Securing Micro-controllers against side-channel attacks: evaluation tools and applications

Balazs Udvarhelyi

January 2024

ICTEAM
Louvain School of Engineering
UCLouvain
Louvain-la-Neuve
Belgium

Thesis Committee:

Pr. Standaert F.-X.	UCLouvain/ICTEAM, Belgium
Pr. Bol D.	UCLouvain/ICTEAM, Belgium
Pr. Buhan I.	Radboud University, Nijmegen, Netherlands
Pr. Eisenbarth T.	Institute for IT Security, University of Lübeck, Germany
Dr. Koeune F.	UCLouvain/ICTEAM, Belgium

Securing Micro-controllers against side-channel attacks: evaluation tools and applications

by Balazs Udvarhelyi

ICTEAM
UCLouvain
Place Sainte-Barbe, 2
1348 Louvain-la-Neuve
Belgium

This work has been funded in parts by the European Union (EU) through the ERC project 724725 (acronym SWORD) and by the EU-Walloon FEDER Project USERMedia (convention 501907-379156).

Acknowledgments

This thesis is the result of 4 years of research with the Crypto Group of UCLouvain. Doing a PhD is not an easy task, but I was lucky be surrounded with passionate people that made this adventure unforgettable! I would like to thank them now.

First and foremost, I would like to express my highest gratitude to my supervisor, François-Xavier Standaert. He first initiated me to the field, then always provided interesting research paths and useful feedback on my work, while allowing me to pursue my own ideas. I would also like to thank the jury members for reading this thesis and the discussions during the private defense.

Then, I would like to thank Olivier Bronchain, who first pushed me towards cryptography during my master's degree, then shepherded me during my master's thesis and first years of PhD, and finally, the many collaborations we had. Thanks to Charles Momin and Gaëtan Cassiers with whom I had many interesting discussions and collaborations! Thanks Charles also for the 4 years past together in the same offices! I would also like to thank Clément Hoffmann, Loïc Masure, Corentin Verhamme and Yaobin Shen for the time we shared at the office but mostly during our bike rides! Thanks to Davide Bellizia for uncovering the black magic related to measurement setups.

I would also like to thank my partner, Emilie and my parents for celebrating the good news with me, but also supporting me during hard times. Thank you also for listening to the stories about cryptography, my work and the related community!

Abstract

Microcontrollers units (MCUs) are nowadays widespread thanks to their versatility and low cost. On one hand, they are ideal candidates for IoT products. On the other, they are the typical target of side-channel attacks. Such attacks exploit unwanted indirect leakages of cryptographic algorithm's implementations. In this thesis, we study the side-channel security of MCUs where in the first part, we contribute to the evaluation methods by providing rules of thumb to design good measurement setups and an efficient method to analyze long traces and large states, which are common when analyzing software leakages.

In the second part, we explore different methods and countermeasures that can be used to secure MCUs against side-channels. We first show that relying on coprocessors is not enough. Then, we show an attack against a 32-bit implementation of ISAP, a leakage-resilient PRF and analyze possible countermeasures to counter our attack. We end this thesis by providing a method for noise generation on MCUs by repurposing already present coprocessors.

Contents

Acronyms

AES	Advanced Encryption Standard
ALU	Arithmetic-logic unit
ASIC	Application-specific integrated circuit
BP	Belief propagation
CMOS	Complementary metal oxide semi-conductor
COTS	Commercial off-the-shelf
CPA	Correlation power analysis
CPU	central processing unit
CWT	Continuous wavelet transform
DMA	Direct Memory Access
DPA	Differential power analysis
DSO	Digital storage oscilloscope
DUT	Device under test
EM	Electromagnetic
FPGA	Field-programmable gate array
FT	Fragment template
GE	Guessing entropy
HAL	Hardware abstraction layer
HDL	Hardware description language
HW	Hamming weight
IoT	Internet of Things
IV	Initialization vector
LDA	Linear discriminant analysis
LR	Linear regression
LSB	Least significant bit
MCU	Microcontroller unit
MI	Mutual information
MSB	Most significant bit
PCA	Principal component analysis
PCB	Printed circuit board
PDF	Probability density function
PI	Perceived information
PINI	Probe isolating non-interference
POI	Point of interest

PRF	Pseudorandom function
RLDA	Regression based linear discriminant analysis
SASCA	Soft Analytical Side-Channel Attacks
SCA	Side-channel attack
SNR	Signal to noise ratio
SPA	Simple power analysis
SR	Success rate
TA	Template attack
TRNG	True random number generator
TVLA	Test vector leakage assessment
XOR	Exclusive or

Introduction 1

Cryptography is the science of protecting information in the presence of adversaries. Among others, it has been used throughout history to secure sensitive communications, protect data, and ensure the privacy and integrity of information. Today, cryptography is omnipresent, its applications have been widened from mostly military to being used everyday, mainly in communications over the internet, for securing local storage from unauthorized access. The classical use-case for cryptography is formed by 2 parties (often called Alice and Bob) that want to communicate over an insecure channel which is controlled by an adversary (often called Eve) who can eavesdrop inject and/or alter messages. Cryptography makes sure that the rightful parties can communicate without adversaries being able to know or modify the messages.

Current standardized cryptographic algorithms are mostly analyzed in a black-box model in which, a fairly powerful adversary can observe the inputs and outputs but is not able to observe the computations. The design of algorithms with a solid mathematical basis combined with external cryptanalysis thanks to public specifications leads to an overall confidence gain in cryptography and its wider adoption. This is in part due to the application of Kerckhoff's principle, that states : *a cryptosystem should remain secure even if everything except the key is public.* Nowadays, standardized schemes, for example the Advanced Encryption Standard (AES) [oST01] are considered secure since no practical attacks have been found in multiple decades despite extensive effort from the community trying to spot weaknesses.

However, in the late 90s, a new type of attack has been introduced [Koc96, KJJ99]. It exploits unwanted physical leakages about the implementation, breaking many algorithms despite them being secure in the black-box model. Side-channel attacks (SCA) are one type of such physical attacks where the adversary passively observes a source of leakage. Common leakage sources include time (execution [Koc96] or cache accesses [Pag02, GSM15]), power consumption [KJJ99] and Electromagnetic (EM) emanations [QS01]. We define the grey-box model

where the adversary is able to collect information on more than the inputs and outputs.

Power and EM SCAs work on the principle that the power consumption of the device under attack is dependent on the data it manipulates. An adversary is therefore able to get information about intermediate variables of a computation that are only dependent on a subset of the key size, typically at the beginning or the end of the execution. Repeating the attack and targeting a different subset of the key each time in a divide and conquer manner allows reducing the search space and in fine recovering the key, making the cryptosystem useless.

Side-channels in practice Performing SCAs consists in first measuring leakage traces that are high temporal resolution measurements of the power consumption during the execution of the algorithm. Then, the adversary creates a model of the leakage for the possible key values and compares the models with the traces using statistical tools. The guessed key is the one whose model fits the most the leakages. This process is illustrated in Figure 1.1. The leakage model used can be based on actual measurements of a similar device that the adversary controls, in which case the leakage model and the attack are called profiled.

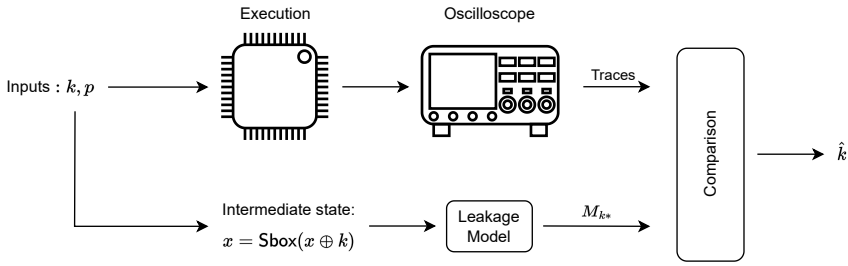


Figure 1.1: Side-channel attack steps.

Depending on the targeted algorithm, the adversary has the possibility to measure traces containing executions with different inputs, we call this Differential power analysis (DPA) or he can only observe one execution, which is called Simple power analysis (SPA). With SPA, in some cases the adversary is able to measure multiple times the same input and average its measurements to reduce the noise.

The number of traces required for a successful attack depends on the amount of information that can be extracted from the measurements. This in turn depends on the side-channel signal and the noise of the measurements, often analyzed together with the Signal to noise ratio

(SNR) [Man04]. Side-channel signal, or data dependency in the power consumption comes from the construction of CMOS logic gates where, at each clock cycle, the rising edge of the clock leads to a different amount of current being drawn from the power source, depending on the changes in state of the registers and memory.

The other part of side-channel measurements is noise. It can be classified into 2 categories, physical and algorithmic. The latter is composed of power consumed by components of the circuit that are not included in the leakage model of the SCA. Algorithmic noise can therefore be reduced if an adversary models a bigger part of the circuit. Physical noise on the other hand is due to electronic noise of the current flowing in the circuit and imperfections of the sensing methods.

The case of software implementations Cryptographic operations vulnerable to SCAs are performed on 2 types of hardware. On one side, specialized circuits (FPGA or ASIC) are specifically tailored to the implementation with near-full control on the product which helps implementing algorithms such that they meet the security requirements. On the other side, software implementations have fundamentally opposite properties. They are using a general purpose central processing unit (CPU) with little or no control at the circuit level.

Software implementations targeted by side-channel attacks run mainly on Microcontroller units (MCUs), the main element of IoT devices that are generally located in publicly accessible areas. Their widespread usage is due to the flexibility of the hardware running the code. It can be corrected and updated using a new firmware, even remotely, without too much effort. Moreover, the development costs are also lower compared to a hardware circuit.

The advantages of software implementations and the vulnerability of MCUs lead to the question of their side-channel security, which is the subject of this thesis. More precisely, we study Commercial off-the-shelf (COTS) MCUs from the side-channel evaluation side and from the implementation design side.

1.1 Side-channel evaluation limitations

One of the difficulties of obtaining good quality side-channel measurements is due to the large measurement setup design space, where good and many bad solutions exist. As measurements have a high impact on the extracted information and therefore the success of an attack,

security evaluations are also prone to overstatements in case of poor measurements.

Then, an other difficulty rises when modeling the leakages. MCUs exist in various bus sizes. SCAs on 8-bit devices are considered easy as the number of possible states in an 8-bit register can be easily modeled by an attacker and their low inherent physical noise offers little protection. However, this is not the case with 32-bit MCUs that are now common. Most attacks still use 8-bit models, which are suboptimal on a 32-bit bus. Naive scaling of current algorithms makes optimal 32-bit leakage models impractical due to the exponentially growing number of parameters to estimate. Moreover, due to the sequential nature of CPUs, the execution of an algorithm uses many clock cycles which leads to longer traces. The processing of such traces requires an additional, dimensionality reduction step, that too, grows exponentially with the bus size. Although, separately, the problem of long traces and large states have good solutions, there is no practical solution that fits both constraints.

1.2 Side-channel design limitations

The design of side-channel secure implementations is inevitably based on countermeasures, this is why, since the discovery of SCA, researchers have been interested in them. Physical ones, like noise generators, or any countermeasure requiring specialized circuitry are not built into COTS MCUs, therefore, a designer is limited to implement algorithmic countermeasures. We will discuss in this thesis implementations with the 2 most studied countermeasures in the literature, masking and shuffling.

First, shuffling [HOM06] aims at hiding the execution time of a specific operation by randomly rearranging independent operations. If n operations are performed, shuffling them provides an improvement of the attack complexity with same factor n, however, the shuffled indexes can also leak side-channel information, therefore reducing the attack complexity [VMKS12].

Masking [ISW03], on the other hand amplifies the naturally present noise in the system by splitting the sensitive variable, like the key into multiple random shares and performing computation on the shares. In this thesis, we focus on Boolean masking where, the sensitive variable is equal to the XOR of its shares. If a masked implementation uses n shares, in the so-called probing model, an adversary must know all the n shares of a variable in order to recover its value. Security in the probing model also implies security in the closer to real-world, noisy leakage

model, where all shares leak noisy observations [DDF14]. However, this requires that the following conditions are met : the noise level must be high enough and independence in the shares' leakages must be guaranteed. Although masking costs are quadratic in the number of shares and require additional randomness, masking comes with exponential security w.r.t. the number of shares that makes it very attractive.

As the side-channel measurements of MCUs are generally less noisy than their hardware counterpart, they contain more information and are therefore more challenging to secure. Therefore, software implementations that rely on masking are therefore costly, as the "sufficient noise" condition is not met, therefore requiring a high number of shares [BS21].

This leads us to an other countermeasure, that is re-keying, a countermeasure implemented in the algorithm itself. It consists in limiting the use of a (long-term) key in order to prevent DPA attacks. It does so by generating, for each encryption, an ephemeral key, that is used only once. The re-keying process itself can either be strongly protected using previously mentioned countermeasures or using a leakage-resilient process [DP10, SPY$^+$10, FPS12]. The latter provides protection against DPA attacks and its security relies on the capacity of an attacker to mount SPA attacks.

Such SPA attacks are known to be successful with 16-bit implementations [BBC$^+$20]. However, with the development of 32-bit ARM Cortex processors and their wide adoption by the industry, newly developed encryption schemes take full advantage of the wider bus. As a 32-bit SPA attack lies in a grey zone between the feasible 8 and 16-bit attacks and impossible 128-bit ones, state of the art leakage-resilient implementations are potentially vulnerable.

1.3 Contributions and outline of this thesis

First, all the necessary background to understand the rest of this thesis is resumed in Chapter 2. The main part of this thesis is divided in 2 parts. The first part provides tools for the evaluator to optimize the extracted information. In Chapter 3, we systematically compare parameters of measurement setups. While no go-to measurement setup is found, we show the impact of security overstatements that a bad setup can lead to. Plus, the evaluator is provided with rules of thumb to optimize its own setup. In Chapter 4, we present an efficient implementation of the Regression based linear discriminant analysis (RLDA), a tool that is well-suited for large states and long traces, alongside an efficient way to

estimate information theoretic metrics on the previously built leakage model.

The second part of this thesis analyzes concrete ways to use MCUs when side-channel protection is needed. In Chapter 5, we show that relying on hardware coprocessors with unknown specifications, leads to key recovery using basic distinguishers with limited reverse-engineering effort. In Chapter 6, we show that re-keying based, leakage-resilient 32-bit implementations can be attacked with SPA with an enumeration power of 2^{64}. This in turn shows that relying solely on re-keying is not enough and 32-bit software implementations might not be secure against SPA attacks. In Chapter 7 we build on the observations of the previous chapter and analyze how countermeasures combine with re-keying. We show first that masking is not combining well since it acts on the noise where SPA is only depending on the signal. Then, although shuffling should provide theoretically a good combination, the recovery of the permutation is possible on MCUs with low noise levels. In the last chapter, we take advantage of the presence of coprocessors and propose a solution by repurposing one as noise engine in order to make attacks like in Chapter 7 more challenging. We show that a reduction by a factor of 2 can be achieved in the information contained in the leakages, factor which can be amplified through the use of masking (and/or shuffling).

Background 2

In this chapter, we present the necessary background to understand the remaining of the thesis. First, we describe the notations used and present the SNR, an univariate evaluation metric. Then, we focus on SCAs that aim to recover the secret key used in an implementation. Enumerating over the whole key space is impossible with today's computational power. Therefore, SCAs model intermediate states of an implementation that are dependent only on a subset of the key, typically a byte. We present multiple leakage models derived from the Template attack (TA) of Chari et al. [CRR02]. An attack allows recovering a subset of the key but can be repeated independently on the same traces targeting other bytes. We then present Soft Analytical Side-Channel Attacks (SASCA)s that take advantage of the leakages of multiple variables and subsets of the key at the same time, combining the information obtained from them. Finally, we present multivariate evaluation metrics and side-channel countermeasures.

2.1 Notations

We use capital letters X for random variables and lower case letters for their realizations, such that x is the realization of X. When clear from the context, we use the shortcut notation of x for $X = x$. We use bold letters $\boldsymbol{x}$ for vectors, capital bold letters $\boldsymbol{X}$ for matrices and calligraphic letters $\mathcal{X}$ for sets. We additionally use the notation $\boldsymbol{X} = (\boldsymbol{x}_1 \dots \boldsymbol{x}_n)$ to denote the matrix whose columns are the vectors $\boldsymbol{x}_i$, and the notation $\mathbb{1}_n$ to denote the vector of length n whose coordinates are all 1.

We use the following conventions: n_s is the number of samples in a trace, n_p and n_a are respectively the number of profiling and attack traces, and p is the subspace dimensionality, reduced with Principal component analysis (PCA) or Linear discriminant analysis (LDA).

In the context of masking, the shares are denoted with subscripts such that x_i is the i-th share of x. In the context of shuffling, random vectors can be indexed with superscripts such that x^i is the i-th element in the vector $\boldsymbol{x}$.

The measured side-channel leakages are realizations of random vectors that we always denote $\boldsymbol{l}$. When the leakage has not been used for profiling a model we denote it as $\boldsymbol{l}^*$. If necessary, we use the subscript to distinguish the leakage source. For example, $\boldsymbol{l}_x$ is the leakage generated by the manipulation of x and $\boldsymbol{l}_{\boldsymbol{x}}$ is the leakage vector originated from the vector $\boldsymbol{x}$.

2.2 SNR

The SNR is a common univariate metric in side-channel analysis. It was first defined by Mangard in [Man04]. For an intermediate variable X, it models the signal as the variance of the mean leakage of each class $x \in \mathcal{X}$, and the noise as the mean of the variance of the leakage of each class.

$$\hat{\mathsf{SNR}} = \frac{\hat{\mathsf{Var}}_x\left(\hat{\mathsf{E}}_i\left(\boldsymbol{l}_{x,i}\right)\right)}{\hat{\mathsf{E}}_x\left(\hat{\mathsf{Var}}_i\left(\boldsymbol{l}_{x,i}\right)\right)}, \tag{2.1}$$

where $\boldsymbol{l}_{x,i}$ corresponds to the i^{th} leakage trace of variable x. $\hat{\mathsf{Var}}_i$ and $\hat{\mathsf{E}}_i$ (resp., $\hat{\mathsf{Var}}_x$ and $\hat{\mathsf{E}}_x$) are the sample variance and the sample mean over the leakages (resp. the classes). The result is a vector of length n_s containing the SNR for each sample in a leakage traces. We note here that the modeled noise is a combination of physical and algorithmic noise. Mangard's SNR is a good estimator for the complexity of univariate attacks like Correlation Power Analysis or (univariate) template attacks as their complexity can be directly linked to the SNR [Man04, MOS11, DFS19]. The SNR can also be used as tool to detect Points of interest (POIs) as the time samples in a measured leakage trace where the SNR is high represent samples for which first-order information can be extracted.

2.3 Gaussian Template attack

The TA, introduced by Chari et al. [CRR02] is the optimal side-channel attack. It is done in 2 phases.

It starts with a profiling phase. During this step, leakage traces and the corresponding plaintexts and keys are given to the adversary. Based on these, he estimates a Probability density function (PDF) of the leakage $\boldsymbol{l}$ given an intermediate state x. The leakage distribution is

assumed to be Gaussian, so that this conditional distribution is

$$\hat{\mathsf{f}}^{\mathsf{GT}}[\boldsymbol{l}^*|X=x] = \frac{1}{\sqrt{(2\pi)^n \left|\hat{\boldsymbol{\Sigma}}\right|}} \cdot \exp\left(-\frac{1}{2}\left(\boldsymbol{l}^* - \hat{\boldsymbol{\mu}}(x)\right)^T \hat{\boldsymbol{\Sigma}}^{-1}\left(\boldsymbol{l}^* - \hat{\boldsymbol{\mu}}(x)\right)\right),$$

where $\hat{\boldsymbol{\mu}}(x)$ and $\hat{\boldsymbol{\Sigma}}$ are the estimated mean vector for the leakages of x and the noise covariance matrix. In all models, the noise covariance matrix is pooled [CK13], meaning that only one matrix is estimated instead of estimating one for each class. As the covariance matrix models the noise, it is sound to assume that it is independent of x.

In the second, attack phase, new leakage traces $\boldsymbol{l}^*$ and only the corresponding plaintext are given to the adversary. Based on these and the estimated PDF, he uses Bayes' theorem to estimate the likelihoods of $\hat{\mathsf{p}}[x|\boldsymbol{l}^*]$ for each trace independently (assuming that the prior distribution of X is known):

$$\hat{\mathsf{p}}^{\mathsf{GT}}[X=x|\boldsymbol{l}^*] = \frac{\hat{\mathsf{f}}^{\mathsf{GT}}[\boldsymbol{l}^*|X=x]\mathsf{Pr}[X=x]}{\sum_{x'\in\mathcal{X}} \hat{\mathsf{f}}^{\mathsf{GT}}[\boldsymbol{l}^*|X=x^*]\mathsf{Pr}[X=x']}\,. \tag{2.2}$$

The adversary performs multiple measurements to mount the attack. From n_a attack traces, he infers the key byte with maximum likelihood as

$$\hat{\mathsf{p}}[k, \boldsymbol{l}_x^*] = \prod_{i=1}^{n_a} \hat{\mathsf{p}}[x(p_i,k)|\boldsymbol{l}_{x,i}^*] \tag{2.3}$$

$$\hat{k} = \underset{k^*\in\mathcal{K}}{\operatorname{argmax}}\ \hat{\mathsf{p}}[k^*, \boldsymbol{l}_x^*]\,, \tag{2.4}$$

where p_i is the plaintext corresponding to the $\boldsymbol{l}_{x,i}^*$ trace and $\boldsymbol{l}_x^*$ is the concatenation of the n_a traces.

In practice, Chari et al.'s TA has limitations in terms of computational and profiling effort when used in software leakages. In the following sections, we present leakage models derived from the optimal TA that result in more efficient methods at the cost of a small loss in precision. The profiling phases differ and lead to different models $\hat{\mathsf{f}}^{\mathsf{M}}$ where M is the name of the model. The second phase being identical to all models, it is simply needed to plug in $\hat{\mathsf{f}}^{\mathsf{M}}$ into Equation 2.2.

2.4 Regression-based Gaussian models

Linear regression (LR)-based profiled attacks have been introduced by Schindler et al. [SLP05]. They can be viewed as a variant of Chari et al.'s TA, where the deterministic part of the leakage function is expressed as a linear combination of basis functions, in order to reduce the number of profiling traces. For every time sample s in a leakage trace, instead of estimating this deterministic part thanks to the mean of the sample for every class (i.e., every possible value $x \in \mathcal{X} = \mathbb{Z}_{2^b}$ of the target variable), the LR-based attack fits a weighted sum of n_b basis functions denoted as β_i:

$$\mathsf{m}_s(x) = \boldsymbol{a}_s^T \cdot \boldsymbol{\beta}(x) = \sum_{i=0}^{n_b-1} a_{i,s}\beta_i(x),$$

where the $a_{i,s}$'s are the weights of the model learned by profiling. A simple choice for the basis functions is the linear basis, where each function β_i corresponds to a single bit of the target value x:

$$\beta_i(x) = \begin{cases} 1 & \text{if } i = 0, \\ 1 & \text{if } \lfloor x/2^{i-1} \rfloor \mod 2 = 1, \\ -1 & \text{otherwise,} \end{cases} \tag{2.5}$$

with $n_b = b + 1$ and b being the number of bits X. The profiling step then collects a set of n_p traces $\boldsymbol{L} = (\boldsymbol{l}_1 \cdots \boldsymbol{l}_{n_p})$ and their corresponding values $\boldsymbol{x} = (x^1, x^2, \ldots, x^{n_p}) \in \mathcal{X}^{n_p}$. Letting $\boldsymbol{l}'_s$ be the sth row of $\boldsymbol{L}$ and $\boldsymbol{\beta}(\boldsymbol{x}) = (\boldsymbol{\beta}(x^1) \ldots \boldsymbol{\beta}(x^{n_p}))$, it minimizes the norm of the residual:

$$\boldsymbol{r}_s^T = (\boldsymbol{l}'_s)^T - \boldsymbol{a}_s^T \cdot \boldsymbol{\beta}(\boldsymbol{x}). \tag{2.6}$$

The minimization of $\|\boldsymbol{r}_s\|$ is an ordinary linear least squares problem that can be transformed into a small linear system using the normal equations:

$$(\boldsymbol{l}'_s)^T \cdot \boldsymbol{\beta}(\boldsymbol{x})^T = \boldsymbol{a}_s^T \cdot \boldsymbol{\beta}(\boldsymbol{x}) \cdot \boldsymbol{\beta}(\boldsymbol{x})^T, \tag{2.7}$$

for each time sample $s \in \{1, \ldots, n_s\}$. Once $\mathsf{m}_s(x)$ is computed, it is used as the mean for a pooled template [CK13], i.e., we estimate a single approximate covariance matrix for all the traces $\hat{\boldsymbol{\Sigma}}$, computed as the empirical covariance of the residual $\boldsymbol{R} = (\boldsymbol{r}_1 \cdots \boldsymbol{r}_{n_s})^T$:

$$\hat{\boldsymbol{\Sigma}} = \frac{1}{n_p} \boldsymbol{R}\boldsymbol{R}^T. \tag{2.8}$$

(There is no need to subtract the mean since $\boldsymbol{r}$ has mean 0 by construction.)

Finally, the modeled probability density for a new leakage trace $\boldsymbol{l}^*$ is:

$$\hat{\mathsf{f}}^{\mathsf{LR}}[\boldsymbol{l}^*|X=x] = \frac{1}{\sqrt{(2\pi)^n \left|\hat{\Sigma}\right|}} \exp\left(-\frac{1}{2}(\boldsymbol{l}^* - \mathbf{m}(x))^T \hat{\boldsymbol{\Sigma}}^{-1}(\boldsymbol{l}^* - \mathbf{m}(x))\right), \tag{2.9}$$

where $\mathbf{m}(x) = (\mathsf{m}_1(x), \ldots, \mathsf{m}_{n_s}(x))$.

2.5 Subspace-based Gaussian templates

Leakage traces can be long, especially in software implementations, and even if the leakage model is limited to the POIs, profiling will require many traces and long computation time due to the need to accurately estimate and invert a large covariance matrix $\hat{\boldsymbol{\Sigma}}$. It is therefore common to use a dimensionality reduction technique to reduce the high number of POIs into a more manageable number of samples while preserving (most of its) useful content [SA08].

2.5.1 Principal Component Analysis

The PCA [APSQ06] is one such technique that relies on finding a linear projection maximizing the inter-class variance.

First, the mean vector of each class $\hat{\boldsymbol{\mu}}(x)$ is estimated. Given a set of traces $\mathcal{L}$, let $\mathcal{L}(x)$ be the subset of traces in $\mathcal{L}$ for which $X = x$, the scatter matrix of the means

$$\boldsymbol{S_B} = \sum_{x \in \mathcal{X}} |\mathcal{L}(x)| \, (\hat{\boldsymbol{\mu}}(x) - \hat{\boldsymbol{\mu}})(\hat{\boldsymbol{\mu}}(x) - \hat{\boldsymbol{\mu}})^T, \tag{2.10}$$

is calculated with $\hat{\boldsymbol{\mu}}$ being the mean of all classes $\hat{\boldsymbol{\mu}} = \frac{1}{|\mathcal{X}|} \sum_{x \in \mathcal{X}} \hat{\boldsymbol{\mu}}(x)$

The projection matrix is obtained by decomposing $\boldsymbol{S_B} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^T$ and taking the p eigenvectors corresponding to the highest eigenvalues which we denote with the matrix $\boldsymbol{W}^{\mathsf{PCA}} \in \mathbb{R}^{n_s \times p}$, where n_s is the number of samples before the projections, usually the number of POIs.

It is now possible to build the PCA-based multivariate Gaussian template by projecting the means and the pooled covariance matrix to the subspace:

$$\hat{\mathsf{f}}^{\mathsf{PCA}}[\boldsymbol{l}^*|X=x] = \frac{1}{\sqrt{(2\pi)^n \left|\hat{\Sigma}_{\boldsymbol{W}^{\mathsf{PCA}}}\right|}} \cdot$$
$$\exp\left(-\frac{1}{2}\left(\boldsymbol{W}^{\mathsf{PCA}^T}(\boldsymbol{l}^* - \hat{\boldsymbol{\mu}}(x))\right)^T \hat{\boldsymbol{\Sigma}}^{-1}_{\boldsymbol{W}^{\mathsf{PCA}}} \left(\boldsymbol{W}^{\mathsf{PCA}^T}(\boldsymbol{l}^* - \hat{\boldsymbol{\mu}}(x))\right)\right),$$

where $\hat{\boldsymbol{\Sigma}}_{\boldsymbol{W}^{\mathsf{PCA}}} = |\mathcal{L}|^{-1}\, \boldsymbol{W}^{\mathsf{PCA}^T} \hat{\Sigma}\, \boldsymbol{W}^{\mathsf{PCA}}$ is the projected noise covariance matrix.

2.5.2 Linear Discriminant Analysis

An other common solution to mitigate the problem of high dimensional Gaussian templates is to apply Fisher's LDA in order to reduce the dimensionality of the trace similarly to the PCA. In a nutshell, LDA finds a linear projection to a fixed-dimensionality subspace that maximizes the side-channel SNR in that subspace. Formally, it finds the projection matrix $\boldsymbol{W}^{\mathsf{LDA}} \in \mathbb{R}^{n_s \times p}$ that maximizes the ratio between the inter-class scatter $\boldsymbol{S_B}$ from Equation 2.10 and the intra-class scatter $\boldsymbol{S_W}$, that is [DHS01]:

$$\arg\max_{\boldsymbol{W}^{\mathsf{LDA}}} \frac{\left|\boldsymbol{W}^T \boldsymbol{S_B} \boldsymbol{W}\right|}{|\boldsymbol{W}^T \boldsymbol{S_W} \boldsymbol{W}|}.$$

The scatter matrices can then be computed as:

$$\begin{aligned}
\hat{\boldsymbol{\mu}}(x) &= \frac{1}{|\mathcal{L}(x)|} \sum_{\boldsymbol{l} \in \mathcal{L}(x)} \boldsymbol{l}, \\
\hat{\boldsymbol{\mu}} &= \frac{1}{|\mathcal{L}|} \sum_{\boldsymbol{l} \in \mathcal{L}} \boldsymbol{l}, \\
\boldsymbol{S_W} &= \sum_{x \in \mathcal{X}} \sum_{\boldsymbol{l} \in \mathcal{L}(x)} (\boldsymbol{l} - \hat{\boldsymbol{\mu}}(x))(\boldsymbol{l} - \hat{\boldsymbol{\mu}}(x))^T,
\end{aligned}$$

and the maximization can be rewritten as the generalized eigendecomposition problem:

$$\boldsymbol{S_B} \boldsymbol{w}_i = \lambda_i \boldsymbol{S_W} \boldsymbol{w}_i,$$

where $\boldsymbol{w}_i$ is the ith column of $\boldsymbol{W}^{\mathsf{LDA}}$ (we consider only the p largest eigenvalues and the associated eigenvectors). Once the projection is known, the traces are projected in the subspace and any model can be built from the projected traces. In the particular case of pooled Gaussian templates, the model becomes:

$$\begin{aligned}
\hat{\mathsf{f}}^{\mathsf{LDA}}[\boldsymbol{l}^* | X = x] = &\frac{1}{\sqrt{(2\pi)^n \left|\hat{\Sigma}_{\boldsymbol{W}^{\mathsf{LDA}}}\right|}} \\
&\exp\left(-\frac{1}{2}\left(\boldsymbol{W}^{\mathsf{LDA}^T}(\boldsymbol{l}^* - \hat{\boldsymbol{\mu}}(x))\right)^T \hat{\boldsymbol{\Sigma}}^{-1}_{\boldsymbol{W}^{\mathsf{LDA}}} \left(\boldsymbol{W}^{\mathsf{LDA}^T}(\boldsymbol{l}^* - \hat{\boldsymbol{\mu}}(x))\right)\right),
\end{aligned}$$

with $\hat{\boldsymbol{\Sigma}}_{\boldsymbol{W}^{\mathsf{LDA}}} = |\mathcal{L}|^{-1}\, \boldsymbol{W}^{\mathsf{LDA}^T} \boldsymbol{S_W} \boldsymbol{W}^{\mathsf{LDA}}$. If the number of dimensions after projection p is small (i.e., $p \ll n_s$, with n_s the number of samples

in the raw traces), the both PCA and LDA use only a small part of the spectrum of the trace's covariance (or equivalently scatter) matrix, which converges fairly fast, hence mitigates the "big covariance estimation" problem.

