any specific dataset. Their role within the library is to provide a common computational basis for deriving the AES quantities required by the analysis procedures.

The implementation of these transformations relies on tabulated forms whenever this is more convenient than repeated direct computation. In particular, the AES SBox is stored explicitly according to the standard substitution table, and the Hamming weight function is tabulated for all byte values between 0 and 255. In addition, for each byte value $x \in \{0, \dots, 255\}$, the quantity $x \oplus \text{SBox}(x)$ is precomputed. This corresponds to the bitwise difference between the input and the output of the AES SBox, and it is used to estimate the power consumption in the case where the SBox output is stored in the same synchronous element as its input.

The actual quantities used in the analysis are then obtained by applying these AES components to the metadata available for each trace, such as plaintext bytes and key bytes. The use of precomputed tables reduces the cost of repeated evaluations and keeps the computation of derived quantities consistent across the library.

Hypothesis matrices and leakage models. Once the AES quantities required by the analysis have been determined, FreeSCA transforms them into the prediction quantities needed by procedures based on key hypotheses. The theoretical definition of leakage models was discussed in Section 1.2 and the present paragraph focuses on how these models are evaluated within the library.

For a fixed byte index b , the current implementation first constructs the matrix of intermediate values associated with all traces and all key-byte hypotheses. In particular, given the plaintext byte $P_i[b]$ of trace i and a hypothesis $k \in \{0, \dots, 255\}$, the xor intermediate is computed as

$$v_{i,k}^{(b)} = P_i[b] \oplus k. \quad (2.15)$$

This produces a matrix $V^{(b)} \in \mathbb{R}^{n \times 256}$, where each row corresponds to one trace and each column to one key hypothesis. Starting from this representation, the supported leakage models are applied in order to generate the corresponding prediction matrix $H^{(b)}$. In the current implementation, the supported models are the following:

- The Hamming weight model applied directly to the xor intermediate,

$$h_{i,k}^{(b)} = \text{hw}(v_{i,k}^{(b)}); \quad (2.16)$$

- The Hamming weight model applied to the SBox output,

$$h_{i,k}^{(b)} = \text{hw}(\text{SBox}(v_{i,k}^{(b)})); \quad (2.17)$$

- The Hamming distance model between SBox input and output,

$$h_{i,k}^{(b)} = \text{hw}(v_{i,k}^{(b)} \oplus \text{SBox}(v_{i,k}^{(b)})). \quad (2.18)$$

From an implementation perspective, these model applications are carried out through the precomputed tables available in the library. Through a common interface, the same matrix $V^{(b)}$ can be reused with different model choices, and both the target byte and the leakage model are handled as configurable library parameters.

Trace partitioning from intermediate values. Intermediate values can also be used to partition the traces into classes. For this purpose, FreeSCA provides a dedicated tool that partitions traces into two output datasets according to the value of a user selected bit of a chosen AES intermediate. In the current implementation, the considered intermediate is the SBox output associated with one targeted byte position. This partition is useful whenever the analysis requires two trace classes defined by a selected property of the targeted intermediate value.

More precisely, let $P_i[b]$ and $K_i[b]$ denote, respectively, the plaintext byte and the key byte associated with trace i at the targeted byte index b . The corresponding AES intermediate is computed as

$$v_i^{(b)} = \text{SBox}(P_i[b] \oplus K_i[b]), \quad (2.19)$$

where $v_i^{(b)}$ is therefore the SBox output associated with the i -th trace at byte position b . A bit position is then selected by the user and each trace m_i is assigned to one of two classes according to whether that bit of $v_i^{(b)}$ is equal to 0 or 1. The resulting partition therefore depends on a specific bit of the selected intermediate value.

From the implementation point of view, the tool can process either a single FSCA dataset or multiple FSCA files. This makes it possible to construct the two output datasets, from one or more input sources, with a predefined number of traces in each class. The tool requires plaintext and key metadata to be present in the processed FSCA files, since both quantities are necessary to evaluate the selected AES intermediate. It also verifies the structural consistency of the input traces and preserves the available metadata in the generated outputs. The final result consists of two FSCA files, corresponding to the two possible values of the selected bit, which can then be reused directly in subsequent procedures requiring a binary partition of the dataset.

2.5. Statistical Tests

This section describes the statistical tests implemented in FreeSCA for leakage detection. These procedures are used to determine whether two trace classes exhibit statistically significant differences and therefore whether the measured data contain evidence of information leakage. In contrast with attack algorithms, their purpose is not to recover the secret key, but to assess the presence of distinguishable leakage under a given partition of the traces.

The current implementation includes three variants of the t -test:

1. First order univariate test;

2. Higher order univariate test based on one-pass moment estimation;
3. Bivariate test defined on pairs of time samples.

The following subsections describe how these procedures are implemented within the library and how their statistics are accumulated efficiently from the trace data.

2.5.1. First Order *t*-test

The theoretical definition of Welch's *t*-test and its use for leakage detection were discussed in Section 1.3.1. In FreeSCA, the first order univariate *t*-test is implemented as a sample-wise procedure applied independently to each time sample of two trace sets. This test is effective when leakage manifests itself as a first order effect in the raw traces, namely as a statistically significant difference between the sample means of the two groups. For this reason, it is the natural starting point when the objective is to detect unmasked leakage or leakage effects that are visible directly on the raw traces.

The input of the tool consists of two trace datasets representing the two groups to be compared. In a non-specific setting, these two sets may correspond for example to fixed and random inputs, whereas in a specific setting they may be obtained from a binary partition derived from an intermediate value, as discussed in Section 1.3.1. The implementation processes the two datasets in batches. If the trace lengths differ, the longer dataset is trimmed so that the test is evaluated on aligned sample positions only.

Since the same computation is repeated independently at each time sample, the sample index is omitted in the following formulas for readability. For each group $g \in \{0, 1\}$, the traces are processed incrementally through Welford's scheme, as in 2.3. At each step, the implementation updates the number n_g of processed traces, the current mean estimate $\hat{\mu}_g$ and an auxiliary quantity $M_{2,g}$, which accumulates the squared deviations required to compute the sample variance at the end of the procedure. Once all traces of the two groups have been processed, the variance estimate used by the test is obtained as

$$\hat{\sigma}_g^2 = \frac{M_{2,g}}{n_g - 1}, \quad g \in \{0, 1\}. \quad (2.20)$$

For the considered sample position, the Welch statistic is then computed as

$$t = \frac{\hat{\mu}_1 - \hat{\mu}_0}{\sqrt{\frac{\hat{\sigma}_1^2}{n_1} + \frac{\hat{\sigma}_0^2}{n_0}}}. \quad (2.21)$$

Repeating the same computation independently for all sample positions yields the final one

dimensional vector of t values over the trace, that is, the statistic profile produced by the tool. After the sample-wise Welch statistic has been computed, the implementation also evaluates the corresponding degrees of freedom through the Welch-Satterthwaite approximation. This approximation is used to associate the Welch statistic with an appropriate reference Student t distribution when the two groups may have different variances. In this way, the statistic can be interpreted using an effective number of degrees of freedom:

$$\nu \approx \frac{\left(\frac{\hat{\sigma}_1^2}{n_1} + \frac{\hat{\sigma}_0^2}{n_0}\right)^2}{\frac{\left(\frac{\hat{\sigma}_1^2}{n_1}\right)^2}{n_1 - 1} + \frac{\left(\frac{\hat{\sigma}_0^2}{n_0}\right)^2}{n_0 - 1}}. \quad (2.22)$$

The quantity ν is not used to define the t statistic itself, but to determine the Student t distribution used to compute the corresponding significance threshold.

Since the statistic is computed independently for each sample position, the resulting profile corresponds to a set of simultaneous statistical tests. If the same significance level α were used for each position, the probability of observing at least one false positive over the whole trace would increase with the number of tested samples. As a result, the interpretation of individual threshold crossings would not reflect the selected global significance level for the full trace. To control this multiple testing effect, the implementation applies a Bonferroni correction by assigning to each individual test a reduced significance level. If U denotes the number of tested sample positions and α the selected global significance level, the corresponding per-test significance level is:

$$\alpha^* = \frac{\alpha}{U}. \quad (2.23)$$

For each sample position j , the associated two-sided threshold is then obtained from the Student t distribution with the corresponding degrees of freedom ν ,

$$\tau = t_{1-\alpha^*/2, \nu}. \quad (2.24)$$

where $t_{1-\alpha^*/2, \nu}$ denotes the $(1 - \alpha^*/2)$ -quantile of the Student t distribution with ν degrees of freedom. The resulting decision rule is therefore applied sample-wise by comparing t with the two-sided thresholds $\pm\tau$.

The tool outputs a CSV file containing the sample-wise t values and a plot showing the statistic together with the corresponding positive and negative Bonferroni thresholds.

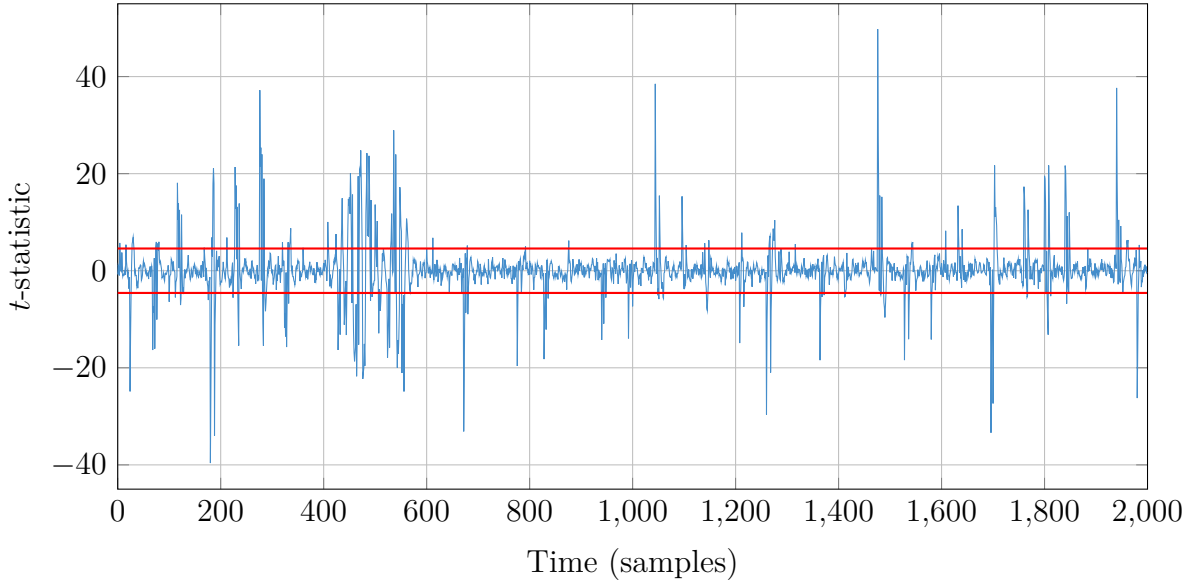


Figure 2.9: Example of t -test profile over 2000 samples, together with the corresponding Bonferroni significance thresholds.

Figure 2.9 shows an example of a t -test profile computed over 2000 samples by comparing one trace group acquired with fixed key and fixed plaintext against a second trace group acquired with fixed key and random plaintext, together with the corresponding Bonferroni thresholds derived for a global significance level $\alpha = 0.01$. Sample positions for which the absolute value of the statistic exceeds the corresponding threshold are interpreted as positions where the two trace sets differ significantly according to the selected significance level. In the context of leakage detection, these positions indicate time samples at which the measured traces provide evidence of data dependence.

2.5.2. Higher Order t -test

The higher order Welch test was introduced in Section 1.3.1. In leakage detection, this variant becomes relevant when the difference between the two groups is not visible at the level of the raw trace mean, but emerges only after transforming the samples through higher-order features. This situation is typical of protected implementations, where countermeasures such as masking are intended to suppress or strongly reduce first order leakage. In a masked implementation, the sensitive value is not manipulated directly, but is first combined with random masks. Consequently, the leakage observed at a single time sample is mainly related to a randomized intermediate value, so its mean may no longer distinguish the two groups considered by the test. The sensitive information is therefore hidden at first order rather than removed, since it can still affect higher order properties

of the leakage distribution or, more generally, the joint behavior of samples corresponding to related masked operations. Higher order tests exploit this effect by applying the statistical test to transformed samples, making dependencies that are cancelled at first order observable again.

For this reason, FreeSCA implements a higher order variant of Welch's test based on central moments [9]. The tool takes as input two trace datasets, representing the two classes to be compared, and an integer order $d \geq 2$, and applies the test independently to each time sample after constructing the corresponding higher order quantity. As in the first order case, the two datasets are processed in batches. If the trace lengths differ, the longer dataset is trimmed so that the statistic is evaluated on aligned sample positions only.

Before introducing the higher order test in its implemented form, it is necessary to recall the definitions of the statistical quantities on which it is based. Since the same computation is repeated independently at each time sample, the sample index is omitted in the following formulas for readability. Let $\mathcal{X}$ denote the scalar random variable associated with one fixed sample position of the traces. In the first order test, the comparison between the two groups is performed directly on the first moment of this variable, namely on its mean. In the higher order setting, the comparison is instead based on quantities derived from moments of order greater than one, so that differences not visible at the level of the mean may still become detectable.

To describe these quantities, let

$$M_d = \mathbb{E}[\mathcal{X}^d] \quad (2.25)$$

denote the moment of order d . Since raw moments depend directly on the position of the distribution with respect to the origin, the implementation relies instead on central moments,

$$CM_d = \mathbb{E}[(\mathcal{X} - \mu)^d], \quad (2.26)$$

where $\mu = \mathbb{E}[\mathcal{X}]$. Central moments characterize the distribution relative to its mean rather than relative to the origin. In particular, the second central moment

$$CM_2 = \mathbb{E}[(\mathcal{X} - \mu)^2] \quad (2.27)$$

coincides with the variance. For orders $d > 2$, the implementation also uses standardized moments, defined as

$$SM_d = \mathbb{E} \left[\left(\frac{\mathcal{X} - \mu}{\sigma} \right)^d \right], \quad (2.28)$$

where $\sigma^2 = CM_2$. By substituting the definition of σ , the same quantity can be written as

$$SM_d = \frac{CM_d}{CM_2^{d/2}}. \quad (2.29)$$

This normalization expresses the moment of order d on a scale relative to the variance of the distribution. These definitions clarify which statistical quantities are required by the higher order test.

For each group $g \in \{0, 1\}$, the implementation must recover the empirical central moments

$$CM_{r,g} = \frac{1}{n_g} \sum_{i=1}^{n_g} (m_i - \hat{\mu}_g)^r, \quad (2.30)$$

where r is used as the index of the moment order, $\hat{\mu}_g$ is the sample mean of group g and n_g is the number of traces in the corresponding group. The values of r required by the implementation depend on the selected test order. Moments up to order d are required to define the higher order feature itself, whereas moments up to order $2d$ are required to evaluate the corresponding variance term used in the final test statistic. For this reason, the implementation must make available central moments up to order $2d$.

To obtain these quantities efficiently, the implementation does not recompute the required central moments from the full trace sets whenever a new trace is processed. Instead, for each order $r = 2, \dots, 2d$ and each group g , the tool maintains a running accumulator $CS_{r,g,n}$, where n is the number of traces processed so far in that group. The quantity $CS_{r,g,n}$ stores the cumulative sum of the terms $(m_i - \hat{\mu}_g)^r$ associated with the central moment of order r , and therefore plays, for that order, the same role that the accumulator M_2 plays for the variance in the first order case. Once all traces of the considered group have been processed, the corresponding central moment is recovered as

$$CM_{r,g} = \frac{CS_{r,g}}{n_g}. \quad (2.31)$$

The one pass recurrences update these accumulators during trace processing, so that the central moments required by the higher order test can be recovered directly from the maintained sums, without recomputation from the full datasets.

Since these accumulators are defined with respect to the group mean, their incremental update requires the mean estimate to be updated at the same time. More precisely, for each group $g \in \{0, 1\}$, after processing the n -th trace value m_n , the mean is updated

according to Welford's scheme as

$$\hat{\mu}_{g,n} = \hat{\mu}_{g,n-1} + \frac{m_n - \hat{\mu}_{g,n-1}}{n}, \quad (2.32)$$

and, for each order $r = 2, \dots, 2d$, the corresponding accumulator is updated through

$$\begin{aligned} CS_{r,g,n} = CS_{r,g,n-1} &+ \sum_{k=1}^{r-2} \binom{r}{k} CS_{r-k,g,n-1} \left(-\frac{m_n - \hat{\mu}_{g,n-1}}{n} \right)^k \\ &+ \left(\frac{n-1}{n} (m_n - \hat{\mu}_{g,n-1}) \right)^r \left(1 - \left(-\frac{1}{n-1} \right)^{r-1} \right). \end{aligned} \quad (2.33)$$

This recurrence makes it possible to maintain, in a single pass over the traces, all the sums needed to recover the central moments required by the test.

At this point, the implementation distinguishes two cases, depending on the selected order. In both cases, the quantity denoted by θ_g is the statistic associated with group g on which the final comparison between the two classes is based. Its role is therefore analogous to that of the group mean in the first order case.