2.5.3 Regression-based LDA

The previous sections outline the respective merits of LR-based attacks and LDA: the efficient profiling of large states (i.e., many bits) for LR-based attacks and the efficient handling of long traces (i.e., many samples) for LDA. Since our goal is to efficiently profile long traces with large states, combining both methods is a natural solution.

In a nutshell, the RLDA model proposed in [CK14] first computes the coefficients $\boldsymbol{a}_s$ of the linear regression for each time sample s in the trace, following the technique presented in Section 2.4. Then, it performs the LDA, but uses the regressed class means $\mathbf{m}(x)$ instead of the directly estimated means $\hat{\boldsymbol{\mu}}(x)$, both in the computation of the scatter matrices $\boldsymbol{S_W}$ and $\boldsymbol{S_B}$, and in the final Gaussian probability formula.

Formally, after computing the regression as in Section 2.4, we define the coefficient matrix $\boldsymbol{A} = (\boldsymbol{a_0} \cdots \boldsymbol{a_{n_s}})^T$, such that $\mathbf{m}(x) = \boldsymbol{A}\boldsymbol{\beta}(x)$. Then, we define the regressed inter- and intra-class scatter matrices $\boldsymbol{S}_{\boldsymbol{B}}^{\mathsf{reg}}$ and $\boldsymbol{S}_{\boldsymbol{W}}^{\mathsf{reg}}$ as:

$$\boldsymbol{S}_{\boldsymbol{B}}^{\mathsf{reg}} = \sum_{x \in \mathcal{X}} |\mathcal{L}(x)| \, (\mathbf{m}(x) - \hat{\boldsymbol{\mu}})(\mathbf{m}(x) - \hat{\boldsymbol{\mu}})^T,$$
$$\boldsymbol{S}_{\boldsymbol{W}}^{\mathsf{reg}} = \sum_{x \in \mathcal{X}} \sum_{\boldsymbol{l} \in \mathcal{L}(x)} (\boldsymbol{l} - \mathbf{m}(x))(\boldsymbol{l} - \mathbf{m}(x))^T.$$

Let us remark that $\boldsymbol{S}_{\boldsymbol{W}}^{\mathsf{reg}}$ can equivalently be defined as the scatter of the residual (which has mean 0): $\boldsymbol{S}_{\boldsymbol{W}}^{\mathsf{reg}} = \boldsymbol{r}^T\boldsymbol{r}$ (see Equation 2.8). The following optimization problem is then solved to obtain the projection matrix $\boldsymbol{W}^{\mathsf{reg}}$ for a chosen dimension parameter p:

$$\boldsymbol{W}^{\mathsf{reg}} = \underset{\boldsymbol{W} \in \mathbb{R}^{n_s \times p}}{\arg\max} \frac{\left|\boldsymbol{W}^T \boldsymbol{S}_{\boldsymbol{B}}^{\mathsf{reg}} \boldsymbol{W}\right|}{\left|\boldsymbol{W}^T \boldsymbol{S}_{\boldsymbol{W}}^{\mathsf{reg}} \boldsymbol{W}\right|}.$$

Finally, the pooled noise covariance matrix can be estimated from the intra-class scatter $\hat{\boldsymbol{\Sigma}}_{\boldsymbol{W}^{\mathsf{reg}}} = |\mathcal{L}|^{-1} \boldsymbol{W}^{\mathsf{reg}T} \boldsymbol{S}_{\boldsymbol{W}}^{\mathsf{reg}} \boldsymbol{W}^{\mathsf{reg}}$, and the model becomes:

$$\hat{\mathsf{f}}^{\mathsf{RLDA}}[\boldsymbol{l}^* | X = x] = 1/\sqrt{(2\pi)^p |\hat{\boldsymbol{\Sigma}}_{\boldsymbol{W}^{\mathsf{reg}}}|}$$
$$\exp\left(-\frac{1}{2}(\boldsymbol{W}^{\mathsf{reg}T}(\boldsymbol{l}^* - \mathbf{m}(x)))^T \hat{\boldsymbol{\Sigma}}_{\boldsymbol{W}^{\mathsf{reg}}}^{-1} (\boldsymbol{W}^{\mathsf{reg}T}(\boldsymbol{l}^* - \mathbf{m}(x)))\right), \quad (2.11)$$

2.6 Soft Analytical Side-channel Attacks

The previous sections showed how we can profile the leakage of an intermediate variable X. However, during the computation of an encryption algorithm, several intermediate states exist which can all leak useful information. Directly profiling many variables in the same template is rapidly impractical as the number of classes grows exponentially. SASCA has been introduced as an efficient way to profile several variables independently and exploit them jointly [VGS14]. For this purpose, it models the relations between the variables with a factor graph and leverages the Belief propagation (BP) algorithm.

Concretely, the factor graph is a bipartite graph containing the variables (circles) on one side and the relations (squares) on the other. The edges represent the relations between the variables. An example is given in Figure 2.1.

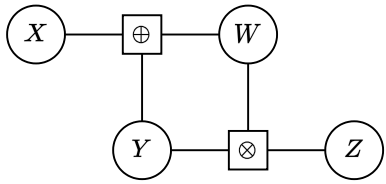


Figure 2.1: Exemplary factor graph for $X \oplus Y = W$ and $Y \otimes Z = W$.

Internally, the BP algorithm iterates over 3 steps. At iteration t, we have:

1. The belief of a variable X is computed as the product of its likelihood and the beliefs coming from its neighbors ∂X.

$$P_X^{t+1}(x) = \hat{\mathsf{p}}(X = x) \prod_{R \in \partial X} m_{R \to X}^t(x),$$

2. From variable X to a relation R, the belief is the product of the beliefs on the variable divided by the belief from the destination relation.

$$m_{X \to R}^{t+1}(x) = P_X^t / m_{R \to X}^t(x).$$

3. From relation R to variable X, the belief is the sum over all compatible values for the relation of the products of the beliefs from all neighbors of R except X, where we denote the compatibility function with ψ_R, which has a value of 1 if the values are compatible

with the relation and 0 otherwise:

$$m^{t+1}_{R\to X}(x) = \sum_{x_i\in\mathcal{X}_i \text{ s.t. } \{X_1,\ldots,X_k\}=\partial R\setminus X} \psi_R(x, x_1, \ldots, x_k) \prod_{i=1}^{k} m^{t}_{X\to R}(x_i),$$

This algorithm is proven to converge on a tree-like structure with a number of iterations corresponding to twice the diameter of the graph. With graphs containing cycles like Figure 2.1, the algorithm becomes a heuristic.

2.7 Information Theoretic metrics

Evaluating the success rate of a multivariate attack cannot be done with the SNR (which is a univariate metric). The Mutual information (MI) between the leakage and target variable is a good candidate for this purpose. However, it is notoriously hard to estimate. We therefore use the Perceived information (PI) as a surrogate, which provides an easy to estimate lower bound [BHM+19]. It can be estimated by sampling the model $\hat{\mathsf{f}}$ as follows:

$$\hat{\mathsf{PI}}(X, \boldsymbol{L}) = \mathrm{H}(X) + \sum_{x\in\mathcal{X}} \mathsf{Pr}[x] \sum_{\boldsymbol{l}'\in\mathcal{L}'(x)} \frac{1}{|\mathcal{L}'(x)|} \log_2 \hat{\mathsf{p}}[x|\boldsymbol{l}'], \tag{2.12}$$

where $\mathrm{H}(X)$ is the entropy of X and $\mathcal{L}'$ is the set of traces used to estimate the PI, which must differ from the set of traces $\mathcal{L}$ used to fit the model.

2.8 Countermeasures

In this section, the 2 countermeasures that are studied in this thesis are presented, namely masking and shuffling. Both are provided with the general idea, followed by a detailed explanation and impact on SCA.

2.8.1 Masking

Masking is a common countermeasure against side-channel attacks. It relies on an encoding of any sensitive variable x into a tuple of n shares:

$$x = x_1 \oplus x_2 \oplus \ldots \oplus x_n,$$

where the $n-1$ first shares are selected uniformly at random, so that an adversary needs knowledge of all the shares in order to recover the sensitive information. This security guarantee is easily expressed in the

abstract probing model [ISW03], which states security if an adversary who can probe up to $n-1$ wires in the circuit does not learn anything about x. In this thesis we work with Boolean masking only.

From an implementation viewpoint, linear operations are performed share-by-share and have a linear complexity in the number of shares. Multiplications of two encodings are more complex: they have quadratic complexity in the number of shares and require additional randomness to remain d-probing secure. Popular algorithms are the ISW multiplication [ISW03], that is represented in Algorithm 1 and 2 and the Probe isolating non-interference (PINI) gadgets introduced in [CS20].

Algorithm 1 Masked addition.

Input: Shares a_i and b_i such that $a = \bigoplus_{i=1}^{n} a_i$ and $b = \bigoplus_{i=1}^{n} b_i$.
Output: Shares c_i such that $c = \bigoplus_{i=1}^{n} c_i$ and $c = a \oplus b$.

1: **for** i in $\{1, 2, ..., n\}$ **do**
2: $\quad c_i \leftarrow a_i \oplus b_i$

Algorithm 2 Masked ISW Multiplication.

Input: Shares a_i and b_i such that $a = \bigoplus_{i=1}^{n} a_i$ and $b = \bigoplus_{i=1}^{n} b_i$. Randomness $\mathcal{R}$.
Output: Shares c_i such that $c = \bigoplus_{i=1}^{n} c_i$ and $c = a \otimes b$.

1: **for** i in $\{1, 2, ..., n\}$ **do**
2: $\quad c_i \leftarrow a_i \otimes b_i$
3: **for** i in $\{1, 2, ..., n\}$ **do**
4: $\quad$ **for** j in $\{i+1, 2, ..., n\}$ **do**
5: $\quad\quad r \xleftarrow{\mathcal{R}} \{0, 1\}$
6: $\quad\quad c_i \leftarrow c_i \oplus r \oplus (a_i \otimes b_j)$
7: $\quad\quad c_j \leftarrow c_j \oplus r \oplus (a_j \otimes b_i)$

Probing security reduces to noisy leakage security [DDF14], which is closer to real-world measurements. The noisy leakage model assumes that all the shares leak noisy observations to the adversary. In this case, the data complexity N of the attack is inversely proportional to the product of the MI between each share and the leakage [BCG+23], so that:

$$N \geq \frac{c}{\prod_{i=1}^{n} \mathrm{MI}(X_i, L)}, \tag{2.13}$$

increases exponentially with the number of shares given that the MI per share is small enough. As mentioned in the introduction, this guarantee

only holds as long as the leakage function ensures independence between the shares' leakages (i.e., as long as it does not recombine the shares). When replacing the MI by the PI in this equation, we no longer have a lower bound but an estimate that measures the particular model used to estimate the PI.

Masked software implementations have the specificity that shares are manipulated serially, allowing the adversary to profile the leakage models $\hat{\mathsf{f}}$ for each share independently. Attacking a masked implementation is done by obtaining the likelihoods on the shares $\hat{\mathsf{p}}[x_i|\boldsymbol{l}]$, and then to recombine them to obtain likelihoods on the target variable:

$$\hat{\mathsf{p}}[x|\boldsymbol{l}] \propto \sum_{\{x_0,\ldots,x_{n-1}|\sum x_i=x\}\in\mathcal{X}^n} \prod_{i=1}^{n} \hat{\mathsf{p}}[x_i|\boldsymbol{l}] \tag{2.14}$$

2.8.2 Shuffling

Shuffling is another popular side-channel countermeasure. While masking randomizes the manipulated data, shuffling randomizes the execution flow. Namely, when an algorithm is composed of independent operations, these can be executed in a random order while maintaining correctness. One typical application of shuffling is an Sbox layer applied to an input vector $\boldsymbol{x}$ of size $|\boldsymbol{x}|$ (e.g., 16 for the AES). Next we detail such an application of shuffling thanks to Algorithm 3. There, the first step is to generate a random uniform permutation $\boldsymbol{\pi}$ over of the set $\{0, 1, \ldots, |\mathbf{x}| - 1\}$. This is done with gen_perm$(\cdot, \cdot)$ that takes as input the permutation size together with randomness $\mathcal{R}$.[1] The second step is to iterate over all the indexes i with $0 \leq i < |\boldsymbol{x}|$. At every iteration, one value $j = \pi_i$ is fetched from the permutation $\boldsymbol{\pi}$. The Sbox is then applied to j-th entry of the input vector and stored at the corresponding index of the output vector.

Algorithm 3 Shuffled Sbox layer.

Input: $\boldsymbol{x}$ and randomness $\mathcal{R}$.
Output: $\boldsymbol{y}$ such that $\forall i, 0 \leq i < |\boldsymbol{x}|, y_i = \text{Sbox}(x_i)$

1: $\boldsymbol{\pi} \leftarrow$ gen_perm$(\mathcal{R}, |\text{x}|)$
2: **for** i in $\{0, 1, ..., |\mathbf{x}| - 1\}$ **do**
3: $\quad j \leftarrow \pi_i$
4: $\quad y_j \leftarrow \text{Sbox}(x_j)$

[1] In this work, we assume that gen_perm$(\cdot, \cdot)$ is pre-computed and the permutation is stored in memory. It can also be generated on-the-fly if needed.

Informally, in order to perform a side-channel attack against a shuffled implementation, an adversary has to map the leakage of x_j to the correct x_i for every iteration. If the permutation is known by the adversary, such an implementation is equivalent to an unprotected one. If it is not, the adversary is forced to perform a so called "integration" attack which sums together every $\boldsymbol{l}_{x_j}$ (resp., $\boldsymbol{l}_{y_j}$) [VMKS12]. This has the effect of turning the leakage of a serial implementation (e.g., software) into the leakage of a parallel one (e.g., hardware) where the adversary has only access to the sum of all leakages leading to an attack complexity multiplied by the size of the permutation.

In practical case studies, the permutation generally leaks partially to the adversary through $\boldsymbol{l}_{\pi_i}$. These leakages can be due to the generation of the permutation π itself, to its storage and loading from memory and to the addresses used to load x_j and to store y_j.

Part I

Evaluation tools

Evaluating measurement setups

3

The design of a measurement setup is the first step in the evaluation of a cryptographic implementation against side-channel analysis. Due to its physical nature, this step inherently carries hard to quantify risks of security overstatements. Noisy setups may indeed lead evaluators to conclude that the measurements are less informative than they actually are, and this gap will then be increased in case a countermeasure aiming at noise amplification, like masking [CJRR99, GP99] or shuffling [HOM06, VMKS12], is implemented. Surprisingly, and despite papers focused on practical side-channel attacks usually describe how they optimized their setups, especially when targeting challenging real-world devices [MBKP11, BGRV15], very few works are dedicated to the systematic evaluation of measurement setups and the impact of their optimization on security evaluations. Besides, and to the best of our knowledge, the most advanced (published) investigations of this topic were performed in specific settings such as the exploitation of static leakages, as recently investigated by Moos et al. [MMR20], or the evaluation of physical effects such as couplings to reduce a masked implementation's security order [CBG+17, CEM18, LBS19]. But when it comes to the impact of measurement setups on the noise level in the context of (standard) attacks exploiting the dynamic part of the leakage, the only works we are aware of are the one of Guilley et al. which puts forward the SNR as a meaningful metric to quantify the quality of side-channel acquisitions [GMS+11], and the one of Merino del Pozo and Standaert that discusses the impact of different setups in the context of leakage detection [PS17].

In this respect, and despite these references are important first steps in specifying relevant comparison metrics and highlighting the existence of an interesting design space, they are still have a limited scope: [GMS+11] estimates its proposed (univariate) metric for a single measurement setup while [PS17] compares different analog amplifiers and filters for a single probing method.

Given that the design space of measurement setups is broader than the parameters studied in the previously cited papers, we study in this chapter four important parameters of 2 measurement setups. Even though this thesis focuses on MCUs, a hardware FPGA target and a software ARM Cortex one were selected as most MCUs propose cryptographic coprocessors like the one used in Chapter 5 and Chapter 8. These coprocessors are hardware implementations and the analysis of this chapter can be applied to them directly. We additionally evaluate the effect of the different parameters for both univariate evaluation metrics like the SNR and multivariate metrics like the PI.

While most of the observations presented in this chapter are admittedly present (implicitly or explicitly) in former experimental works, we provide a useful consolidating effort with their compilation and a quantitative analysis of the losses (which may reach orders of magnitude) a poor measurement setup may imply.

3.1 Setup model & design space

We now introduce our model and design space for measurement setups, alongside with the two devices we have adopted to conduct our investigations.

3.1.1 Setup model

The setup model is illustrated in Figure 3.1. Its goal is to highlight important parameters for the informativeness of the leakages such as the probing method, the Device under test (DUT)'s parameters and the Digital storage oscilloscope (DSO)'s parameters. As reported in [fS19], the choice of those components and how they interact with each other impact sensibly on the final outcome of the practical side-channel security evaluation of a leaking implementation.

A bit more precisely, the current absorbed by the DUT is first monitored by a probe, which has the role to convert the current signal into a voltage signal. This signal can then be amplified using a preamplifier stage in order to increase its magnitude, to mitigate noise in the measurements and to improve electrical characteristics for the following blocks. At the end of the so-called measurement chain, a DSO samples and quantizes the analog voltage signal, converting it in a digital representation. Usually, the sampling operation is handled following a specific timing, that exploits a trigger signal in order to synchronize dif-

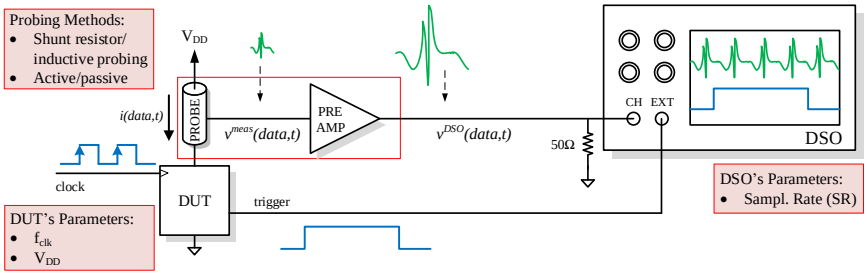


Figure 3.1: Measurement setup model for power analysis evaluation.

ferent measurements.[1] The precise design space that we will consider for each block of the model will be detailed later.

We note that our investigations do not consider the question of filtering, which we view as an orthogonal one, since it can be performed after the measurements took place in order to compensate a too noisy setup.

3.1.2 Platforms

In our investigations, we have used two devices in order to cover both hardware and software implementations of cryptographic algorithms. This choice is motivated by the expected differences between the two types of targets. For example, hardware implementations generally allow better controlling the design aspects (from the level of parallelism to low-level implementation choices) while software implementations are usually more general purpose and serial.

Hardware DUT. Our target hardware DUT is a Xilinx Spartan-6 LX75 FPGA, mounted on a Sakura-G board, implementing an AES-128 processor with a 32-bit architecture. It is illustrated in Figure 3.2. In order to provide synchronization between measurements, we generate a trigger signal on one of the IO pins of the FPGA, rising to logical '1' one cycle before the start of the encryption and setting it back to '0' one cycle after the end of the AES encryption.

Software DUT. Our target software implementation is running on a Cortex-M0 MCU from the STM32F0308 Discovery board. Small modifications were performed on the board. Namely, we added a crystal

[1] The availability of a good trigger may raise additional challenges [BBGV16].

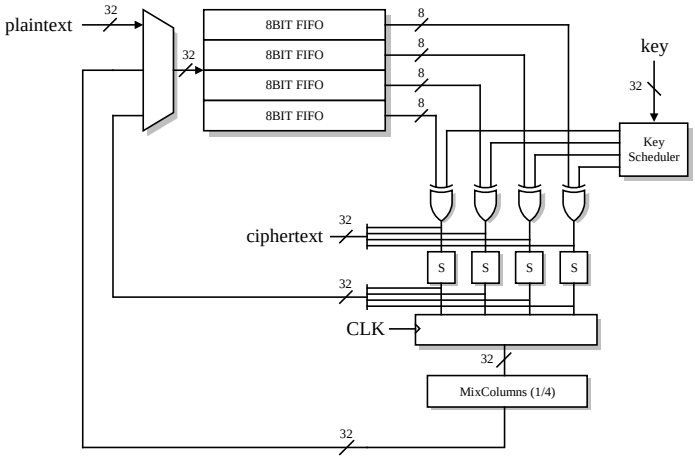


Figure 3.2: Architecture of the 32-bit AES encryption coprocessor.

oscillator to provide a stable clock source for the measurements and decoupling capacitors were desoldered. The MCU is running tiny-AES [2], an open source AES-128 implementation. We used the same trigger methodology as for the hardware DUT.

3.1.3 Design space

We explored our design space and DUTs by testing the following parameters:

First regarding the probing methodology, we used both a 2Ω resistor in series with the power supply voltage and an inductive probe (the Tektronix CT-1, which gives a transresistance of 5mV/mA in the frequency range 25kHz–1GHz [3]). We optionally used a preamplifier, namely a Rohde&Schwarz HZ16 [4] providing a gain of 20dB with a noise figure of 4.5dB, in the frequency range 100kHz–3GHz.

Next, the DUT's clock frequency is an important macroscopic feature of a side-channel trace, since it usually reflects the frequency spectrum where leakage can be found. We chose three clock frequency values (1MHz, 6MHz and 24MHz) for the hardware DUT and three clock fre-

[2] `https://github.com/kokke/tiny-AES-c`

[3] Tektronix AC Current Probes - CT1, CT2, CT6 Data Sheet: `https://download.tek.com/manual/070795702web.pdf`

[4] `Rohde&SchwarzHZ15,HZ16,HZ17datasheet:https://scdn.rohde-schwarz.com/ur/pws/dl_downloads/dl_common_library/dl_brochures_and_datasheets/pdf_1/service_support_30/HZ-15_16_17_bro_en_5213-6687-12_v0100.pdf`

quencies (4MHz, 24MHz and 48MHz) for the software DUT. Those sets of values were chosen to observe the impact of the clock frequency on the shape and distinguishabilty of the leakage cycles.

Similarly, we chose three power supply voltage (0.8V, 1.2V and 1.4V) for the hardware DUT and three power supplies (2.6V, 3.0V and 3.6V) for the software DUT. Those sets of values were chosen in order to observe the impact of working at nominal supply voltage vs. in minimum and maximum corner cases.

Finally, we used a Picoscope 6424E providing a vertical resolution of 12 bits and running at three different sampling rates as DSO. We chose sampling rates values according to the clock frequency of the given DUT, in order to analyze the impact of the collected number of samples per clock cycles (which impacts the acquisition bandwidth and memory requirements). Precisely, we set the sampling rate of the DSO at $\times 1$, $\times 5$, $\times 25$ the chosen DUT's clock frequency.

In total, we performed $4 \times 3 \times 3 \times 3 = 108$ experiments on each DUT. In each experiment, we targeted the first key byte of the first AES round and collected 4×10^6 traces for the hardware DUT and 10^5 for the software one. Both the input plaintexts and keys have been picked up uniformly at random, in order to stimulate the combinational and sequential logics of both platforms.

3.2 Experimental results and discussion

In this section, we present the results of our analyses for the proposed metrics throughout our design space. We first introduce the set of plots (e.g., for the SNR and PI) that summarize our experiments and will be the basis of our discussions. We then extract the best configurations for the measurement setup of both platforms. We finally propose general guidelines for the design of good measurement setups. Given the granularity of the explored design space, we organize this discussion according to the setup model in Section 3.1. We also evaluate the relevance of univariate evaluation metrics as predictors of multivariate ones.

Figure 3.3 shows the highest SNR value we found for each set of parameters for both platforms (in logarithmic scale). We present the results in the form of a matrix where the X-axis contains the different power supply values and sampling speeds, and the Y-axis contains the DUT clock frequencies and the probing method used in each experiment. The thick orange lines delimit the probing methods on the X-axis and the sampling speeds on the Y-axis. Darker blue blocks represent setup parameters where the SNR is higher. Figure 3.4 shows a similar matrix

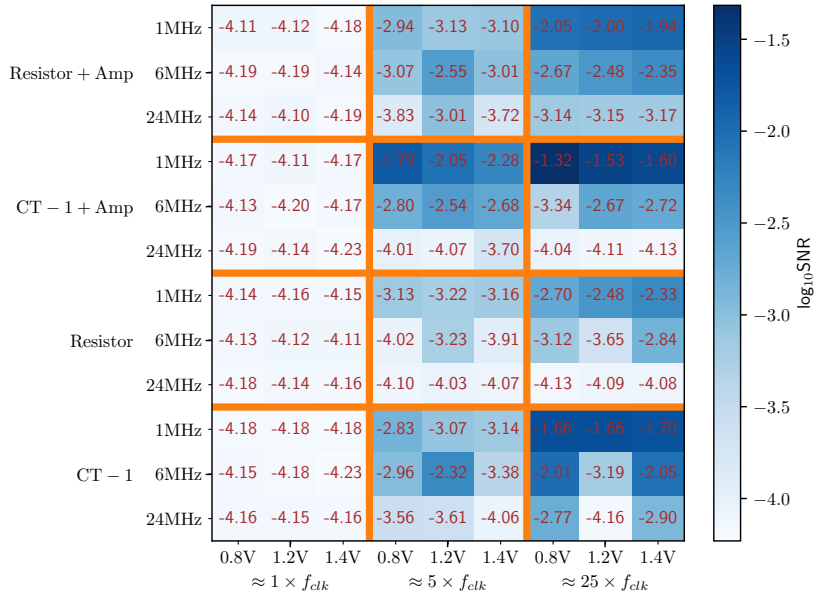
for the PI values obtained after LDA, in order to evaluate the impact of setup choices from a multivariate attack perspective.

A bit more in detail, the SNRs in Figure 3.3 were calculated on the whole leakage trace and the maximum value was then taken. In the hardware case, as shown in Figure 3.3a, the best SNR is obtained using the CT-1 current probe combined with the amplifier, setting the DUT to the slowest clock speed and lowest power supply value, and sampling at the highest rate. In the software case, as shown in Figure 3.3b, the differences are more subtle and many sets of parameters give a peak SNR value close to the best one. The latter is obtained using the resistor combined with the amplifier, setting the DUT to the highest clock speed and sampling at lowest rate (contrary to the hardware case) while still using the lowest power supply value (like in the hardware case).

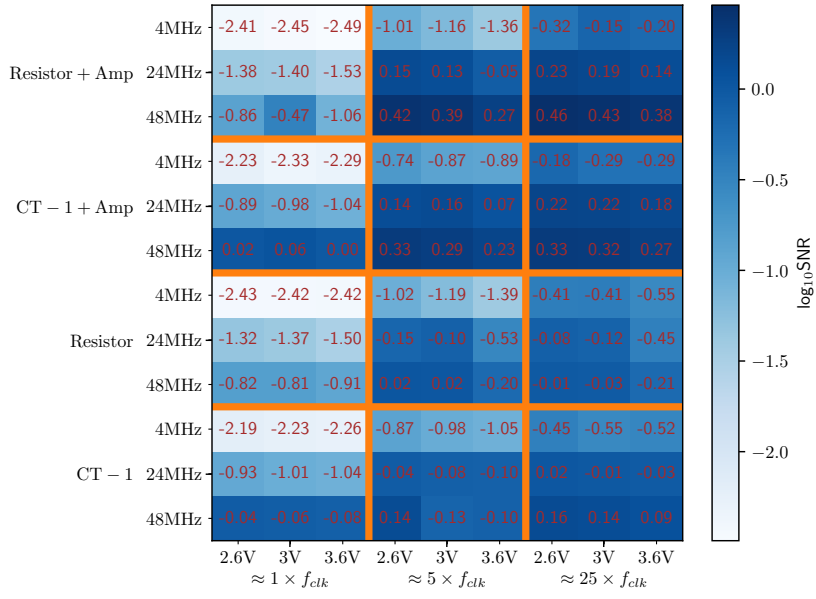
Regarding our multivariate analysis, we calculated the PI for each set of parameters. Concretely, for each experiment independently, we first pre-selected samples based on the SNR traces, keeping the ones above the noise floor for profiling. We then built 8-bit Gaussian templates combined with LDA as presented in Section 2.5.2. We next analyzed the impact of the subspace dimensionality p, trying $p = 1$ up to 25 for the hardware platform and 50 for the software one. We finally kept the p leading to the highest PI, which is reported in Figure 3.4.

For the hardware platform's results reported in Figure 3.4a, we observe that the experiment leading to the highest PI is obtained with the same set of parameters that leads to the highest SNR in Figure 3.3a. Generally speaking, comparing the two metrics, the PI follows the same trend as the SNR in this case.[5] For the software platform's results reported in Figure 3.4b, we see that the effect of the probing method and the power supply voltage are negligible, which is different from the univariate SNR analysis of Figure 3.3b. By contrast, observations regarding the clock frequency and sampling rate remain similar as in the univariate case. We also note that our highest PI value is 7.96 for an 8-bit bus. The difference between to software and hardware platforms can at least partially be attributed to the implementations. In hardware, the implementation has a 32-bit architecture meaning only 8-bits out of 32 are modeled, leading to 24-bits of algorithmic noise. In software, the implementation is table based and therefore performs the Sboxes

[5]The experiments where the $\log_{10}$ PI is -inf correspond to a negative PI, indicating that no information could be extracted from the estimated model.

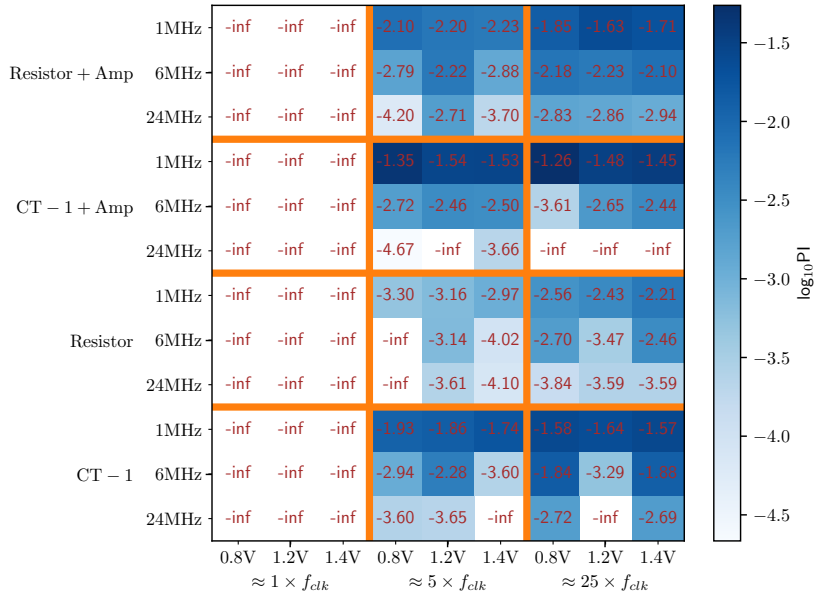


(a) Hardware DUT.

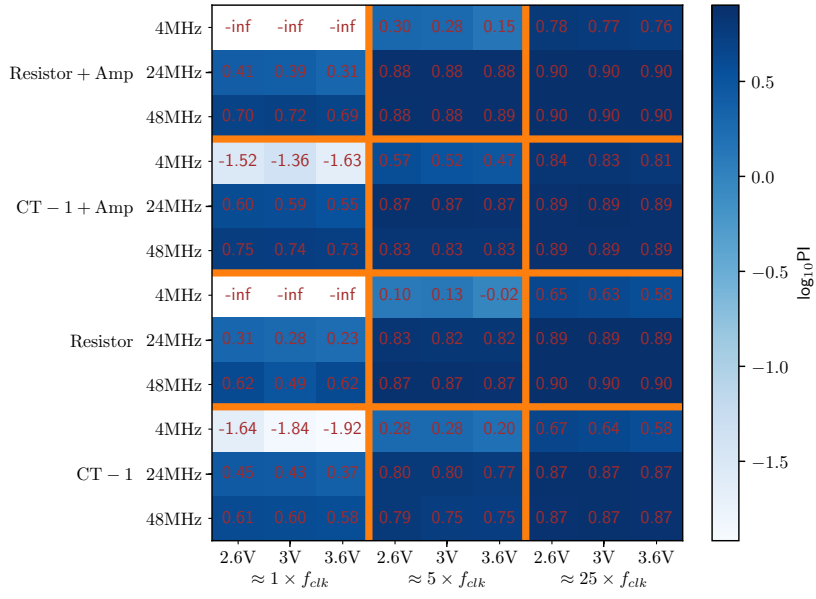


(b) Software DUT.

Figure 3.3: Peak SNR values observed for the hardware (a) and software (b) DUTs.



(a) Hardware DUT.



(b) Software DUT.

Figure 3.4: Peak PI values observed for the hardware (a) and software (b) DUTs.

one-by-one, leading to no algorithmic noise[6]. Moreover, in software, the leakage model is combined from many separate instructions.

As a complement, Figure 3.5 depicts exemplary SNR and leakage traces for both platforms, corresponding to the best cases in Figure 3.3. The SNR traces are in the upper subplots and mean leakage traces in the lower subplots.

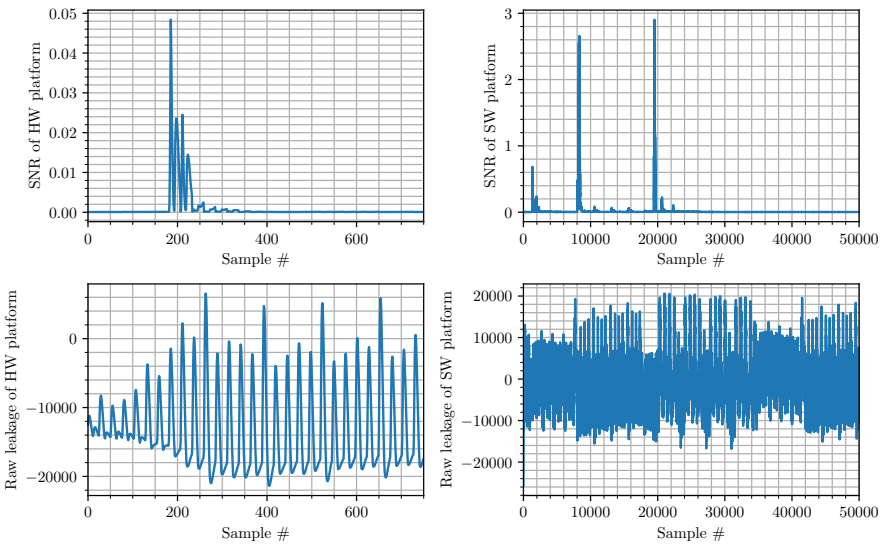


Figure 3.5: Exemplary SNR and leakage traces: hardware (left) and software (right).

These experimental investigations lead to the following observations:

Probe. The choice of a probe was more critical for the hardware platform than the software one in our experiments. In the hardware case, the inductive probe gave better results than the resistor. A plausible explanation is that the CT-1 interferes less with the side-channel signal and is intrinsically less noisy than a shunt resistor. So as long as the target leakage is covered by the probe's bandwidth, it seems to be a good choice. In the software case, both the inductive probe and the resistor gave good results, presumably due to the easier-to-exploit measurements (reflected by the higher SNR and PI values). As for the use of the amplifier, it does not show a significant impact in our experiments.

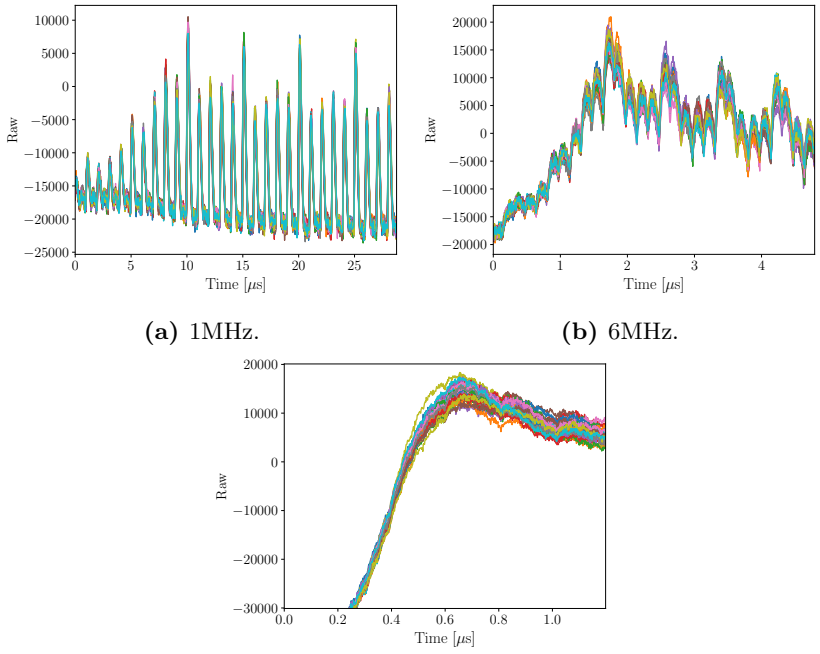
We posit that the observation regarding the inductive (CT-1) probe could change if targeting higher clock frequencies, and the observation

[6]We consider only algorithmic noise from the CPU, some other peripherals might be active though.

regarding the amplifier could change if targeting more advanced technologies or a side-channel signal with lower amplitude (e.g., an electromagnetic one).

Clock frequency. This parameter is in general important for side-channel analysis. Whenever it can be controlled by the adversary, both our hardware and software results suggest the same rule-of-thumb: "*use the highest available clock frequency such that independent clock cycles are easy to distinguish*".

We first illustrate this rule-of-thumb with Figure 3.6. It shows the traces we recorded with the best parameter set and varying clock frequencies for the FPGA platform. At 1MHz, the independent peaks for each clock cycle are clearly distinguishable. At higher clock frequencies, the leakage traces are smoother and the overlapping between the clock cycles in the measurements increases.



(a) 1MHz. **(b)** 6MHz.

(c) 24MHz.

Figure 3.6: Clock frequency effect on the hardware setup.

We next turn to the software case study to explain the first part of the rule-of-thumb (i.e., why it is not advisable to reduce the clock frequency unconditionally). In this respect, we first note that for this software DUT, the clock cycles were clearly distinguishable even for the maximum clock frequency (so the second part of the rule-of-thumb was

fulfilled). In this case, the best SNR and PI values are observed for higher clock frequencies. We explain this effect by observing that all the samples in a clock cycle are not equally informative. During an MCU clock cycle, most of the dynamic power is contained right after the rising edge of the clock as the effect of the registers changing state. The leakage from the remaining of the clock cycle is mostly due to static power and is usually less informative [PSKM15, Mor14]. Therefore, the interest of decreasing the clock frequency can become detrimental while keeping the number of samples per clock cycle fixed.

More precisely, and as illustrated in Figure 3.7, decreasing the clock frequency[7] can lead the collected samples (represented by red diamonds in the figure) to correspond mostly to the static part of the leakage, and to miss the information of the dynamic part (represented by the green rectangles of the figures). Overall, this can lead to a collection of samples that is less informative: the univariate SNR can be lower by missing the most informative sample and the multivariate PI can be lower by cumulatively covering less relevant samples.

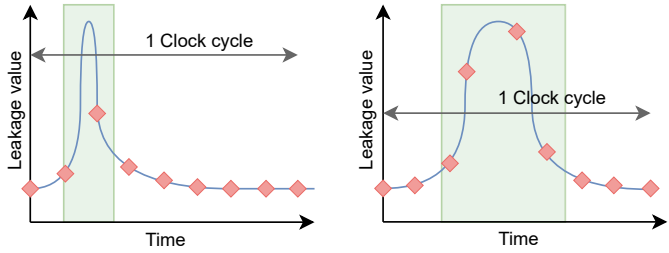


Figure 3.7: Sampling with different DUT clock frequencies.

V_{DD}. Despite being less definitive than the clock frequency, the supply voltage also affects the shape of the leakage traces, as it increases the critical path and therefore spreads the information towards more samples. This naturally causes the multivariate PI to be improved when lowering the supply voltage below the nominal one. Interestingly, we also observed that for both targets and most sets of other parameters, decreasing V_{DD} is also beneficial to the univariate SNR.

A plausible reason for this observation is that both devices are based on CMOS technology (even though from different technology nodes and manufacturers) which generally exhibits smoother transient current when V_{DD} is lower than nominal, due to reduced transconductance of

[7] And while maintaining the same number of samples per clock cycle.

digital cells. This can reduce both the signal and, here more dominantly, the noise of the leakage.

We note that this observation is admittedly technology-dependent: see [VSN15] for a report on several technology nodes. It is also not unconditional: as reported in the same paper, the output noise of a digital cell in subthreshold regime (that corresponds to extremely low V_{DD} values) is not minimal, as transistors exhibit higher resistance and thus contribute more to increase the noise level. So overall, our conclusion regarding the V_{DD} parameter is that reducing it below the nominal value can have marginal interest, especially for multivariate attacks, but is not expected to lead to significant gain/loss factors.

Sampling rate. This parameter is especially critical for the cost of the attacks as it affects the memory requirements to store the leakage measurements.

On the one hand, its selection is related to the clock frequency: as in general when quantizing signals, the sampling rate should at least be chosen larger than the Nyquist frequency. This requirement was confirmed in our experiments, and showed to be more critical (resp., less critical) in the hardware case (resp., software case). This is presumably due to the lower (resp., larger) amount of less (resp., more) informative samples of the hardware (resp., software) case.

On the other hand, in the context of side-channel analysis, a natural question is whether increasing the sampling rate significantly beyond the Nyquist frequency can be useful. Namely, can it lead to more powerful multivariate attacks? By testing a sampling frequency of $\times 1$, $\times 5$ and $\times 25$ the clock frequency, we observed that collecting more samples helps only to a limited extent. In particular, both for the software and the hardware platforms, the gains when moving from $\times 1$ to $\times 5$ the clock frequency are more significant than when moving from $\times 5$ to $\times 25$ the clock frequency, again with more incentive to increase the sampling rate in the hardware case than in the software case. A plausible reason for this difference is once more the more condensed and noisy nature of the hardware leakage (i.e., the fact that it is concentrated in less cycles with more algorithmic noise, rather than spread over more cycles in software).

Univariate vs multivariate evaluations. Eventually, our results indicate that whether the SNR is a good predictor of the PI is quite case-dependent.

If the SNR traces present a single peak or a set of peaks that are close to each other (e.g., within one cycle), they usually indicate correlated leakage coming from a single operation. In this case, which typically cor-

responds to our software experiments (see the right part of Figure 3.5), a good univariate SNR will generally be a good indicator of a good multivariate PI. Multivariate attacks will always be more powerful but the SNR can serve as a first-order comparison metric.

By contrast, if the SNR traces contain multiple peaks separated by several clock cycles, they rather indicate independent leakage coming from different operations. In this case, which typically corresponds to our hardware experiments (see the left part of Figure 3.5), multivariate attacks are expected to be significantly more powerful than univariate ones. So the direct estimation of the multivariate PI is in general a better (i.e., more conclusive) evaluation strategy.

A general methodology would be to always calculate the SNR to assess the leakages. Then, if necessary, estimate the PI when multivariate metrics are necessary.

3.3 Conclusions

In this chapter, we aimed at evaluating the risk of over-estimating the physical security of an implementation due to inadequate parameters' choices when configuring a measurement setup. Our findings show that the losses due to a bad selection of parameters can be significant and lead to strong over-estimations of an implementation's security level. We also use our experiments to highlight the necessity of a multivariate PI metric in the case of a software measurements with leakages from multiple operations. Evaluating the PI of a leakage model requires more computational power than the SNR and it might be hard for some adversaries. In the next chapter, we show efficient algorithms to solve this issue.

Efficient profiling and evaluation of large states and long traces

4

SPA has recently gained interest due to the rise of single-trace horizontal side-channel attacks. Among such works, we can cite Kannwischer et al. with their impact on post-quantum cryptographic algorithms [KPP20a] and Bellizia et al. highlighting their importance for leakage-resistant modes of operation [BBC+20]: on the one hand, they are the main attack vector to break the confidentiality of schemes leveraging an internal re-keying, like Ascon [DEMS21], Spook [BBB+20] or TEDT [BGP+20][1], on the other hand, schemes like ISAP even reduce their long term security (and integrity) to the assumption that such attacks are hard [DEM+20].

While in a DPA setting, an attacker is able to exploit leakages containing algorithmic noise, with SPA, the algorithmic noise biases the measurements and the extracted information is reduced. Therefore, single-trace attacks are optimal when the leakage model corresponds to the bus size.

In that sense, profiling an 8-bit implementation is generally easy, making SASCA readily applicable in this context [VGS14]. By contrast, in the case of large parallel implementations, i.e. 64-bit and above, profiling the whole bus is impossible. In-between these two extremes, the situation is currently more fuzzy. For example, Kannwischer et al. only show simulated attacks against 32-bit implementations under a Hamming weight assumption (which reduces the cardinality of the leakages to consider from 2^{32} to 33), and Bellizia et al. could only target a 16-bit implementation of Keccak. This leads to an uncomfortable situation given the importance of 32-bit embedded microcontrollers in the benchmarking of side-channel resistant lightweight and post-quantum cryptography.[2]

[1] Assuming that the initialization and finalization are strongly protected (e.g., with masking)

[2] `https://csrc.nist.gov/Projects/lightweight-cryptography`, `https://csrc.nist.gov/projects/post-quantum-cryptography`.

The profiling difficulty lies in the combination of dealing with large intermediate values and long leakage traces that makes the direct application of Gaussian template attacks from Section 2.3 hardly realistic [CRR02]. More precisely, when taken separately, these two issues have good solutions. Profiling large target intermediate values is known to be efficiently performed thanks to LR as in Section 2.4. And the problem of estimating the covariance of long leakage traces can be mitigated thanks to dimensionality reduction, e.g., with LDA as in Subsection 2.5.2. The only attempt to combine these two techniques observed that LR and LDA form an effective combination (significantly better than, e.g., LR and PCA), but concluded that estimating 32-bit templates was not practical [CK14].

An attempt at targeting 32-bit implementations are Fragment template (FT) attacks [YK21], which performs independent modeling of (e.g., 8-bit) fragments of target intermediate variables. While their attack was successful, this approach is conceptually unsatisfying, since it implies that a part of the signal is considered as algorithmic noise, which is suboptimal from the information extraction viewpoint.

Based on this state of the art, this chapter presents the optimization of Regression-based Linear Discriminant Analysis, so that 32-bit target intermediate values can be efficiently profiled. First, we detail the optimized technique and how to implement it efficiently. Second, we show how to use RLDA in order to rapidly bound the information theoretic metrics of an implementation.

The baseline RLDA model presented in Subsection 2.5.3, gives a conceptually powerful model which can be trained with relatively few traces. However, RLDA becomes computationally expensive when the number of bits in the state grows. Indeed, the baseline has a complexity $\Theta\left(2^b n_s^2\right)$ for profiling, and $\Theta\left(2^b(b+p)n_s\right)$ for computing the distribution of a variable from one leakage trace.[3] In the following, we mitigate these performance bottlenecks by introducing improved solutions for efficient RLDA. We first describe a profiling algorithm with polynomial complexity, namely $\Theta\left(n_s^2(b+n_p)\right)$ computation and $\Theta\left(n_s^2\right)$ memory, with the additional advantage of being incremental: it can accumulate traces during their acquisition using a constant amount of memory. We next provide an optimized method for computing a proba-

[3] The optimization of [CK14] reduces the attack complexity to $\Theta\left(p(n_s+2^b)\right)$ at the cost of $\Theta\left(2^b p\right)$ memory, which (even with enough memory) is not efficient due to memory bandwidth saturation.

bility distribution from the model in $\Theta\left((n_s + 2^b)p\right)$ computations (i.e., we gain a factor n_s).

Related works. As an alternative to "classical" profiled attacks based on templates, linear regression and dimensionality reduction (see references in the above state of the art), a large amount of recent works investigate the applicability of machine learning and deep learning to the context of profiled side-channel analysis. We cite [HZ12, MPP16] for early results in this direction and [CDP17, MDP20] for more recent ones. Like classical template attacks, these works are hardly applicable to large target intermediate values, due to profiling complexity reasons. One notable exception is the neural estimation of the mutual information proposed in [CLM20]. However, contrary to our work, the latter can only be used to estimate the security level of an implementation, not to mount an attack (in that sense it is only similar to our PI bound of Section 4.3). We also note the recent [KDB+22] which performs analytical attacks against implementations of stream cipher based on a Hamming weight assumption (to reduce profiling complexity).

4.1 Background

Fragment Template Attacks. As a solution to the challenges in running a RLDA on a 32-bit state [CK14], You and Kuhn introduced FT attacks [YK21]. Their main idea is to split the variable of interest into 2 (or more) smaller fragments which can be profiled thanks to RLDA and treat the rest of the state as (algorithmic) noise. In the attack phase, the information extracted from each fragment is then recombined. In their paper, they analyzed the leakage of 32-bit words thanks to four 8-bit fragments, and exploited this model in a SASCA against Keccak. Their SASCA is based on single-bit variables, therefore they additionally marginalize the 8-bit probabilities from each fragment to single-bit probabilities.

4.2 Efficient RLDA implementation

We next introduce our two steps to improve the efficiency of the RLDA. Precisely, we first highlight efficient and incremental formulas in order to compute the regression and the scatter matrices, and we discuss how to compute the projection. We next propose to slightly modify the projection in order to simplify the probability computations.

These efficient profiling computations are summarized in Algorithm 4.

Algorithm 4 Efficient RLDA profiling.

Input: Parameters n_s, b, n_p and p.
Input: Batches of profiling traces and associated intermediate values $(\boldsymbol{L}_1, \boldsymbol{x}_1), (\boldsymbol{L}_2, \boldsymbol{x}_2) \ldots$.
Output: $\boldsymbol{W}^{\mathsf{eff}}$ and $\boldsymbol{A}^{\mathsf{eff}}$ matrices for use in Equation 4.2.

1: $n_b \leftarrow b + 1$
2: $\boldsymbol{B} \leftarrow \boldsymbol{0}^{n_b \times n_b}$; $\boldsymbol{C} \leftarrow \boldsymbol{0}^{n_s \times n_b}$; $\boldsymbol{S_L} \leftarrow \boldsymbol{0}^{n_s \times n_s}$
3: **for** each acquired batch $(\boldsymbol{L}_i, \boldsymbol{x}_i)$ **do**
4: $\quad \boldsymbol{B} \leftarrow \boldsymbol{B} + \boldsymbol{\beta}(\boldsymbol{x}_i)\boldsymbol{\beta}(\boldsymbol{x}_i)^T$
5: $\quad \boldsymbol{C} \leftarrow \boldsymbol{C} + \boldsymbol{L}_i\boldsymbol{\beta}(\boldsymbol{x}_i)^T$
6: $\quad \boldsymbol{S_L} \leftarrow \boldsymbol{S_L} + \boldsymbol{L}_i\boldsymbol{L}_i^T$
7: Solve the system $\boldsymbol{C} = \boldsymbol{A} \cdot \boldsymbol{B}$ for $\boldsymbol{A} \in \mathbb{R}^{n_s \times n_b}$.
8: $\hat{\boldsymbol{\mu}} \leftarrow \boldsymbol{C}_{*,1}/n_p$ ($\boldsymbol{C}_{*,1}$ is the first column of $\boldsymbol{C}$).
9: Compute $\boldsymbol{S}_{\boldsymbol{B}}^{\mathsf{reg}}$ and $\boldsymbol{S}_{\boldsymbol{W}}^{\mathsf{reg}}$ using Proposition 1 and Proposition 2.
10: Find the p largest eigenvalues and associated eigenvectors $\boldsymbol{W}^{\mathsf{reg}}$ that solve Equation 4.1.
11: Compute the symmetric eigendecomposition $\boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^T$ of $\hat{\Sigma}_{\boldsymbol{W}^{\mathsf{reg}}} = \boldsymbol{W}^{\mathsf{reg}T}\boldsymbol{S}_{\boldsymbol{W}}^{\mathsf{reg}}\boldsymbol{W}^{\mathsf{reg}}/n_p$.
12: $\boldsymbol{W}^{\mathsf{norm}} \leftarrow \boldsymbol{V}\boldsymbol{\Lambda}^{-1/2}$
13: $\boldsymbol{W}^{\mathsf{eff}} = \boldsymbol{W}^{\mathsf{reg}}\boldsymbol{W}^{\mathsf{norm}}$
14: $\boldsymbol{A}^{\mathsf{eff}} \leftarrow \boldsymbol{W}^{\mathsf{eff}T}\boldsymbol{A}$

First, for the regression, we have to solve the linear system:

$$\boldsymbol{C} = \boldsymbol{A} \cdot \boldsymbol{B},$$

for the unknown $\boldsymbol{A}$ with:

$$\boldsymbol{B} = \boldsymbol{\beta}(\boldsymbol{x})\boldsymbol{\beta}(\boldsymbol{x})^T = \sum_{i=1}^{n_p} \boldsymbol{\beta}(x_i)\boldsymbol{\beta}(x_i)^T,$$

$$\boldsymbol{C} = \boldsymbol{L}\boldsymbol{\beta}(\boldsymbol{x})^T = \sum_{i=1}^{n_p} \boldsymbol{l}_i\boldsymbol{\beta}(x_i)^T,$$

where $\boldsymbol{B}$ is a full-rank square matrix if the span of the set of bit-vectors x_i is $\mathbb{F}^b$.[4] This system is therefore a re-writing of the normal equations (Equation 2.7) in a matrix form.

Let us remark that $\boldsymbol{B}$ and $\boldsymbol{C}$ can be computed incrementally while the profiling traces are being acquired or loaded. For improved imple-

[4] For example, this happens with overwhelming probability for the regression with the linear basis when x is selected uniformly at random and the profiling set is large enough.

mentation efficiency, one may also decompose $\boldsymbol{x}$ and $\boldsymbol{L}$ in small blocks rather than single elements (resp., columns).

Next, we derive an efficient formula for the inter-class scatter matrix $\boldsymbol{S}_{\boldsymbol{B}}^{\mathsf{reg}}$, avoiding the sum over the 2^b classes, as formalized by the following proposition.

Proposition 1 (Efficient $\boldsymbol{S}_{\boldsymbol{B}}^{\mathsf{reg}}$ formula)**.**

$$\boldsymbol{S}_{\boldsymbol{B}}^{\mathsf{reg}} = \boldsymbol{A}\boldsymbol{B}\boldsymbol{A}^T - n_p\hat{\boldsymbol{\mu}}\hat{\boldsymbol{\mu}}^T.$$

Proof. First, we observe that

$$\sum_{x\in\mathcal{X}} |\mathcal{L}_x|\,\mathbf{m}(x) = \sum_{i=1}^{n_p} \mathbf{m}(x_i) = \sum_{i=1}^{n_p} \boldsymbol{A}\boldsymbol{\beta}(x_i) = \boldsymbol{A}\boldsymbol{\beta}(\boldsymbol{x})\mathbb{1}_{n_p},$$

is the first column of $\boldsymbol{A}\boldsymbol{B}$, since the first column of $\boldsymbol{\beta}(\boldsymbol{x})^T$ is $\mathbb{1}_{n_p}$ by construction (the first basis function is the constant 1). Therefore, it is equal to the first column of $\boldsymbol{C}$, which is itself equal to $\boldsymbol{L}\mathbb{1}_{n_p} = \sum_{i=1}^{n_p} \boldsymbol{l}_i$. Hence

$$\sum_{x\in\mathcal{X}} |\mathcal{L}_x|\,\mathbf{m}(x) = n_p\hat{\boldsymbol{\mu}}$$

by definition of $\hat{\boldsymbol{\mu}}$. We then expand the definition of $\boldsymbol{S}_{\boldsymbol{B}}^{\mathsf{reg}}$:

$$\boldsymbol{S}_{\boldsymbol{B}}^{\mathsf{reg}} = \sum_{x\in\mathcal{X}} |\mathcal{L}_x| \left[\mathbf{m}(x)\mathbf{m}(x)^T - \mathbf{m}(x)\hat{\boldsymbol{\mu}}^T - \hat{\boldsymbol{\mu}}\mathbf{m}(x)^T + \hat{\boldsymbol{\mu}}\hat{\boldsymbol{\mu}}^T\right],$$

which, using the previous result, gives

$$\boldsymbol{S}_{\boldsymbol{B}}^{\mathsf{reg}} = \sum_{i=1}^{n_p} \mathbf{m}(x_i)\mathbf{m}(x_i)^T - n_p\hat{\boldsymbol{\mu}}\hat{\boldsymbol{\mu}}^T.$$

We conclude the proof by remarking that

$$\begin{aligned}\sum_{i=1}^{n_p} \mathbf{m}(x_i)\mathbf{m}(x_i)^T &= \boldsymbol{A}\boldsymbol{\beta}(\boldsymbol{x})(\boldsymbol{A}\boldsymbol{\beta}(\boldsymbol{x}))^T.\,(\boldsymbol{A}\boldsymbol{\beta}(\boldsymbol{x}))\,(\boldsymbol{A}\boldsymbol{\beta}(\boldsymbol{x}))^T \\ &= \boldsymbol{A}\boldsymbol{\beta}(\boldsymbol{x})\boldsymbol{\beta}^T(\boldsymbol{x})\boldsymbol{A}^T \\ &= \boldsymbol{A}\boldsymbol{B}\boldsymbol{A}^T.\end{aligned}$$

□

And for $\boldsymbol{S}_{\boldsymbol{W}}^{\mathsf{reg}}$, we similarly derive the following incremental formula.

Proposition 2 (Efficient $\boldsymbol{S}_{\boldsymbol{W}}^{\mathsf{reg}}$ formula)**.**

$$\boldsymbol{S}_{\boldsymbol{W}}^{\mathsf{reg}} = \boldsymbol{L}\boldsymbol{L}^T - \boldsymbol{C}\boldsymbol{A}^T - \boldsymbol{A}\boldsymbol{C}^T + \boldsymbol{A}\boldsymbol{B}\boldsymbol{A}^T.$$

Proof. We expand the definition of $\boldsymbol{S}_{\boldsymbol{W}}^{\mathsf{reg}}$:

$$\begin{aligned}\boldsymbol{S}_{\boldsymbol{W}}^{\mathsf{reg}} &= \sum_{i=0}^{n_p} \left[\boldsymbol{l}_i\boldsymbol{l}_i^T - \boldsymbol{l}_i\mathbf{m}(x_i)^T - \mathbf{m}(x_i)\boldsymbol{l}_i^T + \mathbf{m}(x_i)\mathbf{m}(x_i)^T\right], \\ &= \boldsymbol{L}\boldsymbol{L}^T - \boldsymbol{L}(\boldsymbol{A}\boldsymbol{\beta}(\boldsymbol{x}))^T - \boldsymbol{A}\boldsymbol{\beta}(\boldsymbol{x})\boldsymbol{L}^T + \boldsymbol{A}\boldsymbol{\beta}(\boldsymbol{x})(\boldsymbol{A}\boldsymbol{\beta}(\boldsymbol{x}))^T,\end{aligned}$$

and the claim then follows from the definitions of $\boldsymbol{B}$ and $\boldsymbol{C}$. □

The matrix $\boldsymbol{L}\boldsymbol{L}^T$ can be computed incrementally as $\sum_{i=1}^{n_p} \boldsymbol{l}_i\boldsymbol{l}_i^T$. Exploiting such a proposition nevertheless requires some attention since it may not always be numerically stable if a lot of traces are needed for profiling. In practice though, assuming that the adversary/evaluator measures traces made of 8-bit or 16-bit integers and the accumulator is a 64-bit integer, the computation remains exact as long as $|\mathcal{L}| < 2^{32}$).[5]

Once the scatter matrices are known, one can partially solve the generalized eigenproblem:

$$\boldsymbol{S}_{\boldsymbol{B}}^{\mathsf{reg}}\boldsymbol{w}_i = \lambda_i \boldsymbol{S}_{\boldsymbol{W}}^{\mathsf{reg}}\boldsymbol{w}_i, \tag{4.1}$$

to get the projection matrix $\boldsymbol{W}^{\mathsf{reg}} = (\boldsymbol{w}_1 \cdots \boldsymbol{w}_p)$. This is fairly efficient since both scatter matrices are symmetric semi-positive definite, and we usually only need a small part of the spectrum (the one corresponding to the $p \ll n_s$ largest eigenvalues). Moreover, if the traces are noisy and $n_p > n_s$, the matrix $\boldsymbol{S}_{\boldsymbol{W}}^{\mathsf{reg}}$ is non-singular with probability 1, which makes the problem easy to solve [PW69]. At this stage, we could compute $\hat{\Sigma}_{\boldsymbol{W}^{\mathsf{reg}}}^{-1}$ and then use Equation 2.11. However, in order to further simplify the model, we will modify the projection to get a Gaussian distribution with unit covariance. Namely, we compute the eigendecomposition of the symmetric positive-definite (if we profile with at least p noisy traces) $p \times p$ matrix: $\hat{\Sigma}_{\boldsymbol{W}^{\mathsf{reg}}} = |\mathcal{L}|^{-1}\, \boldsymbol{W}^{\mathsf{reg}T} \boldsymbol{S}_{\boldsymbol{W}}^{\mathsf{reg}} \boldsymbol{W}^{\mathsf{reg}}$:

$$\hat{\Sigma}_{\boldsymbol{W}^{\mathsf{reg}}} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^T$$

where $\boldsymbol{\Lambda}$ is the diagonal matrix of eigenvalues and $\boldsymbol{V}$ is the orthonormal matrix of eigenvectors. Then, we set $\boldsymbol{W}^{\mathsf{norm}} = \boldsymbol{V}\boldsymbol{\Lambda}^{-1/2}$. The final projection matrix is $\boldsymbol{W}^{\mathsf{eff}} = \boldsymbol{W}^{\mathsf{reg}}\boldsymbol{W}^{\mathsf{norm}}$, and we compute $\boldsymbol{A}^{\mathsf{eff}} = \boldsymbol{W}^{\mathsf{eff}^T}\boldsymbol{A}$. As a result,

$$\boldsymbol{W}^{\mathsf{eff}^T}\boldsymbol{l}^* - \boldsymbol{A}^{\mathsf{eff}}\boldsymbol{\beta}(x) = \boldsymbol{W}^{\mathsf{norm}T}\left(\boldsymbol{W}^{\mathsf{reg}T}(\boldsymbol{l}^* - \mathbf{m}(x))\right)$$

[5] If numerical precision is a concern, an easy improvement is to ensure that the mean of the traces is close to zero, which can be done by computing the mean of a few traces and subtracting it to every trace.

and since $\boldsymbol{W}^{\mathsf{norm}}\boldsymbol{W}^{\mathsf{norm}T} = \hat{\Sigma}^{-1}_{\boldsymbol{W}^{\mathsf{reg}}}$, we can re-write Equation 2.11 as

$$\hat{\mathsf{f}}^{\mathsf{RLDA}}[\boldsymbol{l}^*|X = x] = \alpha \exp\left(-\frac{1}{2}\left\|\boldsymbol{W}^{\mathsf{eff}T}\boldsymbol{l}^* - \boldsymbol{A}^{\mathsf{eff}}\boldsymbol{\beta}(x)\right\|^2\right). \tag{4.2}$$

Let us note that we do not need to compute $\alpha = 1/\sqrt{(2\pi)^p|\hat{\Sigma}_{\boldsymbol{W}^{\mathsf{reg}}}|}$ since we only have to know $\hat{\mathsf{f}}^{\mathsf{RLDA}}[\boldsymbol{l}^*|X = x]$ up to a constant factor to apply Bayes rule.