- **Case $d = 2$.** For second order analysis, the tested feature coincides with the second central moment of the group,

$$\theta_g = CM_{2,g}. \quad (2.34)$$

The variance term required by the Welch statistic is then obtained from the fourth centered moment as

$$\hat{\sigma}_{\theta_g}^2 = CM_{4,g} - CM_{2,g}^2. \quad (2.35)$$

- **Case $d > 2$.** For orders greater than two, the tested feature is the standardized moment of order d ,

$$\theta_g = SM_{d,g} = \frac{CM_{d,g}}{CM_{2,g}^{d/2}}, \quad (2.36)$$

In this case, the tested statistic is the standardized moment of order d , which normalizes the central moment with respect to the group variance. The corresponding variance term is estimated as

$$\hat{\sigma}_{\theta_g}^2 = SM_{2d,g} - SM_{d,g}^2 = \frac{CM_{2d,g} - CM_{d,g}^2}{CM_{2,g}^d}. \quad (2.37)$$

In both cases, a sample correction factor is then applied to the variance estimate of the

tested feature, yielding

$$\hat{\sigma}_{\theta_g, c}^2 = \frac{n_g}{n_g - 1} \hat{\sigma}_{\theta_g}^2. \quad (2.38)$$

Once the quantities θ_0 , θ_1 , $\hat{\sigma}_{\theta_0, c}^2$ and $\hat{\sigma}_{\theta_1, c}^2$ have been obtained, the final statistic is evaluated through the same Welch framework used in the first order case,

$$t = \frac{\theta_1 - \theta_0}{\sqrt{\frac{\hat{\sigma}_{\theta_1, c}^2}{n_1} + \frac{\hat{\sigma}_{\theta_0, c}^2}{n_0}}}. \quad (2.39)$$

Repeating the same computation independently for all sample positions yields the final one dimensional vector of t values over the trace, that is, the statistic profile produced by the tool.

After the sample-wise higher order statistic has been computed, the implementation also evaluates the corresponding degrees of freedom through the Welch-Satterthwaite approximation:

$$\nu = \frac{\left(\frac{\hat{\sigma}_{\theta_1, c}^2}{n_1} + \frac{\hat{\sigma}_{\theta_0, c}^2}{n_0} \right)^2}{\frac{\left(\frac{\hat{\sigma}_{\theta_1, c}^2}{n_1} \right)^2}{n_1 - 1} + \frac{\left(\frac{\hat{\sigma}_{\theta_0, c}^2}{n_0} \right)^2}{n_0 - 1}}. \quad (2.40)$$

As in the first order case, the quantity ν is not used to define the statistic itself, but to determine the reference Student t distribution against which that statistic is interpreted.

Bonferroni thresholds are then derived from the selected global significance level and from the number of tested sample positions. The sequence of operations carried out by the implementation, from the one-pass accumulation of the central-power sums to the final computation of the statistic, is summarized in Algorithm 2.1.

The tool outputs a CSV file containing the sample-wise t values and a figure showing the statistic together with the corresponding positive and negative Bonferroni thresholds. Sample positions for which the absolute value of the statistic exceeds the corresponding threshold are interpreted as positions where the two trace sets differ significantly according to the selected significance level. In the context of leakage detection, these positions indicate time samples at which the transformed traces provide evidence of data dependence under the considered higher order feature. Figure 2.10 presents two t -test profiles computed on masked traces over 700 samples, with trace groups defined by partitioning the dataset according to one bit of the SBox output. The associated Bonferroni thresh-

Input: two trace sets Q_0 and Q_1 , test order $d \geq 2$, global significance level α

Output: statistic profile t , degrees of freedom ν and Bonferroni thresholds $\pm\tau$

```

1 Set  $d_{\max} \leftarrow 2d$ 
2 foreach  $g \in \{0, 1\}$  do
3    $n_g \leftarrow 0$  and  $\hat{\mu}_g \leftarrow 0$ 
4   for  $r \leftarrow 2$  to  $d_{\max}$  do
5      $CS_{r,g} \leftarrow 0$ 
        // One-pass accumulation of means and central moment sums
6 while traces remain to be processed do
7   Read one batch  $B_g$  from each trace set  $Q_g$ , for  $g \in \{0, 1\}$ 
8   foreach  $g \in \{0, 1\}$  do
9     foreach trace  $m$  in  $B_g$  do
10      Update  $n_g$  and  $\hat{\mu}_g$ 
11      for  $r \leftarrow 2$  to  $d_{\max}$  do
12        Update  $CS_{r,g}$  through the one-pass recurrence
        // Recovery of the central moments required by the selected order
13 foreach  $g \in \{0, 1\}$  do
14    $CM_{2,g} \leftarrow CS_{2,g}/n_g$ 
15    $CM_{d,g} \leftarrow CS_{d,g}/n_g$ 
16    $CM_{2d,g} \leftarrow CS_{2d,g}/n_g$ 
        // Definition of the tested feature and of its variance term
17 if  $d = 2$  then
18   foreach  $g \in \{0, 1\}$  do
19      $\theta_g \leftarrow CM_{2,g}$ 
20      $\hat{\sigma}_{\theta_g}^2 \leftarrow CM_{4,g} - CM_{2,g}^2$ 
21 else
22   foreach  $g \in \{0, 1\}$  do
23      $\theta_g \leftarrow CM_{d,g}/CM_{2,g}^{d/2}$ 
24      $\hat{\sigma}_{\theta_g}^2 \leftarrow (CM_{2d,g} - CM_{d,g}^2)/CM_{2,g}^d$ 
25 foreach  $g \in \{0, 1\}$  do
26    $\hat{\sigma}_{\theta_g,c}^2 \leftarrow \frac{n_g}{n_g - 1} \hat{\sigma}_{\theta_g}^2$ 
27  $t \leftarrow \frac{\theta_1 - \theta_0}{\sqrt{\hat{\sigma}_{\theta_1,c}^2/n_1 + \hat{\sigma}_{\theta_0,c}^2/n_0}}$  // Higher order Welch statistic
28 Compute  $\nu$  // Degrees of freedom
29  $\alpha^* \leftarrow \alpha/U$  // Bonferroni per-test level
30  $\tau \leftarrow t_{1-\alpha^*/2,\nu}$  // Two-sided critical threshold
31 return  $t, \nu, \pm\tau$ 

```

Algorithm 2.1: One-pass higher order Welch t-test based on central moments

olds are derived for a global significance level $\alpha = 0.01$. In this example, the comparison between the two profiles shows the absence of statistically significant first order leakage

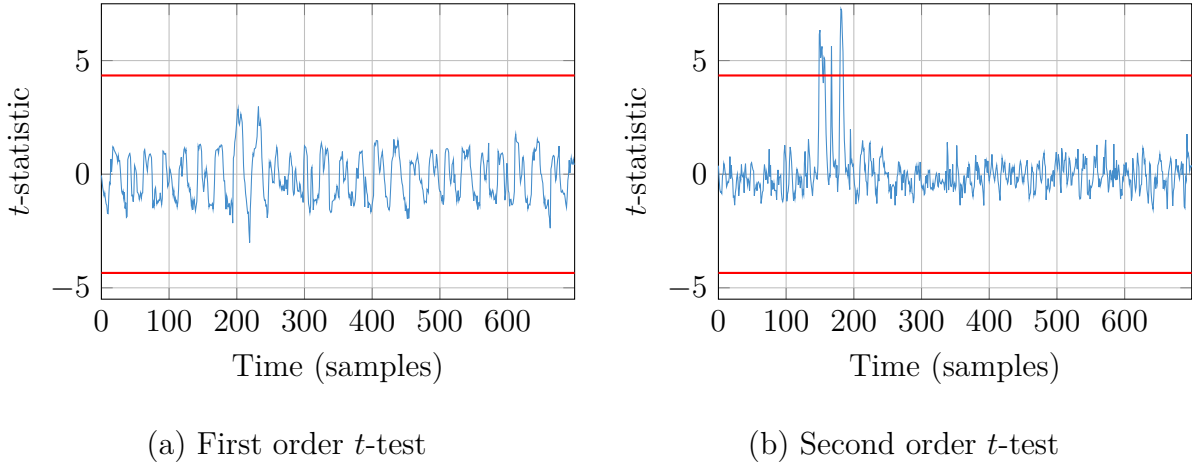


Figure 2.10: Example first order and second order t -test profiles over 700 samples for masked traces, together with the corresponding Bonferroni thresholds.

and the presence of statistically significant second order leakage.

2.5.3. Bivariate t -test

The fork Welch test was introduced in Section 1.3.1. In contrast with the first order univariate test, which compares the two groups sample by sample on the raw traces, and with the higher order moment based test, which applies the comparison to a transformed quantity defined at a single sample position, the bivariate variant is designed to detect leakage that emerges from the joint behavior of two time samples. This is relevant when the informative contribution is distributed across time and is not sufficiently visible when each sample is considered in isolation.

For this reason, FreeSCA implements a bivariate version of Welch's test based on pairs of centered samples. The objective is to compare, for the two groups, not only the behavior at one time index, but the joint behavior of two indices j_1 and j_2 . More precisely, the implementation considers pairs of time indices such that $j_2 = j_1 + \delta$, where $\delta \geq 0$ denotes the temporal distance between the two positions. The analysis is repeated for all values $\delta \in \{0, \dots, \delta_{\max}\}$, where $\delta_{\max}$ is a user-defined upper bound on the distance to be explored. For a fixed value of δ , the test is therefore applied to all valid pairs (j_1, j_2) satisfying $j_2 = j_1 + \delta$ and such that both indices remain within the considered trace length. In this way, the method does not only evaluate whether one time sample exhibits leakage, but also whether two samples separated by at most $\delta_{\max}$ exhibit a statistically significant joint effect.

The tool takes as input two trace datasets representing the two groups to be compared.

As in the previous statistical tests, the two datasets are processed in batches. If their trace lengths differ, the longer dataset is trimmed so that all subsequent operations are defined on aligned time indices.

For each group Q_g , with $g \in \{0, 1\}$, the implementation first estimates the mean trace $\hat{\mu}_g$. In the implementation, these mean traces are estimated incrementally through Welford's scheme, so that the two input datasets can be processed in batches without loading them entirely into memory. More precisely, if $\hat{\mu}_g(j)$ denotes the mean of group Q_g at time index j , then for each trace $i \in Q_g$ and for each pair of indices (j_1, j_2) , the bivariate feature is constructed as

$$y_{i,j_1,j_2}^{(g)} = (m_{i,j_1} - \hat{\mu}_g(j_1))(m_{i,j_2} - \hat{\mu}_g(j_2)). \quad (2.41)$$

For a fixed value of δ , the implementation restricts this construction to the pairs $(j_1, j_1 + \delta)$ with valid index j_1 . If n_t denotes the trace length used by the test, then the admissible values of j_1 range from 0 to $n_t - \delta - 1$. The resulting features are collected, for each group, into a matrix

$$Y_\delta^{(g)} \in \mathbb{R}^{n_g \times (n_t - \delta)}, \quad (2.42)$$

where n_g is the number of traces in group Q_g . In these matrices, each row corresponds to one trace and each column corresponds to one pair of time indices $(j_1, j_1 + \delta)$.

For $\delta = 0$, the construction reduces to the squared centered trace,

$$y_{i,j_1,j_1}^{(g)} = (m_{i,j_1} - \hat{\mu}_g(j_1))^2, \quad (2.43)$$

whereas for $\delta > 0$ it represents the interaction between two distinct time indices.

A practical issue arises when the traces are long. If each trace contains a large number of samples, the exploration of all displacements up to $\delta_{\max}$ may become computationally expensive. To reduce this cost, the implementation applies an averaging-based aggregation step before constructing the bivariate features. This operation can be interpreted as a simple low-pass filtering step followed by resampling, since groups of consecutive samples are replaced by their average. Let r denote the aggregation size. The trace is partitioned into consecutive windows of r samples, and each window is mapped to a single value equal to the average of the samples it contains. In this way, the original trace is converted into a shorter sequence of aggregated samples, whose length is $\lfloor n_t/r \rfloor$, where n_t denotes the original trace length.

When this aggregation step is applied, the same construction is performed on the aggregated representation. The indices j_1 and j_2 then refer to aggregation indices rather than to raw time samples, and δ represents the distance between two aggregation indices. The

resulting feature matrices therefore describe the interaction between pairs of temporal regions, rather than between isolated acquisition samples. This transformation reduces the number of positions involved in the pairwise comparison and makes the exploration of larger displacements more manageable. It is particularly convenient when the acquisition sampling rate is high with respect to the target frequency, so that a single clock cycle or a relevant computation interval is represented by many consecutive samples.

Once the two feature matrices $Y_\delta^{(0)}$ and $Y_\delta^{(1)}$ have been obtained for a fixed value of δ , Welch's test is applied column-wise as in the univariate case, but on bivariate features. More precisely, for each valid position j_1 , the two samples compared by the test are

$$\{y_{i,j_1,j_1+\delta}^{(0)}\}_{i \in Q_0} \quad \text{and} \quad \{y_{i,j_1,j_1+\delta}^{(1)}\}_{i \in Q_1}. \quad (2.44)$$

The corresponding statistic is therefore

$$t_\delta(j_1) = \frac{\hat{\mu}_{1,\delta}(j_1) - \hat{\mu}_{0,\delta}(j_1)}{\sqrt{\frac{\hat{\sigma}_{1,\delta}^2(j_1)}{n_1} + \frac{\hat{\sigma}_{0,\delta}^2(j_1)}{n_0}}}, \quad (2.45)$$

where $\hat{\mu}_{g,\delta}(j_1)$ and $\hat{\sigma}_{g,\delta}^2(j_1)$ denote the mean and variance of the bivariate feature for group Q_g , displacement δ and position j_1 . For that fixed displacement, the column-wise application of Welch's test therefore produces a one dimensional vector $\mathbf{t}_\delta \in \mathbb{R}^{n_t-\delta}$, whose entries are the values $t_\delta(j_1)$ for all admissible j_1 .

Repeating the computation for all values $\delta \in \{0, \dots, \delta_{\max}\}$ produces a sequence of t -statistic vectors indexed by j_1 . Collecting these vectors over all displacements defines the two-dimensional statistic map associated with the fork test. This map is represented as a heatmap whose horizontal axis corresponds to j_1 and whose vertical axis corresponds to δ . Each cell is therefore associated with the pair of positions $(j_1, j_1 + \delta)$. This representation makes it possible to identify at which temporal separations statistically significant effects arise and how these effects are distributed along the trace.

The implementation also derives confidence interval bounds. In the notation introduced in Section 1.1, the estimated parameter $\hat{\theta}$ is in the present case the difference between the two means of the bivariate feature,

$$\hat{\theta}(j_1, \delta) = \hat{\mu}_{1,\delta}(j_1) - \hat{\mu}_{0,\delta}(j_1), \quad (2.46)$$

where $\hat{\mu}_{g,\delta}(j_1)$ denotes the mean of the feature associated with group Q_g , displacement δ ,

and position j_1 . The corresponding standard error is

$$\widehat{\text{SE}}(j_1, \delta) = \sqrt{\frac{\hat{\sigma}_{1,\delta}^2(j_1)}{n_1} + \frac{\hat{\sigma}_{0,\delta}^2(j_1)}{n_0}}, \quad (2.47)$$

where $\hat{\sigma}_{g,\delta}^2(j_1)$ is the variance of the same feature in group Q_g .

To construct the confidence interval, the implementation first determines the total number U of tested pairs (j_1, δ) and applies a Bonferroni correction to the selected global significance level α , obtaining the per-test significance level

$$\alpha^* = \frac{\alpha}{U}. \quad (2.48)$$

For each pair (j_1, δ) , the Welch degrees of freedom $\nu(j_1, \delta)$ are then estimated and the corresponding critical value is obtained from the Student's t distribution as

$$q(j_1, \delta) = t_{1-\alpha^*/2, \nu(j_1, \delta)}, \quad (2.49)$$

where $t_{1-\alpha^*/2, \nu(j_1, \delta)}$ denotes the $(1 - \alpha^*/2)$ -quantile of the Student's t distribution with $\nu(j_1, \delta)$ degrees of freedom.

The confidence interval is therefore constructed as

$$\hat{\theta}(j_1, \delta) \pm q(j_1, \delta) \widehat{\text{SE}}(j_1, \delta). \quad (2.50)$$

In addition, the resulting confidence interval bounds are arranged into a square matrix indexed directly by the two positions j_1 and j_2 . In this matrix, the upper triangular part contains the lower bounds, whereas the lower triangular part contains the upper bounds. The sequence of operations carried out by the implementation, from the estimation of the mean traces on the aggregated representation to the construction of the statistic map and of the confidence interval matrix, is summarized in Algorithm 2.2. The tool outputs a CSV file containing, for each pair (δ, j_1) , the corresponding value of the Welch statistic $t_\delta(j_1)$, together with the heatmap of the statistic map and the heatmap associated with the confidence interval matrix.

2.6. Implemented Side Channel Attacks

This section describes the side channel attack procedures currently implemented in FreeSCA. In contrast with the statistical tests discussed in 2.5, whose purpose is to determine

Input: two trace sets Q_0 and Q_1 , maximum displacement $\delta_{\max}$, aggregation size r , global significance level α

Output: statistic map T_{map} and confidence interval map CI_{map}

```

1 Trim the two inputs to the same trace length
2  $n_t \leftarrow \lfloor \text{trimmed trace length} / r \rfloor$ 

    // First pass: estimate the mean trace of each group after
    aggregation step
3 foreach  $g \in \{0, 1\}$  do
4    $n_g \leftarrow 0$  and  $\hat{\mu}_g \leftarrow 0$ 
5 while traces remain to be processed do
6   Read one batch  $B_g$  from each trace set  $Q_g$ , for  $g \in \{0, 1\}$ 
7   Replace each trace in  $B_g$  with its aggregated samples using aggregation size  $r$ 
8   Update  $n_g$  and  $\hat{\mu}_g$  through Welford's scheme

    // Initialization of the Welch tests for all displacements
9 for  $\delta \leftarrow 0$  to  $\delta_{\max}$  do
10  foreach  $g \in \{0, 1\}$  do
11     $n_{g,\delta} \leftarrow 0$ ,  $\hat{\mu}_{g,\delta} \leftarrow 0$  and  $M_{2,g,\delta} \leftarrow 0$ 

    // Second pass: bivariate features and one-pass Welch updates
12 while traces remain to be processed do
13   Read one batch  $B_g$  from each trace set  $Q_g$ , for  $g \in \{0, 1\}$ 
14   Replace each trace in  $B_g$  with its aggregated samples using aggregation size  $r$ 
15   Center it using  $\hat{\mu}_g$ 
16   for  $\delta \leftarrow 0$  to  $\delta_{\max}$  do
17     Construct  $Y_\delta^{(g)}$  from the features  $y_{i,j_1,j_1+\delta}^{(g)}$ 
18     Update  $n_{g,\delta}$ ,  $\hat{\mu}_{g,\delta}$  and  $M_{2,g,\delta}$  from  $Y_\delta^{(g)}$  through Welford's scheme

    // Statistic vectors and confidence bounds for all displacements
19  $U \leftarrow \sum_{\delta=0}^{\delta_{\max}} (n_t - \delta)$  // Total number of tested pairs
20  $\alpha^* \leftarrow \alpha / U$ 
21 for  $\delta \leftarrow 0$  to  $\delta_{\max}$  do
22    $\hat{\sigma}_{g,\delta}^2 \leftarrow M_{2,g,\delta} / (n_{g,\delta} - 1)$ , for  $g \in \{0, 1\}$  // Feature variance
23    $\mathbf{t}_\delta \leftarrow (t_\delta(j_1))_{j_1=0}^{n_t-\delta-1}$  // Statistic vector for displacement  $\delta$ 
24   Compute the corresponding degrees of freedom  $\nu_\delta$ 
25   Compute  $\mathbf{l}_\delta, \mathbf{u}_\delta$  using  $\alpha^*$  and  $\nu_\delta$  // Lower and upper CI bounds
26   Store  $\mathbf{t}_\delta$  in  $T_{\text{map}}$  and  $\mathbf{l}_\delta, \mathbf{u}_\delta$  in the corresponding bound matrices
27 Construct  $CI_{\text{map}}$  from the lower bound and upper bound matrices
28 return  $T_{\text{map}}, CI_{\text{map}}$ 

```

Algorithm 2.2: Bivariate Welch t -test with maximum displacement $\delta_{\max}$

whether the traces exhibit statistically significant leakage, the procedures considered in this section are designed to exploit that leakage in order to rank or identify key hypotheses.

The current implementation includes two attacks based on different analysis settings:

- **Correlation Power Analysis (CPA):** implemented as a non profiled vertical attack, in which key hypotheses are evaluated directly on the measured traces through a leakage model and a statistical score.
- **Template Attack (TA):** implemented as a profiled attack, in which a preliminary training phase is used to characterize the leakage distributions associated with different classes before the attack is applied to traces acquired under an unknown key.

The following subsections focus on their implementation within the library, with particular attention to the required inputs, the quantities computed internally and the structure of the produced outputs.

2.6.1. Correlation Power Analysis

The theoretical principle of CPA was introduced in Section 1.4.1, where the attack was presented as a non profiled vertical method based on Pearson correlation.

In the current implementation, CPA follows the standard Divide and Conquer strategy and attacks one key byte at a time. More precisely, for a selected byte index b , all the hypotheses $k \in \{0, \dots, 255\}$ for the corresponding key byte are evaluated independently, and the resulting scores are used to rank the candidates. This choice is consistent with the usual application of CPA, where the secret key is decomposed into smaller subkeys and each subkey is attacked separately. For this reason, the first step of the implementation is the selection of the target byte. In FreeSCA, the attacked byte is handled as a configurable analysis parameter. The current implementation reads this choice from the global configuration, so that the same attack procedure can be executed on different byte positions without modifying the attack logic itself.

The attack also requires the metadata needed to derive the predicted intermediate values for all traces and all key hypotheses. In the current implementation, this means that plaintexts must be available in the input dataset. More precisely, for the selected byte b , the implementation constructs the matrix of intermediate values

$$v_{i,k}^{(b)} = P_i[b] \oplus k, \quad (2.51)$$

where $P_i[b]$ is the b -th plaintext byte of trace i , and k is one of the 256 key-byte hypotheses. The matrix $V^{(b)}$ therefore contains, for each trace and for each hypothesis, the

corresponding xor intermediate.

Once the xor intermediate matrix has been obtained, the implementation derives the AES quantities required by the selected attack configuration. This step corresponds to recomputing, under each key hypothesis, the portion of the AES algorithm that is assumed to influence the measured leakage. Depending on the selected configuration, this may involve only the xor intermediate, the corresponding SBox output, or both quantities when a transition is considered. The power consumption model is then applied to the resulting quantities in order to obtain the prediction matrix $H^{(b)} \in \mathbb{R}^{n \times 256}$, where n is the number of traces. In this matrix, the k -th column provides the prediction vector associated with the key hypothesis k . In the current implementation, the supported configurations are:

- the xor intermediate $v_{i,k}^{(b)}$, mapped through the Hamming weight model;
- the SBox output $\text{SBox}(v_{i,k}^{(b)})$, mapped through the Hamming weight model;
- the pair formed by $v_{i,k}^{(b)}$ and $\text{SBox}(v_{i,k}^{(b)})$, mapped through the Hamming distance model, computed as the Hamming weight of their bitwise difference.

As for the attacked byte, the attack configuration is also handled as a configurable parameter and is selected through the global configuration.

At this stage, the implementation has access to the measured trace matrix $M \in \mathbb{R}^{n \times t}$, where t denotes the number of time samples per trace, and to the prediction matrix $H^{(b)}$ associated with the selected byte and leakage model. The objective of CPA is to quantify, for each key hypothesis, the statistical dependence between the measured leakage and the corresponding prediction vector. This comparison is carried out for all time samples of the traces and for all key hypotheses, so as to determine which hypothesis exhibits the strongest agreement with the observed data under the selected model.

For a fixed time sample j , let

$$\mathbf{m} = (m_1, \dots, m_n) \in \mathbb{R}^n \quad (2.52)$$

denote the vector collecting the measured leakage values at position j across the n traces. In addition, let

$$\mathbf{h}_k = (h_{1,k}, \dots, h_{n,k}) \in \mathbb{R}^n \quad (2.53)$$

denote the prediction vector associated with the key hypothesis k , that is the k -th column of the prediction matrix $H^{(b)}$. The CPA score at the considered time sample is obtained by comparing $\mathbf{m}$ with $\mathbf{h}_k$ for every key hypothesis k .

As discussed in Section 1.3.4, the adopted score is the Pearson correlation coefficient. In the current implementation, this quantity is not evaluated by explicitly centering the vectors at each step. Instead, for each considered time sample j and for each key hypothesis k , the computation is expressed in terms of quantities that can be accumulated incrementally while the traces are processed in batches. More precisely, the implementation maintains

$$\sum_{i=1}^n m_i, \quad \sum_{i=1}^n m_i^2, \quad \sum_{i=1}^n h_{i,k}, \quad \sum_{i=1}^n h_{i,k}^2, \quad \sum_{i=1}^n h_{i,k} m_i. \quad (2.54)$$

In order to express the Pearson coefficient in terms of quantities that can be accumulated incrementally, the centered formula is expanded and rewritten by replacing the sample means with their explicit definitions. The corresponding algebraic steps are reported in Appendix B and lead to the equivalent expression

$$r_k = \frac{n \sum_{i=1}^n h_{i,k} m_i - (\sum_{i=1}^n h_{i,k}) (\sum_{i=1}^n m_i)}{\sqrt{\left(n \sum_{i=1}^n m_i^2 - (\sum_{i=1}^n m_i)^2\right) \left(n \sum_{i=1}^n h_{i,k}^2 - (\sum_{i=1}^n h_{i,k})^2\right)}}. \quad (2.55)$$

This form is algebraically equivalent to the standard Pearson Formula 1.46, but it is more convenient from an implementation point of view, since it allows the correlation matrix to be computed without storing all traces and prediction vectors once the required sums have been updated.

Repeating this computation for all time samples produces a correlation matrix whose rows correspond to key hypotheses and whose columns correspond to time samples. In this form, the attack provides a score profile over time for every key hypothesis, which can be used both to localize the most informative leakage samples and to rank the candidate key values.

The same accumulated quantities in 2.54 can also be exploited before the full dataset has been processed, in order to monitor the evolution of the attack as additional traces are incorporated. For this reason, the implementation optionally supports an incremental mode. In this setting, after each processed batch, the current correlation matrix is evaluated from the partial sums and, for each key hypothesis k , the implementation records the quantities

$$\max_j r_k(j) \quad \text{and} \quad \max_j |r_k(j)|. \quad (2.56)$$

The first quantity preserves the sign of the largest observed correlation, whereas the second retains only its magnitude. This incremental representation is useful because it

makes it possible to assess how quickly the hypotheses separate as more measurements are incorporated. In particular, it allows one to observe whether a given hypothesis begins to exhibit a stable correlation peak after a limited number of traces, or whether a larger dataset is required before a clear distinction emerges.

In addition to the correlation coefficients, the implementation also provides confidence intervals for a selected CPA score. In the notation introduced in Section 1.1, the parameter estimate $\hat{\theta}$ is, in the present case, the Pearson correlation coefficient $r_k(j)$ associated with hypothesis k at time sample j . The direct construction of a confidence interval for the correlation coefficient is not generally carried out on $r_k(j)$ itself, because this estimator is inherently bivariate and its distribution depends on the joint distribution of the two variables whose dependence is being estimated. Deriving the interval directly would therefore require transforming this distribution into a known bivariate form. For this reason, the implementation relies on Fisher's transformation,

$$z_k(j) = \operatorname{atanh}(r_k(j)), \quad (2.57)$$

which maps the correlation estimator to a random variable with an approximately Gaussian univariate distribution. Under this standard approximation, the transformed quantity has standard error

$$\widehat{\text{SE}}(z_k(j)) = \frac{1}{\sqrt{n-3}}, \quad (2.58)$$

where n denotes the number of processed traces. The confidence interval is therefore first constructed in the transformed domain as

$$z_k(j) \pm q \frac{1}{\sqrt{n-3}}, \quad (2.59)$$

where q is the Gaussian critical value associated with the selected significance level α . The two bounds are then mapped back through the inverse transformation, yielding the lower and upper bounds for the original correlation coefficient $r_k(j)$. These bounds complement the point estimate by associating an interval of plausible values with the observed correlation coefficient.

The implemented CPA procedure returns its results both in tabular form and through graphical representations. The correlation matrix provides, for each key hypothesis and for each time sample, the corresponding value of the Pearson coefficient, thus supporting the identification of the most informative leakage samples and of the hypotheses that are most consistent with the measured traces. When the incremental mode is enabled, the evolution of the correlation maxima is also reported, both in signed form and in absolute

value, so as to highlight how the separation between hypotheses develops as more traces are incorporated. Taken together, these outputs support the interpretation of the attack over the time axis and the evaluation of the amount of data required for a stable hypothesis ranking.

2.6.2. Template Attack

The theoretical principle of the Template Attack was introduced in Section 1.4.2, where it was presented as a profiled side channel attack. In contrast with CPA, which evaluates hypotheses directly on the measured traces, the Template Attack relies on a preliminary profiling phase in which a statistical model of the leakage is estimated from labeled data and is then used in the attack phase. In the current implementation, the procedure is formulated as a classification problem over a set of labels. The following paragraphs describe how the corresponding templates are constructed, how they are evaluated on previously unseen traces and which quantities are produced by the attack.

The implementation is first described through its general design choices and is then discussed by distinguishing the two phases on which Template Attacks are based:

1. In the profiling phase, one statistical template is estimated for each class from labeled traces acquired under known conditions.
2. In the attack phase, the resulting templates are used to classify previously unseen traces.

In the current implementation, the class structure is specified through a JSON label map, which associates each class label with the corresponding traceset. Each entry therefore defines one profiling class and the traces used to estimate its template. This organization is independent of the specific interpretation of the labels and can be applied to any finite set of classes, provided that one traceset is available for each of them. Depending on the chosen labeling criterion, the classes may correspond to the two values of a selected bit, to the Hamming weight classes or to the 256 possible values of a byte.

Within the current implementation, the same label structure is also adopted in the attack phase, where the previously estimated templates are evaluated on traces associated with the same set of classes. In this context, the label map is used mainly for evaluation purposes, since it provides the reference classes needed to measure the classification performance of the attack, for example in terms of accuracy. The current implementation also supports a simplified workflow in which the same tracesets are reused for both phases. In this case, a fixed number of initial traces per class is used to estimate the templates, while

the remaining traces are reserved for the attack phase and classified using the learned model.

Before the templates are estimated or evaluated, the traces are aligned and trimmed according to the selected preprocessing settings, so that all feature vectors associated with the different classes have the same dimension and can be used consistently in both phases of the procedure.

Profiling phase. For each label ℓ , the profiling phase estimates one statistical template from the traces belonging to that class. In the current implementation, each template is modeled as a multivariate Gaussian distribution, as recalled in Section 1.1. The corresponding parameters are therefore a mean vector $\boldsymbol{\mu}_\ell$ and a covariance matrix Σ_ℓ , which describe, respectively, the average behavior of the traces in class ℓ and their joint variability across the considered sample positions.

More precisely, let $\mathbf{m}_i^{(\ell)} \in \mathbb{R}^t$ denote the i -th profiling trace associated with label ℓ , where t is the trace dimension and let n_ℓ denote the number of profiling traces available for that class. The template parameters are estimated as

$$\boldsymbol{\mu}_\ell = \frac{1}{n_\ell} \sum_{i=1}^{n_\ell} \mathbf{m}_i^{(\ell)}, \quad (2.60)$$

and

$$\Sigma_\ell = \frac{1}{n_\ell - 1} \sum_{i=1}^{n_\ell} \left(\mathbf{m}_i^{(\ell)} - \boldsymbol{\mu}_\ell \right) \left(\mathbf{m}_i^{(\ell)} - \boldsymbol{\mu}_\ell \right)^\top. \quad (2.61)$$

From an implementation point of view, these quantities are not estimated by loading all traces of the class at once. Instead, the profiling traces are processed in batches and the template parameters are computed incrementally. The estimation is carried out in two passes over the traces of each class. In the first pass, the mean vector $\hat{\boldsymbol{\mu}}_{\ell,n}$ is estimated through the Welford's algorithm 2.3. Once the class mean has been estimated, in the second pass the covariance matrix is accumulated using the previously computed mean vector $\hat{\boldsymbol{\mu}}_{\ell,n}$. More precisely, if $Q_{\ell,n} \in \mathbb{R}^{t \times t}$ denotes the auxiliary matrix used to accumulate the covariance terms for class ℓ , the update has the form

$$Q_{\ell,n} = Q_{\ell,n-1} + \left(\mathbf{m}_n^{(\ell)} - \hat{\boldsymbol{\mu}}_{\ell,n-1} \right) \left(\mathbf{m}_n^{(\ell)} - \hat{\boldsymbol{\mu}}_{\ell,n} \right)^\top, \quad (2.62)$$

and the covariance estimate is then obtained as

$$\hat{\Sigma}_{\ell,n} = \frac{Q_{\ell,n}}{n - 1}. \quad (2.63)$$

This organization avoids the need to store all traces of the class simultaneously in memory and remains applicable also when both the trace dimension and the number of profiling traces are large.

Attack phase. Once the profiling parameters have been estimated, the attack phase uses them to score previously unseen traces under the available class models. Accordingly, each class template is interpreted in probabilistic form. In the current implementation, the template associated with label ℓ is interpreted as a multivariate Gaussian model $\mathcal{N}(\boldsymbol{\mu}_\ell, \Sigma_\ell)$, where $\boldsymbol{\mu}_\ell$ and Σ_ℓ are the mean vector and covariance matrix estimated during the profiling phase. As reported in Section 1.1, this model assigns a probability density value to any trace vector $\mathbf{m} \in \mathbb{R}^t$, thus quantifying how compatible the observed trace is with the statistical behavior of class ℓ . More precisely, for a trace $\mathbf{m}$, the Gaussian density associated with class ℓ is

$$p(\mathbf{m} \mid \ell) = \frac{1}{\sqrt{(2\pi)^t \det(\Sigma_\ell)}} \exp \left(-\frac{1}{2} (\mathbf{m} - \boldsymbol{\mu}_\ell)^\top \Sigma_\ell^{-1} (\mathbf{m} - \boldsymbol{\mu}_\ell) \right), \quad (2.64)$$

where the quadratic form $(\mathbf{m} - \boldsymbol{\mu}_\ell)^\top \Sigma_\ell^{-1} (\mathbf{m} - \boldsymbol{\mu}_\ell)$ measures the discrepancy between the trace $\mathbf{m}$ and the mean trace of class ℓ , while taking into account the covariance structure of that class. It therefore assigns a larger penalty to traces that are less compatible with the template. This quantity is the squared Mahalanobis distance between $\mathbf{m}$ and the Gaussian model associated with label ℓ .

From an implementation point of view, the density is not evaluated directly. Instead, the attack uses its logarithm,

$$\log p(\mathbf{m} \mid \ell) = -\frac{1}{2} \left(t \log(2\pi) + \log \det(\Sigma_\ell) + (\mathbf{m} - \boldsymbol{\mu}_\ell)^\top \Sigma_\ell^{-1} (\mathbf{m} - \boldsymbol{\mu}_\ell) \right). \quad (2.65)$$

The logarithmic form is preferred in the implementation because it provides a more convenient numerical representation of the Gaussian score while preserving the ordering of the class likelihoods. The resulting decision rule is therefore unchanged, i.e., maximizing $p(\mathbf{m} \mid \ell)$ over the class labels is equivalent to maximizing its logarithm.