Profiling complexity. Overall, RLDA profiling using Algorithm 4 has a computational complexity of $\Theta\left(n_s^2(b + n_p)\right)$ and memory usage $\Theta\left(n_s^2\right)$.[6] Remarkably, and contrary to the baseline method of Subsection 2.5.3, this complexity is not exponential in b.

Attack complexity. For one attack trace $\boldsymbol{l}^*$, computing the full distribution of X using Equation 4.2 and Equation 2.2 has a computational cost $\Theta\left((n_s + 2^b)p\right)$, instead of $\Theta\left(2^b(b + p)n_s\right)$ for a naive implementation, since we can amortize the cost of computing $\boldsymbol{A}^{\mathsf{eff}}\boldsymbol{\beta}(x)$ to $\Theta\left(p\right)$ on average by caching some intermediate sums. The main memory usage is dominated by the storage of the resulting distribution which has cost $\Theta\left(2^b\right)$, while the cache used for the above optimization is small: $\Theta\left(pb\right)$. When n_a traces are available for the attack, we compute Equation 4.2 independently for each trace and multiply the resulting distributions, which has computational complexity $\Theta\left(n_a(n_s + 2^b)p\right)$.

Multi-trace optimization. There are cases where multiple traces correspond to the same value of the variable X. For example, there is only one value for X in the SPA setting, and the number of attack traces n_a can be larger than the number of possible values for X in a traditional DPA setting. In such cases, the following optimization reduces the computation cost of computing the RLDA on n_a traces that share a common X value to the cost of a single-trace attack, up to the cost of summing all the attack traces together, i.e., time complexity of $\Theta\left(n_a n_s + (n_s + 2^b)p\right)$.

It is well-known that Equation 4.2 can be simplified by removing the quadratic term in $\boldsymbol{l}^*$ from the sum [CK13], which is what makes LDA

[6] We simplified these expressions by assuming that $b \leq n_s$ and $b \leq n_p$.

"linear":

$$\left\|\boldsymbol{W}^{\mathsf{eff}^T}\boldsymbol{l}^* - \boldsymbol{A}^{\mathsf{eff}}\boldsymbol{\beta}(x)\right\|^2 = \left\|\boldsymbol{W}^{\mathsf{eff}^T}\boldsymbol{l}^*\right\|^2 - 2(\boldsymbol{W}^{\mathsf{eff}^T}\boldsymbol{l}^*)^T\boldsymbol{A}^{\mathsf{eff}}\boldsymbol{\beta}(x) + \left\|\boldsymbol{A}^{\mathsf{eff}}\boldsymbol{\beta}(x)\right\|^2.$$

This scales the terms by a factor independent of x, which disappears when applying the normalization of Equation 2.2, leading to a formula linear in $\boldsymbol{l}^*$. Then, the product of the distributions for the n_a traces is turned into a sum of the exponents. We revisit this technique by keeping the exponent in the form of a squared norm, which makes it very efficient to compute and avoids numerical overflows in the exponential (which otherwise would require a normalization pass before computing the exponential to avoid overflows), while maintaining the explicit computations of probabilities to discriminate the key (contrary to [CK13]). Our technique is derived as follows:

$$\begin{aligned}\hat{\mathsf{f}}^{\mathsf{RLDA}}&[(\boldsymbol{l}_1^*,\ldots,\boldsymbol{l}_{n_a}^*)|X=x] \\ &= \prod_{i=1}^{n_a}\alpha\exp\left(-\frac{1}{2}\left\|\boldsymbol{W}^{\mathsf{eff}^T}\boldsymbol{l}_i^* - \boldsymbol{A}^{\mathsf{eff}}\boldsymbol{\beta}(x)\right\|^2\right), \\ &= \alpha_{(\boldsymbol{l}_1^*,\ldots,\boldsymbol{l}_{n_a}^*)}\exp\left(-\frac{1}{2n_a}\left\|\boldsymbol{W}^{\mathsf{eff}^T}\sum_{i=1}^{n_a}\boldsymbol{l}_i^* - n_a\boldsymbol{A}^{\mathsf{eff}}\boldsymbol{\beta}(x)\right\|^2\right),\end{aligned} \tag{4.3}$$

where the normalization factor $\alpha_{(\boldsymbol{l}_1^*,\ldots,\boldsymbol{l}_{n_a}^*)}$ does not have to be computed, as it cancels out in Equation 2.2. The correctness of Equation 4.3 follows from the equality:

$$\sum_{i=1}^{n_a}\left\|\boldsymbol{W}^{\mathsf{eff}^T}\boldsymbol{l}_i^* - \boldsymbol{A}^{\mathsf{eff}}\boldsymbol{\beta}(x)\right\|^2 = \frac{1}{n_a}\left\|\boldsymbol{W}^{\mathsf{eff}^T}\left(\sum_{i=1}^{n_a}\boldsymbol{l}_i^*\right) - n_a\boldsymbol{A}^{\mathsf{eff}}\boldsymbol{\beta}(x)\right\|^2 + \omega_{(\boldsymbol{l}_1^*,\ldots,\boldsymbol{l}_{n_a}^*)},$$

where $\omega_{(\boldsymbol{l}_1^*,\ldots,\boldsymbol{l}_{n_a}^*)}$ depends only on $\boldsymbol{l}_i^*$ and $\boldsymbol{W}^{\mathsf{eff}}$.

4.3 Efficient PI bound

The previous section described a tool enabling to evaluate a model for large (e.g., 32-bit) target intermediate values. As an adversary/evaluator, such a tool can directly be used to mount improved attacks, which

we will further detail in Chapter 6. But as an evaluator, it may also happen that one is just interested to gauge the security level of an implementation, without mounting explicit attacks. This is typically what happens when using shortcut formulas like [DFS19, dCGRP19] for divide-and-conquer attacks and [GGSB20] for analytical attacks. In this case, the evaluation problem essentially reduces to the estimation of an information theoretic metric like the PI.

Looking at Equation 2.12, estimating the PI by sampling boils down to estimating a model $|\mathcal{L}'|$ times – the value of $|\mathcal{L}'|$ depending on the security level. However, in order to turn the leakage (conditional) likelihoods $\hat{\mathsf{f}}[\boldsymbol{l}|x]$ into probabilities $\hat{\mathsf{p}}[x|\boldsymbol{l}]$ for the target intermediate value, Bayes's rule must be applied. This implies evaluating the model over all the possible classes $x' \in \mathcal{X}$, where $\mathcal{X}$ grows exponentially with the number of bits b of X. For intermediate target sizes, like the 32-bit ones we consider in this work, it may result in a situation where despite feasible, evaluating the model exhaustively $\Theta\left(|\mathcal{L}'| \cdot |\mathcal{X}|\right)$ times becomes cumbersome. In this section, we therefore propose a way to improve the efficiency of the PI estimation, by replacing its exact computation with cheaper bounds. Precisely, we show a solution to reduces the $\Theta\left(|\mathcal{L}'| \cdot |\mathcal{X}|\right)$ computational complexity of the exact estimation down to $\Theta\left(|\mathcal{L}'| \cdot S + |\mathcal{X}| \log S\right)$ for the approximated one, at a $\Theta\left(S\right)$ memory cost, where the parameter S determines the tightness of the bounds.

Concretely, instead of computing $\hat{\mathsf{p}}[x|\boldsymbol{l}]$ exactly for every x, we will bound it by clustering some x values according to some metric. These bounds will then be translated into bounds on the estimated PI. More precisely, using the RLDA model, we have:

$$\hat{\mathsf{f}}[X = x|\boldsymbol{l}] = \frac{\exp(-\frac{1}{2}\left\|l - \mathsf{m}(x)\right\|^2)}{\sum_{x'=0}^{2^b-1} \exp(-\frac{1}{2}\left\|l - \mathsf{m}(x')\right\|^2)}.$$

Our core idea is that if $\mathsf{m}(x')$ and $\mathsf{m}(x'')$ are close to each other, then the corresponding terms in the denominator will have close values, hence we can compute the term $\exp(-\frac{1}{2}\left\|l - \mathsf{m}(x')\right\|^2)$ for x' and use it to infer bounds for the term $\exp(-\frac{1}{2}\left\|l - \mathsf{m}(x'')\right\|^2)$. As a result, our goal (to gain efficiency) is to find many x'' classes close to each class x' for which we compute the exponentiation. For this purpose, we perform a preprocessing to identify the classes that are close to each other, then compute all the relevant terms, and finally put these results together to get bounds for all $\hat{\mathsf{f}}[X = x|\boldsymbol{l}]$ values.

The preprocessing is parameterized by a distance threshold $t \geq 0$ and builds a set $\mathcal{X}' \subset \mathcal{X}$ and a map $\sigma : \mathcal{X} \to \mathcal{X}'$ such that

$\|\mathsf{m}(x) - \mathsf{m}(\sigma(x))\| \leq t$ for every $x \in \mathcal{X}$. Each $x' \in \mathcal{X}'$ is named a cluster center, and the pre-images of x' form a cluster. For every cluster center, we compute the size of its cluster $s(x')$. The construction of $\mathcal{X}'$ and $s(\cdot)$ is done according to Algorithm 5, where the map σ is implicit: $\sigma(x)$ is the nearest point to x at the time x is processed. Its correctness can be verified by observing that the expected properties on $\sigma(.)$ and $s(.)$ are kept satisfied at every iteration (when considering only the subset of values $x \in \mathcal{X}$ that have already been processed). We illustrate the result of the clustering in Figure 4.1. Let $S = |\mathcal{X}'|$, the time complexity of this algorithm is $\mathcal{O}(|X| \cdot \log S)$ and its memory complexity is $\mathcal{O}(\mathcal{X})$ when computing σ^{-1}, or $\mathcal{O}(S)$ otherwise.

Algorithm 5 Clustering.

Input: A set of classes $\mathcal{X}$, a model $\mathsf{m} : \mathcal{X} \to \mathbb{R}^p$ and a threshold distance t.

Output: A set of cluster centers $\mathcal{X}'$ (represented by nn in the algorithm), cluster sizes $s(\cdot)$ and list of classes for each cluster, next denoted as σ^{-1}.

▷ `NN` is a set of elements in $(\mathcal{X}, \mathbb{R}^p)$ with logarithmic cost (in its size) `insert` and $\mathtt{nn} : \mathbb{R}^p \to \mathcal{X}$ (nearest neighbor) query (retrieves the stored $x \in \mathcal{X}$ that corresponds to closest stored vector). We instantiate it with a k-d tree [Ben75].

1: $\mathcal{X}' \leftarrow \emptyset$
2: $s \leftarrow$ `Map.newEmpty()`
3: $nn \leftarrow$ `NN.newEmpty()`
4: **for** $x \in \mathcal{X}$ **do**
5: $\quad x' \leftarrow nn.\mathtt{nearest}(\mathsf{m}(x))$
6: $\quad$ **if** $\|\mathsf{m}(x) - \mathsf{m}(x')\| > t$ **then**
7: $\quad\quad \mathcal{X}' \leftarrow \mathcal{X}' \cup \{x\}$
8: $\quad\quad nn.\mathtt{insert}(x, \mathsf{m}(x))$
9: $\quad\quad s(x) \leftarrow 0$
10: $\quad\quad \sigma^{-1}(x) \leftarrow \emptyset$
11: $\quad\quad x' \leftarrow x$
12: $\quad s(x') \leftarrow s(x') + 1$
13: $\quad \sigma^{-1}(x') \leftarrow \sigma^{-1}(x') \cup \{x\}$ ▷ Only when using Equation 4.5 and not Equation 4.4.

Once clustering is performed, we can compute probability bounds knowing only the cluster centers and the cluster sizes using the following proposition.

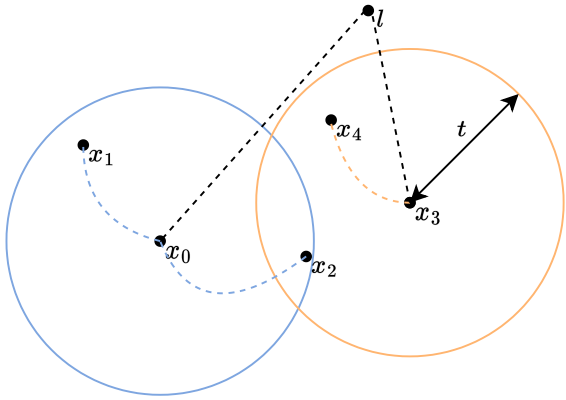


Figure 4.1: Illustration of the clustering-based PI computation for the set of classes $\mathcal{X} = \{x_0, \ldots, x_4\}$. The cluster centers are $\mathcal{X}' = \{x_0, x_3\}$ in our example, and the bound from Proposition 3 is $3l_{x_0} + 2l_{x_3} \leq \sum_{x \in \mathcal{X}} \alpha\left(\|\boldsymbol{l} - x_i\|^2\right) \leq 3u_{x_0} + 2u_{x_3}$.

Proposition 3 (Probability bounding with clustering)**.** *Let $\mathcal{X}'$ and $s(\cdot)$ be the outputs of Algorithm 5 for the inputs $\mathcal{X}$, m and t, and let us denote $\alpha(y) = \exp(-\frac{1}{2}y)$. Then, for any $\boldsymbol{l} \in \mathbb{R}^p$:*

$$\sum_{x' \in \mathcal{X}'} s(x') \cdot l_{x'} \leq \sum_{x \in \mathcal{X}} \alpha\left(\|\boldsymbol{l} - \mathsf{m}(x)\|^2\right) \leq \sum_{x' \in \mathcal{X}'} s(x') \cdot u_{x'},$$

where:

$$l_{x'} = \alpha\left(\left(\|\boldsymbol{l} - \mathsf{m}(x')\| + t\right)^2\right),$$
$$u_{x'} = \alpha\left(\left(\max\left(0, \|\boldsymbol{l} - \mathsf{m}(x')\| - t\right)\right)^2\right).$$

Proof. Let $x \in \mathcal{X}$ and $x' = \sigma(x)$ (hence $\|\mathsf{m}(x) - \mathsf{m}(x')\| \leq t$), using the triangular inequality we get:

$$\|\boldsymbol{l} - \mathsf{m}(x)\|^2 \leq \left(\|\boldsymbol{l} - \mathsf{m}(x')\| + \|\mathsf{m}(x) - \mathsf{m}(x')\|\right)^2 \leq \left(\|\boldsymbol{l} - \mathsf{m}(x')\| + t\right)^2.$$

We proceed similarly to calculate the lower bound. Starting from the triangular inequality $\|\boldsymbol{l} - \mathsf{m}(x')\| \leq \|\boldsymbol{l} - \mathsf{m}(x)\| + \|\mathsf{m}(x) - \mathsf{m}(x')\|$, we get $\|\boldsymbol{l} - \mathsf{m}(x)\| \geq \|\boldsymbol{l} - \mathsf{m}(x')\| - t$, and then $\|\boldsymbol{l} - \mathsf{m}(x)\|^2 \geq \left(\max\left(0, \|\boldsymbol{l} - \mathsf{m}(x')\| - t\right)\right)^2$.

We now have $\alpha^{-1}(u_{x'}) \leq \|\boldsymbol{l} - \mathsf{m}(x)\|^2 \leq \alpha^{-1}(l_{x'})$, which leads to the expected result by observing that $\alpha(\cdot)$ is a decreasing function and then summing over all $x \in \mathcal{X}$. □

Corollary 1. *With the same hypotheses as Proposition 3, and assuming that:*

$$\hat{\mathsf{f}}[X = x|\boldsymbol{l}] = \frac{\alpha\left(\|l - \mathsf{m}(x)\|^2\right)}{\sum_{x' \in \mathcal{X}} \alpha\left(\|l - \mathsf{m}(x')\|^2\right)},$$

the following holds:

$$\frac{\alpha\left(\|l - \mathsf{m}(x)\|^2\right)}{\sum_{x' \in \mathcal{X}'} s(x') \cdot u_{x'}} \leq \hat{\mathsf{f}}[X = x|\boldsymbol{l}] \leq \frac{\alpha\left(\|l - \mathsf{m}(x)\|^2\right)}{\sum_{x' \in \mathcal{X}'} s(x') \cdot l_{x'}}. \tag{4.4}$$

Proof. The two inequalities are trivial consequences of Proposition 3. □

We can then bound the PI by using the probability bounds of Corollary 1 in Equation 2.12, since this formula is increasing with $\hat{\mathsf{p}}[X = x|\boldsymbol{l}]$. The trade-off between the tightness of the bound and the computation cost is controlled with the parameter t.

When this trade-off is not satisfactory, we next propose an improvement of the method. It is based on the observation that the largest contributors to the gap between the probability upper- and lower-bounds are the clusters that are near to $\boldsymbol{l}$. Indeed, the value of $l_{x'}$ and $u_{x'}$ are small for cluster centers x' that are far from $\boldsymbol{l}$, compared to the closer ones, due to exponential shrinking with $\|\boldsymbol{l} - \mathsf{m}(x')\|^2$. Therefore, even a large relative gap $u_{x'}/l_{x'}$ for these far clusters does not translate into a large difference on the overall sums $\sum_{x' \in \mathcal{X}'} s(x') \cdot l_{x'}$ and $\sum_{x' \in \mathcal{X}'} s(x') \cdot u_{x'}$. Conversely, the gap on the closest clusters will have the most impact on the overall tightness of the final bounds. We therefore propose to nullify this gap by computing the exact probability $\hat{\mathsf{f}}[\boldsymbol{l}|X = y]$ for all classes y in the close clusters, instead of relying on the bounds. Concretely, we implement this by storing a list of all classes in each cluster when running Algorithm 5 (i.e., we store the map $\sigma^{-1} : \mathcal{X}' \to \mathcal{P}(\mathcal{X})$).[7] Then, when evaluating probability bounds for some leakage $\boldsymbol{l}$, we look in the *nn* structure (from Algorithm 5) for the cluster centers that are the closest to $\boldsymbol{l}$, and add them to a set $\mathcal{X}''$, stopping before the total size of the associated clusters exceeds a computational cost threshold ζ (i.e., we ensure $\sum_{x'' \in \mathcal{X}''} s(x'') \leq \zeta$).[8] Finally, we use the following bounds:

$$\frac{\alpha\left(\|l - \mathsf{m}(x)\|^2\right)}{Z + \sum_{x' \in \mathcal{X}' \setminus \mathcal{X}''} s(x') \cdot u_{x'}} \leq \hat{\mathsf{f}}[X = x|\boldsymbol{l}] \leq \frac{\alpha\left(\|l - \mathsf{m}(x)\|^2\right)}{Z + \sum_{x' \in \mathcal{X}' \setminus \mathcal{X}''} s(x') \cdot l_{x'}}, \tag{4.5}$$

[7] Which significantly increases the memory usage of the algorithm but not its computational complexity.

[8] Taking ζ as a small multiple of the number of clusters $|\mathcal{X}'|$ works well in practice.

where:

$$Z = \sum_{y \in \mathcal{Y}} \alpha\left(\|l - \mathsf{m}(y)\|^2\right), \quad \text{with} \quad \mathcal{Y} = \bigcup_{x'' \in \mathcal{X}''} \sigma^{-1}(x'').$$

Remark. We assumed a uniformly distributed X. The method can be adapted to a non-uniform X by storing the total probability of the clusters in $s(\cdot)$ instead of their sizes.

4.4 Atomic experiments

In this section, we investigate the ability of RLDA to exploit the leakage of a 32-bit ALU for a single isolated instruction and we analyze the obtained PI bounds. We compare the PI of the RLDA model with the one of state-of-the art models: LDA, LR and FT.

4.4.1 Experimental setup

We designed this first experiment to produce well isolated and easy to interpret leakages. The target 32-bit ARM instruction is a single XOR between two values stored in registers. As shown in Figure 4.2, this instruction is surrounded by `nop` instructions to separate the leakage of this instruction from other leakages (e.g., caused by memory accesses). The XOR operands are independent uniform random values, and the target value is the result of the operation. As a result, the leakage of the operands cannot be exploited by the RLDA (like in a 1st-order masked implementation). The register writeback leakage is also reduced by overwriting an operand: this produces mainly a transition leakage which is then equivalent to leakage from the other operand. The impact of a more realistic setting including pipeline noise will be discussed in 6.

```
ldrd r5,r6,[r0] // Load 2 random b-bit values (other bits
    set to 0)
NOP6 // Macro that generates 6 nop instructions, to flush
    the pipeline.
bl <trigger_up>
NOP6
eor.w r5,r5,r6 // Target instruction
NOP6
bl <trigger_low>
```

Figure 4.2: Target ARM assembly code for the atomic experiment.

Measurements were performed on the ChipWhisperer CW308 board, with an STM32F303 target (Cortex-M4, 3-stage pipeline). The target

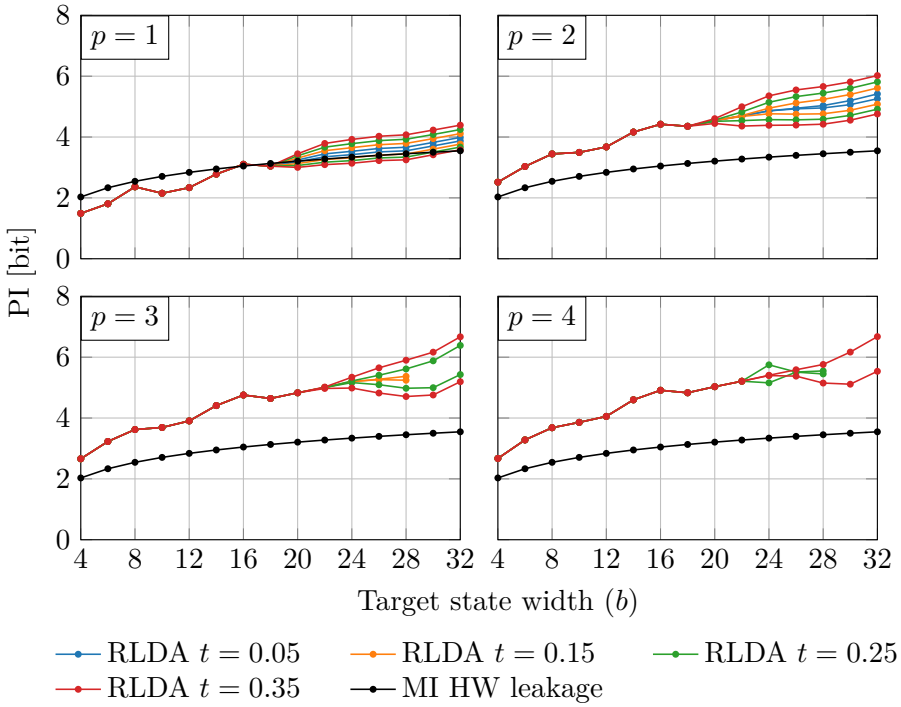


Figure 4.3: PI bounds for a XOR instruction leakage profiled with the RLDA model, for various values of the model parameters b, p, and of the PI estimation clustering parameter t. Identically colored lines represent the upper- and the lower-bound on the PI. The black line corresponds to the MI obtained from simulated Hamming weight leakages.

was running an 80 MHz clock derived internally from a 8 MHz external crystal. We used the Tektronix CT-1 AC current probe and we sampled our signal with the Picoscope 6424E at 625 MSample/s with 12-bit resolution. For each experiment, we used 1.3 million profiling traces and 1000 evaluation traces.

4.4.2 Analyzing various bus sizes

We analyzed the PI of the RLDA model (with a linear basis) when profiling the result of the XOR operation for different simulated bus sizes by keeping some bits of the operands set at 0. We profiled with multiple subspace dimensionalities: $p = 1, 2, 3, 4$ using the full leakage trace $n_s = 500$ samples (i.e., no POI selection). We also evaluated the tightness of our PI bound by running it with multiple clustering thresholds $t = 0.05, 0.15, 0.25, 0.35$, and aborting if the number of clusters $|\mathcal{X}'|$ exceeds 2^{20}, to limit the computational cost. We allowed the exact calculation

of the likelihood for $\zeta = \max(2^{16}, 16 \cdot |\mathcal{X}'|)$ classes, which leads to a reasonnable overhead in execution time when there are many clusters and ensures that we calculate the exact PI when the cost to do so is low. The results are in Figure 4.3, where we add the MI of a Hamming weight model for comparison purposes.

We first observe that the PI of the model grows with the number of target bits b (the entropy of the target variables increases as well). For $p = 1$, the PI of our model is close the to the MI of Hamming weight leakages, and it exceeds it for all values of b when $p \geq 2$. Moreover, the PI of the model increases with p (for any fixed $b > p$), but saturates quickly: the improvement between $p = 3$ and $p = 4$, which matches the intuition that most the leakage lies in a low-dimentional subspace.[9] Next, regarding the tightness of the clustering-based PI bounds, we observe that it decreases as the threshold t increases. This leads to an accuracy vs. efficiency trade-off, since the efficiency mostly depends on the number of clusters (reported in Table 4.1). This number is limited by two parameters: the total number of classes 2^b, and the number of clusters needed to cover the space of possible leakage values. Intuitively, this space is p-dimensional and the extreme values depend on the SNR of the leakage in this projected space (the noise variance is normalized to 1), while we try to cover it with balls of radius t. As for the computation cost: for the largest case ($b = 32$, $p = 4$), the RLDA profiling runs in about 18 s, the clustering takes approximately 4 h and the PI bounds computation using the evaluation traces takes about 10 s. The implementation is multi-threaded, with the exception of the clustering and was run on a 2.7 GHz AMD CPU. The k-d tree implementation has not been optimized.

4.4.3 Comparison with related leakage models

We also compared the RLDA with other profiled leakage models. Namely, we considered the LDA of Subsection 2.5.2 profiled using all the 500 samples of the traces (like RLDA), the LR-based profiling of Section 2.4 combined with a POI selection taking the p points with the highest SNR in the traces (POI selection is needed in this case, since we were not able to accurately estimate the large covariance matrices without this preprocessing), and the FT of Section 4.1 with 8-bit fragments (the PI reported is then the sum of the PI on all fragments) —

[9] We did not study larger p values due to limitations of our PI estimation method (the clustering with k-d tree becomes less efficent). However, the RLDA is not limited to such low dimensions: its main limit is the increased risk of over-fitting, which can be limited with a sufficiently large profiling dataset.

Table 4.1: Logarithm of the number of clusters ($\log_2(\mathcal{X}')$) for the atomic experiment. The cell color represents the ratio $|\mathcal{X}'| / |\mathcal{X}|$: darker cells reflect less effective clustering.

b	p	t				p	t			
		0.05	0.15	0.25	0.35		0.05	0.15	0.25	0.35
8	1	6.9	6.0	5.3	5.0	2	8.0	7.6	7.4	7.3
16		9.5	8.1	7.4	6.9		14.8	12.8	11.5	10.9
24		10.5	8.8	8.1	7.8		17.5	14.7	13.3	12.4
32		11.2	9.6	8.8	8.4		19.0	16.0	14.6	13.6
8	3	8.0	7.6	7.6	7.6	4	8.0	7.6	7.6	7.6
16		15.9	14.6	13.2	12.5		16.0	15.1	14.6	13.5
24			18.0	16.1	14.9				19.0	16.8
32				18.3	16.9					19.6

this model is denoted as FT8. For completeness, we also analyzed FT with 1-bit fragments (denoted as FT1) and FT with 8-bit fragments marginalized to single)-bit variables (denoted as mFT8). The results for the cases $b = 16$ and $b = 32$ are shown in Figure 4.4. For the 16-bit bus, all models are analyzed. For the 32-bit case, we could only run FT and RLDA, since the other have a too high computational cost.

Starting with the 16-bit bus experiment, the PI of the RLDA can be evaluated exactly (i.e., no clustering is needed) and the resulting model achieves the highest PI of all models. The gains over FT are expected since these models artificially increase the amount of algorithmic noise due to their fragmentation process. The gains over the LR-based attack are presumably due to the suboptimal selection of POIs (i.e., selecting multiple univariate POIs with the SNR does not ensure that their multivariate selection is good, for example due to correlated leakage samples that may bring limited additional information). As a side note, the SNR might not be the optimal heuristic for finding the POI for a univariate attack as we see from Figure 4.4, the 6th POI for the LR model brings more information than the first 5[10]. As for the LDA, it performs worse than the RLDA due to higher class mean estimation errors. Despite the large total profiling dataset, each class mean estimation for the LDA relies on only 20 traces per class on average, since there are 2^{16} classes in total. For a similar reason, the PI of the LDA model slowly decreases after $p = 6$. This effect does not appear for the RLDA, thanks to the regularization effect of the linear regression.

[10]This was confirmed by an independent experiment not shown here. Therefore the improvement of the 6 dimension LR vs 5 is not only due to a better multivariate selection.

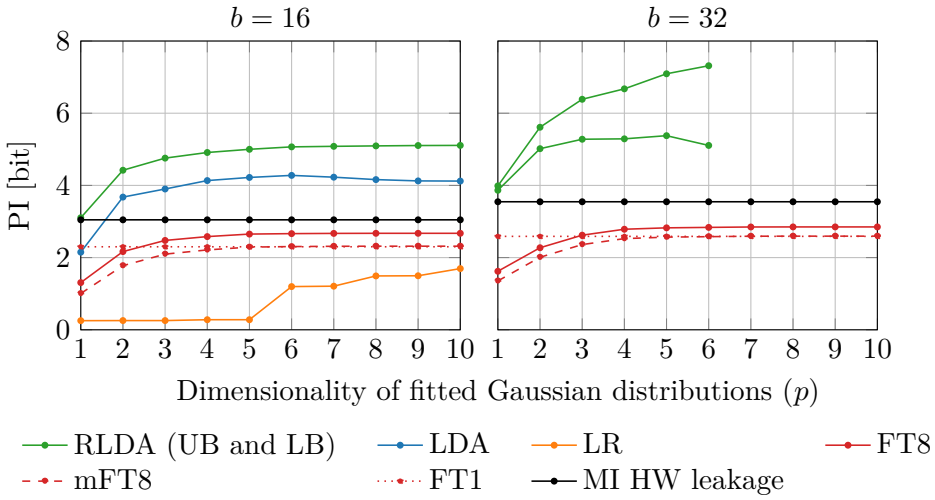


Figure 4.4: Comparison of profiled models for 16- and 32-bit bus.

When moving to the 32-bit state, we need the clustering to estimate the PI of the RLDA, leading to uncertainty on the value of the PI for the RLDA. As in the 16-bit case, we observe that FT performs quite poorly compared to RLDA. The gap between the bounds for RLDA becomes wide for $p > 2$, and we could not compute meaningful bounds for $p > 6$. However, the results with the 16-bit bus suggest that there is not much information to gain by increasing p significantly beyond 5. Besides, and despite we cannot cheaply estimate the PI in those cases, we can still use the resulting models to run attacks.

4.5 Conclusions

In this chapter we proposed optimized solutions to compute and estimate the PI of the RLDA profiled model, allowing it to scale up to 32-bit targets. The RLDA model is fairly simple and it could be improved towards better generality and accuracy, which could be useful to attack less leaky targets. The emphasis of this chapter being on efficiency, we published the code in SCAlib [CB23], an open-source library for side-channel analysis, oriented towards performance.

Among possible research directions, one could consider the use of a larger (e.g., quadratic) basis for the regression or the use of Quadratic Discriminant Analysis instead of LDA. Whether efficiently profiling long traces for large target intermediate values is possible with more advanced (e.g., deep learning) models is an interesting open problem as well.

Part II

Applications

Using coprocessors: Obscurity does not help 5

Microcontrollers are more and more equipped with cryptographic coprocessors. However, their documentation is limited to the needs of the end user, i.e. how to use the coprocessor and occasionally performance metrics. In this chapter we analyze MCUs with such peripherals and we question if the end user can rely on them when side-channel security is needed.

Side-channel attacks in similar settings have been published, with the common denominator being that attacks typically require a (possibly long and tedious) reverse engineering effort, because of either badly documented components or limited public implementation details. Among the examples of such real-world attacks we are aware of, we note the ones against Keeloq [EKM+08], the Xilinx bitstream encryption [MKP12, MS16], SIM cards [ZYSQ13, LYS+15], Hue Smart lamps [RSWO17] and Thread communication stacks [DK18].

We complete this state-of-the-art and extend the investigation of side-channel attacks against real-world devices to 32-bit MCU, that are becoming increasingly popular for lightweight embedded systems. In particular, we study the side-channel resistance of two COTS MCUs with Cortex-M4 processors that include a hardware coprocessor. We focus on MCUs from two different manufacturers, namely NXP Semiconductors and ST Microelectronics. We insist that none of them claimed to provide a high physical security level (although the NXP device includes some light countermeasure [Sem15]).

Our investigations show that limited information on the coprocessor's architecture does not prevent the identification of simple yet powerful attacks (while a better understanding of the targets could lead to further optimizations [BS20]). For this purpose, we first propose a variant of the Test vector leakage assessment (TVLA) methodology, which we denote as "one-hot" and is well suited to the analysis of hardware implementations: it allows us to locate some target operations and to infer the coprocessor architecture (e.g., the degree of parallelism) with low data complexity.

While the following attacks are certainly suboptimal, this chapter is a useful reminder that the absence of documentation on internal construction of the cryptographic core can be easily circumvented with common side-channel detection tools. The main purpose of such coprocessors should remain computation acceleration.

5.1 Background

This section contains additional background required to understand the content of this chapter.

T-Test The t-test [GJJR11, CMG+] is a common detection tool for side-channel leakages. It is based on Welch's t-test [Wel47], a statistical test used to highlight a difference between the means of two populations respectively denoted as μ_1 and μ_2. The test is performed by computing

$$t = \frac{\hat{\mu}_1 - \hat{\mu}_2}{\sqrt{\frac{\hat{\sigma}_1^2}{N_1} + \frac{\hat{\sigma}_2^2}{N_2}}}\,, \tag{5.1}$$

where σ_i and N_i are the standard deviation and the sample size of the population i. If the maximum $|t|$ is larger than 4.5, a difference between the means is very likely to be present (with a p-value smaller than 10^{-5}). In a side-channel context, it is generally used to highlight dependencies between the mean of the traces and the manipulated variables. To do so, the two tested populations are leakage traces $\boldsymbol{l}_i$ for two different sets of inputs. The t-test is then applied to all the time samples of the leakage traces independently. The implementation is said to be leaking when at least one sample presents a t-value larger than 4.5.

Continuous Wavelet Transform. In order to reduce the impact of noise on the measurements, a Continuous wavelet transform (CWT) can be used [DSE+12]. Similarly to Fourier Transforms, the CWT is a representation of a given signal in another domain. The CWT domain can be interpreted as the frequency content (f) across time (t'). Formally, the CWT of a time signal $x(t)$ is written as

$$X_\omega(f, t') = \frac{1}{|f|} \int_{-\infty}^{\infty} x(t) \cdot \psi\left(\frac{t - t'}{f}\right) dt\,, \tag{5.2}$$

where $\psi(\cdot)$ is a given wavelet. It is therefore the convolution of the signal with a scaled wavelet. Several wavelets have been tested. The best results were obtained with the Ricker wavelet which we use in this work.

For computational reasons, we limited the wavelet width to the smallest one that was maintaining the signal. In the side-channel context, a CWT can be applied to the traces before any other processing.[1] Since the signal manipulated is then in the CWT domain, an SNR computed on it will highlight where the key-dependent signal lies across both time and frequency. This allows removing frequencies and time samples that are not signal-dependent, making it useful for noise reduction.

Correlation power analysis Correlation power analysis (CPA) is exploiting Pearson's Correlation $\hat{\rho}(\cdot,\cdot)$ between the observed traces $\boldsymbol{l}$ and leakage model of an intermediate variable X denoted $\boldsymbol{M}_x$ [BCO04]. X is key-dependent and in the case of the AES, it is usually one output byte of the first Sbox layer. The inferred key byte k_i is the one leading to the highest correlation such that

$$\hat{k_i} = \underset{k^* \in \mathcal{K}}{\operatorname{argmax}} \quad \hat{\rho}(\boldsymbol{M}_{x(k^*,p_i)}, \boldsymbol{l}), \tag{5.3}$$

with $x(k,p) = \mathsf{Sbox}(k \oplus p)$ and p_i the ith plaintext byte.

Informally, it is expected that the most accurate model will be the one of the correct key. The model can be selected based on engineering intuition (e.g., assuming the leakages to be proportional to the Hamming weight of x), which we denote as the non-profiled CPA. The model can be also profiled and corresponds to the estimated mean of the traces for each value of x. The profiled leakage model corresponds to the vectors $\boldsymbol{\mu}_x$ of a univariate Gaussian template attacks of Section 2.3. Compared to the profiled CPA, templates can exploit multivariate leakages. However, in a univariate setting with a sufficiently noisy environment, these two are equivalent [MOS11].

5.2 Targets and setup

First, this section gives a rationale behind the choice of two MCUs as well as their specificities. Second, it describes the conditions under which these were monitored during their security evaluation.

5.2.1 Targets

In this study, we are interested in the security of AES coprocessors in low-cost off-the-shelf components. We found out that `ARM Cortex-M4`

[1] Concretely, we only evaluate Equation 5.2 at a finite number of coordinates since exploring the entire continuous domain is unpractical.

devices are among the cheapest components with widespread AES hardware acceleration option. For diversity, we chose one component fulfilling these criteria from two well-known manufacturers in the MCU industry, namely NXP and STM32.

NXP. The selected MCU from NXP is the Kinetis K82 `MK82FN256-VLL15`. It comes with two cryptographic coprocessors. The one under investigation is the `LP Trusted Cryptography` module. It reports a countermeasure that inserts noise into the power consumption with a random mask [2]. A `DPAMaskSeed` register is present to reseed the core, which is advised after 50,000 encryptions. This target has been mounted on a custom Printed circuit board (PCB) in order to limit potential noise due to additional components.

STM32. The selected MCU from STM32 is the `STM32L422CB` [ST18]. The AES coprocessor of this processor does not have countermeasures against side-channel attacks mentioned. The target has been mounted on a custom PCB too.

5.2.2 Evaluation setup

In order to cover a good range of threat models, we evaluated the two aforementioned targets under different conditions, also reflecting the fact that such low-cost MCUs can be used for a wide range of applications, going from low-energy to more computationally intensive tasks.

The first parameter of our evaluations is the clock frequency. Targets were evaluated with a low clock frequency of 8[MHz] as well as close to their maximum frequency (i.e., 100[MHz] for the NXP target and 80[MHz] for the STM32 target). In both cases, the clock is derived from a 8[MHz] on-board crystal.

Second, both the EM emanations and the power consumption were recorded for the evaluations. These two signals are simultaneously measured by a `Picoscope 5244d` at 500[MSample/s] with an 8-bit resolution. The EM leakage was obtained using an H near-field probe from the HZ-15, probe set from Rohde&Schwartz. The EM probe used was impedance matched and preamplified using the Rohde&Schwartz HZ-16 preamplifier. Several positions were tested by hand for the probes and the position with the highest signal was kept. Therefore, the measurement was done above the target for the NXP MCU. For the STM32,

[2] See the device's reference manual: `https://www.nxp.com/webapp/Download?colCode=K82P121M150SF5RM`

no exploitable emanations were observed above the chip. Hence, the measurements were made thanks to the emanations of a power supply pin of the MCU.

The power consumption was measured through a shunt resistor and without amplification. The resistor is of 5[Ohm] for the NXP target and 10[Ohm] for the STM32 target. These values were chosen as high as possible, leading to a greater signal, but without triggering a brown out reset. The MCUs are accessing the AES cores using the Hardware abstraction layer (HAL) published by the manufacturers. The scope is triggered just before the HAL call.

5.3 Architecture inference

In absence of detailed specifications of the target hardware architectures, a first step in the following side-channel attacks is to infer a sufficient understanding enabling us to identify good target intermediate variables. We next describe our methodology for this purpose, followed by its results on the two targets.

At a high-level, we use a (fast) variant of the TVLA to locate the encryption and the execution of the first-round Sboxes. Then, we use the (slower) SNR to precisely identify the time samples corresponding to each key byte.

1. We start with the variant of the non-specific TVLA which we next denote as one-hot TVLA. We perform 16 well-chosen fixed-vs-fixed t-tests [DS16], such that the two sets of inputs induce a difference of a single byte in the first round of the AES encryption (hence the one-hot terminology). An independent t-test is then executed for each byte of the AES state. Overall, this requires only 17 sets of measurements as one of the sets is common across all the t-tests. An illustration of this method is given in Figure 5.1.

2. Second, and as a complement, the SNR is evaluated for each byte of the first Sbox layer, (detected thanks to the first step), in order to obtain the POIs for each of the Sbox outputs, that are then considered for profiling.

Note that the one-hot TVLA lies between specific and non-specific t-tests. A non-specific t-test leaks everywhere on the leakage trace and is not suitable for POI detection. Non-specific t-tests can be fixed-vs-random as in [GJJR11, CMG+] or fixed-vs-fixed as in [DS16]. They typically allow faster leakage detection thanks to their reduced number

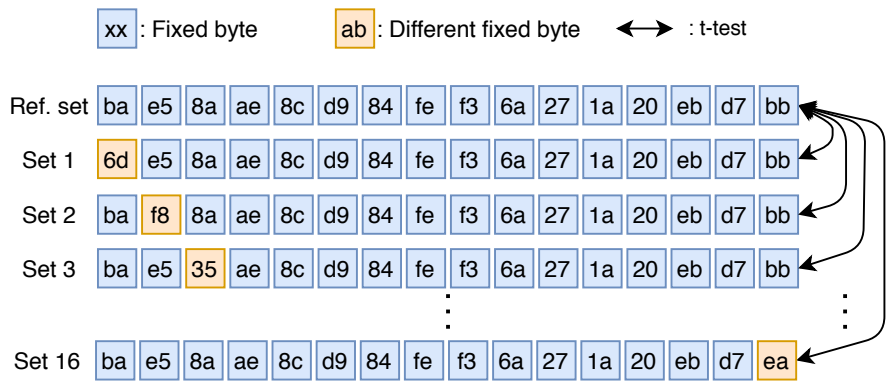


Figure 5.1: Illustration of the one-hot TVLA.

of classes. By contrast, specific t-tests (like SNR computations) allow POI detection at the cost of a higher number of classes to estimate. Due to the structure of the one-hot TVLA, leakage is detected only for the single byte which is different during the first operations of a block cipher. Then, for the later rounds, leakage is spotted everywhere due to the diffusion property. As a result, the one-hot TVLA is more specific for the first AES round and non-specific for the later rounds. By comparing the position of significant TVLA peaks for each byte, we can deduce the position of the first round of the AES.

Note also that such an intermediate between non-specific and specific tests was already proposed. For example, the semi-fixed vs. random test in [CMG+] combines a random set and one set with "well-chosen" values for similar reasons. Yet, the one-hot approach has two advantages compared to this previous proposal. First, the semi-fixed vs. random test will become specific as the size of the semi-fixed test increases (while the one-hot TVLA works with two classes per target byte, which can reduce its data complexity). Second, the semi-fixed vs. random test works as long as the model assumptions used to select the "well-chosen" values are correct (while such good model assumptions may not be available at this stage of an evaluation and are anyway not desirable for detection).

The combination of the two proposed steps can be an interesting trade-off for evaluators. While a good part of the most informative points' positions can be identified with the one-hot TVLA, it remains that the corresponding peaks do not have the quantitative meaning that the SNR carries, as for example discussed in [DS16]. Besides, while the first step could directly be based on the SNR, the TVLA has the advantage of requiring a small data and time complexity to be evaluated [SM16]. Since the SNR is not computed on the whole trace but only on

the first round, it reduces the time/memory complexity required for the detection. For example, when performed on traces of the same size, the one-hot TVLA requires about 100 times less memory than the SNR computed for 256 classes.

5.3.1 NXP Kinetis

The results of the methodology presented above are shown in Figure 5.2 with the mean trace on Figure 5.2a. We observe that a greater signal is present from samples 30,000 to 40,000. This is equivalent to 160 MCU cycles, leading to the suggestion of an implementation serialized on 1 Sbox. This will be verified with measurements as NXP does not provide the cycle count for this AES core.

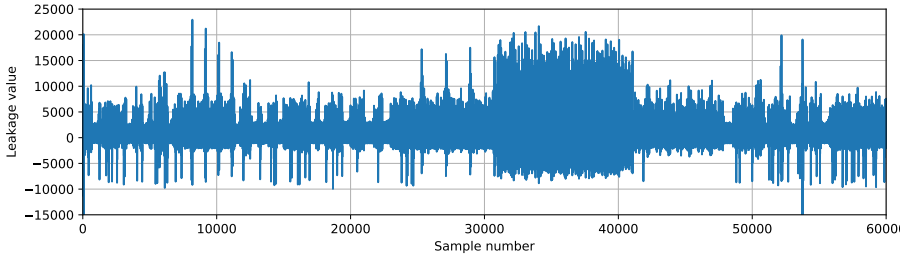
We observe that no leakage is spotted with the one-hot TVLA up to sample 25,000. This first part of the trace corresponds to the key scheduling of the AES (which we confirmed by using a set of inputs with different keys). More interestingly, three distinctly leaking parts can then be highlighted, confirming the interest of the one-hot TVLA: first, the loading of the plaintext around sample 27,000; second, the key addition and the Sbox execution (here given for an exemplary byte) which correspond to the two peaks between samples 28,000 and 32,000; finally, all the peaks after sample 33,000 which is where the test starts to be non-specific due to the diffusion happening after the first AES round.

We next estimated the SNR of the 16 AES Sboxes for all the samples corresponding to the first AES round identified with the one-hot TVLA. The results of Figure 5.2c and Figure 5.2d show that each Sbox leads to well identified peaks and these peaks are spaced by a single cycle. Information about all the bytes is therefore available. By computing the SNR on the second round Sbox, we finally observe on Figure 5.3a that no cycles are lost between rounds. The MixColumns and key addition operations are therefore interleaved with the Sboxes.

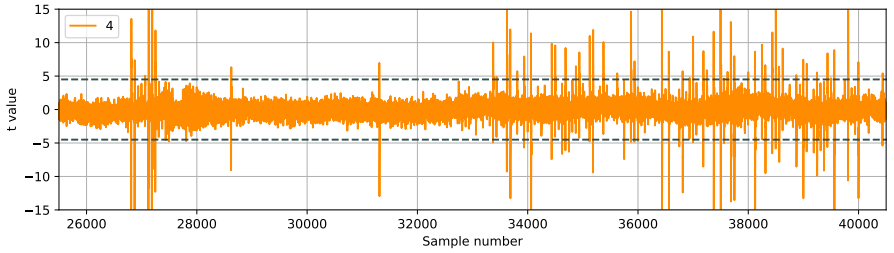
As a result, the architecture is inferred to be serialized on 8 bits for the Sboxes and the other operations such as MixColumns performed in parallel to the Sboxes.

5.3.2 STM32

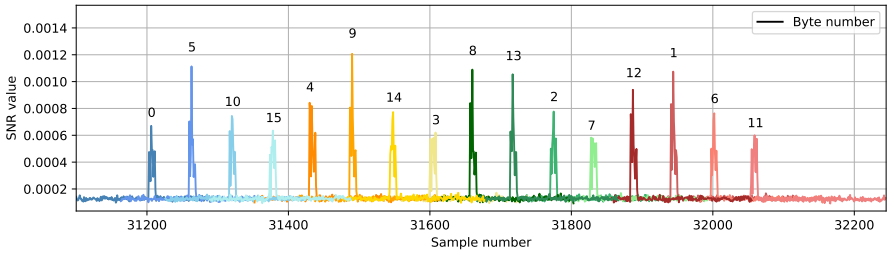
The same methodology was applied to the STM32 target and is reported in Figure 5.4. The mean power trace is given in Figure 5.4a. A repeating pattern is present and its length corresponds to the 214 MCU cycles



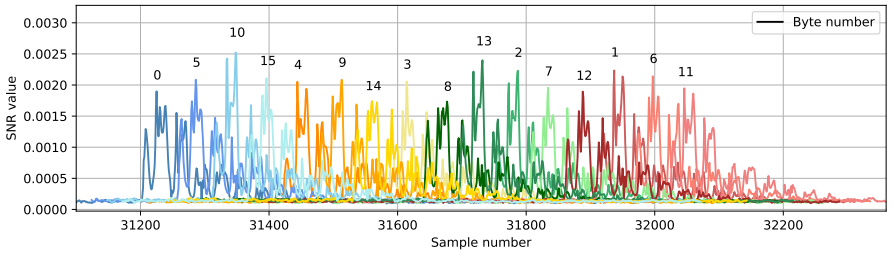
(a) Mean power leakage trace.



(b) Exemplary t-test on EM traces for byte 4.



(c) SNR on EM traces. Each peak is annotated with its corresponding byte number.

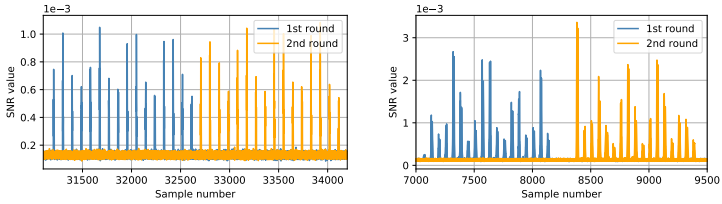


(d) SNR on power traces. Each peak is annotated with its corresponding byte number.

Figure 5.2: Architecture inference methodology on the NXP target (low clock freq.).

presented in the reference manual [3].[4] This again suggests an implemen-

[3]See : `https://www.st.com/resource/en/reference_manual/dm00151940-stm32l41xxx42xxx43xxx44xxx45xxx46xxx-advanced-armbased-32bit-mcus-stmicroelectronics.pdf`



(a) NXP **(b)** STM

Figure 5.3: SNR on first and second rounds for both devices

tation serialized on 1 Sbox. The application of the one-hot TVLA in Figure 5.4b leads to similar intuitions as for the NXP target and we can identify the plaintext loading (around sample 3500), the key addition and S-boxes (around samples 6000 and 7000) and the non-specific leakages after the first AES round (starting after sample 8,200).

These preliminary results are then confirmed with the SNR estimations of Figure 5.4c for EM measurements and Figure 5.4d for power ones. Compared to the NXP target, we notice a more significant disparity between the peak SNR levels of the different bytes (this difference is observed for both power and EM leakages). The largest SNRs on each byte are spaced by exactly one cycle. The SNR on a byte is also significant (but smaller) one cycle after the first peak (except for bytes 10 and 15 with power leakage). Eventually, by computing the SNR on the second round, we deduce that the first round and the second one are spaced by 5 cycles, as represented on Figure 5.3b.

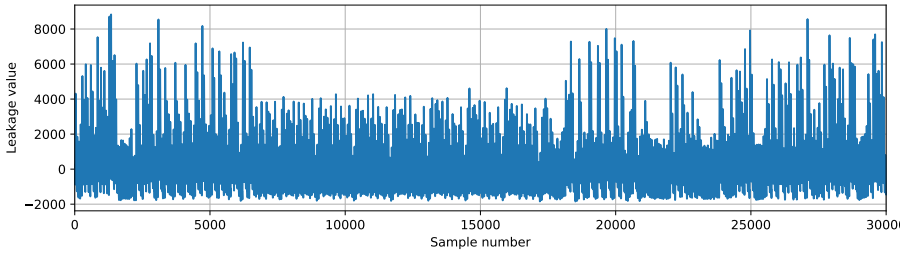
Putting things together, the architecture can be inferred to be serialized on 8 bits for the Sbox layer. The MixColumns operation and key addition are serialized on 32 bits and are not interleaved with Sboxes. Given that the operations are not interleaved, it could also explain the higher SNR levels with the STM32 platform as less algorithmic noise is present. Such an architecture might also leak information through the distance between two consecutive Sboxes. We tried to exploit such leakages but did not obtained better SNRs.

Overall, by comparing the STM32 and the NXP targets, we can conclude that the two manufacturers propose quite similar architectures.

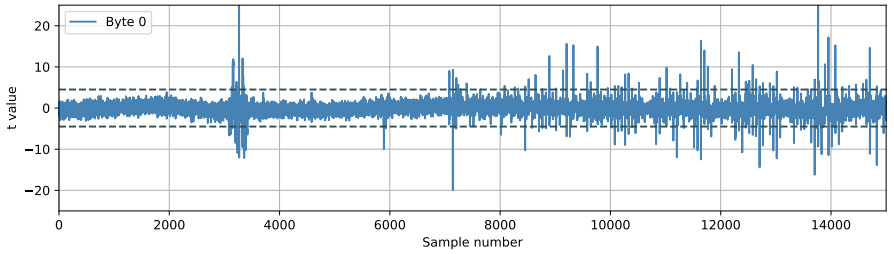
5.4 Attacks

In order to evaluate the side-channel security provided by the two targets, we performed key recovery with CPA with various leakage models

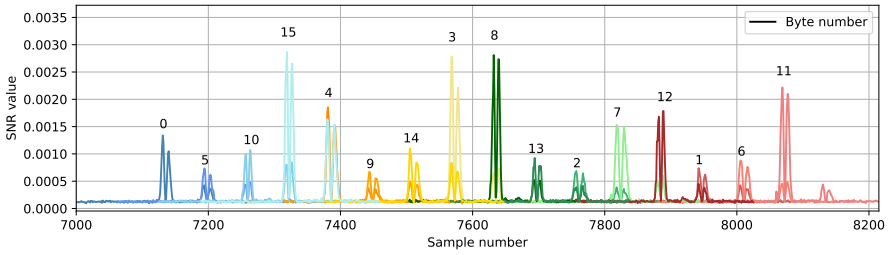
[4]This number corresponds to the AES core and excludes data loadings.



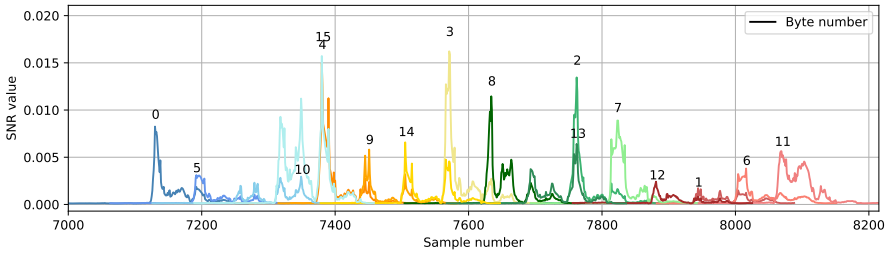
(a) Mean power leakage trace.



(b) Exemplary t-test on EM traces for byte 0.



(c) SNR on EM traces. Each peak is annotated with its corresponding byte number.



(d) SNR on power traces. Each peak is annotated with its corresponding byte number.

Figure 5.4: Architecture inference methodology on the STM32 target (low clock freq.).

and TA. Next, we first describe our attack strategy and then discuss attack results.

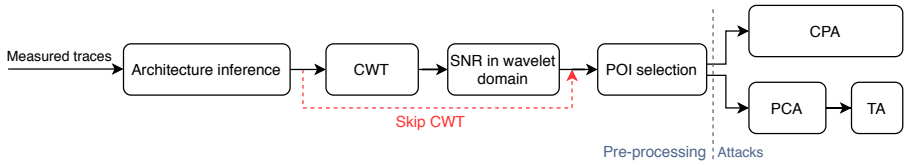


Figure 5.5: General attack strategy.

5.4.1 Attack strategy

The different steps of the presented attacks we performed are illustrated in Figure 5.5. All these steps (including the pre-processing) are performed independently on the key bytes. More precisely, these steps are described as follows.

- First, we use the results of the architecture inference from Section 5.3 to identify the most significant SNR peaks. For each attack, we reduce the trace length by two cycles before and four cycles after this peak.
- Second, the CWT (see Equation 5.2) is optionally applied to the shorter traces. When applied, the next steps are performed in the wavelet domain.
- If applicable, and in order to identify the POIs in the wavelet domain, we compute again the SNR on the previously obtained signal.
- In all cases, we only keep the points that have significant SNR values. This is done by filtering all the points with SNR smaller than $2 \cdot 10^{-4}$. This value depends on the noise in the measurements and the number of traces used for SNR calculation. In our case it corresponds with a small margin, to the peak SNR value observed where no signal is present (later called the noise floor)[5].
- Eventually, two attacks are performed on the pre-processed traces. The CPA from Equation 5.1 is directly applied. For the TA from Section 2.3, a PCA in additionally performed in order to reduce the signal to 5 dimensions. The subspace dimensionality was chosen as a compromise between the computational efficiency of the resulting attack and the amount of information extracted.

The above attacks all use a profiled leakage model. For the CPA, it corresponds to the mean leakage of each output value of the Sboxes.

[5]For example: Samples 7000 to 7100 in Figure 5.4c).

CPAs with non profiled leakage models were also performed and are discussed below.

5.4.2 Attacks results and discussions

We now compare different attacks against our two targets for both low and high clock frequencies and for both EM and power traces. In all cases, we used $2 \cdot 10^6$ profiling traces. The results of these attacks are summarized in Table 5.1. It contains the number of traces we need to reduce the rank of the 128-bit master key below 2^{32}, using the rank estimation algorithm of [PSG16].[6] As a complement, we also provide the graphs reporting the evolution of the guessing entropy for the best attack against each target in Appendix A.

We next discuss the influence of some parameters by looking at the minimum number of traces to reach a key rank lower than 2^{32}. We start with the targets' parameters and measurement setup, and follow with the attack strategy.

The **clock frequency** has a limited influence: for the NXP target, the best attack requires 3.5 more traces at high frequency than at low frequency; for the STM32 one, the best attacks are nearly equivalent at low and high frequencies. This is most likely due to the fact that these frequencies remain moderate and do not lead to particular challenges in terms of sampling frequencies.

The **power and EM** measurements do not differ strongly either (in terms of best attack complexity), both channels can lead to powerful key recoveries. Yet, the attack exploiting power leakage is roughly twice faster than the one based on EM traces in most of the cases. EM is only the best for the NXP target at high frequency, where it requires about 2.5 less traces than its power counterpart.

By contrast, **TA and the CPA** do not always lead to similar results. In most cases, the best attack is a CPA. Even though the TA is an optimal attack and should outperform the CPA, this can be explained by the limited number of traces we used for profiling. The generated template was probably too inaccurate, leading to a worse result. That is, the CPA is a univariate attack and only requires estimating mean values. The TA is a multivariate attack and in most cases, exploiting the additional dimensions is not useful given our profiling effort. Interestingly, the only case where TA outperform the CPA is with the power

[6] For CPA attacks, as they do not output probabilities but correlation scores, the rank estimation algorithm is not optimal and may not represent the worst-case [CPS16]. We then use it as a heuristic bound on the security level of our targets.

Device		2^{32} key rank		Full key recovery	
		CPA	TA	CPA	TA
NXP 8MHz Figure A.3	EM	27800	5200	N.A.	27400
	EM with CWT	3800	2400	7400	8800
	Power	6400	2000	15600	6200
	Power with CWT	12600	1200	27600	5400
NXP 100MHz Figure A.4	EM	26000	N.A.	N.A.	N.A.
	EM with CWT	4200	4400	11000	26600
	Power	N.A.	N.A.	N.A.	N.A.
	Power with CWT	10000	N.A.	N.A.	N.A.
STM32 8MHz Figure A.1	EM	16800	N.A.	N.A.	N.A.
	EM with CWT	3800	N.A.	11000	N.A.
	Power	4200	N.A.	17400	N.A.
	Power with CWT	1800	38000	5400	N.A.
STM32 80MHz Figure A.2	EM	30000	N.A.	N.A.	N.A.
	EM with CWT	2200	N.A.	7200	N.A.
	Power	1600	N.A.	7400	N.A.
	Power with CWT	1600	N.A.	6200	N.A.

Table 5.1: Number of traces needed for every attack. The maximum number of traces was set to 40k traces. The best attack for each target is highlighted in a different color. N.A. signifies the need for more than 40k traces to succeed.

traces of the NXP target. This quite nicely matches the intuition of Figure 5.2d where we see that the signal is less sparse in this case. So for this target, the additional information of the additional dimensions is sufficient to compensate a more expensive profiling.

Note that with more profiling efforts, TA should at least reach the level of performance of CPA [MOS11]). In view of the low complexity of the simple attacks we put forward in Table 5.1, we did not look for such further optimizations.

The **CWT** pre-processing is almost always improving attack complexity. The disparities observed between the different leakages are indeed reduced thanks to the CWT. We also note that the gain of the CWT in the EM case is of a factor between 4 and 10, while it never exceeds 2 in the power case. We assume this difference is due to the richer frequency content of the EM signals.

Although partially successful, our attacks without profiled leakage models were much worse. A combination of Hamming weight and Hamming distance models was necessary to recover the key on the STM32 target. Yet, the corresponding attack required 40,000 traces. As for

the NXP target, a profiled model was necessary with our measurements and we were not able to recover any byte of the key with 75,000 traces. This suggest that both accelerators have leakages that are only weakly correlated to a Hamming Weight predictions.

Finally, we note that the **NXP core countermeasure** was activated. Namely, all the experiments against the NXP target reported in this section were performed while reseeding the `DPA Mask Seed` register after 50,000 encryptions, as stated in the reference manual. This setting was compared with two other ones where we either did not reseed the register or we reseeded it with the same value before each encryption in order to cancel the randomization. While slightly modifying the shape of the measurements, these variations did not appear to imply noticeable variations of the corresponding security levels.[7]

In summary, and quite unsurprisingly given their similar architectures, the STM32 and NXP targets exhibit quite similar security levels and both can be broken with a few thousands traces with standard attack techniques. Interestingly, the orders of magnitude differences in data complexities show here the impact of the measurement setups as studied in Chapter 3 and the potential security overstatements.

5.5 Conclusions

Our results highlight that the lack of precise understanding of the target architectures does not prevent the simple identification of target intermediate variables in an implementation, directly leading to concrete attacks.

The results then exhibit the limited security against side-channel attacks that unprotected hardware coprocessors in mid-range 32-bit MCUs provide. Those coprocessors should therefore be viewed as performance improvers, not as a direct solution for securing an embedded implementation. However, in Chapter 8, we propose an alternative usage to such coprocessors by repurposing them as noise engines, in order to hide the signal of a software implementation.

[7] We are aware of two independent teams who observed similar results.

Single trace attacks on ISAP-A

6

ISAP [DEM+17, DEM+20] is an authenticated encryption scheme submitted to the NIST Light-weight Cryptography Standardization Process.[1] It comes with claims of improved resistance against side-channel attacks thanks to leakage-resilient features. Precisely, it embeds a re-keying process that mixes a long-term key with public data (e.g., nonces) at a low rate, which can be viewed as a permutation-based variant of the tree-based leakage-resilient Pseudorandom function (PRF) constructions discussed in [DP10, SPY+10, FPS12]. The main underlying idea of this construction is that it allows reducing the need to resist against DPA to the need to resist SPA. Since ISAP's re-keying is quite expensive, this idea is then used sparsely (i.e., for initialization and finalization only), in the spirit of a leveled implementations [PSV15]. Concretely, the relevance of this design therefore highly depends on the difficulty to prevent SPA.

Our goal in this chapter is therefore to mount an attack against a 32-bit software implementation of ISAP-A [DEM+20]. More precisely, we focus on the initialization of ISAP's re-keying which constitutes an interesting target since it heavily relies on the fact that the implementation of the Ascon-p permutation is secure against SPA. Besides, the Ascon-p permutation allows for a fully in-register implementation, which is in contrast with the Keccak permutation considered in [YK21], that requires many (leaky) memory accesses, and the ISAP-K implementation considered in [BBC+20] that was only 16-bit (and therefore more leaky as well).