To evaluate the Gaussian score in practice, the covariance matrix is not used directly in its stored form. For each class ℓ , the covariance estimate is first symmetrized by replacing it with

$$\Sigma_\ell \leftarrow \frac{1}{2} (\Sigma_\ell + \Sigma_\ell^\top), \quad (2.66)$$

and then diagonalized as

$$\Sigma_\ell = E_\ell \Lambda_\ell E_\ell^\top, \quad (2.67)$$

where E_ℓ is the matrix of eigenvectors of Σ_ℓ and Λ_ℓ is the diagonal matrix containing the corresponding eigenvalues. Since Σ_ℓ is real and symmetric, this representation is equivalent to a singular value decomposition. This representation is used to evaluate the covariance dependent terms appearing in the log-probability Equation 2.65. In particular, if $\lambda_{\ell,u}$ denotes the u -th eigenvalue of Σ_ℓ , then

$$\log \det(\Sigma_\ell) = \sum_{u=1}^t \log \lambda_{\ell,u}, \quad (2.68)$$

while the inverse covariance matrix can be written as

$$\Sigma_\ell^{-1} = E_\ell \Lambda_\ell^{-1} E_\ell^\top. \quad (2.69)$$

Once the score function has been defined for every class, each previously unseen trace $\mathbf{m}$ is evaluated under all the class labels provided in the input map. Let $\mathcal{L}$ denote this set of labels. The assigned label, denoted by $\hat{\ell}(\mathbf{m})$, is then the one maximizing the corresponding log-probability:

$$\hat{\ell}(\mathbf{m}) = \arg \max_{\ell \in \mathcal{L}} \log p(\mathbf{m} \mid \ell). \quad (2.70)$$

The attack phase therefore consists in selecting, for each trace, the class whose template provides the largest log-likelihood value. In the general evaluation setting, the true label of each attack trace is given by the label of the traceset from which it is taken. The attack result can therefore be assessed by comparing the predicted label $\hat{\ell}(\mathbf{m})$ with the corresponding class label. Let n denote the total number of classified traces, and let n_{corr} and n_{err} denote, respectively, the numbers of correctly and incorrectly classified traces. The classification accuracy is then computed as

$$a = \frac{n_{\text{corr}}}{n}. \quad (2.71)$$

This evaluation procedure is consistent with the general formulation of the implemented attack, since it does not depend on a specific semantic interpretation of the labels.

Generated outputs. As a final consideration, the implemented procedure produces outputs associated with both phases of the attack. During profiling, one template is generated for each class label and stored through two files containing the corresponding mean vector $\boldsymbol{\mu}_\ell$ and covariance matrix Σ_ℓ . Taken together, these files provide the statistical model learned from the profiling traces and reused in the attack phase. During the attack phase, the main result is the classification accuracy obtained over the attack traces. This quantity summarizes how effectively the estimated templates distinguish the consid-

ered classes under the adopted Gaussian model. In addition, the implementation provides graphical representations of selected class means, of their difference, and of the difference between the corresponding covariance matrices. These outputs support the interpretation of the learned templates by highlighting the main statistical differences between classes.

The implemented attack therefore realizes the standard profiled Template Attack workflow, in which class-dependent Gaussian models are first estimated and are then used to classify unseen traces according to their log-likelihood.

3 | Experimental Evaluation

This chapter presents the experimental results obtained with the implemented library and is organized into three main sections. The first section describes the experimental setup, including the considered trace datasets, the target devices and the main acquisition and preprocessing characteristics relevant to the analysis. The second section is devoted to the statistical tests implemented in the library. In particular, it discusses the results obtained with the different variants of Welch’s t -test considered in this work, namely the first order test, the higher order test and the bivariate test. The third section focuses on the two attack techniques implemented in the library, Correlation Power Analysis and Template Attack, and discusses the corresponding experimental results.

3.1. Experimental Setup

This section introduces the two main trace datasets considered throughout the experimental evaluation, one used as reference for the unmasked setting and one used as reference for the masked setting.

- The unmasked experiments are conducted on traces acquired with the ChipWhisperer platform. The target device is CW308_STM32F4, with an STM32F405RGT6 microcontroller, and the cryptographic implementation is software-based and unprotected. The target frequency is 7.36 MHz, whereas the sampling frequency is 29.45 MHz. Each trace corresponds to a complete execution of AES-128 and contains 32000 samples.
- The masked experiments are conducted on the ASCAD dataset. The target device is an ATmega8515 and the cryptographic implementation is software-based and protected by masking. The target frequency is 4 MHz, whereas the acquisition sampling frequency is 2 GHz. In the original acquisition campaign, each trace contains 100000 samples. In the preprocessed version considered in this work, each trace is reduced to a window of 700 samples, corresponding to the time interval selected around the operation of interest.

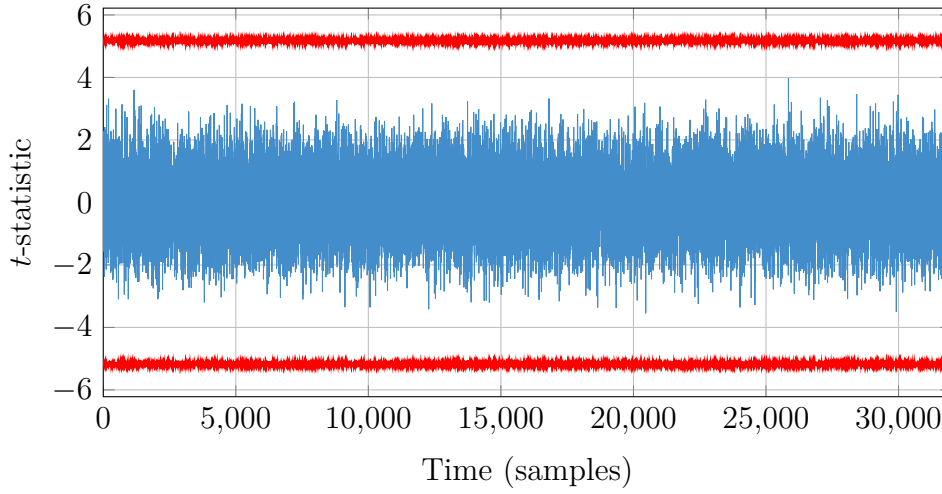


Figure 3.1: Control experiment for the first order Welch t -test on unmasked ChipWhisperer traces. A set of 1000 fixed-key fixed-plaintext traces is split into two disjoint groups. The red curves denote the Bonferroni thresholds for $\alpha = 0.01$.

Both datasets were converted into the FSCA format through the dedicated converters introduced in Section 2.2.2. In the converted ChipWhisperer traces, the metadata include plaintexts, keys and ciphertexts, whereas the converted ASCAD traces also retain the corresponding masks.

3.2. Statistical Test Results

This section presents the experimental results obtained with the statistical tests implemented in the library. For each method, the considered dataset, the adopted configuration and the main analysis parameters are specified explicitly. The corresponding plots are then used to examine the leakage behavior observed in the masked and unmasked settings.

3.2.1. First Order t -test

The first order Welch t -test is first applied to the unmasked ChipWhisperer traces in a non specific setting. As a preliminary validation step, a set of 1000 traces acquired with fixed key and fixed plaintext is split into two disjoint groups, and the point-wise test is applied between them. Since both groups are obtained under the same acquisition conditions, no systematic statistical difference is expected. This control experiment is used to verify that the analysis pipeline does not introduce artificial peaks due to the splitting procedure or to the implementation of the test. The global significance level is set to $\alpha = 0.01$, and the corresponding Bonferroni correction is applied over all tested sample positions.

Figure 3.1 shows that the statistic remains within the Bonferroni bounds over the whole trace, without threshold crossings. This confirms the expected behavior of the control experiment and supports the correctness of the implemented analysis pipeline in this setting.

Two additional fixed-set comparisons are then considered by inspecting the corresponding *t*-test profiles on the first 5000 samples. Each group contains 1000 traces acquired under fixed-key fixed-plaintext conditions. In the first comparison, the two groups differ only in the plaintext value, whereas in the second comparison they differ only in the key value.

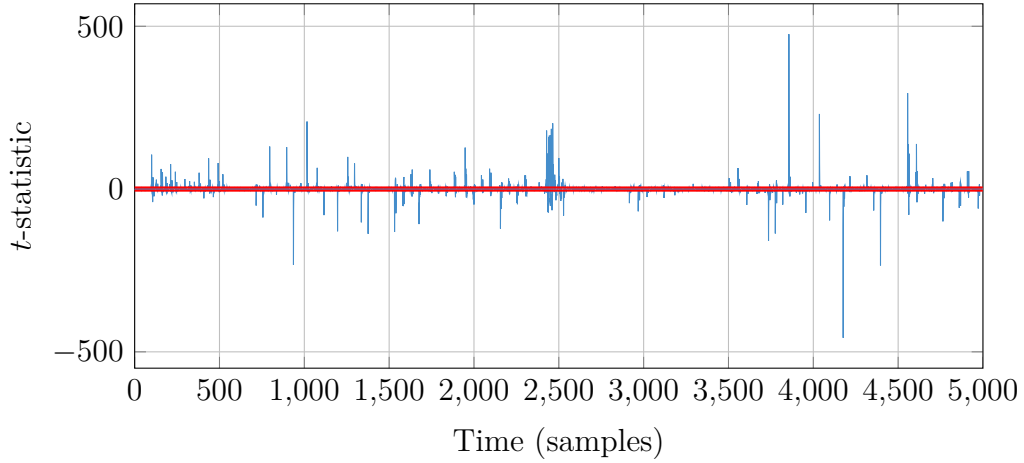
Figure 3.2 shows that the two fixed-set comparisons lead to statistically significant peaks in different portions of the considered time window. Since the two experiments isolate respectively the variation of the plaintext and of the key, the corresponding peaks can be interpreted as plaintext-dependent and key-dependent effects. Their different temporal locations suggest that these effects are associated with distinct stages of the AES execution, consistently with the loading and early processing of plaintext and key material.

The analysis is then extended to a fixed-versus-random setting. The two compared groups correspond respectively to traces acquired with fixed key and fixed plaintext, and to traces acquired with fixed key and random plaintext. In this case, the objective is to assess whether the dependence on the processed data is directly visible at first order on the raw traces. The analysis is carried out by comparing two groups of 1000 unmasked traces each. The global significance level is $\alpha = 0.01$ and the corresponding Bonferroni correction is applied over all tested sample positions.

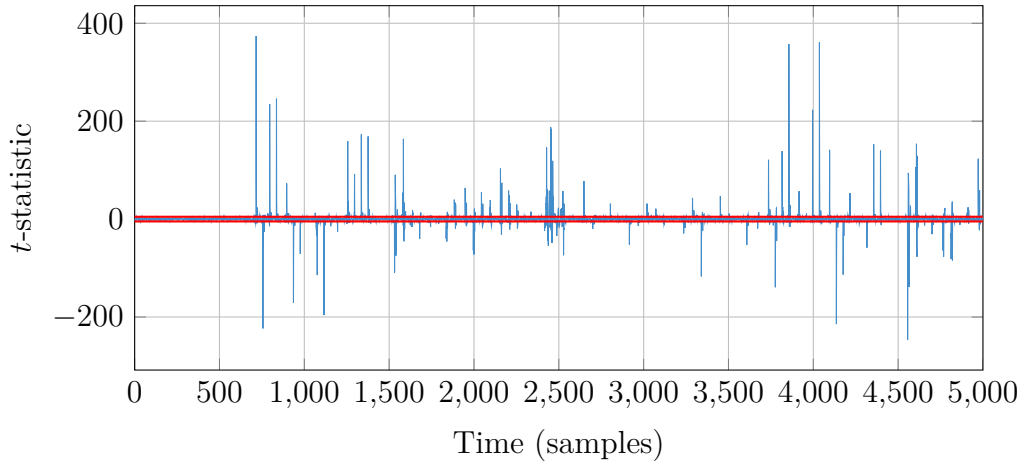
Figure 3.3 shows the resulting statistic profile together with the corresponding Bonferroni thresholds. In this experiment, the threshold is approximately equal to ± 5.13 over the whole trace. The statistic exceeds this bound over multiple portions of the acquisition, with peak values around 76.4 and -66.6 , thereby providing clear evidence of detectable first order leakage in the unmasked setting.

As an additional experiment, the same test is also applied by comparing traces acquired with fixed key and fixed plaintext against traces acquired with random key and fixed plaintext. The analysis is performed on two groups of 1000 unmasked traces each, using the same global significance level and the same Bonferroni correction. This second setting is used to examine how the first order statistic behaves when the variability is introduced through the key rather than through the plaintext, that is, when the changing quantity is the secret parameter influencing the internal AES computation.

Figure 3.4 shows that the statistic again exceeds the Bonferroni threshold over multiple



(a) Different plaintexts



(b) Different keys

Figure 3.2: First order Welch t -test profiles on unmasked ChipWhisperer traces. Panel (a) compares two fixed acquisition sets differing only in the plaintext value. Panel (b) compares two fixed acquisition sets differing only in the key value.

portions of the trace when the key is varied while the plaintext is kept fixed. The result still provides clear evidence of first order detectable leakage in the unmasked setting.

Since the traces correspond to a full AES execution, the profile also exhibits a sequence of oscillatory regions distributed along the trace. This behavior is consistent with the repeated structure of the AES computation and plausibly reflects the succession of its main processing steps.

The first order Welch t -test is then applied to the masked ASCAD traces in a specific

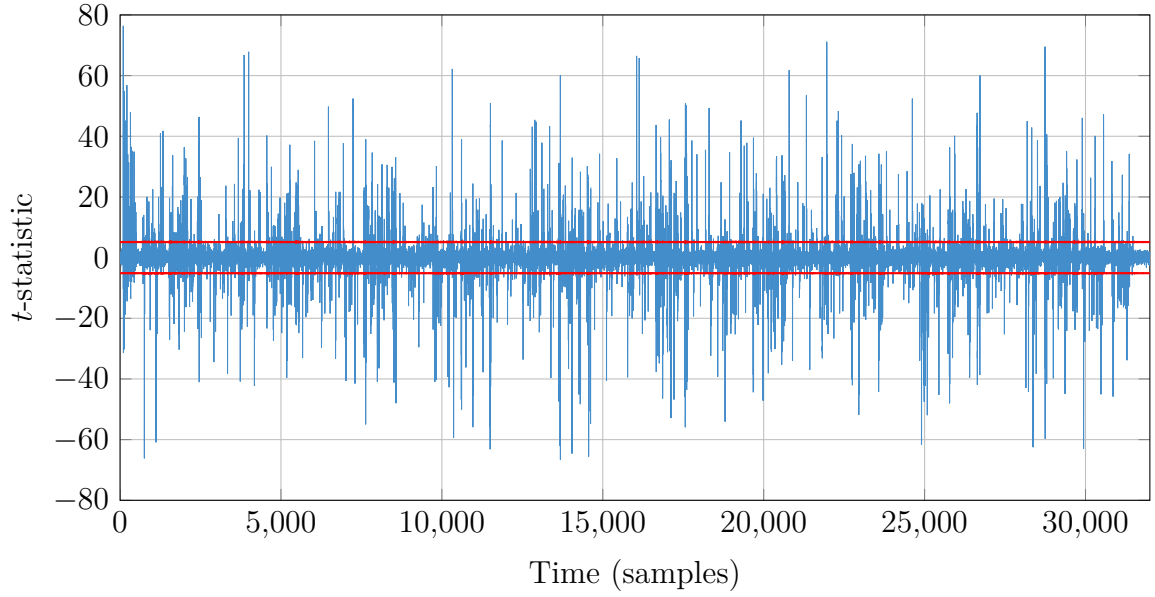


Figure 3.3: First order Welch t -test profile on unmasked ChipWhisperer traces for the fixed-key fixed-plaintext versus fixed-key random-plaintext comparison. The red curves denote the Bonferroni thresholds for $\alpha = 0.01$.

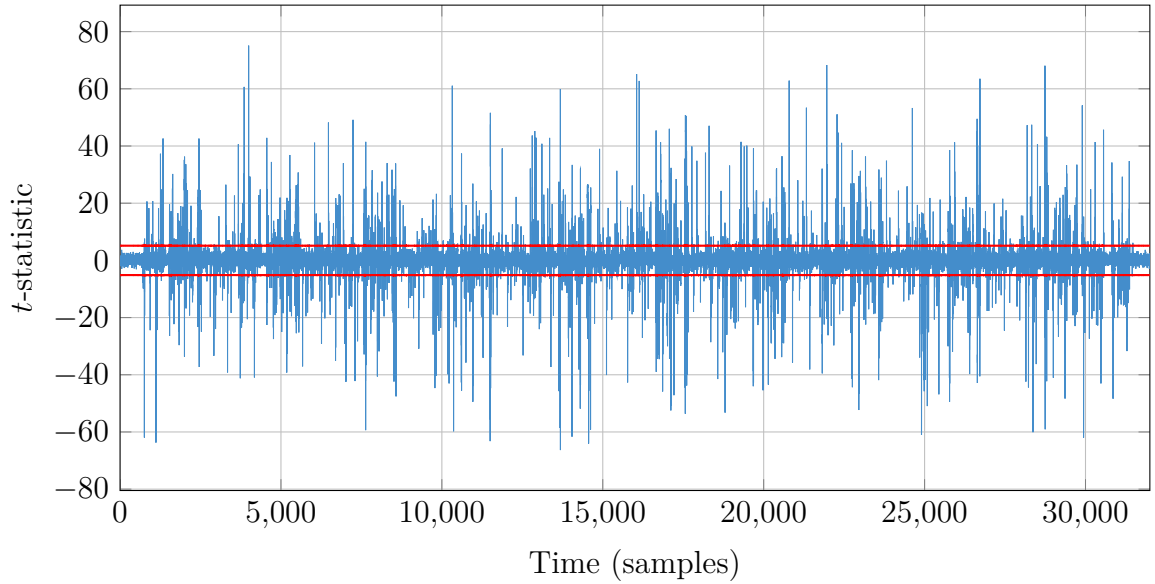


Figure 3.4: First order Welch t -test profile on unmasked ChipWhisperer traces for the fixed-key fixed-plaintext versus random-key fixed-plaintext comparison. The red curves denote the Bonferroni thresholds for $\alpha = 0.01$.

setting. In this case, the two groups are obtained through the partitioning criterion introduced in 2.4, by selecting byte index $b = 2$ and considering bit 0 of the corresponding

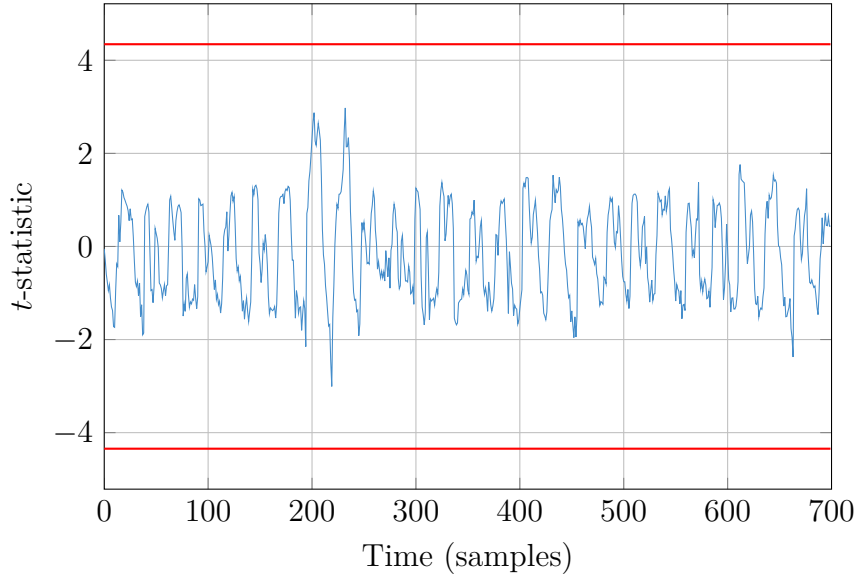


Figure 3.5: First order Welch t -test profile on masked ASCAD traces for the partition induced by bit 0 of the SBox output associated with byte 2. The red curves denote the Bonferroni thresholds for $\alpha = 0.01$.

SBox output. More precisely, the intermediate value associated with trace i is

$$v_i^{(2)} = \text{SBox}(P_i[2] \oplus K_i[2]), \quad (3.1)$$

and the binary label is defined as

$$s_i = (v_i^{(2)})_0 \in \{0, 1\}. \quad (3.2)$$

The two groups used by the test are therefore

$$Q_0 = \{i : s_i = 0\}, \quad Q_1 = \{i : s_i = 1\}. \quad (3.3)$$

The analysis is carried out on two groups of 2000 masked traces each. The global significance level is $\alpha = 0.01$ and the corresponding Bonferroni correction is applied over all tested sample positions.

Figure 3.5 shows the resulting statistic profile together with the corresponding Bonferroni thresholds. In this experiment, the threshold is approximately equal to ± 4.34 . As can be seen in Figure 3.5, the statistic remains within the corresponding bounds over the whole trace, and no sample position exceeds the significance threshold. Under this partitioning criterion, the masked traces therefore do not exhibit detectable first order leakage. This

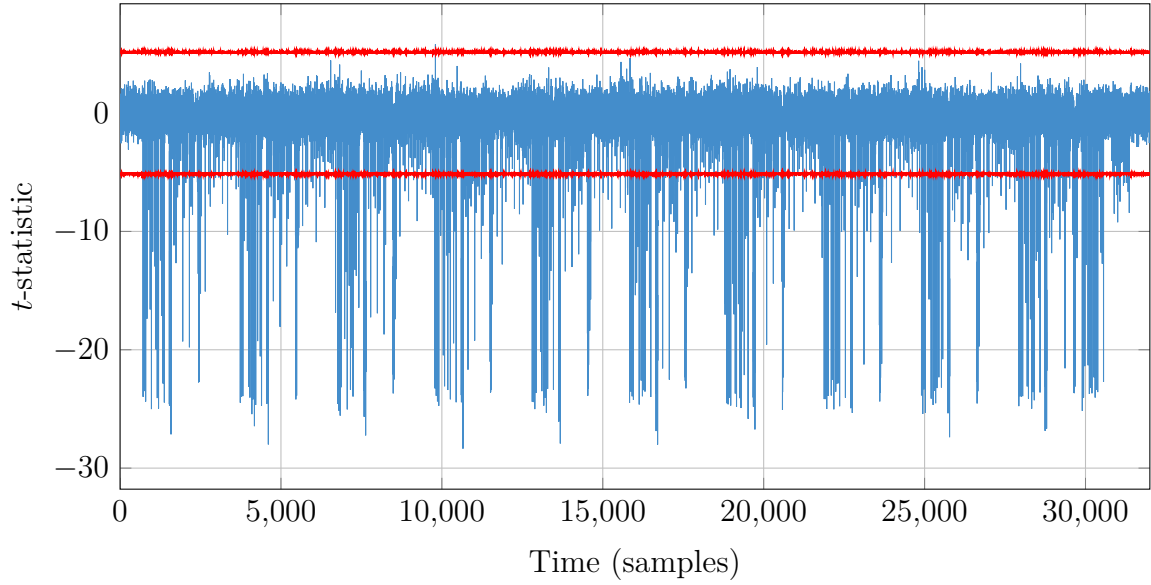


Figure 3.6: Second order Welch t -test profile on unmasked ChipWhisperer traces for the fixed-key fixed-plaintext versus random-key fixed-plaintext comparison. The red curves denote the Bonferroni thresholds for $\alpha = 0.01$.

result is consistent with the protected nature of the implementation and motivates the use of higher order statistical tests.

3.2.2. Higher Order t -test

The higher order Welch t -test is then considered in order to assess whether leakage that is not visible at first order can be revealed after the corresponding higher order transformation. In the unmasked ChipWhisperer setting, the first order test already provided clear evidence of leakage. Nevertheless, a second order profile is reported as a reference, by comparing two groups of 1000 unmasked traces each, acquired respectively with fixed key and fixed plaintext, and with random key and fixed plaintext. In this setting, the higher order test is not introduced to establish the presence of leakage, which is already evident at first order, but to examine how the statistic profile changes under the second order transformation.

Figure 3.6 shows the resulting second order statistic profile together with the corresponding Bonferroni thresholds. In this experiment, the threshold is approximately equal to ± 5.13 over the whole trace. The statistic exceeds this bound over several portions of the acquisition, confirming that leakage remains clearly detectable also after the second order transformation. Compared with the first order profile, the statistic is concentrated on one side of the axis, in agreement with the form of the adopted second order transformation.

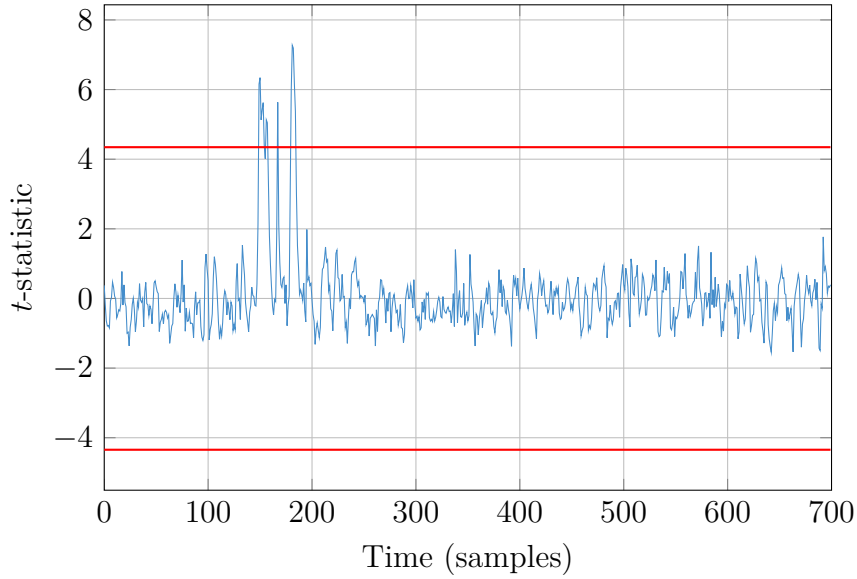


Figure 3.7: Second order Welch t -test profile on masked ASCAD traces for the partition induced by bit 0 of the SBox output associated with byte 2. The red curves denote the Bonferroni thresholds for $\alpha = 0.01$.

The significant regions remain distributed over multiple portions of the acquisition, and their arrangement makes the repeated temporal structure of the full AES execution more apparent.

The higher order analysis is then applied to the masked ASCAD traces. The specific partitioning criterion introduced in Equation 3.1 is retained, that is, the two groups are constructed from byte index $b = 2$ by selecting bit 0 of the corresponding SBox output. The global significance level is $\alpha = 0.01$ and the Bonferroni correction is applied over all tested sample positions. In this setting, the first order statistic did not show significant deviations, so that the higher order test becomes the natural tool to investigate whether the masking leaves detectable residual dependencies at higher order.

Figure 3.7 shows the resulting second order statistic profile together with the corresponding Bonferroni thresholds. In this experiment, the threshold is approximately equal to ± 4.34 over the whole trace. In contrast with Figure 3.5 of the first order, the statistic exceeds this bound over a limited number of regions, thereby providing evidence of detectable second order leakage under the considered partitioning criterion. This result indicates that, although the first order dependency is not visible, the protected implementation leaves a statistically significant effect after the second order transformation.

In the masked setting, the second order profile already provides statistically significant evidence of leakage. For this reason, the main discussion is restricted to the second order

result, which is sufficient to establish the presence of detectable higher order leakage under the considered partitioning criterion.

3.2.3. Bivariate t-test

The bivariate Welch t -test is applied to the masked ASCAD traces under the same specific partitioning criterion introduced in 3.3. The analysis is carried out on two groups of 2000 masked traces each. Before constructing the bivariate features, the traces are reduced through averaging-based aggregation with aggregation size $r = 20$. This yields an aggregated trace length of

$$n_t = \left\lfloor \frac{700}{20} \right\rfloor = 35, \quad (3.4)$$

where 700 is the original trace length. The maximum displacement is then fixed to $\delta_{\max} = 34$, so that all admissible displacements over the aggregated trace are included in the analysis. In the following, j denotes an aggregation index, so that each bivariate feature is associated with a pair $(j, j + \delta)$.

Figure 3.8 reports the absolute value of the bivariate Welch statistic over the full set of admissible pairs of aggregation indices. The horizontal axis is indexed by the first aggregation index j , while the vertical axis represents the displacement δ , so that each cell corresponds to the pair $(j, j + \delta)$. The highest values are concentrated in a limited number of regions rather than being uniformly distributed over the triangular support. In particular, the most prominent cluster is located around $j \approx 6 - 9$ and $\delta \approx 15 - 21$, with the strongest concentration around $\delta \approx 18 - 21$. This corresponds to pairs with second aggregation index $j_2 \approx 21 - 30$. The map therefore indicates that the detectable bivariate dependency is associated with specific temporal pairs rather than with a diffuse effect over the whole trace. The coordinates of the most prominent region are consistent with the significant region observed in the second order univariate profile of Figure 3.7.

Figure 3.9 provides the complementary representation of the same result in terms of Bonferroni confidence intervals. In this case, the matrix is indexed directly by the two aggregation indices (j_1, j_2) , so that each temporal pair is represented by two symmetric entries, i.e., the lower bound in the upper triangular part and the upper bound in the lower triangular part. This representation supports two complementary interpretations. First, it provides a direct significance assessment for each pair of aggregation indices, since a pair is significant whenever the corresponding confidence interval excludes zero. Second, because the map is indexed directly by (j_1, j_2) , it localizes where the significant pairs occur over the two dimensional temporal domain. In the present result, the most evident structures are concentrated for $j_1 \approx 6 - 9$ and $j_2 \approx 21 - 30$, with the strongest

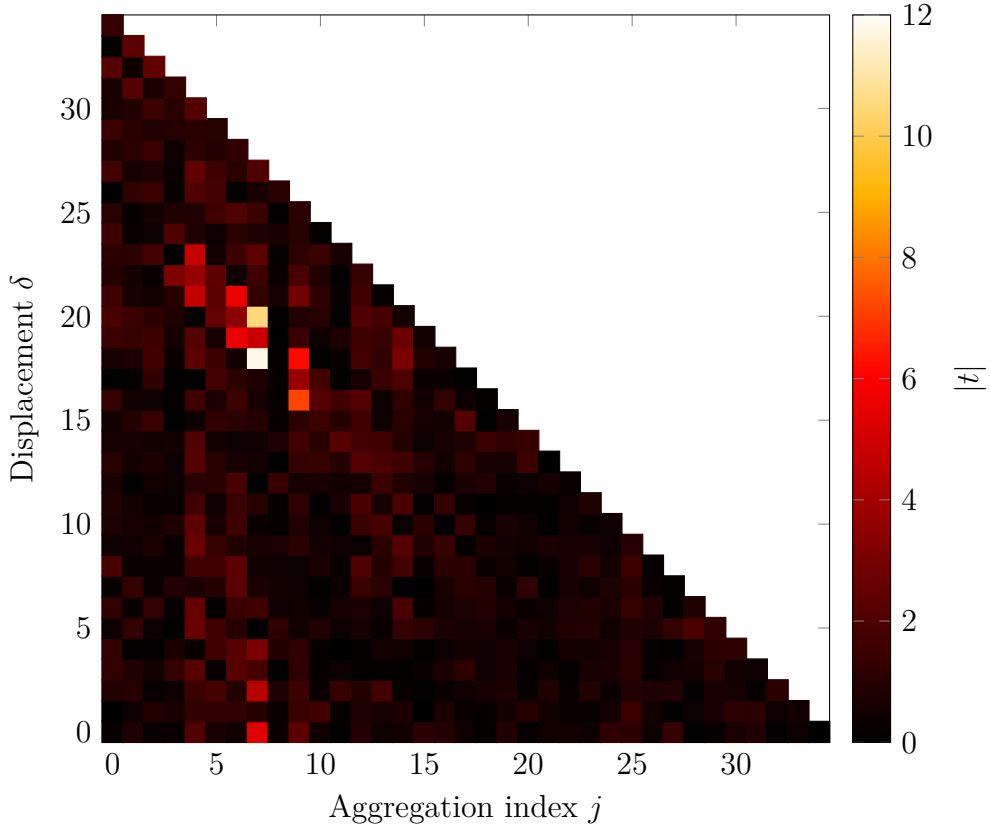


Figure 3.8: Bivariate Welch t -test statistic map on masked ASCAD traces with aggregation size $r = 20$ and maximum displacement $\delta_{\max} = 34$. The heatmap reports the absolute value of the statistic for all admissible $(j, j + \delta)$ pairs.

concentration around $j_2 \approx 25 - 28$. This is consistent with the cluster already highlighted in the statistic map and with the significant region observed in the second order univariate profile of Figure 3.7. The confidence interval representation therefore confirms that the detectable bivariate dependency is concentrated on specific temporal pairs rather than being uniformly distributed over the whole trace.

The results discussed in this section are summarized in Table 3.1. It should be noted that these statistical tests are used to assess the presence of detectable leakage in the measured traces and do not, by themselves, imply successful key recovery.

Overall, the summary confirms that detectable leakage appears already at first order in the unmasked setting, whereas the masked setting requires higher order or bivariate analysis to reveal statistically significant dependencies.

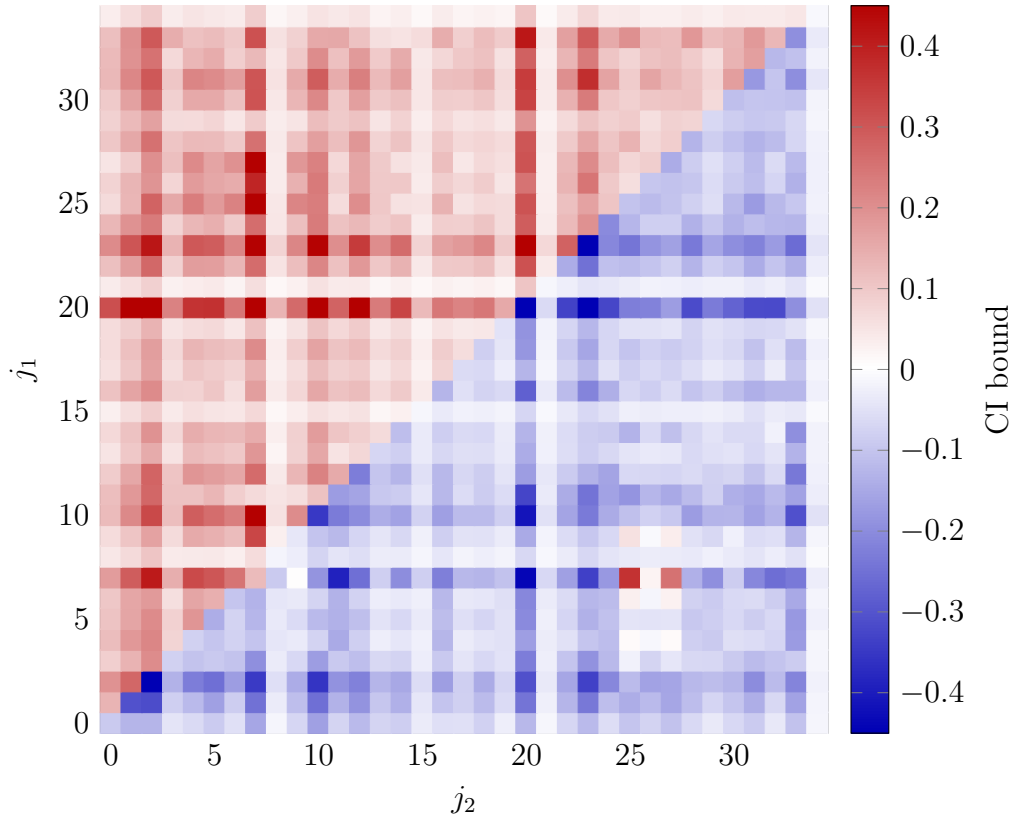


Figure 3.9: Confidence interval map associated with the bivariate Welch t -test on masked ASCAD traces with aggregation size $r = 20$ and maximum displacement $\delta_{\max} = 34$. The upper triangular part contains the lower bounds, whereas the lower triangular part contains the upper bounds of the Bonferroni confidence intervals.