As we aim to demonstrate a single-trace attack, it is crucial to extract as much information as possible from 32-bit leakages, which is enabled by the use of the efficient RLDA model of Chapter 4. Concretely, we target the first round of the initial permutation of ISAP, whose input is the long-term key and a fixed Initialization vector (IV). We profile all the states of the S-box layer using RLDA, then mount a SASCA with

[1] `https://csrc.nist.gov/projects/lightweight-cryptography`.

the factor graph of Figure 6.1, and we finally compute the rank of the correct key using the algorithm of [PSG16].

In the following sections, we first give more details on the target implementation and our attack methodology. We then present and discuss the attack results.

6.1 Attack target and methodology

Our attack target is the C reference implementation of the Ascon-p permutation running on a Cortex-M4 MCU compiled with optimizations.[2] We keep only the S-box computation in the trace and use the same side-channel acquisition setup as in Subsection 4.4.1. Each acquisition is repeated 100 times to give averaged traces that contain less noise, which is allowed by the threat model of ISAP. The initial state of the permutation is the initial state for the IsapRk (f = enc), that is: $(K_0, K_1, \mathrm{IV}_{\mathrm{KE}}, 0, 0)$ with $\mathrm{IV}_{\mathrm{KE}}$ = `0x01804001_0C01060C`. The permutation state is stored in 10 32-bit registers, which fits into the register file of the target. Hence, there is no register spilling, which would cause increased leakage. Each 64-bit word of state is split into two registers: the 32 LSBs and the 32 MSBs, that we refer to as the lower and upper parts of the states.

We used 100k averaged profiling traces to build a RLDA model with $p = 10$ for both parts of the non-zero variables in the Ascon S-box, plus the distance between the key words (i.e., $K_0 \oplus K_1$). After getting the distribution for all the target variables using RLDA, we run a SASCA using the graph in Figure 6.1 twice, setting the IV to the known constant: one for the lower part and one for the upper part of the state. Each SASCA is executed with two belief propagation iterations, which is sufficient to propagate all the variables information to the key and limits the instability of the loopy belief propagation algorithm.

We compare our RLDA-based attack with the state of the art by running the same attack with the marginalized 8-bit fragment templates model (mFT8) of [YK21], using the same parameters and set of attack and profiling traces as for the RLDA.

[2] `https://github.com/ascon/ascon-c/blob/33abe4bee86/crypto_aead/ascon128av12/ref/round.h`. We use `arm-none-eabi-gcc` version 9.2.1 with flags `-Os -mthumb -mcpu=cortex-m4`.

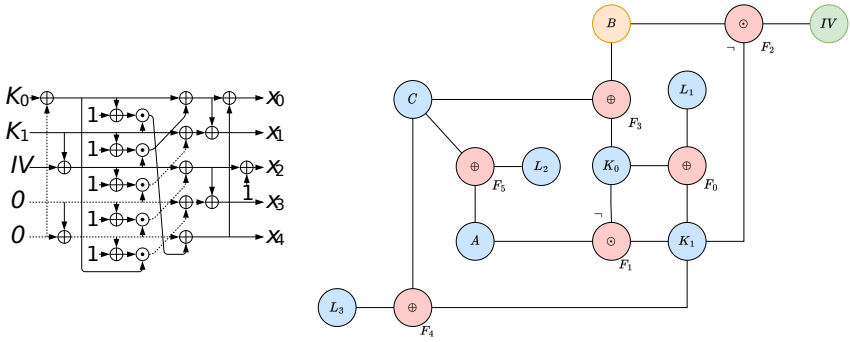


Figure 6.1: Ascon S-box for the initialization of ISAP' re-keying (the dashed lines are 0) and corresponding factor graph (the $\neg$, $\oplus$ and $\odot$ symbols are bitwise NOT, XOR and AND).

6.2 Attack results and discussion

We ran the attack on 6 randomly chosen keys, leading to the final key rank distributions shown in Table 6.1. For the RLDA model, after the SASCA, the correct key is within enumeration power (i.e., rank less than 2^{64}) in a majority of the cases. The rank of the key pre-SASCA (i.e., using only the leakage on the key words) is most of the time below 2^{70}, which remains within enumeration range for determined adversaries. We also observe a strong improvement of the ranks compared to fragment templates.

We additionally computed the PI of our RLDA models (under the same settings as Subsection 4.4.2 but reduced to $p = 4$, hence they are worse than the models we used for the attacks), which are shown in Table 6.2. We observe that the PI is sometimes lower and sometimes higher than the one of the experiment in Subsection 4.4.2. Two opposing effects can explain these differences. First, the pipeline is now full, leading to multiple states leaking simultaneously and therefore generating more computational/algorithmic noise. Second, the increased amount of computation on every state increases the leakage. In particular, computations such as K_1&IV create leakage on a subset of the bits of a state, adding information on K_1. We note that the mFT8 has lower PI than the RLDA (by roughly 1.5 to 8 bits, depending on the variable), which explains the worse key ranks.

We finally discuss the execution time of the attacks, when run on 8 cores of a AMD Epyc 7501 2.0 GHz CPU. First, the profiling of the RLDA takes less than 2 minutes for all 14 target variables. Then, the computation of the distributions from the RLDA for one attack trace

Table 6.1: Attack of a 32-bit ISAP-A implementation for 6 randomly chosen keys. The number given is the base 2 logarithm of the rank of the correct key.

	k_0	k_1	k_2	k_3	k_4	k_5
RLDA Pre-SASCA	63.9	69.4	70.6	67.4	69.4	65.2
RLDA Post-SASCA	51.3	62.0	61.1	66.3	74.3	48.2
mFT8 Pre-SASCA	89.5	92.7	101.8	92.7	104.6	77.7
mFT8 Post-SASCA	80.1	87.7	104.9	91.6	102.5	75.8

Table 6.2: PI bounds for profiled variables in the ISAP-A implem. ($p = 4$, $t = 0.5$).

Variable	$\mathrm{PI}^{\mathrm{RLDA}}_{\text{lower part}}$	$\mathrm{PI}^{\mathrm{mFT8}}_{\text{lower part}}$	$\mathrm{PI}^{\mathrm{RLDA}}_{\text{upper part}}$	$\mathrm{PI}^{\mathrm{mFT8}}_{\text{upper part}}$
K_0	9.918 ± 0.653	6.233	10.132 ± 0.575	5.204
K_1	15.162 ± 0.000	10.408	11.416 ± 0.175	6.897
A	14.853 ± 0.088	6.53	12.879 ± 0.297	4.91
C	8.726 ± 1.111	6.231	9.547 ± 0.805	5.13
L_1	8.485 ± 1.217	5.816	6.806 ± 1.830	5.046
L_2	7.904 ± 0.148	5.905	7.659 ± 0.143	4.706
L_3	8.087 ± 1.376	5.929	7.327 ± 1.640	5.12

takes less than 15 second per variable. The belief propagation for one attack is performed in about 2.7 h (it is single-threaded). As for the memory usage, the most expensive step is the SASCA, for which our (far from optimal) prototype implementation stored 35 distributions of 32 GiB each, summing to 1.13 TiB of RAM.

6.3 Conclusions and open problems

In this chapter, we used the optimized RLDA model as the basis for a single-trace SASCA against an implementation of the Ascon-p S-box on a 32-bit MCU. An important application for such an attack is the ISAP-A re-keying. Our results indeed show that the SPA security on which this re-keying relies can be attacked when implemented on unprotected 32-bit microcontrollers. Our current results target the initialization of the re-keying (which could be pre-computed) and open various directions for further research. Since our attack is bottlenecked by the SASCA, due to its heuristic nature and its time complexity, it would be interesting to investigate sparse and computationally efficient representations for distributions. This could help in growing the attacked states towards

64 bits, where computing full distributions is infeasible, and to extend our attacks beyond the initialization of ISAP's re-keying. In the next chapter, we analyze potential countermeasures that could provide additional security to re-keying based schemes like ISAP and render attacks like the one in this chapter impossible.

Combining countermeasures with re-keying

7

In the previous chapter, we presented an SPA attack against ISAP-A, initiated with the Ascon permutation. ISAP also proposes to use Keccak at its base, which has already been attacked by Bellizia et al. in [BBC+20] and Kannwischer et al. in [KPP20b]. In the latter, Kannwischer et al. suggested that SPA security on low-end microcontrollers could be obtained by combining the re-keying of the ISAP design with algorithmic-level countermeasures like masking, which has been applied to the Keccak permutation in [GSM17], or shuffling [HOM06, VMKS12], for which the application to bitslice permutation-based designs remains to be investigated.

Combining countermeasures is a popular idea in the side-channel literature. In general, the hope is that the complexity to attack a combined countermeasure will be the product of the complexities to attack its components. For example, in case side-channel measurements are sufficiently noisy, it is known that such a multiplicative effect happens when combining masking and shuffling [RPD09, ABG+22]. In this chapter, we question whether the same multiplicative effect takes place when combining re-keying with masking or shuffling, as proposed in [KPP20b].

For masking, we answer the question negatively in a definitive manner by showing that a divide-and-conquer side-channel attack of its combination with re-keying is always possible. That is, the complexity to attack a masked leakage-resilient PRF is only the sum of the complexities to (1) extract the useful signal of its masked state and (2) exploit this useful signal in an analytical attack. The latter can be explained by the fact that the useful signal of a masked state is the same as the useful signal of an unprotected state since masking can only amplify the noise of an implementation.[1] In other words, and independent of the level of noise in the measurements, combining re-keying with masking will never

[1] Concretely, it could even make the situation worse since the computational overheads of some masked computations (e.g., multiplications) could even increase the

lead to a multiplicative effect. At best, the masked implementation can become hard to attack. But in this case, the significant overheads of the re-keying scheme (which iterates the permutation n times to digest an n-bit value) becomes a waste, since this re-keying does not bring any significant additional security benefit.

For shuffling, the situation is different since it is known that its combination with a leakage-resilient PRF is at least theoretically founded [GPSG14]. Intuitively, the reason is that in case of sufficient noise, the shuffling countermeasure is modifying the shape of the signal since it emulates a parallel implementation that would combine (e.g., sum) the deterministic parts of multiple byte's leakage functions into a single leakage sample. So the security of this proposal boils down to the question whether the noise of a low-end embedded device such as considered in [BBC+20, KPP20b] and in the previous chapter is always sufficient for this emulation to take place. We answer the question negatively by describing a multivariate attack against a shuffled implementation of Keccak in an ARM Cortex M0. We show that we can recover its permutation in a single trace. It implies that a trivial side-channel dissection (in the sense of [BS20]) of the shuffling + re-keying combination is possible, leading to the same powerful single-trace attacks as without shuffling. So while shuffling ISAP in a more noisy device remains a good strategy in order to prevent averaging traces, it is not a sufficient one to gain high confidence in low-end embedded devices where shuffling can be the target of highly multivariate attacks that further circumvent the already low noise available on such devices.

So overall, our results mitigate the hope that countermeasures aiming to amplify the side-channel noise or to limit the side-channel signal always combine well when a leakage-resilient authenticated encryption scheme like ISAP is implemented on a low-end embedded software platform. They rather suggest that the best use cases for such schemes remain the ones where no additional countermeasures are needed, like larger parallel hardware implementations.

Related work. To some extent, our conclusion regarding masking and re-keying could be inferred from a previous work of Belaïd et al. [BGS15]. It concluded that the cost vs. security trade-off of a standalone leakage-resilient primitive is better than its combination with masking when the leakage is sufficiently bounded, and that masking alone is preferable otherwise. We consolidate this conclusion by exhibiting the poor (additive)

signal, which we do not investigate since quite implementation-specific and leading to the same conclusion that masking and re-keying do not combine well.

combination of complexities that such a mix implies in general. As for shuffling, it is also known that multivariate attacks can be quite damaging against them and the analysis in [GPSG14] was coming with a cautionary note in this direction. Yet, our results show the sensitivity of such security evaluations to small variations of the attack methodology. In particular, the main addition that we made compared to this previous analysis is to extract more information thanks to a dimensionality reduction step [SA08].

We finally note that we take the ISAP scheme as a case study, but our conclusions also apply to the application of single-trace attacks in the context of post-quantum cryptographic primitives investigated in [KPP20b].

7.1 Re-keying + masking

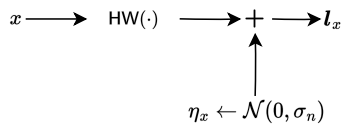
In order to break the re-keying scheme of ISAP with an SPA, the first step is to recover a maximum of information on each of the intermediate variables. To do so, the adversary is allowed to blend measurements, meaning that she can observe multiple leakages for the same secret input. The second step is to recombine this partial information on the intermediate variable to obtain a key guess [VGS14, KPP20b, BBC+20]. Next, we study the case where masking is combined with re-keying.

Since our goal is to show that re-keying and masking generically combine badly, we perform our investigations in a simulated setting where the noise level is tightly controlled. This allows us to illustrate our claims in an easily interpretable context and to show that they hold even in case masking is perfectly implemented (e.g., without independence flaws). We next present the parameters selected for our simulations and then discuss the obtained results.

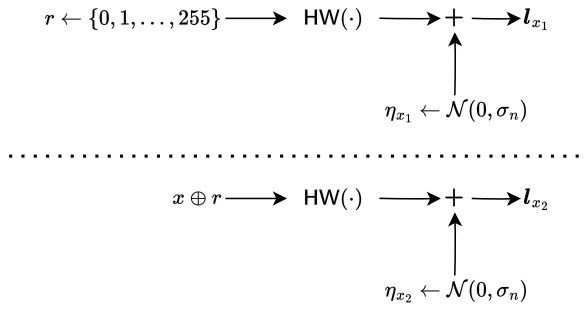
7.1.1 Simulation settings

For our simulations, the leakage of a given variable is its Hamming weight (HW) with additional Gaussian noise (as previously considered in the simulated setting of [KPP20b]). We consider different noise levels σ_n^2, such that the SNR of our leakages corresponds to 0.1, 1 and 10. Our simulation settings are summarized in Figure 7.1, where the sensitive variable x is an 8-bit word. For the unprotected setting (see Figure 7.1a), the leakage of each variable x is written as $\boldsymbol{l}_x$. For the first-order masking (see Figure 7.1b), the leakage on both shares x_1 and x_2 are respectively denoted as $\boldsymbol{l}_{x_1}$ and $\boldsymbol{l}_{x_2}$.

We performed our experiments 1000 times to average the results. For each experiment, we used 5000 measurements of the leakages $\boldsymbol{l}_x$, $\boldsymbol{l}_{x_1}$ and $\boldsymbol{l}_{x_2}$.



(a) Unmasked implementation.



(b) Masked implementation.

Figure 7.1: Simulation settings with Hamming weight leakages and Gaussian noise.

7.1.2 Security analysis

In the following, we will first detail how the adversary can blend measurements in both the masked and unmasked settings. Then, we detail what is his Success rate (SR) [SMY09] in recovering an intermediate variable x first for unmasked and then for masked implementation. The SR is the probability that the adversary recovers the correct value of the secret variable, which we denote as

$$\mathrm{SR}_x = \Pr[x = \hat{x}], \tag{7.1}$$

where the subscript notation defines the target variable (in this case, x). We will show that both protected and unprotected implementation's asymptotic SR are equivalent, making the effect of masking and shuffling independent for the adversary.

In order to blend independent measurements, and to exploit multiple traces that manipulate the same secret x, the adversary can use Equa-

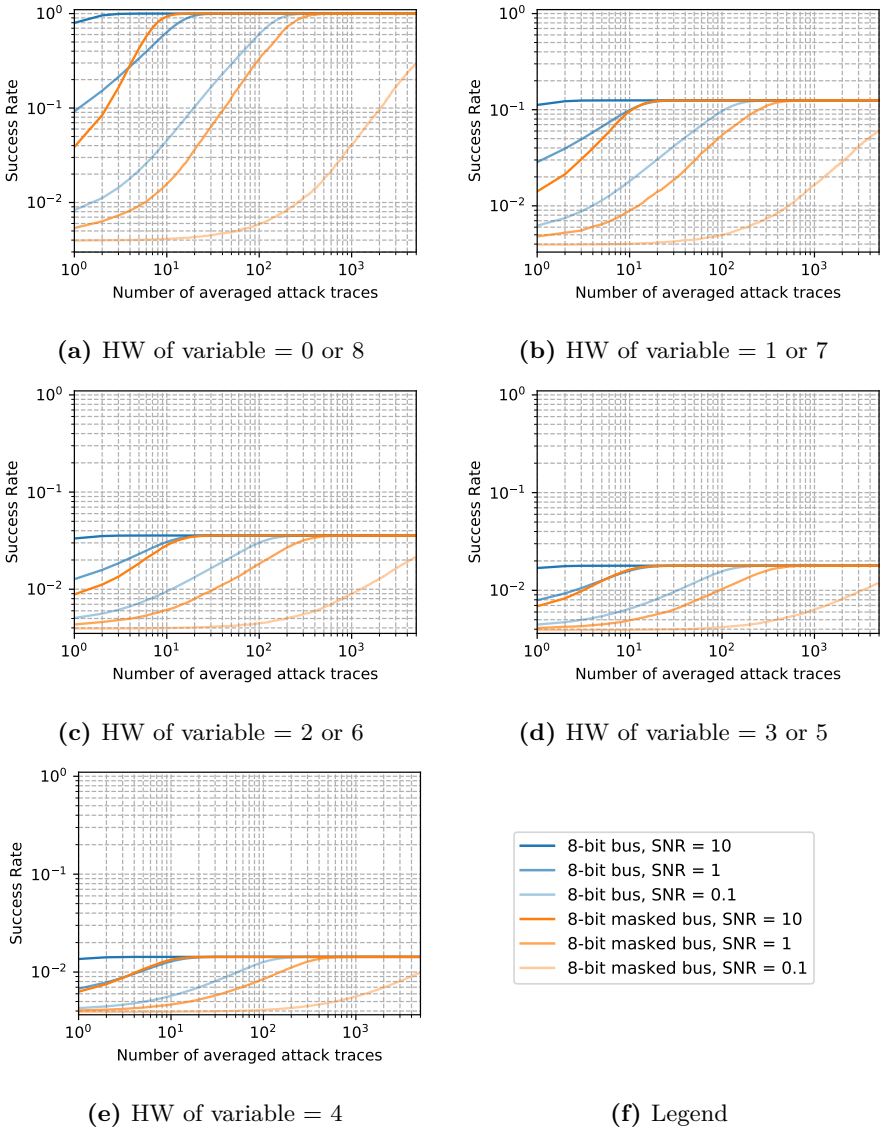
tion 2.3 to combine likelihoods of each leakage trace $\boldsymbol{l}^*$. In the masked case, Equation 2.13 has to be used.

Unmasked implementation. In the unmasked setting, computing Equation 2.3 is equivalent to directly averaging all the leakage traces. It allows recovering the mean of the leakages $\hat{\boldsymbol{\mu}}(x)$, which is $\mathsf{HW}(x)$ in our simulation since it averages the additive Gaussian noise η (see Figure 7.1a). By averaging, the adversary can increase the SNR and have a tighter approximation of $\mathsf{HW}(x)$.

The success rate of this adversary is reported on the blue curves of Figure 7.2 for different Hamming weights and SNRs. Since we focus on a SPA setting, the amount of information that can be extracted from the leakage (and therefore the SR) indeed depends on this Hamming weight. The x-axis corresponds the number of averaged traces t, the y-axis is the SR_x and the different plots are for different $\mathsf{HW}(x)$. We observe that: (*i*) the SR_x first increases with t and then saturates: this is because the noise is averaged and so the estimate of leakage mean becomes more accurate up to the point where there is no noise left; (*ii*) lowering the SNR (i.e., adding noise), slows down the convergence of SR_x but does not affect its asymptotic value: this is because the asymptotic value of SR_x only depends on the side-channel signal (i.e., the deterministic part of the – here Hamming weight – leakage function); (*iii*) this asymptotic value is larger for extreme Hamming weights (e.g., it is worth 1 for the Hamming weights 0 and 8) and lower for the intermediate Hamming weights: this is because multiple values of x then lead to the same HW, making it impossible for the adversary to recover it with probability one. More precisely, SR_x is inversely proportional to the number of values with the same HW. For example, when $\mathsf{HW}(x) = 1$ (or $\mathsf{HW}(x) = 7$), the asymptotic SR_x is equal to $\mathrm{SR}_x = 1/8 = 0.125$ as we count 8 values on an 8-bit bus that have $\mathsf{HW}(x) = 1$ (resp., $\mathsf{HW}(x) = 7$).

Masked implementation. We now observe that moving to a masked implementation leads to essentially similar observations. For this purpose, we first report the histograms of the two-dimensional leakages corresponding to a masked encoding with two shares in Figure 7.3. The x-axis corresponds to the leakage on the first share ($\boldsymbol{l}_{x_1}$) and the y-axis corresponds to the leakage on the second share ($\boldsymbol{l}_{x_2}$). Each subplot is for a different Hamming weight.

Clearly, we observe that just as in the unprotected case where the adversary could distinguish 9 distributions corresponding to the 9 Hamming weights, we still have 9 different distributions in the masked case.



(a) HW of variable = 0 or 8

(b) HW of variable = 1 or 7

(c) HW of variable = 2 or 6

(d) HW of variable = 3 or 5

(e) HW of variable = 4

(f) Legend

Figure 7.2: Success rate of a value-recovery attack against unprotected and masked implementations at different noise levels and for different target values.

As a result, the side-channel signal that can be obtained when masking is the same as in the case of an unprotected implementation (up to the comment of Footnote 1) and this actually holds independently of the leakage function: masking amplifies the noise but does not affect the signal.

This fact is directly reflected in the orange SR_x curves of the masked implementation that are reported in Figure 7.2. On the one hand, the

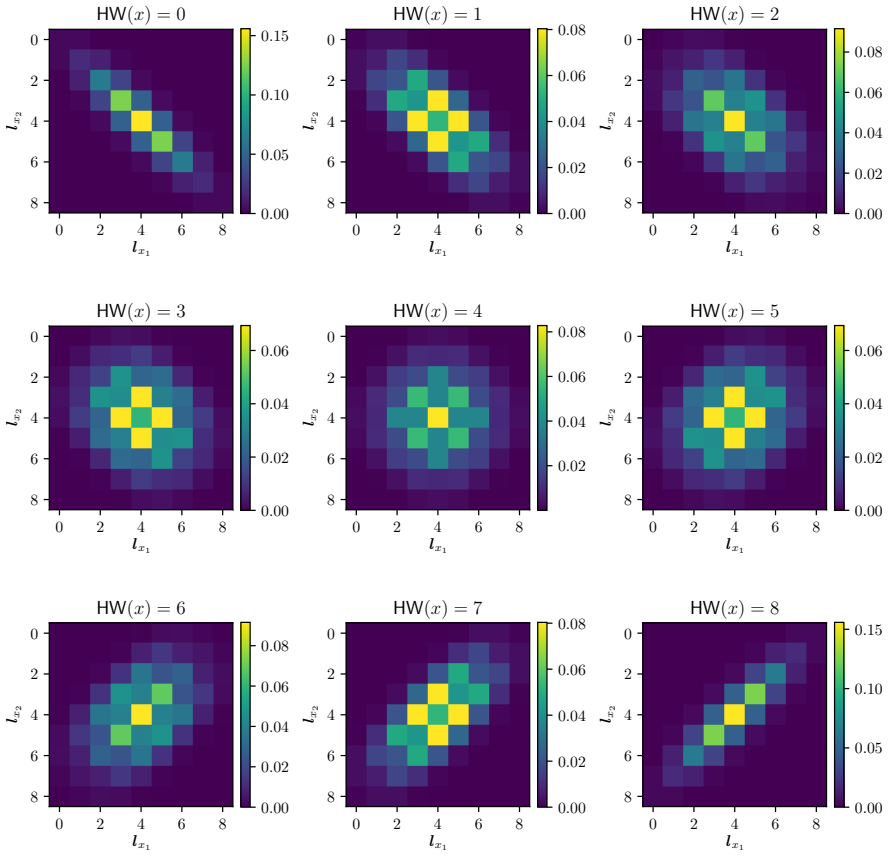


Figure 7.3: Masked PDFs $\hat{\mathsf{f}}[\boldsymbol{l}_{x_1}, \boldsymbol{l}_{x_2} | \boldsymbol{x}]$ for different $\mathsf{HW}(\boldsymbol{x})$ values and SNR 10.

asymptotic values are equal to the ones of the unprotected case. On the other hand, converging to this asymptotic value (i.e., extracting all the signal) will need more measurements as expected from masking security proofs [DFS15].

As a result, attacking a combination of masking and re-keying can always proceed in two steps with additive complexities. The first one is the attack against all intermediate values x to recover partial information. The number of traces required for each of them is impacted by masking but not the amount of signal that can be recovered asymptotically. The second one is the attack that recombines all these partial information with SASCA, which remains unchanged. So no multiplicative effect takes place in this combination of countermeasures which can

always be broken with a divide-and-conquer approach. As mentioned in the introduction of this chapter, this result is expected since the security of a PRF-based re-keying scheme depends on the amount of side-channel signal that a leaking implementation provides. Therefore, a countermeasure aiming at amplifying the noise cannot be of any help to satisfy this assumption: it can only make the signal extraction more difficult, as would be provided by masking used as a stand-alone countermeasure. This conclusion holds for any number of shares.

7.2 Re-keying + shuffling

Since combining re-keying with masking cannot lead to a strong (multiplicative) impact on the security of an implementation, we now consider shuffling as an alternative option. The situation is different in this case since in order to make the shuffling ineffective, the adversary has to recover the permutation $\boldsymbol{\pi}$ that is used to randomize the execution of the underlying operations in a single measurement. There is no possibility to combine multiple leakages for this part of the attack since the permutation is an ephemeral secret. In case of success, the situation is similar to an unprotected setting and the leakage of x can be averaged again. If not, the shuffling affects the shape of the leakage function, ultimately emulating a parallel implementation where the shuffled operations cannot be distinguished. While this ideal situation is not expected to happen [VMKS12] due to leakages on the permutation, the question we tackle in this section is whether shuffling always adds some complexity to the attack. We next show that in the case of a low-noise MCU (e.g., an ARM Cortex M0), the leakage may be enough to recover the whole permutation with a single trace, making the combination of shuffling and re-keying irrelevant.

7.2.1 Implementation and measurement setup

The previous published attacks against re-keying schemes focused on the Keccak permutation [KPP20b, BDPA] and its usage in the ISAP authenticated encryption scheme [BBC+20]. Next, we describe both our software implementation and the measurements setup that we keep as close to the one used in [BBC+20] as possible.

Regarding the software, we modified the reference implementation of ISAP [2]. We leveraged the shuffling with double indexing from Algorithm 3 and modified the Keccak implementation to combine shuffling

[2] `https://github.com/isap-lwc/isap-code-package`

and re-keying. More precisely, since a Keccak state contains 25 *lanes*, we generate permutations $\boldsymbol{\pi}$ with 25 elements and shuffle 25 independent operations.[3] The permutations are generated before the re-keying process and stored in memory. Therefore during the attack, the leakages $\boldsymbol{l}_{\pi_i}$ on the permutation indices π_i is independent of the permutation generation. They only result from the memory loadings and from other bijectively related leaking variables (e.g., memory addresses).

Regarding the measurement setup, our implementation is running on an STM32F0308 Discovery board, based on an ARM Cortex M0 as in [BBC+20]. An additional crystal was added to the board to provide a stable clock source for the side-channel measurements. Decoupling capacitors were also removed. We measured our traces thanks to the Tektronix CT-1 AC current probe and a Picoscope 5244D oscilloscope. The clock frequency of our MCU was set to the maximum value of 48MHz and we sampled at 500 MSamples/s.

7.2.2 Leakage modeling

In order to recover the permutation indices π_i, our adversary needs to obtain probabilities $\hat{\mathsf{p}}(\pi_i|\boldsymbol{l}^*)$. To do so, he uses the LDA-based templates from Subsection 2.5.2. More precisely, he first computes the SNR for each permutation index π_i [Man04]. Then, he selects the POIs for each of them by only considering samples above the noise floor. On average, we used $n = 450$ samples for each permutation index. For each of them, we kept the number of subspace dimensions p as a parameter.

7.2.3 Permutation index recovery

We first focus on an adversary attempting to recover each of the permutation indexes π_i within the permutation $\boldsymbol{\pi}$ independently, next denoted as $\mathcal{A}_1$. To do so, she uses the previously described templates. Namely, with a single leakage observation $\boldsymbol{l}^*$, she assumes a value $\hat{\pi}_i$ for the permutation index π_i with Equation 2.4. On Figure 7.4, we present the average SR_{π_i} of the first permutation within the shuffling implementation of ISAP described in Subsection 7.2.1. We observe that by increasing the number of dimensions in the linear subspace p, the SR_{π_i} increases. Its maximum is of 99.18% with $p = 15$ when all our 20,000 profiling traces are used. We can also see that the use of 5000 profiling traces and $p = 7$ is enough to obtain the asymptotic SR values.

[3] It is not always possible to find 25 independent operations within the Keccak round function. Yet, we will show that even in this best case (for the designer) where there are 25 independent operations, shuffling is ineffective.

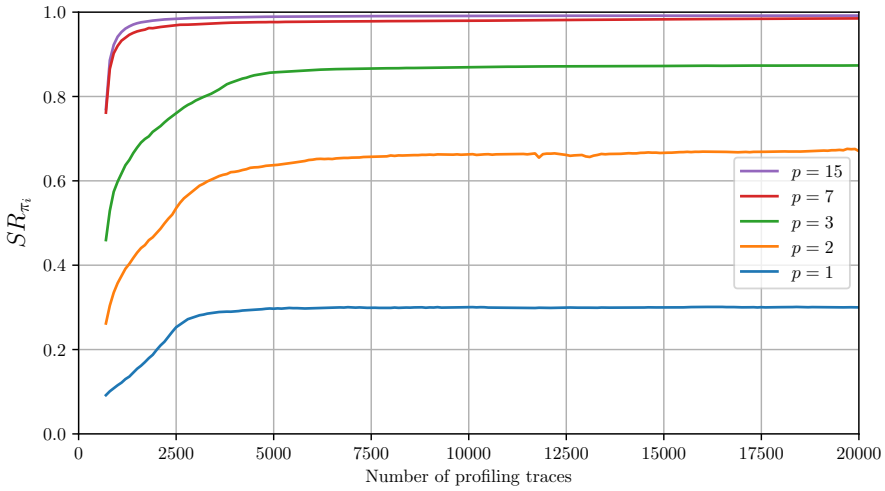


Figure 7.4: Estimated SR_{π_i} for $1 \leq p \leq 15$ and various number of profiling traces.

7.2.4 Full permutation recovery

We now discuss how to turn the information about the permutation indexes π_i into a full permutation recovery. To do so, we introduce the corresponding success rate $\mathrm{SR}_{\boldsymbol{\pi}}$ that is defined as

$$\mathrm{SR}_{\boldsymbol{\pi}} = \Pr[\pi_i = \hat{\pi}_i,\ \forall i]. \tag{7.2}$$

A first intuitive solution, denoted $\mathcal{A}_2$, is to derive an estimate for the full permutation $\hat{\boldsymbol{\pi}}$ by leveraging the $\mathcal{A}_1$ adversary and directly plugging the obtained π_i within $\hat{\boldsymbol{\pi}}$. The SR of these two adversaries are linked as $\mathrm{SR}_{\boldsymbol{\pi}} = \prod_i \mathrm{SR}_{\pi_i}$.

From this, we observe that even though the observed SR_{π_i} in Figure 7.4 is close to 100%, it is not sufficient to recover the full permutation with overwhelming probability. Namely we have that $\mathrm{SR}_{\boldsymbol{\pi}} \approx 0.9918^{25} \approx 0.8139$.

In order to improve these results, the adversary $\mathcal{A}_3$ that we describe in Algorithm 6 takes advantage of a characteristic of permutations which are under attack. Namely, $\forall i, j$ such that $i \neq j$ we have $\pi_i \neq \pi_j$ and this constraint is not enforced with the straightforward divide & conquer adversary $\mathcal{A}_2$. A simple option to exploit this constraint is to enumerate the $\hat{\boldsymbol{\pi}}$'s from the most to the least probable, and to keep as guess the first one being a permutation.

Various enumeration algorithms exist in the literature: $\mathcal{A}_3$ takes the key enumeration algorithm of [VGRS12] to instantiate Enumerate($\cdot$),

Algorithm 6 Permutation adversary $\mathcal{A}_3$.

Input: Lists for probabilities $\hat{\mathrm{p}}(\pi_i|\boldsymbol{l}^*)$ $\forall i$ and $\forall \pi_i$.
Output: Permutation guess $\hat{\boldsymbol{\pi}}$ with $\hat{\pi}_i \neq \hat{\pi}_j$ $\forall i, j$ and $i \neq j$.

1: **for** $\hat{\boldsymbol{\pi}} \leftarrow$ Enumerate$(\hat{\mathrm{p}}(\pi_0|\boldsymbol{l}^*), \hat{\mathrm{p}}(\pi_1|\boldsymbol{l}^*), \ldots)$ **do**
2: **if** $\hat{\pi}_i \neq \hat{\pi}_j$ $\forall i, j$ and $i \neq j$ **then**
3: **return** $\hat{\boldsymbol{\pi}}$

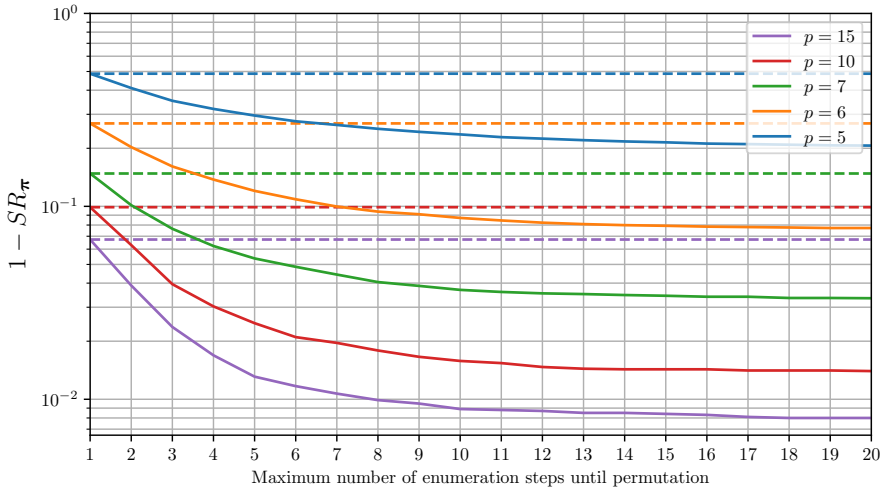


Figure 7.5: $1 - \mathrm{SR}_{\boldsymbol{\pi}}$ for different subspace dimensions p and as function of the enumeration depth. Dashed lines represent the value without enumeration.

which has the advantage to be optimal. The $\mathrm{SR}_{\boldsymbol{\pi}}$ of this adversary is reported in Figure 7.5 where the y-axis is $1 - \mathrm{SR}_{\boldsymbol{\pi}}$ in log-scale. The x-axis is the maximum number of permutations enumerated (i.e., the maximum number iterations of the loop) in Algorithm 6. The dashed horizontal lines are for a single iteration and correspond to $\mathcal{A}_2$. First, we observe that by enabling more enumeration steps, the SR increases. Second, we observe that the SR also increases with the number of dimensions p in the template's linear subspace. Putting things together, for the best templates ($p = 15$), we observe that $\mathrm{SR}_{\boldsymbol{\pi}}$ starts from 0.814 if no enumeration is allowed and increases up to 0.995 when setting the maximum number of steps to 20. Enumerating more was useless in our case, since with high probability, a permutation was found with at most 20 steps. Hence, the computational cost of the enumeration is negligible compared to the rest of the attack.

Our experiments therefore illustrate that recovering a leaking permutation on the considered low-end platform is possible with low complexity. An adversary can then cancel the impact of the shuffling countermeasure, making analytical attacks against re-keying like [KPP20b, BBC+20] possible. We finally note that various other attacks could be applied against the shuffling and refer to [VMKS12] for a survey.

7.3 Conclusions

As shown in the previously cited papers [KPP20b, BBC+20] and Chapter 6, single-trace attacks are an important threat against leakage-resilient PRFs such as used in the ISAP authenticated encryption scheme. The suggestion of Kannwischer et al. was to mitigate them by combining the re-keying scheme that leakage-resilient PRFs leverage with masking or shuffling. We first showed that combining such PRFs with masking is in general not a good idea as it just adds up the complexities to attack the two countermeasures separately. We then showed that combining these PRFs with shuffling, while theoretically appealing, can be practically challenging. For low-end embedded devices that are typical targets for such a combination (since they correspond to the targets of the single-trace attacks), implementing shuffling securely requires preventing multivariate side-channel attacks aiming at recovering the shuffling permutation (which are easy in low-noise settings). This second conclusion is not a general claim and shuffling could be effectively combined with re-keying on more noisy targets. But it shows that preventing single-trace attacks on low-end devices for which single-trace attacks are especially a concern is not trivial. Whether this can be achieved on the Cortex M0 device we analyzed is an interesting open question, which raises the challenge of hiding its leaky load/store operations. Overall, the combined results of this chapter and Chapter 6 suggest that ISAP may not be the best option to ensure side-channel security on low-end devices and finds a more natural application to more parallel (possibly hardware) architectures.

Noise generation on MCUs and its impact on masking

8

In Chapter 7, we showed the full recovery of a permutation of a shuffled implementation with a very high success rate. The noise levels of such low-end Cortex-M0 MCU being low, they contribute to the success of the attack. On a similar platform, a recent work showed that state-of-the-art masked implementations of the AES can be broken nearly independently of the security order[1] [BS21]. In both cases, the conclusion is that implementing countermeasures on such low-end devices is hard and costly as it requires large permutations for shuffling or a high number of shares for masking. Moreover, the serial nature of MCUs lets the adversary mount horizontal attacks that combine the leakages of multiple operations in order to further reduce the noise [BCPZ16], like in [BS21] and Chapter 6.

In this chapter, we are therefore interested in possibilities to increase the noise level of COTS devices. Our general idea relies on the presence of peripherals alongside the main CPU of recent MCUs, which could be run in parallel and provide additional algorithmic noise. Such an algorithmic noise could in turn make masking more effective and possibly reduce the number of shares needed to reach a given security level.

In order to evaluate the effectiveness of such a possibility, we run state-of-the-art attacks against a masked bitslice implementation of the AES with different number of shares. We show that running dummy coprocessor operations in parallel to the main (masked) computations indeed improves the noise level by a factor $f \approx 2$ in our experiments, leading to an increase of the best attacks' complexity by a factor $\approx 2^n$ for n shares. As a side result, we also show that the MCU we consider, which uses a more advanced technology than the one of Chapter 7 and [BS21], leads to slightly less informative leakages even without running coprocessors in parallel. These combined observations lead to improved

[1] Exploiting the low noise provided the best strategy over physical defaults like transitions that may show up in software implementations.

security for masked software implementations. Reaching high security levels remains expensive but overheads are reduced. For example, on out target, we were not able to attack a 4-share implementation with additional noise in less than one million traces.

We note that our work is focused on power measurements (or more generally, global leakages). This excludes advanced adversaries taking advantage of high-resolution localized measurements like [UHSS17, UHS+18], which generally work best in an invasive attack setting (i.e., after depackaging). We leave the investigation of such advanced attacks as an interesting open problem.

8.1 Noise Generation

In this section, we detail the implementation of our coprocessor-based noise engine. In order to be characterized as noise, the coprocessor should process hard-to-predict data, with ideally random states at each cycle. This way, an adversary is not able to predict the data and its leakage, which would make the noise deterministic and easy to filter. It should ideally be slow (i.e., take many cycles of computation) but quick to configure such that interrupts of the main task are sparse and short. The coprocessor should also rely on a large (parallel) architecture in order to generate more algorithmic noise and to better hide the leakage of the CPU that runs a masked implementation.

In practice, we configured a Direct Memory Access (DMA) peripheral which loads the buffer to process into to coprocessor and unloads it without the CPU being interrupted. The buffer can contain multiple blocks of data to process. When the buffer is fully processed, the CPU gets interrupted, the DMA and the coprocessor are reset and the cycle can restart. The buffer size therefore gives a trade-off parameter between the memory used and the overhead in execution time due to the interrupts. We note that the suitable coprocessors of the MCU we used did not allow circular DMA transfers.[2] Hence we needed to reset after processing the buffer. A representation of this process is shown in Figure 8.1.

Natural candidates that fit the requirements are cryptographic coprocessors. We choose for our experiments was an AES-128 implementation with a 128-bit architecture, performing the encryption of one block of data in 16 cycles (i.e., one cycle per round). In order for its states

[2]Which are DMA transfers where the DMA automatically resets its pointer to the start address at the end of the buffer, without input from the CPU.

to remain unpredictable, we put this coprocessor in a leakage-resilient mode of operation for which it is expected that a large parallel implementation provides sufficient security. Concretely, we used a construction similar to the one in [BMPS21] where it is shown that the 128-bit coprocessor maintains 64 bits of security as long as we re-key it every 100 encryptions. We used the True random number generator (TRNG) of the MCU to generate the random buffer and initializing the coprocessor with a random key. The TRNG generated a 32-bit random word every 40 cycles. After each completed buffer, we generate a new random key with the TRNG and encrypt the buffer with the new key. Overall, this strategy keeps performance overheads due to resets quite small (a few percents).

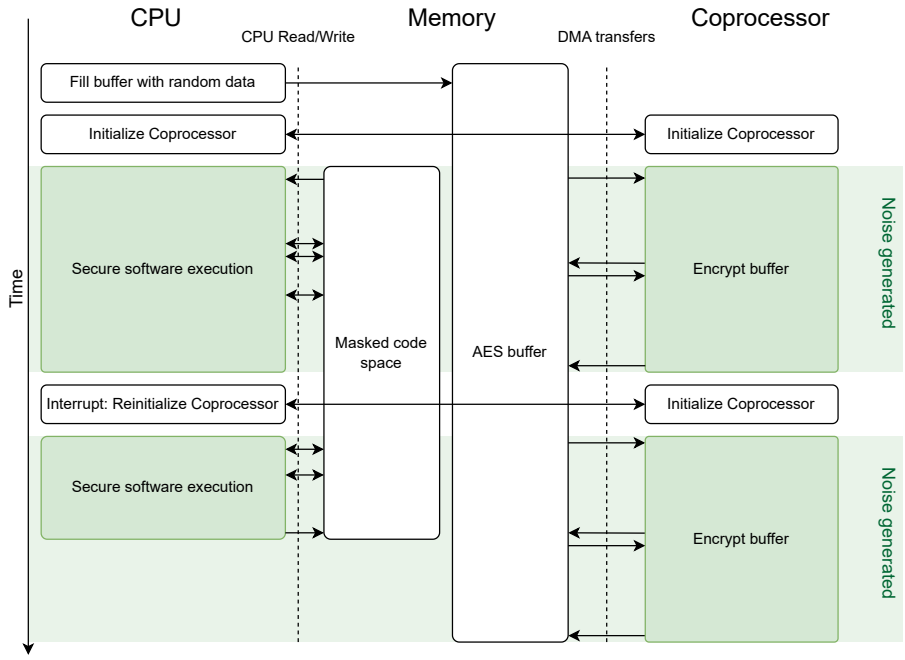


Figure 8.1: Execution scheme of the noise engine.

We note that reducing the buffer size would increase the computation time of our masked implementation, but improve the security of the leakage-resilient mode of operation running in parallel. It could also improve the security of the masked implementation due to the desynchronization of the traces that it would imply (since interrupts could then act as somewhat random delays).

8.2 Evaluation Setup

We now describe the implementation we analyze and our measurement setup.

8.2.1 Description of the Target

The implementation under test is the bitsliced AES implementation of Goudarzi and Rivain [GR17] with state-of-the-art PINI gadgets from [CS20]. We precisely used the assembly code provided in [BC22]. We compiled our code with `-os` to avoid compilation efforts that might alter the code up to the point of breaking side-channel countermeasures. Bitslicing allows efficient masked software implementations. It works by splitting each word and placing each bit into a different register, so that bitwise operations can be applied at the register level. Given the size of the CPU, we are not limited to one bit per register and can easily parallelize all the 16 S-boxes of the AES in 32-bit registers. In the case of masked implementations, each register is shared into n registers such that in one register, only bits of one share are present. This avoids recombining shares in the barrel shifter, also known as the shareslicing issue [GMPO20].

8.2.2 Measurement Setup

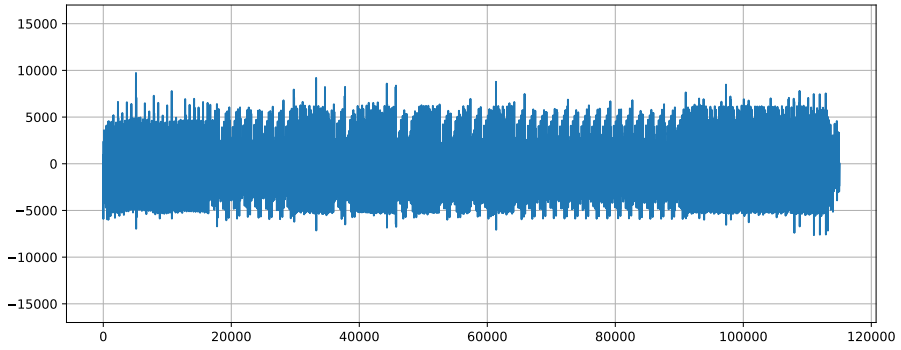
Our measurements were performed on a Chipwhisperer CW308 board with the `STM32F415RG` daughterboard. This board includes an ARM Cortex-M4 CPU with the required peripherals. We used a Tektronix CT-1 AC current probe on the dedicated measuring pins.[3] For sampling, we used the Picoscope 5244D 12-bit oscilloscope at its maximum speed of 500MS/S. The clock of our DUT was set at 40MHz and derived with internal PLLs from an 8MHz crystal on the CW308 board. We note that we also performed measurements to make sure our measurement setup produces results close to Bronchain and Standaert's measurement setup [BS21]. For this, we kept our setup identical with the exception of the daughterboard being a `STM32F051R4` with a Cortex-M0 CPU.

8.3 Side-Channel Metrics

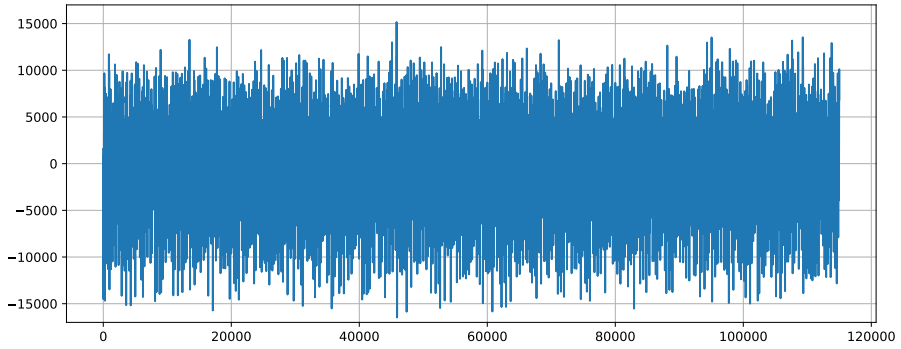
In this section, we compare two side-channel metrics, namely the SNR and the PI. Beforehand, we show in Figure 8.2 the effect of the coprocessor on the measurements by showing mean leakage traces with and

[3] This effectively shorts the onboard shunt resistor.

without the noise generation. The mean leakage traces were computed over 100k non-averaged traces and represent the execution of the first S-box layer of the AES. We clearly see the impact of the coprocessor: without it, groups of operations are distinguishable; when activated, they are hidden within the generated noise.



(a) Without noise generation.



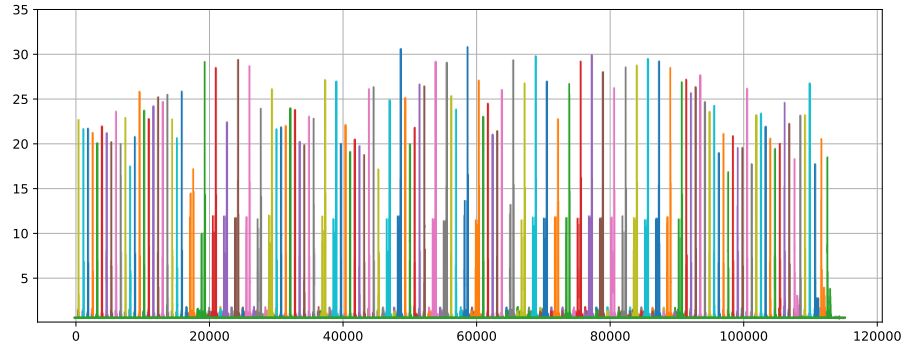
(b) With noise generation.

Figure 8.2: Comparison of mean leakage traces, based on 100k traces.

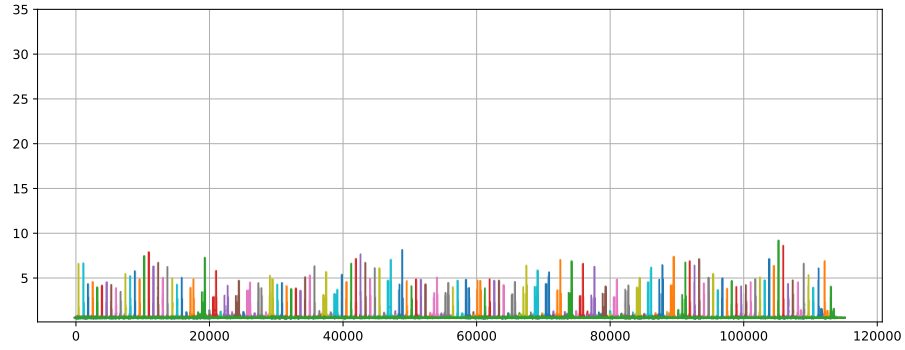
8.3.1 SNR Comparison

First, we compare the SNR with and without the noise engine for each state in the computation of the first S-box Layer. The target states have a 16-bit width, corresponding to the 16 S-Boxes computed in parallel. In each word, 16 bits correspond to the 16 S-boxes and the 16 remaining ones are set to zero. Thus, each bit of the bus is modeled and no algorithmic noise comes from the bus. Every state corresponds to a share of an intermediate variable and is processed independently. We used 500k traces with random inputs to compute the SNR.

In Figure 8.3, we show the (123×2) SNR curves calculated for each 16-bit intermediate state of the AES bitslice S-box for $n = 2$ shares, with (bottom) and without (top) noise generation. We observe two types of SNR peaks in the upper figure: the tighter ones are the XOR (and NXOR) gadgets, the more spread out, and slightly higher ones correspond to the AND gadgets. The impact of the noise generation appears on the bottom figure, where all time samples exhibit a reduced SNR. There is no significant alteration of the shape of the SNR curves and we observed a similar reduction for implementations with more shares.



(a) Without noise generation



(b) With noise generation

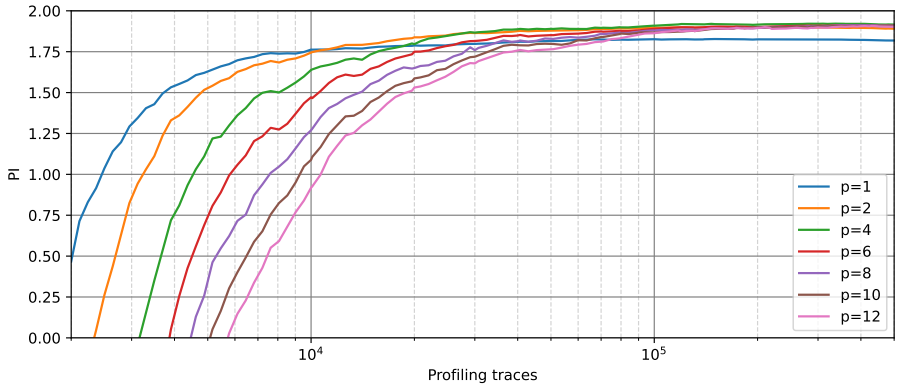
Figure 8.3: SNR for each share of the first S-box layer masked with two shares.

8.3.2 PI Comparison

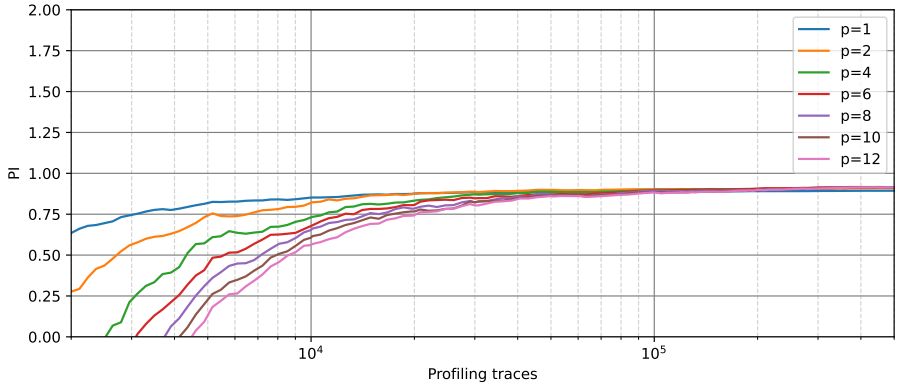
In order to evaluate the reduction of the PI due to the coprocessor, we first detail the steps followed to generate our leakage models. First, for each intermediate state, we find our POIs which we define as the time samples that are higher than the noise floor of the SNR. It leads to

around 2000 samples for each state share. We then profile these intermediate state shares with 500k traces using the RLDA leakage model on 16-bits, with 10 dimensions in the subspace. Finally, 5000 fresh traces are used to evaluate the PI.

To justify these parameters, Figure 8.4 shows the extracted information for an exemplary state share as a function of the number of profiling traces used, with and without the noise generation. It can be seen that all models converge with $\approx$ 100k traces. Furthermore, increasing the number of dimensions (e.g., beyond p=4) has limited impact, especially in the noisy case (bottom figure).



(a) Without noise generation



(b) With noise generation

Figure 8.4: PI per share obtained for an exemplary variable.

We next show the PI per share of the input of the S-box layer in Table 8.1, computed from a 2-share implementation. We observe that the 8 words of the AES state have slightly different levels of information,

but the noise produced by the coprocessor consistently reduces it. This is independent of the share that is being modeled. We also note that the PI per share of the linear operations is stable with the number of shares (since these operations are applied share-wise). Since our following attacks will primarily exploit this information, we do not detail the evaluation of the PI per share for the non-linear operations that takes place for larger numbers of shares (due to multiple shares manipulations). On average, we observed a reduction by an approximate factor 2 for the PI per share when activating the noise generation. Based on this value, we can extrapolate the security gains that the noise generation brings thanks to Equation 2.13: it should increase the data complexity by an approximate factor 2^n with n shares.[4]

	Share #	Word 0	Word 1	Word 2	Word 3
No noise	0	1.90	1.53	1.21	1.89
	1	1.97	1.64	1.29	1.97
With noise	0	0.95	0.71	0.42	0.72
	1	1.06	0.67	0.46	0.76
	Share #	Word 4	Word 5	Word 6	Word 7
No noise	0	0.81	2.00	2.47	3.12
	1	0.67	2.12	2.60	3.91
With noise	0	0.37	0.92	1.06	1.69
	1	0.30	1.10	1.12	1.78

Table 8.1: PI per share: input of the S-box layer, 2 shares, 16-bit RLDA model.

For completeness, we also computed the results obtained with two 8-bit LDA-based models, in order to compare them with the 16-bit RLDA-based model. As shown in Table 8.2, this leads to significant reduction of the perceived information. This last table is interesting since it uses a similar model as [BS21] and shows significantly lower PI values that this previous work obtained with a Cortex-M0 `STM32F0` MCUs. We repeated the measurements of [BS21] with our measurement setup (just plugging/unplugging the devices) and obtained similar results. So we posit that the changes are due to the higher complexity of the Cortex-M4 and a change of technology node, from 180nm for the F0 line to

[4]This is assuming that the independence condition holds to a sufficient degree. Since the gadgets we use were tested against such defaults in [BC22] and the possible reduction of the security order due to transitions is orthogonal to the noise issue we discuss, we did not reproduce this part of the experiments in the paper.

90nm for the more recent (and lower power) F4 line we evaluate in this chapter.[5]

	Share #	Word 0	Word 1	Word 2	Word 3
No noise	0	0.89	0.75	0.76	0.98
	1	0.98	0.96	0.86	1.05
With noise	0	0.62	0.49	0.35	0.52
	1	0.70	0.49	0.40	0.55
	Share #	Word 4	Word 5	Word 6	Word 7
No noise	0	0.57	0.87	1.25	1.56
	1	0.50	1.18	1.56	2.39
With noise	0	0.30	0.59	0.75	1.01
	1	0.26	0.79	0.77	1.21

Table 8.2: PI per share: input of the S-box layer, 2 shares, two 8-bit LDA models.

8.4 Attack results

In this section, we finally present concrete attacks against implementations with different masking orders, and discuss the effectiveness of the proposed noise generation. First, a baseline template attack is shown against the key addition. Next, we compare our results to the attack presented in [BS21]: a SASCA exploiting the leakages of all the intermediate states of the S-box layer.

This baseline attack is a textbook template attack on each of the 8 words of the key addition. We profile each share of each word as explained in Subsection 8.3.2, using the 16-bit RLDA models. Then, we recombine the likelihoods obtained on our shares to obtain the likelihood of the unmasked variable as shown in Equation 2.14. Figure 8.5 shows the results of attacks against 3 masked implementations, with 2, 3 and 4 shares. For each number of shares, we measured a dataset with and without noise generation. As a reference, we also ran the attack against an implementation without masking. The figure shows the mean rank (in bits) of the key as a function of the number of traces used in the attacks [SMY09]. We used the rank estimation algorithm presented in [PSG16]. The horizontal black line represents a rank of 32 bits, set

[5] https://blog.st.com/stm32g0-mainstream-90-nm-mcu/
https://www.st.com/en/microcontrollers-microprocessors/stm32f405-415.html

as a limit for the success of the attack (below that rank, enumeration is almost instantaneous).

First, looking at the dashed lines, the impact of masking is clearly visible. For each additional share, roughly 10 times more traces are needed. Next, the impact of the noise generation can be seen in the rightwards shift when moving from the dashed curves to the plain curves. As expected form our previous extrapolation, the security gain brought by the additional algorithmic noise is amplified as the number of shares increases. This is reflected by the gap between curves of the same color, that grows roughly as 2^n. With four shares, we were not able to reduce the key rank below 2^{32} with one million traces.

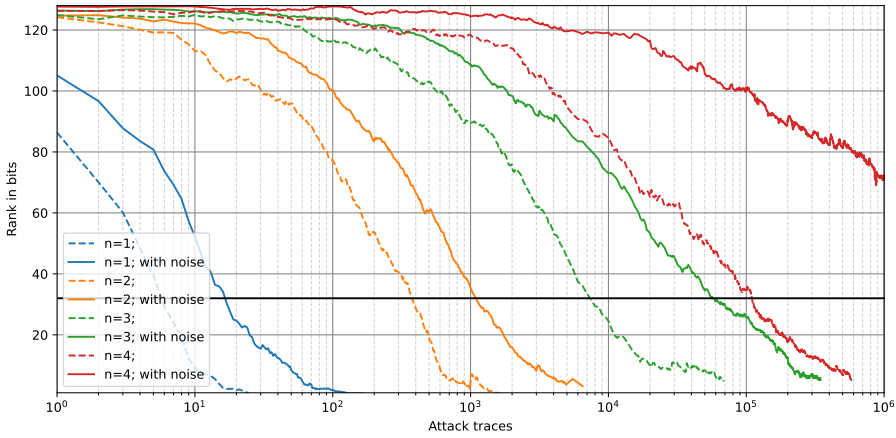


Figure 8.5: Key rank for $\neq$ number of shares, with and without noise generation.

For completeness, we repeated the SASCA of [BS21], where the adversary computes probabilities on the intermediate variables using Equation 2.14 and combines them thanks to BP. We performed this attack using the same datasets as the baseline attack. All variables were profiled using the same RLDA models and parameters. The factor graph of the AES being cyclic, we studied the number of iterations and observed the best results after 5 or 6 iterations, depending on the cases. Iterating the BP algorithm further lead to worse ranks on the key.

The results of the SASCA and baseline attacks are compared in Table 8.3, which gives the number of traces required to reach the rank of 2^{32} (mostly because the improvements are hardly distinguishable from Figure 8.5). The improvement of the SASCA over the baseline attack is limited and decreases with the masking order. We posit that this

limited effectiveness is due to the lower information obtained on the target intermediate variables combined with the information loss when propagating beliefs through XOR operations.

	$n = 2$		$n = 4$		$n = 4$	
Noise generation	Off	On	Off	On	Off	On
TA	373	1101	7500	36500	110500	NA.
SASCA	295	1024	7000	36000	109000	NA.
Gain in %	21	7	7	1.3	1.3	NA.

Table 8.3: Data complexity to obtain a 2^{32} rank with SASCA (vs. template attack).

8.5 Conclusions

The versatility of low-end MCUs makes them appealing solutions for deployment in various applications. Yet, their simplicity also makes them good targets for physical attacks. Previous works showed that their low-noise makes them vulnerable to horizontal side-channel analysis. In this chapter, we showed that the move to more advanced manufacturing technologies combined with the generation of algorithmic noise thanks to coprocessors running in parallel to masked software implementations can improve this situation with limited performance overheads. It would be interesting to further improve the side-channel security of software implementations by combining the noise engines we propose with other heuristic means to reduce the leakage (e.g., via time randomizations).

Conclusion 9

In this thesis, we showed that protecting MCUs against side-channel attacks is hard. This is essentially due to the low inherent noise of ARM Cortex processors, making algorithmic countermeasures very costly or ineffective. First we demonstrated attacks on an AES coprocessor with unknown specifications where basic attacks were already successful. Then, we attacked a re-keying based 32-bit software implementation with SPA and showed that masking and shuffling do not combine well with it. Finally, we ended on a positive note where we showed that coprocessors can be repurposed as noise engines with significant gains in attack complexity.

9.1 Leakage resilience and coprocessors

As we've shown that cryptographic coprocessors do not prevent side-channel attacks but only make them harder, similar behavior is expected with attacks targeting modes of operation without claims of side-channel resistance [OC16, BBB+22]. Re-keying based leakage resilient schemes like ISAP remain the most challenging targets due to their 32-bit implementation. As attacks like in Chapter 6 will certainly be improved in the future, it shows the limit of leakage-resilient schemes based on primitives implemented in software. Leveraging coprocessors for those leakage-resilient schemes is currently the most promising solution as studied in [BPPS17, BGP+20, USS+20, BMPS21]. We note here that ISAP instantiated with coprocessor based permutations would present similar characteristics.

If the above mentioned schemes can not be used in a design, noise engines with coprocessors at their base can be promising too in combination with other countermeasures. The bottom line being that today, secure and efficient design on MCUs cannot solely rely on coprocessors or software, but is best achieved with the usage of both.

From a more general point of view, MCUs with detailed open-source Hardware description language (HDL) source codes can circumvent analyses such as in Chapter 5, keeping the effort of a designer/evaluator away

from observed leakages with unexpected origins, typically glitches and transitions. Although not generalized, with Risc-V based MCUs, industry slowly tends towards such open-source HDL. The development of such MCU would be in fact the application of Kerckhoff's principle at a larger, implementation level, and one could expect that this can lead to a paradigm shift, similar to the one observed with cryptographic algorithms in 70s.