Table 3.1: Summary of the statistical test results. FK: fixed key, RK: random key, FP: fixed plaintext, RP: random plaintext.

Test	Dataset	Setting	Outcome
1st order	CW unmasked	FK-FP vs FK-RP	leakage detected
1st order	CW unmasked	FK-FP vs RK-FP	leakage detected
1st order	ASCAD masked	bit 0 of SBox output, $b = 2$	no detectable leakage
2nd order	CW unmasked	FK-FP vs RK-FP	leakage detected
2nd order	ASCAD masked	bit 0 of SBox output, $b = 2$	leakage detected
Bivariate	ASCAD masked	bit 0 of SBox output, $b = 2$	leakage detected

3.3. Attack Results

This section presents the experimental results obtained with the two attack techniques implemented in the library, namely Correlation Power Analysis and Template Attack. For

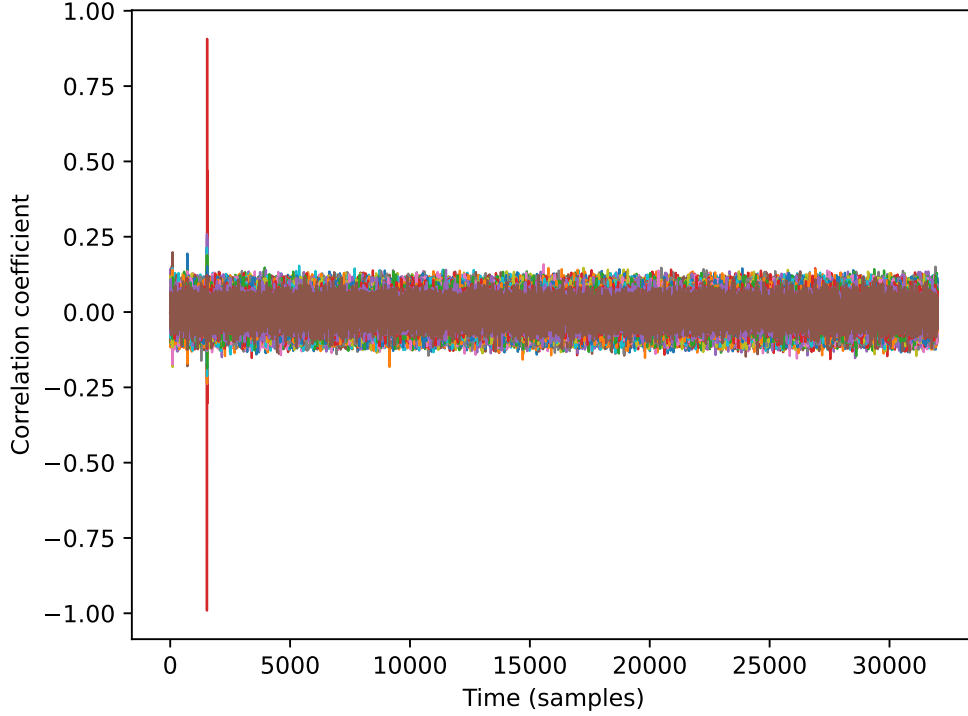


Figure 3.10: CPA correlation profiles on the unmasked ChipWhisperer traces for all 256 key hypotheses of the attacked byte 0. Each curve reports the temporal Pearson correlation coefficient between the measured leakage and the corresponding prediction vector.

each method, the considered traces, the adopted experimental configuration and the main analysis parameters are specified explicitly. Whenever they are included in the considered setting, the corresponding reduction strategies are discussed as well.

3.3.1. Correlation Power Analysis

The CPA experiment is conducted on the unmasked ChipWhisperer traces. The analysis targets byte index $b = 0$ and adopts the Hamming weight of the SBox output as leakage model. For the attacked byte, the hypothetical leakage is evaluated for all 256 key candidates. The analysis is carried out on 1000 unmasked traces of length 32000 samples. Since the computation requires the evaluation of hypothetical intermediate values for each key hypothesis, the plaintext metadata are used to derive the corresponding prediction matrix. The results are discussed through the temporal correlation profiles and through their incremental evolution as the number of processed traces increases.

Figure 3.10 reports the temporal correlation profiles obtained for all 256 key hypotheses of

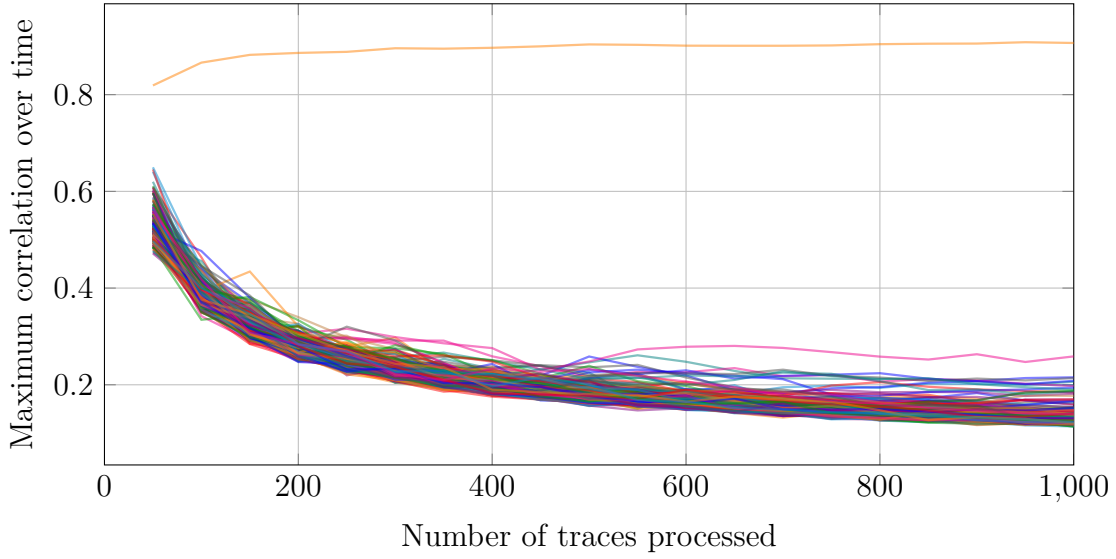


Figure 3.11: Incremental evolution of the maximum temporal correlation for all CPA key hypotheses on the unmasked ChipWhisperer traces.

the attacked byte. Each curve represents the Pearson correlation coefficient between the measured traces and the prediction vector associated with one key candidate. The figure shows that one hypothesis clearly dominates the others over a limited portion of the trace, indicating a strong agreement between the measured leakage and the adopted prediction model. In the present experiment, the largest absolute correlation reaches approximately 0.991 around sample 1532, whereas the strongest competing hypothesis remains below about 0.260. This large separation indicates that the selected model is highly effective in the considered unmasked setting and that the leakage relevant to the attacked byte is concentrated in a specific temporal region of the acquisition.

To better assess how the discrimination between hypotheses evolves as additional traces are processed, the incremental maxima of the correlation profiles are considered next.

Figure 3.11 shows the incremental evolution of the maximum correlation over time for all key hypotheses. For each processed batch, the reported value corresponds to the largest temporal correlation reached by the corresponding hypothesis up to that point. This representation makes it possible to evaluate how quickly the correct key candidate becomes distinguishable as the number of available traces increases. In the present experiment, the dominant hypothesis already reaches a maximum signed correlation of about 0.866 after 100 traces, 0.898 after 300 traces, and 0.907 at 1000 traces. The progression indicates that the separation between the leading hypothesis and the remaining candidates becomes stable relatively early in the acquisition process.

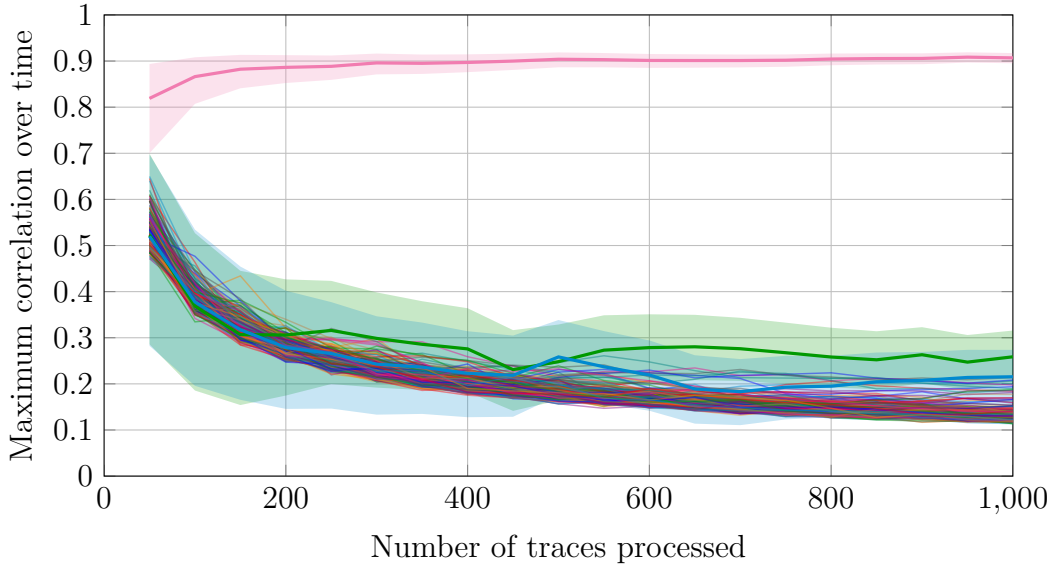


Figure 3.12: Incremental evolution of the maximum temporal correlation with confidence intervals for the top three CPA key hypotheses on the unmasked ChipWhisperer traces.

To complement the incremental analysis, confidence intervals are computed for the incremental correlation estimates. For readability, Figure 3.12 reports the confidence intervals only for the three key hypotheses with the largest final correlation values.

The shaded regions in Figure 3.12 represent the uncertainty associated with the estimated maximum correlation as the number of processed traces increases. The interval associated with the dominant hypothesis remains well separated from the competing hypotheses, confirming that the observed ranking is stable. As the number of traces increases, the confidence intervals become narrower, indicating a reduction in the uncertainty of the correlation estimates.

The analysis is then repeated by considering the absolute value of the correlation, which is the quantity typically used to rank CPA key hypotheses.

Figure 3.13 reports the incremental evolution of the maximum absolute correlation for all key hypotheses. This representation is especially relevant because CPA ranking is typically based on the largest absolute correlation observed over time. In the considered experiment, the dominant hypothesis already reaches an absolute correlation close to 0.991 after 100 traces and remains nearly unchanged thereafter. The profile also shows a substantial separation from the strongest wrong candidate over the whole acquisition range. This behavior indicates that, in the present unmasked setting, the correct hypothesis is distinguished reliably after a relatively small number of traces.

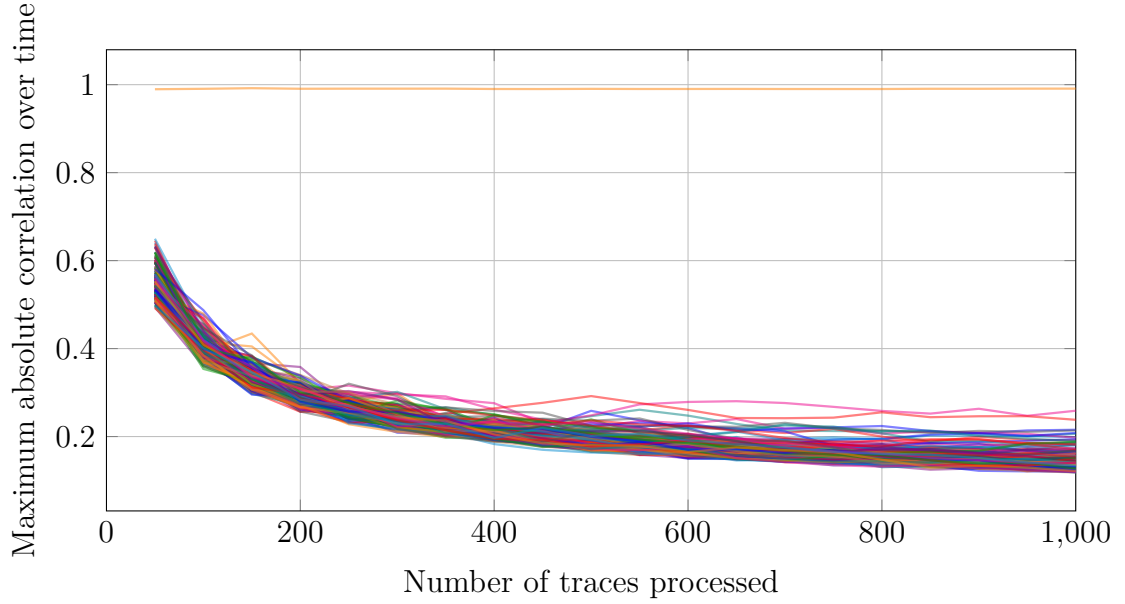


Figure 3.13: Incremental evolution of the maximum absolute temporal correlation for all CPA key hypotheses on the unmasked ChipWhisperer traces.

3.3.2. Template Attack

Template Attack is evaluated on the unmasked ChipWhisperer traces in a binary classification setting. The starting dataset is the fixed-key random-plaintext acquisition set. In the present implementation, the template attack is evaluated in a binary classification setting rather than in a full key ranking setting. The two classes are defined from bit 0 of the SBox output associated with byte index $b = 0$. Denoting by

$$v_i^{(0)} = \text{SBox}(P_i[0] \oplus K_i[0]), \quad (3.5)$$

the corresponding intermediate for trace i , the class label is given by

$$s_i = (v_i^{(0)})_0 \in \{0, 1\}. \quad (3.6)$$

After the partitioning step, two balanced sets of 400 traces each are obtained. For each class, 250 traces are used for profiling and 150 for attack. In the profiling phase, the template parameters, i.e., the class mean vector and covariance matrix, are estimated from the training subset. Since the original traces are 32000 samples long, the reported experiments rely on feature reduction before template estimation. All feature reduction methods implemented in the library are considered in the experimental comparison. For sample selection methods, top- k denotes the reduced representation obtained by retaining

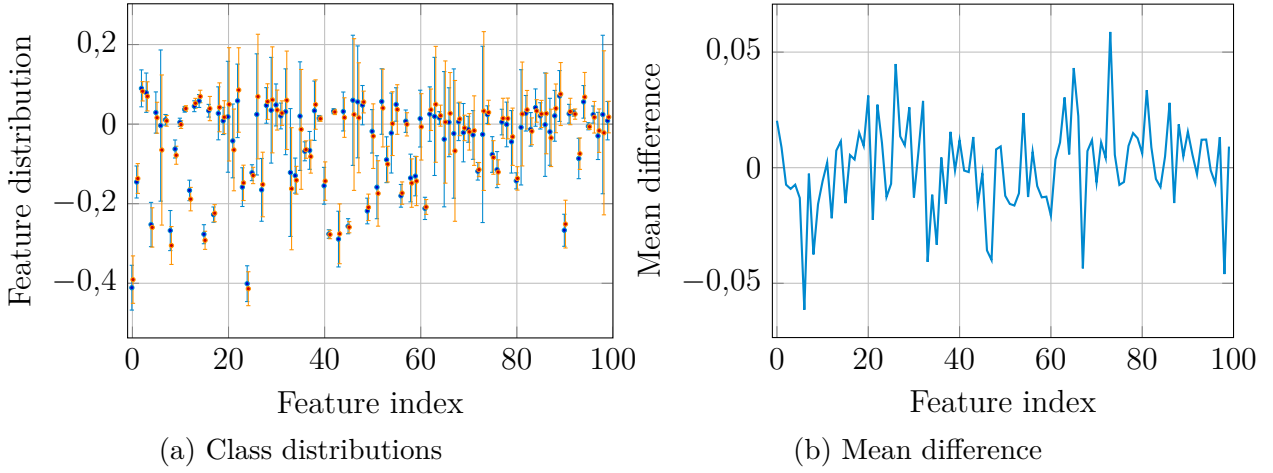


Figure 3.14: Class distributions and mean difference for the Template Attack with SNR-based reduction to the top $k = 100$ features. Panel (a) reports, for each retained feature, the estimated class means as points, with vertical whiskers corresponding to the class standard deviation. Panel (b) reports the difference between the two class means.

the k samples with the highest score according to the selected criterion, whereas for PCA it denotes the projection onto the first k principal components.

The attack performance is evaluated in terms of classification accuracy, defined as

$$\text{Accuracy} = \frac{N_{\text{corr}}}{N_{\text{corr}} + N_{\text{err}}}. \quad (3.7)$$

The complete accuracy results are reported in Table 3.2.

In order to inspect the effect of feature reduction in more detail, the following analysis focuses on the SNR-based reduced representations. Two cases are considered, obtained by retaining the top $k = 100$ and top $k = 10$ samples, respectively. For the $k = 100$ reduced representation, the attack yields an accuracy of 0.576. The corresponding templates are analyzed through the estimated class means and their variability, the mean difference, and the covariance difference, reported in Figures 3.14 and 3.15.

Figure 3.14 reports the estimated class means and their variability for the $k = 100$ reduced representation, together with the difference between the two class mean vectors. The figure shows that the separation between the two classes is not concentrated on a single retained feature, and that the mean differences are generally moderate with respect to the variability of the corresponding features. This suggests that the classification is based on the combined contribution of several weakly discriminating features, which is consistent with the limited accuracy obtained for $k = 100$.

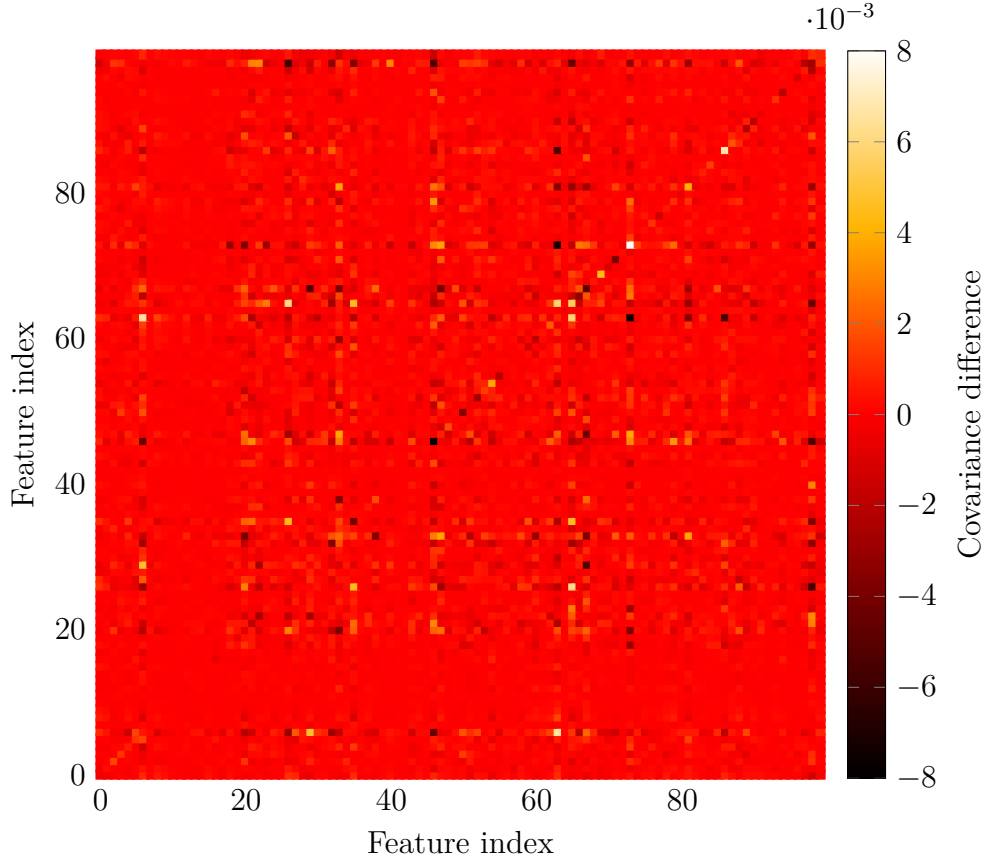


Figure 3.15: Difference between the covariance matrices estimated for the two classes in the $k = 100$ reduced representation. The color scale reports the signed covariance difference.

Figure 3.15 provides complementary information by showing the difference between the covariance structures of the two classes. The visible patterns indicate that the distinction between the templates is not limited to their mean values, but also involves the dependence structure of the retained features. At the same time, the relatively high dimensionality of this representation makes the class separation less effective in terms of final classification accuracy.

For the $k = 10$ reduced representation, the attack accuracy increases to 0.650. The corresponding templates are analyzed through the estimated class means and their variability, the mean difference, and the covariance difference, reported in Figures 3.16 and 3.17.

Figure 3.16 reports the estimated class means and their variability for the $k = 10$ reduced representation, together with the corresponding mean difference. Also in this case, the two classes are not sharply separated on individual features. However, the same type of class-dependent variation is represented in a more compact feature space. This is consistent

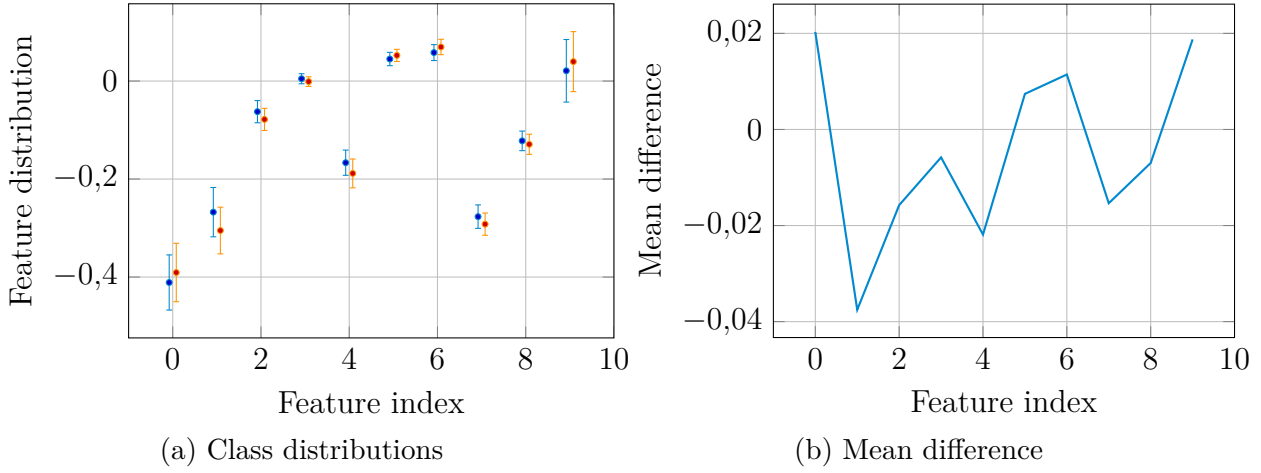


Figure 3.16: Class distributions and mean difference for the Template Attack with SNR-based reduction to the top $k = 10$ features. Panel (a) reports, for each retained feature, the estimated class means as points, with vertical whiskers corresponding to the class standard deviation. Panel (b) reports the difference between the two class means.

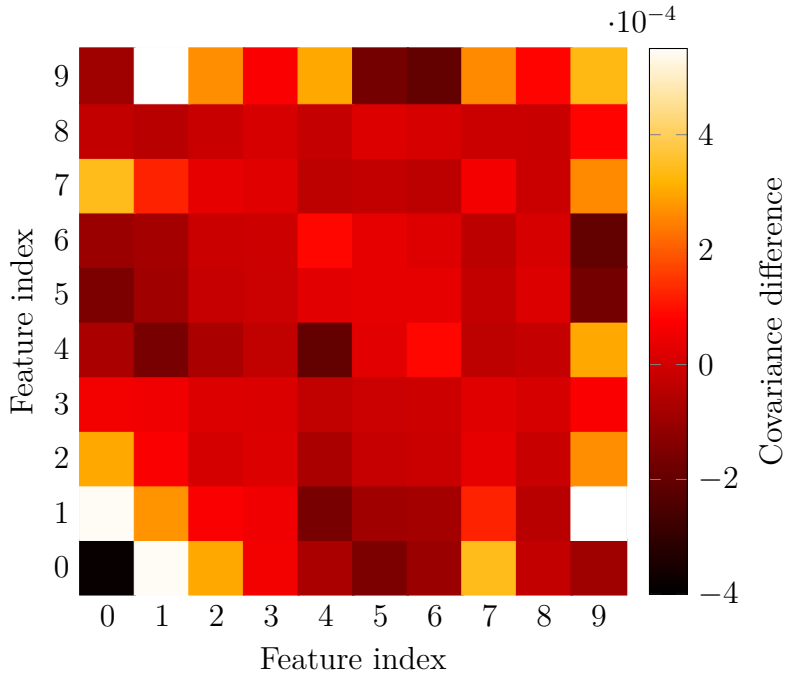


Figure 3.17: Difference between the covariance matrices estimated for the two classes in the $k = 10$ reduced representation. The color scale reports the signed covariance difference.

with the higher accuracy obtained for $k = 10$, since fewer weakly informative features are included in the template estimation.

Figure 3.17 reports the covariance difference associated with the $k = 10$ templates. As

Table 3.2: Summary of Template Attack accuracy for the considered feature-reduction methods. The columns report the accuracy obtained for each value of k , where k denotes the number of retained samples for SNR, NICV and SOST, and the number of retained principal components for PCA.

Feature reduction	Accuracy			
	$k = 10$	$k = 20$	$k = 50$	$k = 100$
SNR	0.650	0.613	0.543	0.576
NICV	0.650	0.613	0.543	0.576
SOST	0.607	0.613	0.580	0.560
PCA	0.587	0.573	0.510	0.500

in the $k = 100$ case, the two classes differ not only in their mean values but also in their covariance structure. In the present experiment, this lower dimensional representation is associated with the highest classification accuracy among the considered settings. A plausible interpretation is that the reduced feature space retains the most informative components while making the template estimation less affected by weakly informative dimensions.

A direct comparison between the two settings shows that the more compact reduced representation provides the better result in the present experiment. This trend is also visible in Table 3.2, where the reduced representations with $k = 10$ consistently achieve higher accuracy than the corresponding representations with $k = 100$. The higher accuracy obtained for $k = 10$ is consistent with the CPA results, which indicate that the informative leakage is concentrated in a limited temporal region. Under these conditions, retaining a larger number of features does not necessarily improve the template attack. On the contrary, the additional components may introduce weakly informative variations that are learned by the estimated class statistics together with the relevant signal. A more compact SNR-based representation therefore appears to preserve the dominant informative region while reducing the influence of less relevant components.

It should finally be noted that the present experiment evaluates Template Attack in a simplified binary classification setting, where the traces are partitioned according to one bit of the targeted intermediate. In a more complete formulation of the method, the profiling phase would typically be carried out over a finer partition of the intermediate space, for instance by considering all 256 possible byte values of the targeted variable. The reported results should therefore be interpreted as a validation of the implemented template analysis procedure in a simplified controlled setting.

4 | Conclusion

This thesis addressed side channel analysis from both a methodological and a practical perspective. The work started from the study of the main concepts and techniques described in the literature and led to the design and implementation of FreeSCA, a modular Python library conceived to support practical experimentation within a coherent analysis framework. The proposed framework was developed and validated in the context of side channel analysis on AES-128 implementations.

A first relevant outcome of this work is that the proposed framework provides a unified environment for the main stages of side channel analysis. Through the FSCA trace format and the implemented conversion tools, datasets originating from different sources can be imported, organized and processed within the same workflow. This makes it possible to move from heterogeneous trace collections to a more uniform analysis setting, while maintaining a structure that is modular and naturally extensible. The adoption of an incremental approach across the library further contributes to its practical usability, since large trace datasets can be handled in a way that remains compatible with realistic experimental conditions.

The experimental results also show that the implemented methods behave consistently with the theoretical expectations. The statistical tests were able to highlight meaningful differences between masked and unmasked settings, thereby confirming the ability of the framework to support leakage assessment in a coherent way. At the same time, the attack experiments showed that the library can be used not only to detect the presence of leakage, but also to evaluate its practical exploitability through representative attack techniques such as Correlation Power Analysis and Template Attack. Taken together, these results indicate that FreeSCA is not only a collection of implemented methods, but a usable framework for organizing, executing and validating side channel analysis workflows.

The modular structure of the proposed framework naturally suggests several directions for future development. A first relevant extension concerns the further enrichment of the evaluation layer of the library. The current framework already includes several metrics and reduction methods, such as SNR, NICV, SOST and PCA, which support the analysis

and preprocessing of trace datasets. A natural next step would be to complement the existing metrics with tools aimed at quantifying the practical effort required by an attack. In this perspective, a dedicated effort evaluation module could be integrated to compute indicators such as required traces, rank evolution, success rate and guessing entropy. This would provide a more complete assessment of attack effectiveness and of the practical exploitability of the observed leakage.

A second line of development concerns the extension of the methods currently supported by the framework. In particular, the attack module could be expanded with additional techniques, including classical methods such as Differential Power Analysis and Mutual Information Analysis, as well as more advanced approaches already discussed in the state of the art, such as SASCA, exSASCA and deep learning based side channel analysis.

A further possible development concerns the statistical leakage detection module. Beyond the current t -test based methods, the framework could be extended with alternative criteria for characterizing and comparing leakage behavior, including information theoretic approaches based on mutual information. This would enlarge the analytical scope of the library and make it possible to study side channel leakage from complementary statistical perspectives.

Bibliography

- [1] Florian Bache, Jonas Wloka, Pascal Sasdrich, and Tim Güneysu. Multivariate TVLA - efficient side-channel evaluation using confidence intervals. *IEEE Trans. Computers*, 74(3):790–804, 2025.
- [2] Christophe Clavier, Jean-Luc Danger, Guillaume Duc, M. Abdelaziz Elaabid, Benoît Gérard, Sylvain Guilley, Annelie Heuser, Michael Kasper, Yang Li, Victor Lomné, Daisuke Nakatsu, Kazuo Ohta, Kazuo Sakiyama, Laurent Sauvage, Werner Schindler, Marc Stöttinger, Nicolas Veyrat-Charvillon, Matthieu Walle, and Antoine Wurcker. Practical improvements of side-channel attacks on AES: feedback from the 2nd DPA contest. *J. Cryptogr. Eng.*, 4(4):259–274, 2014.
- [3] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks - revealing the secrets of smart cards*. Springer, 2007.
- [4] National Institute of Standards and Technology. Advanced Encryption Standard (AES). Federal Information Processing Standards Publication FIPS 197, Department of Commerce, Washington, D.C., May 2023. Originally published in 2001; updated May 9, 2023.
- [5] Colin O’Flynn and Zhizhang (David) Chen. Chipwhisperer: An open-source platform for hardware embedded security research. In Emmanuel Prouff, editor, *Constructive Side-Channel Analysis and Secure Design - 5th International Workshop, COSADE 2014, Paris, France, April 13-15, 2014. Revised Selected Papers*, Lecture Notes in Computer Science, pages 243–260. Springer, 2014.
- [6] Kostas Papagiannopoulos, Ognjen Glamocanin, Melissa Azouaoui, Dorian Ros, Francesco Regazzoni, and Mirjana Stojilovic. The side-channel metrics cheat sheet. *ACM Comput. Surv.*, 55(10):216:1–216:38, 2023.
- [7] Stjepan Picek, Guilherme Perin, Luca Mariot, Lichao Wu, and Lejla Batina. Sok: Deep learning-based physical side-channel analysis. *IACR Cryptol. ePrint Arch.*, page 1092, 2021.
- [8] Emmanuel Prouff, Remi Strullu, Ryad Benadjila, Eleonora Cagli, and Cecile Du-

- mas. Study of deep learning techniques for side-channel analysis and introduction to ASCAD database. Cryptology ePrint Archive, Paper 2018/053, 2018.
- [9] Tobias Schneider and Amir Moradi. Leakage assessment methodology - A clear roadmap for side-channel evaluations. In Tim Güneysu and Helena Handschuh, editors, *Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings*, volume 9293 of *Lecture Notes in Computer Science*, pages 495–513. Springer, 2015.
- [10] Nicolas Veyrat-Charvillon, Benoît Gérard, and François-Xavier Standaert. Soft analytical side-channel attacks. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 282–296. Springer, 2014.
- [11] Thomas Wedenig, Rishub Nagpal, Gaëtan Cassiers, Stefan Mangard, and Robert Peharz. Exact soft analytical side-channel attacks using tractable circuits. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pages 52472–52483. PMLR / OpenReview.net, 2024.
- [12] Wikipedia contributors. Algorithms for calculating variance — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Algorithms_for_calculating_variance&oldid=1345359962, 2026. [Online; accessed 15-April-2026].
- [13] Wikipedia contributors. Pearson correlation coefficient — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Pearson_correlation_coefficient&oldid=1346864658, 2026. [Online; accessed 15-April-2026].

A | Appendix A

This appendix reports the incremental estimators used in the implementation for mean, variance and covariance. These update schemes are employed whenever traces are processed in batches, so that the required statistics can be updated without storing all observations in memory.

The implementation of trace mean and variance estimation follows Welford's one-pass scheme [12]. In the present setting, each processed observation is a trace vector and the update is therefore applied in vector form to all time samples simultaneously. As a result, the maintained quantities are vectors indexed by the trace samples, while the variance is recovered component-wise at the end of the procedure. The corresponding recurrence is reported below, together with the pseudocode of the estimator used in the implementation.