9.2 Open problems

SASCA We've shown both in Chapter 6 and Chapter 8 that our SASCA improved results but not drastically. We posit this to the PI over entropy ratio being smaller than in other works [BS21]. Studying the behavior of other message-passing methods in a similar scenario is an interesting path.

Sparse distributions RLDA allowed practical distribution calculation for 32-bit states. However, one distribution takes 32Gb of RAM, meaning that SASCA with multiple variables requires huge amount of memory. We observed that the distributions of our attack in Chapter 6 were highly sparse. In that sense, at 32-bits, a sparse distribution could have saved us more than 95% of RAM usage. Implementing a sparse BP could lead to more advances factor graphs and better attacks.

Noise generation is also an important subject. The proposed noise generation method in Chapter 8 is effective against adversaries measuring the global leakage of the MCU. However, the capabilities of more advanced adversaries with one or multiple high resolution EM probes might reduce the impact of such additional noise.

Publications used in this thesis

[BUS21] Davide Bellizia, Balazs Udvarhelyi, and François-Xavier Standaert. Towards a better understanding of side-channel analysis measurements setups. In *CARDIS*, volume 13173 of *Lecture Notes in Computer Science*, pages 64–79. Springer, 2021.

[CDSU23] Gaëtan Cassiers, Henri Devillez, François-Xavier Standaert, and Balazs Udvarhelyi. Efficient regression-based linear discriminant analysis for side-channel security evaluations towards analytical attacks against 32-bit implementations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2023(3):270–293, 2023.

[UBS21] Balazs Udvarhelyi, Olivier Bronchain, and François-Xavier Standaert. Security analysis of deterministic re-keying with masking and shuffling: Application to ISAP. In *COSADE*, volume 12910 of *Lecture Notes in Computer Science*, pages 168–183. Springer, 2021.

[US23] Balazs Udvarhelyi and François-Xavier Standaert. Leveraging coprocessors as noise engines in off-the-shelf microcontrollers. In *CARDIS*, Lecture Notes in Computer Science. Springer, 2023.

[UvWBS20] Balazs Udvarhelyi, Antoine van Wassenhove, Olivier Bronchain, and François-Xavier Standaert. On the security of off-the-shelf microcontrollers: Hardware is not enough. In *CARDIS*, volume 12609 of *Lecture Notes in Computer Science*, pages 103–118. Springer, 2020.

Other Publications

[BBB+20] Davide Bellizia, Francesco Berti, Olivier Bronchain, Gaëtan Cassiers, Sébastien Duval, Chun Guo, Gregor Leander, Gaëtan Leurent, Itamar Levi, Charles Momin, Olivier Pereira, Thomas Peters, François-Xavier Standaert, Balazs Udvarhelyi, and Friedrich Wiemer. Spook: Sponge-based leakage-resistant authenticated encryption with a masked tweakable block cipher. *IACR Trans. Symmetric Cryptol.*, 2020(S1):295–349, 2020.

[BBB+22] Francesco Berti, Shivam Bhasin, Jakub Breier, Xiaolu Hou, Romain Poussier, François-Xavier Standaert, and Balazs Udvarhelyi. A finer-grain analysis of the leakage (non) resilience of OCB. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(1):461–481, 2022.

[HMM+23] Clément Hoffmann, Pierrick Méaux, Charles Momin, Yann Rotella, François-Xavier Standaert, and Balazs Udvarhelyi. Learning with physical rounding for linear and quadratic leakage functions. In *CRYPTO*, Lecture Notes in Computer Science. Springer, 2023.

Bibliography

[ABG+22] Melissa Azouaoui, Olivier Bronchain, Vincent Grosso, Kostas Papagiannopoulos, and François-Xavier Standaert. Bitslice masking and improved shuffling: How and when to mix them in software? *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(2):140–165, 2022.

[APSQ06] Cédric Archambeau, Eric Peeters, François-Xavier Standaert, and Jean-Jacques Quisquater. Template attacks in principal subspaces. In *CHES*, volume 4249 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2006.

[BBB+20] Davide Bellizia, Francesco Berti, Olivier Bronchain, Gaëtan Cassiers, Sébastien Duval, Chun Guo, Gregor Leander, Gaëtan Leurent, Itamar Levi, Charles Momin, Olivier Pereira, Thomas Peters, François-Xavier Standaert, Balazs Udvarhelyi, and Friedrich Wiemer. Spook: Sponge-based leakage-resistant authenticated encryption with a masked tweakable block cipher. *IACR Trans. Symmetric Cryptol.*, 2020(S1):295–349, 2020.

[BBB+22] Francesco Berti, Shivam Bhasin, Jakub Breier, Xiaolu Hou, Romain Poussier, François-Xavier Standaert, and Balazs Udvarhelyi. A finer-grain analysis of the leakage (non) resilience of OCB. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(1):461–481, 2022.

[BBC+20] Davide Bellizia, Olivier Bronchain, Gaëtan Cassiers, Vincent Grosso, Chun Guo, Charles Momin, Olivier Pereira, Thomas Peters, and François-Xavier Standaert. Mode-level vs. implementation-level physical security in symmetric cryptography - A practical guide through the leakage-resistance jungle. In *CRYPTO (1)*, volume 12170 of *Lecture Notes in Computer Science*, pages 369–400. Springer, 2020.

[BBGV16] Arthur Beckers, Josep Balasch, Benedikt Gierlichs, and Ingrid Verbauwhede. Design and implementation of a waveform-matching based triggering system. In *COSADE*,

volume 9689 of *Lecture Notes in Computer Science*. Springer, 2016.

[BC22] Olivier Bronchain and Gaëtan Cassiers. Bitslicing arithmetic/boolean masking conversions for fun and profit with application to lattice-based kems. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(4):553–588, 2022.

[BCG+23] Julien Béguinot, Wei Cheng, Sylvain Guilley, Yi Liu, Loïc Masure, Olivier Rioul, and François-Xavier Standaert. Removing the field size loss from duc et al.'s conjectured bound for masked encodings. In *COSADE*, volume 13979 of *Lecture Notes in Computer Science*, pages 86–104. Springer, 2023.

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In *CHES*, volume 3156 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2004.

[BCPZ16] Alberto Battistello, Jean-Sébastien Coron, Emmanuel Prouff, and Rina Zeitoun. Horizontal side-channel attacks and countermeasures on the ISW masking scheme. In *CHES*, volume 9813 of *Lecture Notes in Computer Science*, pages 23–39. Springer, 2016.

[BDPA] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. The KECCAK reference. `https://keccak.team/files/Keccak-reference-3.0.pdf`.

[Ben75] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975.

[BGP+20] Francesco Berti, Chun Guo, Olivier Pereira, Thomas Peters, and François-Xavier Standaert. Tedt, a leakage-resist AEAD mode for high physical security applications. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(1):256–320, 2020.

[BGRV15] Josep Balasch, Benedikt Gierlichs, Oscar Reparaz, and Ingrid Verbauwhede. Dpa, bitslicing and masking at 1 ghz. In *CHES*, volume 9293 of *Lecture Notes in Computer Science*, pages 599–619. Springer, 2015.

[BGS15] Sonia Belaïd, Vincent Grosso, and François-Xavier Standaert. Masking and leakage-resilient primitives: One, the other(s) or both? *Cryptogr. Commun.*, 7(1):163–184, 2015.

[BHM+19] Olivier Bronchain, Julien M. Hendrickx, Clément Massart, Alex Olshevsky, and François-Xavier Standaert. Leakage certification revisited: Bounding model errors in side-channel security evaluations. In *CRYPTO (1)*, volume 11692 of *Lecture Notes in Computer Science*, pages 713–737. Springer, 2019.

[BMPS21] Olivier Bronchain, Charles Momin, Thomas Peters, and François-Xavier Standaert. Improved leakage-resistant authenticated encryption based on hardware AES coprocessors. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(3):641–676, 2021.

[BPPS17] Francesco Berti, Olivier Pereira, Thomas Peters, and François-Xavier Standaert. On leakage-resilient authenticated encryption with decryption leakages. *IACR Trans. Symmetric Cryptol.*, 2017(3):271–293, 2017.

[BS20] Olivier Bronchain and François-Xavier Standaert. Side-channel countermeasures' dissection and the limits of closed source security evaluations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(2):1–25, 2020.

[BS21] Olivier Bronchain and François-Xavier Standaert. Breaking masked implementations with many shares on 32-bit software platforms or when the security order does not matter. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(3):202–234, 2021.

[CB23] Gaëtan Cassiers and Olivier Bronchain. Scalib: A side-channel analysis library. *Journal of Open Source Software*, 8(86):5196, 2023.

[CBG+17] Thomas De Cnudde, Begül Bilgin, Benedikt Gierlichs, Ventzislav Nikov, Svetla Nikova, and Vincent Rijmen. Does coupling affect the security of masked implementations? In *COSADE*, volume 10348 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2017.

[CDP17] Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. Convolutional neural networks with data augmentation against

jitter-based countermeasures - profiling attacks without pre-processing. In *CHES*, volume 10529 of *Lecture Notes in Computer Science*, pages 45–68. Springer, 2017.

[CEM18] Thomas De Cnudde, Maik Ender, and Amir Moradi. Hardware masking, revisited. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(2):123–148, 2018.

[CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999.

[CK13] Omar Choudary and Markus G. Kuhn. Efficient template attacks. In *CARDIS*, volume 8419 of *Lecture Notes in Computer Science*, pages 253–270. Springer, 2013.

[CK14] Marios O. Choudary and Markus G. Kuhn. Efficient stochastic methods: Profiled attacks beyond 8 bits. In *CARDIS*, volume 8968 of *Lecture Notes in Computer Science*, pages 85–103. Springer, 2014.

[CLM20] Valence Cristiani, Maxime Lecomte, and Philippe Maurine. Leakage assessment through neural estimation of the mutual information. In *ACNS Workshops*, volume 12418 of *Lecture Notes in Computer Science*, pages 144–162. Springer, 2020.

[CMG+] Jeremy Cooper, Elke De Mulder, Gilbert Goodwill, Josh Jaffe, Gary Kenworthy, and Pankaj Rohatgi. Test vector leakage assessment (TVLA) methodology in practice (extended abstract). ICMC 2013. `http://icmc-2013.org/wp/wp-content/uploads/2013/09/goodwillkenworthtestvector.pdf`.

[CPS16] Marios O. Choudary, Romain Poussier, and François-Xavier Standaert. Score-based vs. probability-based enumeration - A cautionary note. In *INDOCRYPT*, volume 10095 of *Lecture Notes in Computer Science*, pages 137–152, 2016.

[CRR02] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In *CHES*, volume 2523 of *Lecture Notes in Computer Science*, pages 13–28. Springer, 2002.

[CS20] Gaëtan Cassiers and François-Xavier Standaert. Trivially and efficiently composing masked gadgets with probe isolating non-interference. *IEEE Trans. Inf. Forensics Secur.*, 15:2542–2555, 2020.

[dCGRP19] Eloi de Chérisey, Sylvain Guilley, Olivier Rioul, and Pablo Piantanida. Best information is most successful mutual information and success rate in side-channel analysis. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(2):49–79, 2019.

[DDF14] Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying leakage models: From probing attacks to noisy leakage. In *EUROCRYPT*, volume 8441 of *Lecture Notes in Computer Science*, pages 423–440. Springer, 2014.

[DEM+17] Christoph Dobraunig, Maria Eichlseder, Stefan Mangard, Florian Mendel, and Thomas Unterluggauer. ISAP - towards side-channel secure authenticated encryption. *IACR Trans. Symmetric Cryptol.*, 2017(1):80–105, 2017.

[DEM+20] Christoph Dobraunig, Maria Eichlseder, Stefan Mangard, Florian Mendel, Bart Mennink, Robert Primas, and Thomas Unterluggauer. Isap v2.0. *IACR Trans. Symmetric Cryptol.*, 2020(S1):390–416, 2020.

[DEMS21] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2: Lightweight authenticated encryption and hashing. *J. Cryptol.*, 34(3):33, 2021.

[DFS15] Alexandre Duc, Sebastian Faust, and François-Xavier Standaert. Making masking security proofs concrete - or how to evaluate the security of any leaking device. In *EUROCRYPT (1)*, volume 9056 of *Lecture Notes in Computer Science*, pages 401–429. Springer, 2015.

[DFS19] Alexandre Duc, Sebastian Faust, and François-Xavier Standaert. Making masking security proofs concrete (or how to evaluate the security of any leaking device), extended version. *J. Cryptol.*, 32(4):1263–1297, 2019.

[DHS01] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern classification, 2nd Edition.* Wiley, 2001.

[DK18] Daniel Dinu and Ilya Kizhvatov. EM analysis in the IoT context: Lessons learned from an attack on thread. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(1):73–97, 2018.

[DP10] Yevgeniy Dodis and Krzysztof Pietrzak. Leakage-resilient pseudorandom functions and side-channel attacks on Feistel networks. In *CRYPTO*, volume 6223 of *Lecture Notes in Computer Science*, pages 21–40. Springer, 2010.

[DS16] François Durvaux and François-Xavier Standaert. From improved leakage detection to the detection of points of interests in leakage traces. In *EUROCRYPT (1)*, volume 9665 of *Lecture Notes in Computer Science*, pages 240–262. Springer, 2016.

[DSE+12] Nicolas Debande, Youssef Souissi, M. Abdelaziz Elaabid, Sylvain Guilley, and Jean-Luc Danger. Wavelet transform based pre-processing for side channel analysis. In *MICRO Workshops*, pages 32–38. IEEE Computer Society, 2012.

[EKM+08] Thomas Eisenbarth, Timo Kasper, Amir Moradi, Christof Paar, Mahmoud Salmasizadeh, and Mohammad T. Manzuri Shalmani. On the power of power analysis in the real world: A complete break of the KeeLoqCode hopping scheme. In *CRYPTO*, volume 5157 of *Lecture Notes in Computer Science*, pages 203–220. Springer, 2008.

[FPS12] Sebastian Faust, Krzysztof Pietrzak, and Joachim Schipper. Practical leakage-resilient symmetric cryptography. In *CHES*, volume 7428 of *Lecture Notes in Computer Science*, pages 213–232. Springer, 2012.

[fS19] International Organization for Standardization. It security techniques - test tool requirements and test tool calibration methods for use in testing non-invasive attack mitigation techniques in cryptographic modules - part 1: Test tools and techniques, Geneva (CH) 2019. ISO/IEC 20082-1.

[GGSB20] Qian Guo, Vincent Grosso, François-Xavier Standaert, and Olivier Bronchain. Modeling soft analytical side-channel attacks from a coding theory viewpoint. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(4):209–238, 2020.

[GJJR11] Gilbert Goodwill, Benjamin Jun, Josh Jaffe, and Pankaj Rohatgi. A testing methodology for side channel resistance validation. NIST non-invasive attack testing workshop, 2011. `http://csrc.nist.gov/news_events/non-invasive-attack-testing-workshop/papers/08_Goodwill.pdf`.

[GMPO20] Si Gao, Ben Marshall, Dan Page, and Elisabeth Oswald. Share-slicing: Friend or foe? *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(1):152–174, 2020.

[GMS+11] Sylvain Guilley, Houssem Maghrebi, Youssef Souissi, Laurent Sauvage, and Jean-Luc Danger. Quantifying the quality of side-channel acquisitions. In *COSADE*, pages 16–28, 2011.

[GP99] Louis Goubin and Jacques Patarin. DES and differential power analysis (the "duplication" method). In *CHES*, volume 1717 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 1999.

[GPSG14] Vincent Grosso, Romain Poussier, François-Xavier Standaert, and Lubos Gaspar. Combining leakage-resilient PRFs and shuffling - towards bounded security for small embedded devices. In *CARDIS*, volume 8968 of *Lecture Notes in Computer Science*, pages 122–136. Springer, 2014.

[GR17] Dahmun Goudarzi and Matthieu Rivain. How fast can higher-order masking be in software? In *EUROCRYPT (1)*, volume 10210 of *Lecture Notes in Computer Science*, pages 567–597, 2017.

[GSM15] Daniel Gruss, Raphael Spreitzer, and Stefan Mangard. Cache template attacks: Automating attacks on inclusive last-level caches. In *USENIX Security Symposium*, pages 897–912. USENIX Association, 2015.

[GSM17] Hannes Groß, David Schaffenrath, and Stefan Mangard. Higher-order side-channel protected implementations of KECCAK. In *DSD*, pages 205–212. IEEE Computer Society, 2017.

[HOM06] Christoph Herbst, Elisabeth Oswald, and Stefan Mangard. An AES smart card implementation resistant to power analysis attacks. In *ACNS*, volume 3989 of *Lecture Notes in Computer Science*, pages 239–252, 2006.

[HZ12] Annelie Heuser and Michael Zohner. Intelligent machine homicide - breaking cryptographic devices using support vector machines. In *COSADE*, volume 7275 of *Lecture Notes in Computer Science*, pages 249–264. Springer, 2012.

[ISW03] Yuval Ishai, Amit Sahai, and David A. Wagner. Private circuits: Securing hardware against probing attacks. In *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, 2003.

[KDB+22] Satyam Kumar, Vishnu Asutosh Dasu, Anubhab Baksi, Santanu Sarkar, Dirmanto Jap, Jakub Breier, and Shivam Bhasin. Side channel attack on stream ciphers: A three-step approach to state/key recovery. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(2):166–191, 2022.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[Koc96] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *CRYPTO*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.

[KPP20a] Matthias J. Kannwischer, Peter Pessl, and Robert Primas. Single-trace attacks on keccak. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(3):243–268, 2020.

[KPP20b] Matthias J. Kannwischer, Peter Pessl, and Robert Primas. Single-trace attacks on Keccak. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(3):243–268, 2020.

[LBS19] Itamar Levi, Davide Bellizia, and François-Xavier Standaert. Reducing a masked implementation's effective security order with setup manipulations and an explanation based on externally-amplified couplings. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(2):293–317, 2019.

[LYS+15] Junrong Liu, Yu Yu, François-Xavier Standaert, Zheng Guo, Dawu Gu, Wei Sun, Yijie Ge, and Xinjun Xie. Small tweaks do not help: Differential power analysis of MILENAGE implementations in 3g/4g USIM cards. In *ESORICS (1)*, volume 9326 of *Lecture Notes in Computer Science*, pages 468–480. Springer, 2015.

[Man04] Stefan Mangard. Hardware countermeasures against DPA ? A statistical analysis of their effectiveness. In *CT-RSA*, volume 2964 of *Lecture Notes in Computer Science*, pages 222–235. Springer, 2004.

[MBKP11] Amir Moradi, Alessandro Barenghi, Timo Kasper, and Christof Paar. On the vulnerability of FPGA bitstream encryption against power analysis attacks: extracting keys from Xilinx Virtex-II FPGAs. In *ACM Conference on Computer and Communications Security*, pages 111–124. ACM, 2011.

[MDP20] Loïc Masure, Cécile Dumas, and Emmanuel Prouff. A comprehensive study of deep learning for side-channel analysis. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(1):348–375, 2020.

[MKP12] Amir Moradi, Markus Kasper, and Christof Paar. Black-box side-channel attacks highlight the importance of countermeasures - an analysis of the xilinx virtex-4 and virtex-5 bitstream encryption mechanism. In *CT-RSA*, volume 7178 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2012.

[MMR20] Thorben Moos, Amir Moradi, and Bastian Richter. Static power side-channel analysis - an investigation of measurement factors. *IEEE Trans. Very Large Scale Integr. Syst.*, 28(2):376–389, 2020.

[Mor14] Amir Moradi. Side-channel leakage through static power - should we care about in practice? In *CHES*, volume 8731 of *Lecture Notes in Computer Science*, pages 562–579. Springer, 2014.

[MOS11] Stefan Mangard, Elisabeth Oswald, and François-Xavier Standaert. One for all - all for one: unifying standard differential power analysis attacks. *IET Inf. Secur.*, 5(2):100–110, 2011.

[MPP16] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking cryptographic implementations using deep learning techniques. In *SPACE*, volume 10076 of *Lecture Notes in Computer Science*, pages 3–26. Springer, 2016.

[MS16] Amir Moradi and Tobias Schneider. Improved side-channel analysis attacks on xilinx bitstream encryption of 5, 6, and 7 series. In *COSADE*, volume 9689 of *Lecture Notes in Computer Science*, pages 71–87. Springer, 2016.

[OC16] Colin O'Flynn and Zhizhang Chen. Power analysis attacks against IEEE 802.15.4 nodes. In *COSADE*, volume 9689 of *Lecture Notes in Computer Science*, pages 55–70. Springer, 2016.

[oST01] National Institute of Standards and Technology. Advanced encryption standard. *NIST FIPS PUB 197*, 2001.

[Pag02] D. Page. Theoretical use of cache memory as a cryptanalytic side-channel. Cryptology ePrint Archive, Paper 2002/169, 2002. `https://eprint.iacr.org/2002/169`.

[PS17] Santos Merino Del Pozo and François-Xavier Standaert. Getting the most out of leakage detection - statistical tools and measurement setups hand in hand. In *COSADE*, volume 10348 of *Lecture Notes in Computer Science*, pages 264–281. Springer, 2017.

[PSG16] Romain Poussier, François-Xavier Standaert, and Vincent Grosso. Simple key enumeration (and rank estimation) using histograms: An integrated approach. In *CHES*, volume 9813 of *Lecture Notes in Computer Science*, pages 61–81. Springer, 2016.

[PSKM15] Santos Merino Del Pozo, François-Xavier Standaert, Dina Kamel, and Amir Moradi. Side-channel attacks from static power: when should we care? In *DATE*, pages 145–150. ACM, 2015.

[PSV15] Olivier Pereira, François-Xavier Standaert, and Srinivas Vivek. Leakage-resilient authentication and encryption from symmetric cryptographic primitives. In *CCS*, pages 96–108. ACM, 2015.

[PW69] G. Peters and James Hardy Wilkinson. Eigenvalues of ax = λbx with band symmetric A and B. *Comput. J.*, 12(4):398–404, 1969.

[QS01] Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (EMA): measures and counter-measures for

smart cards. In *E-smart*, volume 2140 of *Lecture Notes in Computer Science*, pages 200–210. Springer, 2001.

[RPD09] Matthieu Rivain, Emmanuel Prouff, and Julien Doget. Higher-order masking and shuffling for software implementations of block ciphers. In *CHES*, volume 5747 of *Lecture Notes in Computer Science*, pages 171–188. Springer, 2009.

[RSWO17] Eyal Ronen, Adi Shamir, Achi-Or Weingarten, and Colin O'Flynn. IoT goes nuclear: Creating a ZigBee chain reaction. In *IEEE Symposium on Security and Privacy*, pages 195–212. IEEE Computer Society, 2017.

[SA08] François-Xavier Standaert and Cédric Archambeau. Using subspace-based template attacks to compare and combine power and electromagnetic information leakages. In *CHES*, volume 5154 of *Lecture Notes in Computer Science*, pages 411–425. Springer, 2008.

[Sem15] NXP Semiconductors. Kinetis k82 datasheet, 2015. `https://www.nxp.com/docs/en/data-sheet/K82P121M150SF5.pdf`.

[SLP05] Werner Schindler, Kerstin Lemke, and Christof Paar. A stochastic model for differential side channel cryptanalysis. In *CHES*, volume 3659 of *Lecture Notes in Computer Science*, pages 30–46. Springer, 2005.

[SM16] Tobias Schneider and Amir Moradi. Leakage assessment methodology - extended version. *J. Cryptogr. Eng.*, 6(2):85–99, 2016.

[SMY09] François-Xavier Standaert, Tal Malkin, and Moti Yung. A unified framework for the analysis of side-channel key recovery attacks. In *EUROCRYPT*, volume 5479 of *Lecture Notes in Computer Science*, pages 443–461. Springer, 2009.

[SPY+10] François-Xavier Standaert, Olivier Pereira, Yu Yu, Jean-Jacques Quisquater, Moti Yung, and Elisabeth Oswald. Leakage resilient cryptography in practice. In *Towards Hardware-Intrinsic Security*, Information Security and Cryptography, pages 99–134. Springer, 2010.

[ST18] ST. Stm32l422cb datasheet, 2018. `https://www.st.com/resource/en/datasheet/stm32l422cb.pdf`.

[UHS+18] Florian Unterstein, Johann Heyszl, Fabrizio De Santis, Robert Specht, and Georg Sigl. High-resolution EM attacks against leakage-resilient prfs explained - and an improved construction. In *CT-RSA*, volume 10808 of *Lecture Notes in Computer Science*, pages 413–434. Springer, 2018.

[UHSS17] Florian Unterstein, Johann Heyszl, Fabrizio De Santis, and Robert Specht. Dissecting leakage resilient prfs with multivariate localized EM attacks - A practical security evaluation on FPGA. In *COSADE*, volume 10348 of *Lecture Notes in Computer Science*, pages 34–49. Springer, 2017.

[USS+20] Florian Unterstein, Marc Schink, Thomas Schamberger, Lars Tebelmann, Manuel Ilg, and Johann Heyszl. Retrofitting leakage resilient authenticated encryption to microcontrollers. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(4):365–388, 2020.

[VGRS12] Nicolas Veyrat-Charvillon, Benoît Gérard, Mathieu Renauld, and François-Xavier Standaert. An optimal key enumeration algorithm and its application to side-channel attacks. In *Selected Areas in Cryptography*, volume 7707 of *Lecture Notes in Computer Science*, pages 390–406. Springer, 2012.

[VGS14] Nicolas Veyrat-Charvillon, Benoît Gérard, and François-Xavier Standaert. Soft analytical side-channel attacks. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT (1)*, volume 8873 of *Lecture Notes in Computer Science*, pages 282–296. Springer, 2014.

[VMKS12] Nicolas Veyrat-Charvillon, Marcel Medwed, Stéphanie Kerckhof, and François-Xavier Standaert. Shuffling against side-channel attacks: A comprehensive study with cautionary note. In *ASIACRYPT*, volume 7658 of *Lecture Notes in Computer Science*, pages 740–757. Springer, 2012.

[VSN15] Francisco Veirano, Fernando Silveira, and Lirida Navinery. Is intrinsic noise a limiting factor for subthreshold digital logic in nanoscale cmos? In *2015 International Workshop on CMOS Variability (VARI)*, pages 45–50, 2015.

[Wel47] B. L. Welch. The generalization of 'student's' problem when several different population variances are involved. *Biometrika*, 34(1-2):28–35, 1947.

[YK21] Shih-Chun You and Markus G. Kuhn. Single-trace fragment template attack on a 32-bit implementation of keccak. In *CARDIS*, volume 13173 of *Lecture Notes in Computer Science*, pages 3–23. Springer, 2021.

[ZYSQ13] Yuanyuan Zhou, Yu Yu, François-Xavier Standaert, and Jean-Jacques Quisquater. On the need of physical security for small embedded devices: A case study with COMP128-1 implementations in SIM cards. In *Financial Cryptography*, volume 7859 of *Lecture Notes in Computer Science*, pages 230–238. Springer, 2013.

Part III

Appendices

Additional figures for best attacks' on coprocessors

A

On the left, we present the Guessing entropy (GE) [SMY09] for each key byte and on the right, the estimated rank of the correct key using the algorithm in [PSG16]. We averaged both the GE and the estimated rank over 25 attacks.

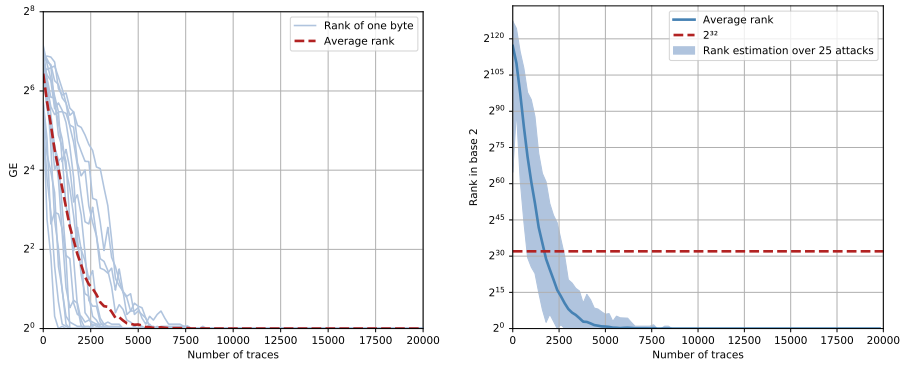


Figure A.1: CPA on the STM at 8MHz with power leakage and CWT.

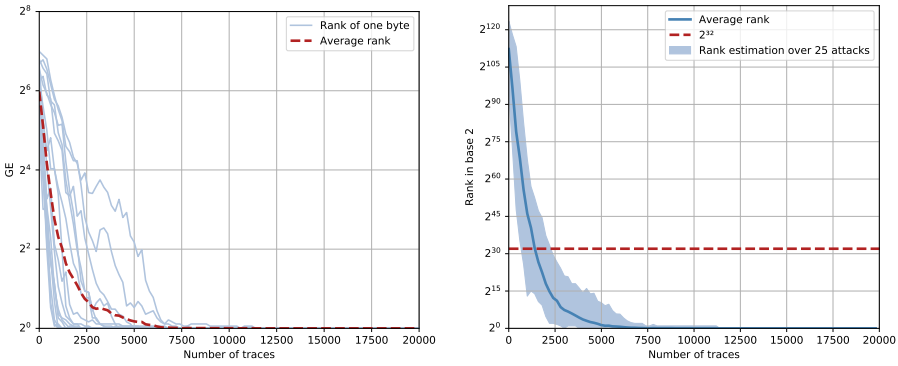


Figure A.2: CPA on the STM at 80MHz with power leakage and CWT.

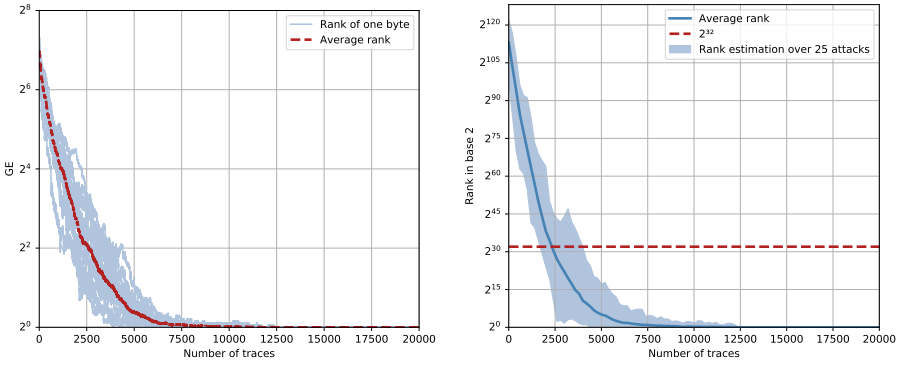


Figure A.3: Template attack on the NXP at 8MHz with EM leakage and CWT.

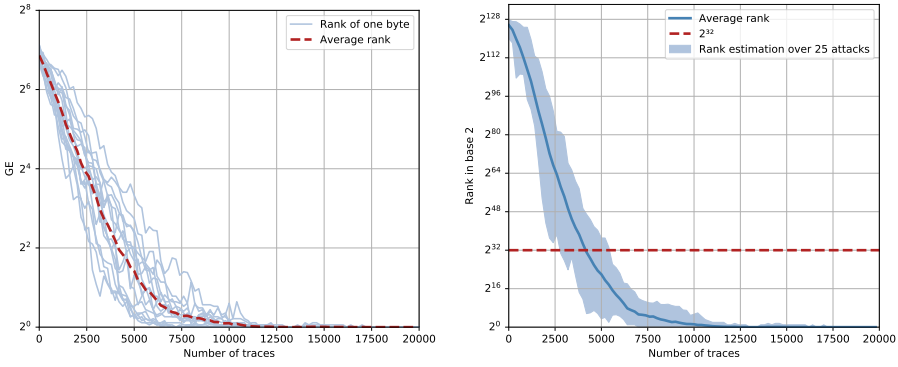


Figure A.4: CPA on the NXP at 100MHz with EM leakage and CWT.