Let $\mathbf{m}_n \in \mathbb{R}^t$ denote the n -th trace processed by the estimator, where t is the trace length. Let $\hat{\boldsymbol{\mu}}_{n-1} \in \mathbb{R}^t$ denote the current mean trace after $n - 1$ updates. Welford's recurrence updates the mean as

$$\hat{\boldsymbol{\mu}}_n = \hat{\boldsymbol{\mu}}_{n-1} + \frac{\mathbf{m}_n - \hat{\boldsymbol{\mu}}_{n-1}}{n}. \quad (\text{A.1})$$

At the same time, the auxiliary accumulator vector $M_{2,n} \in \mathbb{R}^t$ is updated as

$$M_{2,n} = M_{2,n-1} + (\mathbf{m}_n - \hat{\boldsymbol{\mu}}_{n-1})(\mathbf{m}_n - \hat{\boldsymbol{\mu}}_n), \quad (\text{A.2})$$

where the product between the two vectors is taken component-wise. Once all traces have been processed, the sample variance vector is recovered as

$$\hat{\boldsymbol{\sigma}}^2 = \frac{M_{2,n}}{n - 1}. \quad (\text{A.3})$$

Algorithm A.1 summarizes the corresponding estimator used in the implementation.

In the implementation, covariance is estimated through a one-pass update scheme that uses the same mean recurrence as the variance case, but replaces the component-wise accumulator with a matrix accumulator that captures the joint variation of all pairs of

Input: t : trace length

Data: n : number of processed traces,

$\hat{\boldsymbol{\mu}} \in \mathbb{R}^t$: current mean trace,

$M_2 \in \mathbb{R}^t$: accumulated squared deviations

```

1 function Init( $t$ ):
2    $n \leftarrow 0$ 
3    $\hat{\boldsymbol{\mu}} \leftarrow \emptyset$ 
4    $M_2 \leftarrow \emptyset$ 
5 function AddTrace( $\mathbf{m} \in \mathbb{R}^t$ ):
6   if  $n = 0$  then
7      $\hat{\boldsymbol{\mu}} \leftarrow 0 \in \mathbb{R}^t$ 
8      $M_2 \leftarrow 0 \in \mathbb{R}^t$ 
9    $n \leftarrow n + 1$ 
10   $\boldsymbol{\delta} \leftarrow \mathbf{m} - \hat{\boldsymbol{\mu}}$            // deviation from the previous mean trace
11   $\hat{\boldsymbol{\mu}} \leftarrow \hat{\boldsymbol{\mu}} + \boldsymbol{\delta}/n$ 
12   $\boldsymbol{\delta}_2 \leftarrow \mathbf{m} - \hat{\boldsymbol{\mu}}$            // deviation from the updated mean trace
13   $M_2 \leftarrow M_2 + \boldsymbol{\delta} \boldsymbol{\delta}_2$            // component-wise product
14 function AddTraces( $M$ ):
15   foreach trace  $\mathbf{m}$  in  $M$  do
16     AddTrace( $\mathbf{m}$ )
17 function GetMean():
18   return  $\hat{\boldsymbol{\mu}}$ 
19 function GetVar():
20   if  $n \leq 1$  then
21     return None
22   return  $M_2/(n - 1)$            // sample variance, component-wise

```

Algorithm A.1: Incremental mean and variance estimator

trace samples. In this setting, each processed observation is a trace vector $\mathbf{m}_n \in \mathbb{R}^t$, where t denotes the trace length. The estimator maintains the current mean vector $\hat{\boldsymbol{\mu}}_n \in \mathbb{R}^t$ together with a matrix accumulator

$$Q_n \in \mathbb{R}^{t \times t}, \quad (\text{A.4})$$

from which the covariance matrix is recovered at the end of the procedure.

If $\hat{\boldsymbol{\mu}}_{n-1}$ denotes the mean vector after $n - 1$ updates, the arrival of a new trace $\mathbf{m}_n$ first updates the mean according to

$$\hat{\boldsymbol{\mu}}_n = \hat{\boldsymbol{\mu}}_{n-1} + \frac{\mathbf{m}_n - \hat{\boldsymbol{\mu}}_{n-1}}{n}. \quad (\text{A.5})$$

As in the variance case, it is convenient to introduce the two deviations

$$\boldsymbol{\delta} = \mathbf{m}_n - \hat{\boldsymbol{\mu}}_{n-1}, \quad \boldsymbol{\delta}_2 = \mathbf{m}_n - \hat{\boldsymbol{\mu}}_n. \quad (\text{A.6})$$

The covariance accumulator is then updated through

$$Q_n = Q_{n-1} + \boldsymbol{\delta} \boldsymbol{\delta}_2^T. \quad (\text{A.7})$$

Here, $\boldsymbol{\delta} \boldsymbol{\delta}_2^T$ is a $t \times t$ matrix, so that the update is performed simultaneously on all covariance terms.

Once all samples have been processed, the sample covariance matrix is obtained as

$$\hat{\Sigma}_n = \frac{Q_n}{n-1}. \quad (\text{A.8})$$

Algorithm A.2 summarizes the corresponding estimator used in the implementation.

Input: t : feature dimension

Data: n : number of processed traces,

$\hat{\boldsymbol{\mu}} \in \mathbb{R}^t$: current mean vector,

$Q \in \mathbb{R}^{t \times t}$: accumulated covariance deviations

```

1 function Init( $t$ ):
2    $n \leftarrow 0$ 
3    $\hat{\boldsymbol{\mu}} \leftarrow \mathbf{0} \in \mathbb{R}^t$                                      // zero mean vector
4    $Q \leftarrow \mathbf{0} \in \mathbb{R}^{t \times t}$                              // zero covariance accumulator
5 function AddTrace( $\mathbf{m} \in \mathbb{R}^t$ ):
6    $n \leftarrow n + 1$ 
7    $\boldsymbol{\delta} \leftarrow \mathbf{m} - \hat{\boldsymbol{\mu}}$                                 // deviation from the previous mean vector
8    $\hat{\boldsymbol{\mu}} \leftarrow \hat{\boldsymbol{\mu}} + \boldsymbol{\delta}/n$ 
9    $\boldsymbol{\delta}_2 \leftarrow \mathbf{m} - \hat{\boldsymbol{\mu}}$                                 // deviation from the updated mean vector
10   $Q \leftarrow Q + \boldsymbol{\delta} \boldsymbol{\delta}_2^T$ 
11 function AddTraces( $M$ ):
12   foreach trace  $\mathbf{m}$  in  $M$  do
13     AddTrace( $\mathbf{m}$ )
14 function GetCovarianceMatrix():
15   if  $n \leq 1$  then
16     return None
17   return  $Q/(n-1)$                                            // sample covariance matrix

```

Algorithm A.2: Incremental covariance matrix estimator

B | Appendix B

In this appendix, the algebraic steps leading from the standard centered expression of the Pearson correlation coefficient to the equivalent form used in the CPA implementation are reported explicitly [13].

For readability, the dependence on the attacked byte b and on the considered time sample j is kept implicit throughout the derivation.

For a fixed time sample j , let

$$\mathbf{m} = (m_1, \dots, m_n) \in \mathbb{R}^n \quad (\text{B.1})$$

denote the vector collecting the measured leakage values across the n traces at the considered sample position. For a fixed attacked byte and for a fixed key hypothesis k , let

$$\mathbf{h}_k = (h_{1,k}, \dots, h_{n,k}) \in \mathbb{R}^n \quad (\text{B.2})$$

denote the corresponding prediction vector. The Pearson correlation coefficient between $\mathbf{m}$ and $\mathbf{h}_k$ is

$$r_k = \frac{\sum_{i=1}^n (m_i - \bar{m})(h_{i,k} - \bar{h}_k)}{\sqrt{\sum_{i=1}^n (m_i - \bar{m})^2} \sqrt{\sum_{i=1}^n (h_{i,k} - \bar{h}_k)^2}}, \quad (\text{B.3})$$

where

$$\bar{m} = \frac{1}{n} \sum_{i=1}^n m_i, \quad \bar{h}_k = \frac{1}{n} \sum_{i=1}^n h_{i,k}. \quad (\text{B.4})$$

The goal is to rewrite B.3 in terms of sums that can be accumulated incrementally while the traces are processed in batches. To do so, the numerator and the two denominator

terms are treated separately.

Expanding the products in the numerator and in the two denominator terms yields

$$r_k = \frac{\sum_{i=1}^n (m_i h_{i,k} - m_i \bar{h}_k - \bar{m} h_{i,k} + \bar{m} \bar{h}_k)}{\sqrt{\left(\sum_{i=1}^n m_i^2 - 2\bar{m} \sum_{i=1}^n m_i + n\bar{m}^2\right) \left(\sum_{i=1}^n h_{i,k}^2 - 2\bar{h}_k \sum_{i=1}^n h_{i,k} + n\bar{h}_k^2\right)}}. \quad (\text{B.5})$$

Replacing $\bar{m}$ and $\bar{h}_k$ with their explicit definitions in B.4 and collecting the resulting terms, gives

$$r_k = \frac{\sum_{i=1}^n m_i h_{i,k} - \frac{1}{n} \left(\sum_{i=1}^n m_i\right) \left(\sum_{i=1}^n h_{i,k}\right)}{\sqrt{\left(\sum_{i=1}^n m_i^2 - \frac{1}{n} \left(\sum_{i=1}^n m_i\right)^2\right) \left(\sum_{i=1}^n h_{i,k}^2 - \frac{1}{n} \left(\sum_{i=1}^n h_{i,k}\right)^2\right)}}. \quad (\text{B.6})$$

Finally, multiplying both numerator and denominator by n yields the equivalent computational form adopted in the implementation:

$$r_k = \frac{n \sum_{i=1}^n h_{i,k} m_i - \left(\sum_{i=1}^n h_{i,k}\right) \left(\sum_{i=1}^n m_i\right)}{\sqrt{\left(n \sum_{i=1}^n m_i^2 - \left(\sum_{i=1}^n m_i\right)^2\right) \left(n \sum_{i=1}^n h_{i,k}^2 - \left(\sum_{i=1}^n h_{i,k}\right)^2\right)}}. \quad (\text{B.7})$$

This expression is algebraically equivalent to the standard centered Pearson formula, but it is more convenient for implementation purposes. Indeed, it depends only on the sums

$$\sum_{i=1}^n m_i, \quad \sum_{i=1}^n m_i^2, \quad \sum_{i=1}^n h_{i,k}, \quad \sum_{i=1}^n h_{i,k}^2, \quad \sum_{i=1}^n h_{i,k} m_i, \quad (\text{B.8})$$

which can be accumulated incrementally while the traces are processed, without the need to store all the measured and predicted values explicitly.

List of Figures

1.1	Gaussian probability density functions for different values of μ and σ^2 . . .	8
1.2	Two-sided critical values for the standard normal distribution.	9
1.3	Student's t densities for different degrees of freedom ν , compared with the standard normal density.	10
1.4	Two-sided rejection regions for a Student's t reference distribution.	12
1.5	High-level structure of AES-128 encryption and round transformations. . .	13
1.6	Graphical decomposition of Shannon entropies: the overlap of $H(\mathcal{X})$ and $H(\mathcal{Y})$ corresponds to $I(\mathcal{X}; \mathcal{Y})$	21
1.7	Example with $n_{\text{bytes}} = 2$: per-byte histograms $\mathbf{h}_0$ and $\mathbf{h}_1$ obtained by binning the log-scores $w_0(k)$ and $w_1(k)$ into $n_{\text{bin}} = 6$ uniform intervals. . .	26
1.8	Example with $n_{\text{bytes}} = 2$: global histogram $\mathbf{f} = \mathbf{h}_0 * \mathbf{h}_1$ obtained by convolution. The bin $u^* = 6$ corresponds to the correct key.	27
1.9	Example of factor-graph fragment for AES, with variable nodes (known plaintext, key-byte hypothesis, and intermediates), factor nodes (xor and SBox) and soft evidence attached to an intermediate.	30
2.1	General structure of the FreeSCA library, with blue annotations indicating the role of the main components.	32
2.2	Simplified overview of the FreeSCA workflow.	34
2.3	General structure of a FSCA file.	35
2.4	Organization of the FSCA header.	36
2.5	General structure of a trace record in the FSCA format.	37
2.6	Example SNR and NICV profiles obtained from a two class trace partition over 2000 samples.	42
2.7	Example SOST profile obtained from a two class trace partition over 2000 samples.	43
2.8	Example cumulative energy profile obtained from the PCA decomposition. The red dashed lines indicate the number of principal components required to retain 80% and 99% of the cumulative energy, respectively.	46

2.9	Example of t -test profile over 2000 samples, together with the corresponding Bonferroni significance thresholds.	52
2.10	Example first order and second order t -test profiles over 700 samples for masked traces, together with the corresponding Bonferroni thresholds. . . .	58
3.1	Control experiment for the first order Welch t -test on unmasked ChipWhisperer traces. A set of 1000 fixed-key fixed-plaintext traces is split into two disjoint groups. The red curves denote the Bonferroni thresholds for $\alpha = 0.01$	74
3.2	First order Welch t -test profiles on unmasked ChipWhisperer traces. Panel (a) compares two fixed acquisition sets differing only in the plaintext value. Panel (b) compares two fixed acquisition sets differing only in the key value. . . .	76
3.3	First order Welch t -test profile on unmasked ChipWhisperer traces for the fixed-key fixed-plaintext versus fixed-key random-plaintext comparison. The red curves denote the Bonferroni thresholds for $\alpha = 0.01$	77
3.4	First order Welch t -test profile on unmasked ChipWhisperer traces for the fixed-key fixed-plaintext versus random-key fixed-plaintext comparison. The red curves denote the Bonferroni thresholds for $\alpha = 0.01$	77
3.5	First order Welch t -test profile on masked ASCAD traces for the partition induced by bit 0 of the SBox output associated with byte 2. The red curves denote the Bonferroni thresholds for $\alpha = 0.01$	78
3.6	Second order Welch t -test profile on unmasked ChipWhisperer traces for the fixed-key fixed-plaintext versus random-key fixed-plaintext comparison. The red curves denote the Bonferroni thresholds for $\alpha = 0.01$	79
3.7	Second order Welch t -test profile on masked ASCAD traces for the partition induced by bit 0 of the SBox output associated with byte 2. The red curves denote the Bonferroni thresholds for $\alpha = 0.01$	80
3.8	Bivariate Welch t -test statistic map on masked ASCAD traces with aggregation size $r = 20$ and maximum displacement $\delta_{\max} = 34$. The heatmap reports the absolute value of the statistic for all admissible $(j, j + \delta)$ pairs. . . .	82
3.9	Confidence interval map associated with the bivariate Welch t -test on masked ASCAD traces with aggregation size $r = 20$ and maximum displacement $\delta_{\max} = 34$. The upper triangular part contains the lower bounds, whereas the lower triangular part contains the upper bounds of the Bonferroni confidence intervals.	83

3.10	CPA correlation profiles on the unmasked ChipWhisperer traces for all 256 key hypotheses of the attacked byte 0. Each curve reports the temporal Pearson correlation coefficient between the measured leakage and the corresponding prediction vector.	84
3.11	Incremental evolution of the maximum temporal correlation for all CPA key hypotheses on the unmasked ChipWhisperer traces.	85
3.12	Incremental evolution of the maximum temporal correlation with confidence intervals for the top three CPA key hypotheses on the unmasked ChipWhisperer traces.	86
3.13	Incremental evolution of the maximum absolute temporal correlation for all CPA key hypotheses on the unmasked ChipWhisperer traces.	87
3.14	Class distributions and mean difference for the Template Attack with SNR-based reduction to the top $k = 100$ features. Panel (a) reports, for each retained feature, the estimated class means as points, with vertical whiskers corresponding to the class standard deviation. Panel (b) reports the difference between the two class means.	88
3.15	Difference between the covariance matrices estimated for the two classes in the $k = 100$ reduced representation. The color scale reports the signed covariance difference.	89
3.16	Class distributions and mean difference for the Template Attack with SNR-based reduction to the top $k = 10$ features. Panel (a) reports, for each retained feature, the estimated class means as points, with vertical whiskers corresponding to the class standard deviation. Panel (b) reports the difference between the two class means.	90
3.17	Difference between the covariance matrices estimated for the two classes in the $k = 10$ reduced representation. The color scale reports the signed covariance difference.	90

List of Tables

2.1 Summary of the dataset converters currently supported by FreeSCA. . . . 39

3.1 Summary of the statistical test results. FK: fixed key, RK: random key,
FP: fixed plaintext, RP: random plaintext. 83

3.2 Summary of Template Attack accuracy for the considered feature-reduction
methods. The columns report the accuracy obtained for each value of k ,
where k denotes the number of retained samples for SNR, NICV and SOST,
and the number of retained principal components for PCA. 91

List of Symbols

Variable	Description
$\mathcal{X}$	scalar random variable
n	number of traces
$\hat{\mu}$	sample mean
$\hat{\sigma}^2$	sample variance
Σ	covariance matrix
M	trace matrix
$\mathbf{m}_i$	i -th trace
b	target byte index
P	plaintext
K	secret key
k	key-byte hypothesis
$V^{(b)}$	intermediate values matrix for byte index b
$H^{(b)}$	prediction matrix for byte index b
g	group index
Q_g	trace group indexed by g
t	Welch t -statistic
ν	degrees of freedom
α	significance level
α^*	Bonferroni corrected significance level
τ	decision threshold
r_k	Pearson correlation coefficient for key-byte hypothesis k

Acknowledgements

I would like to thank Professor Alessandro Barengi for his guidance, valuable advice, and the perspective he offered throughout the development of this thesis. I would also like to thank Marco Gianvecchio for his assistance during the development of this thesis.

I would like to thank my family for their constant support throughout my academic journey. I am also deeply grateful to Nicolas for always being by my side and for his love, patience, and support.

